PCS: A Productive Computational Science Platform

David Ojika[1], Ann Gordon-Ross[1], Herman Lam[1], Shinjae Yoo[2], Younggang Cui[2], Zhihua Dong[2], Kirstin Kleese Van Dam[2], Seyong Lee[3], Thorsten Kurth[4]
[1]University of Florida, Gainesville, USA
[2]Brookhaven National Lab., Upton, USA
[3]Oak Ridge National Lab., Oak Ridge, USA
[4]Lawrence Berkeley National Lab., Berkeley, USA

***Abstract*—As modern supercomputers continue to be increasingly heterogeneous with diverse computational accelerators (graphics processing units (GPUs), field-programmable gate arrays (FPGAs), application specific integrated circuits (ASICs), etc.), software becomes a critical design aspect. Exploiting this new computational power requires increased software design time and effort to make valuable scientific discovery in the face of the complicated programming environments introduced by these accelerators. To address these challenges, we propose unifying multiple programming models into a single programming environment to facilitate large-scale, accelerator-aware, heterogeneous computing for next-generation scientific applications. This paper presents PCS, a productive computational science platform for cluster-scale heterogeneous computing. Focusing FPGAs, we describe the key concepts of the PCS platform and differentiate PCS from the current state-of-the-art, propose a new multi-FPGA architecture for graph-centric workloads (e.g., deep learning, etc.) with discussions on ongoing work.**

Keywords—heterogeneous computing, programming model, FPGA, scientific computing, programming environments

I. Introduction

As Moore's law scaling comes to an end, ever increasing processing demands of modern applications can no longer be satisfied with traditional homogenous computing systems. These applications (e.g., big data analytics, artificial intelligence, etc.) are characterized by large data volume and non-traditional processing, often requiring computational accelerators to boost performance, maintain reliability, and meet energy efficiency goals. Consequently, computer vendors are continuing to push for innovative processor and memory system architectures to meets these processing demands. General-purpose graphics processing units (GPGPUs) can be used for workload acceleration. GPGPUs have become a traditional staple in the scientific computing community, which has witnessed an increased need for artificial intelligence (AI) and has introduced an application programming paradigm shift. Recently, there has been an explosive emergence of special-purpose accelerators for AI, from large industry organizations to small startups, using acceleration techniques ranging from soft data processing units (DPUs) with high reconfigurability (e.g., FPGAs) to hard DPUs with less reconfigurability but high performance (e.g., ASICs). This diversity further compounds the complexity in the paradigm shift towards heterogeneous-accelerator-based platforms and the associated *models of computation* (parallel computation models) that guide programmability to simplify usage.

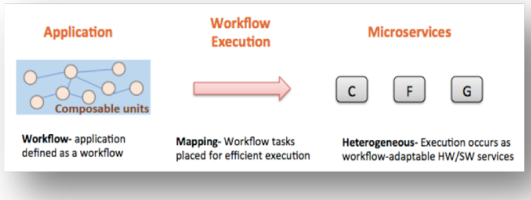

Figure 1. High-level overview of PCS.

Given these emerging architecture trends, no single parallel computation model can effectively leverage heterogeneous-based platforms in a cluster environment due to the diverse user, application, and resource availability requirements. This diversity includes, but is not limited to, software, algorithmic, workflow, data transfer, curation, analysis and accelerator sharing requirements [16]. Overcoming these complexity challenges will require transformative new designs for future supercomputers and systems. Concomitantly, innovative methods are required to flexibly map emerging adaptable AI algorithms to leverage heterogeneous hardware resources without necessitating drastic programming environment changes that would otherwise hinder productivity.

To address these challenges, we propose the productive computational science (PCS (pronounced "Pieces")) platform, which provides a programming abstraction that is accelerator-system agnostic, focusing on scalability and productivity to meet the demands of rapidly changing AI workloads and heterogeneous architectures (Fig 1). Based on the DOE Summary Report from the June 2017 Summit on Extreme Heterogeneity [15], which describes the current state of affairs in heterogeneous systems for scientific computing, we have identified five primary research thrust areas as follows and are correspondingly depicted in Fig 2.

1. Domain-specific language (DSL)
2. Scalable accelerator-aware orchestration
3. Throughput-optimized distributed runtime system
4. Operating system support for scalable heterogeneous systems
5. Programming models for latency-optimized accelerator-to-accelerator intercommunication

II. DESIGN

The PCS platform consists of hardware and middleware layers, together supporting Domain-specific Language (DSL) programming for computational/data-science users. The hardware layer comprises a cluster of FPGAs that accelerates users' applications for fast execution. Within the middleware layer, a partitioned global address space (PGAS) distributed memory abstracts these FPGAs in the global runtime as a unified pool of FPGA computing resources, which in turn provides programmatic interface and runtime system support to the orchestration layer for application task scheduling. Users interact with the orchestration layer to write DSL instructions that dictate how applications execute on the FPGA pool.

A. Motivation

Computational science applications rarely leverage accelerators, such as FPGAs, for workload acceleration, eliminating the FPGA's potential for improving performance and energy efficiency. PCS's goal is to discover Pareto-optimal mappings of application workloads on heterogeneous clusters featuring FPGAs as the primary accelerator. In particular, but without loss of generality, we focus on graph-parallel tasks, such as neural networks to increase system utilization and datacenter efficiency.

To aid this mapping strategy, we propose a novel acceleration architecture, an *edge neural network (ENN)*, based on the bulk synchronous programming (BSP) model. This model is both high-performance and flexible to address the throughput, latency, and energy-efficiency computing demands of modern computational science applications, such as in real-time prediction using deep learning [19][20][21][22].

B. Edge Neural Network

ENN leverages the *edge-centricity* of neural networks to optimally partition and map neural network graphs across a pool of FPGAs while respecting application metrics: throughput, latency, fidelity, energy, and resource utilization. An ENN may be juxtaposed with persistent neural networks (PNNs) [2] with the added advantage of reduced latency. In PNNs, deep neural network (DNN) models are pinned in several FPGA memories with O(1 MB) of on-chip memory per FPGA. Within each FPGA node, an embedded soft-core processor orchestrates instruction scheduling of FPGA- and CPU- targeted portions of the neural network graph. While the soft-core processor can enable flexibility of DNN instruction execution, it places CPU control-logic within the FPGA's DNN dataflow, introducing additional scheduling latency, however, this latency is negligible for non-latency-critical applications.

ENN moves this scheduling operation out of the DNN dataflow into a higher-order orchestration layer—typically a top-of-rack (TOR)/datacenter software—and fuses neural network instructions with latency-optimized message queues using Active Messages [3] to allow for fast stream-based processing across the entire cluster. This architecture deviates from the traditional single program multiple data (SPMD) model of computation commonly employed in modern high-performance computing (HPC) clusters to a systolic, graph-parallel, dataflow pattern with multiple program single data (MPSD).

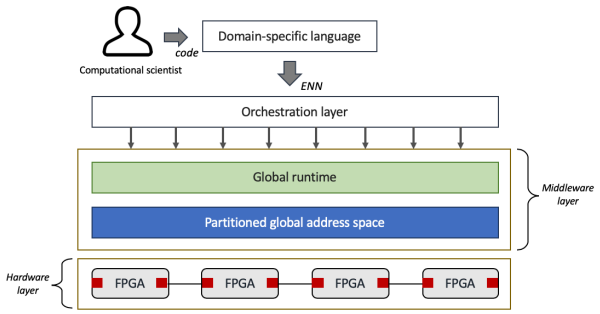

Figure 2. PCS platform architecture supported by five reearch areas, from hardware to domain-specific language.

SPMD, the standard parallelization technique adopted by scientific applications, partitions application data among a collection of worker processes. A bulk-synchronous computation/communication pattern like BSP [4] is used to effect overall computation, typically with the help of message passing. While this technique works well for batch-oriented jobs, it is unsuitable for nontraditional HPC problems, such as machine learning with large, iterative algorithms. On the other hand, an MPSD model follows a *systolic* architecture, partitioning application code across multiple processing elements (PEs), and only allows communication across distinct groups of PEs. This architecture replaces a single processor with an array of PEs interconnected in regular patterns with no control flow and self-orchestrating data movement directly between PEs, which leads to fewer external memory accesses, thus reducing the latency and increasing the throughput.

Therefore, we propose a hybrid SPMD-MPSD FPGA architecture that combines the bulk-synchronous programming model advantage of SPMD (particularly, communication cost modeling) together with the streaming capability of a systolic-based design with the addition of a partitioned global address space (PGAS) memory model for data partitioning and synchronization [5]. As with any distributed system, data partitioning is a key aspect of our proposed multi-FPGA architecture. Although PGASs facilitate distributed shared memory, their data partitioning ideology is not beneficial to our purpose because it defeats the purpose of our hybrid SPMD-MPSD architecture: using inherent dataflow pattern and communication cost to effect computing. Consequently, we can decouple native data partitioning from the PGAS and (statically) define computation code paths within a new bulk-synchronous model that is "SPMD-MPSD-aware." Since the underlying FPGAs must be aware of this new BSP model, we propose GASNet-FPGA, a port of GASNet [6] on FPGAs, to support PGAS on FPGAs, provide remote memory

communication and synchronization, and facilitate MPSD/systolic-based processing.

With this overall ENN architecture (hybrid SPMD-MPSD and GASNet-FPGA), scheduling (instruction fetch and control logic as with PNNs) is completely removed from the DNN dataflow, and the entire DNN program can execute in a true dataflow pattern across multiple FPGAs in a cluster at PE (local FPGA node) processing rate. Using this structure, there is inter FPGA data communication but little to no control dependency between the FPGA nodes. The architecture can even be pipelined at the cluster/system level and hide the implementation details within a customized SPMD-MPSD library so that developers of SPMD-centric applications may continue to use SPMD programming constructs, data parallelism, at increased throughput while the underlying FPGA accelerators execute in MPSD and/or SPMD fashion.

C. Differentiation with Existing Work

1) No intermediary switch network between the FPGAs: Contrary to dataflow architectures like ENNs, a switch-based design adds an additional network hop each time two adjacent FPGAs communicate. While this additional communication might be negligible [2], it is unnecessary for graph-parallel problems with well-defined communication patterns, such as in molecular dynamics [17] or deep learning, which is one of the major application areas of our work. We investigate a foundational set of networking primitives with GASNet-FPGA to facilitate a deterministic, latency-optimized, direct FPGA-to-FPGA communication infrastructure with hardware microservices, obviating the need for CPU involvement in the data plane, supporting multi-tenancy, and increasing overall server resource utilization and application throughput.

2) Decoupling data partitioning from the dataflow: PGAS is a widely used distributed shared memory model in scientific computing applications, allowing large-scale cluster computing. With this memory model, developers can use a tool like Berkeley's UPC and UPC++ [8][9] to write SPMD applications targeting any PGAS backend. This forces the programmer to adhere to strict SPMD-style programming regardless of the underlying implementation of the backend hardware. By separating data partitioning (a core aspect of the SPMD/PGAS model) from the underlying hardware, DNN architectures can be independently optimized against specific FPGA implementation backends and improve performance through a dataflow scheme. We study a systolic-array-oriented architecture (hybrid SPMD-MPSD) that parallelizes and pipelines neural network graphs by sub-graphing graphs across a multitude of FPGAs, while eliminating/minimizing intra sub-graph communication.

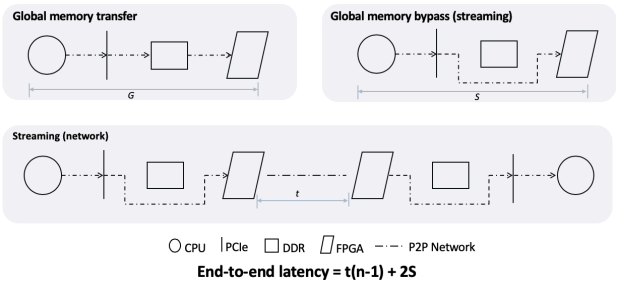

Figure 3. Modeling end-to-end latency in inter-FPGA communication.

3) Using SPMD programming for MPSD computing, which simplifies ease-of-use: Unlike GPUs, FPGA-based acceleration has received little attention in the scientific computing community largely because of the specialized programming and management complexity of using FPGAs in these applications. Many efforts address that this challenge either force users to learn a new programming language or tool chain, or are proprietary and vendor-specific. By providing a programming interface that mimics the widely familiar SPMD programming model and handles data partitioning abstractly, our work can focus on improving user experience and simplifying ease-of-use. We develop a novel performance prediction model and implement this prediction model as a BSP library for UPC/UPC++. The new BSP library allows simplified FPGA-to-scientific-application integration, and facilitates FPGA software-ecosystem (e.g., domain-specific language, orchestration middleware, etc.) development for the scientific computing community.

III. Ongoing Work

Since the overall goal of PCS is to provide a productive computational science platform for accelerating scientific computing, efforts are underway to quantify end-to-end performance and productivity gains using the PCS hardware and middleware layers respectively. For performance characterization, we are currently evaluating MLPerf [14], and plan to evaluate larger models. Larger models with sufficiently large layer counts that comfortably map over a multitude of FPGAs far beyond our current prototype (discussed in the next section) would more accurately characterize the performance gains in terms of performance efficiency and overall latency.

For productivity evaluation, we start with a software-only solution and systematically (using the developed performance model) move selected application partitions into the FPGA and compare both developer productivity—as a result of using the PCS middleware—and application speedup as compared to a state-of-the-art server with a non-ENN-like architecture, such as NVIDIA DGX-2 comprising GPUs with cuDNN library and DeepStream SDK runtime system.

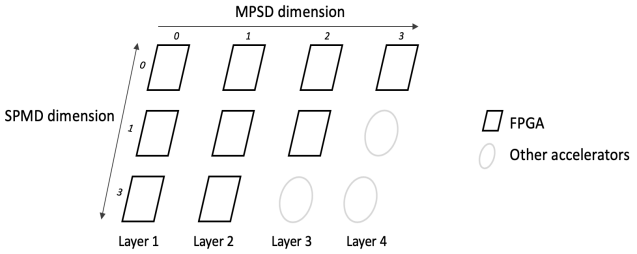

Figure 4. An example layer-wise pipelining of a neural network graph on a compute pool with 9 FPGA accelerators. To minimize inter-graph communication, FPGAs exchange data freely in the SPMD dimension, but less frequently in the MPSD dimension.

Based on the accumulated experience, we will glean detailed insights as to why ENN, and thus a BSP/PGAS-based hybrid model, is a good programming model for deep learning, and potentially graph-parallel algorithms on heterogeneous computing systems, describe minimum infrastructure requirements (e.g., GASNet, Active Messages), and outline a vision for continued exploration of the BSP and PGAS models for a wider scope of applications including genomics, databases, etc.

IV. Preliminary Results

We implemented an initial prototype of PCS, focusing on the hardware layer, particularly the convolutional neural network (CNN) execution on FPGAs. In this section, we discuss an example use-case of running a representative workload on a general-purpose FPGA system with some analysis of the results and preliminary estimation of performance improvement in PCS compared to the original system. To make the example more general, we use the AWS F1 instance with a Xilinx UltraScale+ FPGA and describe our experimental approach for realizing a PCS version.

A. Compute-pool heterogeneity

While CNN graphs consist of compute primitives (e.g., convolution, rectified linear unit, local response normalization, etc.) that readily benefit from hardware acceleration, in some special cases a graph may have primitives that are less amenable to a certain type of compute device, or the hardware version of the primitive may be lacking, and thus require heterogenous execution of the primitives, for example, on FPGAs + CPUs. In such scenario, PCS should hide this heterogeneity from the users and orchestrate data movement between desperate devices. Of course, this can have a negative effect on performance. One approach to minimizing this overhead is exploiting the graph's layer sequencing and mapping successive layers to an ordered arrangement of device types, similar to Fig 4.

B. Layer-wise Pipelining

With latency in mind, a graph may be pipelined by unrolling the graph layer-wise across successive FPGAs - up to the number of layers in the graph - subject to the graph's weight size in bytes. Since the FPGA has limited on-chip memory, multiple FPGAs can be assigned to each layer in the graph as

TABLE I. Computational and Memory Requirements of AlexNet Model

	CONV	*POOL*	*LRN*	*FC*
Total Ops (10^6)	591	0.5	5.5	37.7
Percent (%)	**93**	0.1	0.80	6
Weight size (MB)	10	0.0	0.0	224
Percent (%)	4.3	0.0	0.0	**95**

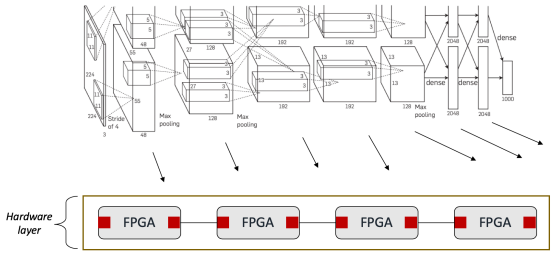

Figure 5. Mapping AlexNet model on PCS.

depicted in Fig 4. With this approach, the layers now have access to fast (aggregate) on-chip memory as well as additional computing resources to further improve throughput. To examine what is the absolute worst-case latency in the *MPSD dimension*, the graph's dataflow path, assuming an aggregate on-chip memory with infinite size (along the *SPMD dimension*), we unroll the graph up to *n* layers, where *n* is the total number of layers in the graph. We then sum the data access time, compute time, and inter-FPGA communication time, as illustrated in Fig 3 to derive the end-to-end latency.

C. Case Study

For the application, we use AlexNet model [23] trained with ImageNet [24] implemented in OpenCL [25]. As shown in Table 1, the model consists of compute primitives: convolution (CONV), pooling (POOL), local response normalization (LRN), and fully-connected (FC). While the CONV primitive is the most compute intensive operation in the model but has relatively low memory footprint (only 10 MB), the FC primitive has majority share of the model size: more than 200 MB, much more than a single FPGA can store in its internal memory. This makes the model a good candidate workload for PCS and layer-wise pipelining as illustrated in Fig 5.

On the general-purpose system (Fig 6), the model executes layer after layer, iteratively mapping all but the FC

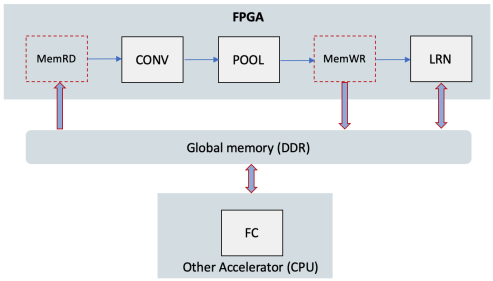

Figure 6. The dataflow architecture of a general-purpose CNN processing system deployed on an AWS F1 UltraScale+ FPGA.

primitive on the FPGA, transferring data back and forth with global memory on each execution of a layer, incurring multiple times of DDR access. The FC primitive, in the last two layers of the model, executes on the CPU. Using the layer-wise pipelining technique, we can eliminate the need for global memory and reduce latency by pinning model weights in the FPGAs' internal block RAMs. With this aggregately larger and faster memory, we can successively map the FC primitive along the SPMD dimension, improving overall latency and throughput.

D. Experiment

We use a single FPGA and assume an infinity large BRAM, in the SPMD dimension, for storing the model weights. This is a fair assumption considering a BRAM is at least an order of magnitude faster than DDR, the global memory of the general-purpose system under test. Because the graph's dataflow is pipelined (Section IV-B), hiding the inter-node latency in the MPSD dimension, we make the assumption of executing each successive model layer (i.e., sub-graph) within our single-FPGA PCS prototype. We then profile the computation and memory access pattern of the model across all its layers and estimate the performance improvement of the SPMD-MPSD layer-wise pipelining technique over the original system as shown in Fig 7.

In total, global memory read/write operations constitutes on average 60% of the total time spent per iteration of the graph. Without batching the input data to hide the DDR latency, this significant overhead makes the general-purpose system inefficient for streaming requests and latency-sensitive applications. Meanwhile, Table 2 shows that the faster, on-chip BRAM memory remains highly underutilized. Conversely, by employing the hybrid SPMD-MPSD design technique and pinning model weights persistently in the BRAM, we project up to 45% improvement in the overall latency while maximizing the on-chip memory utilization.

E. Discussions

1) Middleware support: In our current case-study, we mapped the graph layers to the FPGA, manually; however, this process can be automated by middleware. In PCS, the middleware layer is critically important to deriving the full potential of the FPGA pool, which can be implemented as a set of *microservices*, allowing for the speed, fault tolerance, flexibility and scale of a microservices-based cluster, necessitating the need for a high-performance orchestration system. In future versions of PCS, the orchestration system will be part of a dedicated host sever, interacting with the global runtime system via remote memory access (RMA) protocols.

2) Designing the global runtime system: The global runtime system in part runs on each FPGA host as well as within each corresponding FPGA to form a distinct microservice per model sub-graph. With this sub-graph, the orchestration layer has the task of composing multiple microservices based on the input graph structure supplied by PCS users. In general, the global runtime provides a global view of sets of FPGAs in the cluster: both intra-node (multiple FPGAs interconnected within a node) on the SPMD dimension and inter-node (multiple intra-nodes

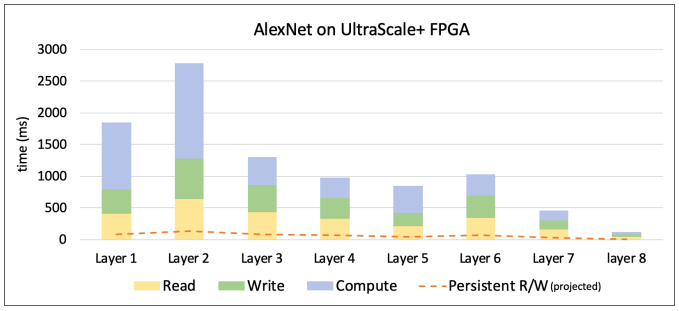

Figure 7. Computation-Communication profile of AlexNet model and the projected performance of pinned model weights (Persistent R/W).

TABLE II. FPGA RESOURCE UTILIZATION

Unit Type	*Used*	*Available*	*Usage %*
Flip Flop	78,048	2,364,000	3.3
LUT	117,736	1,182,000	9.96
DSP	206	6,840	3.01
BRAM	203	75.9 mb	Very low

interconnected via a traditional datacenter switch network such as InfiniBand) in the MPSD dimension. With this unified view, the global runtime system now has the additional task of load-balancing intra-node versus inter-node communication while accommodating for compute-pool heterogeneity and bulk synchronizations.

3) Minimizing synchronization: Decoupling the sub-graphs computation from their bulk synchronization allows us to use the one-sided communication model of PGAS. This provides the benefit of supporting the unstructured computation and irregular communication pattern within the entire graph. Recent trends show that network vendors are increasingly offloading communication protocol stacks onto network hardware, including GASNet protocols. This type of hardware acceleration can allow for the exploitation of fast one-sided communication and allow us to reduce the communication overhead in our proposed GASNet-FPGA implementation, combined with Active Messages to reduce over synchronization. In such scenario, the FPGA hardware should also implement RMA logic.

4) Improving productivity with DSL: Although more advanced users may leverage the proposed ENN-based BSP library and RMA APIs to compose custom applications in UPC++, abstraction with a DSL-based DNN framework such as TensorFlow or Keras will be more impactful. With most existing DNN frameworks supporting the ONNX model format, the PCS DSL can focus on source-to-source translation, inputting graphs, spitting them up into sub-graphs, re-combining the sub-graphs as RMA function calls to generate the final ENN architecture. Separately, the computing units can be written in OpenCL, offline-compiled and synthesized into hardware bitstreams using the traditional OpenCL development flow from vendors such as Intel and Xilinx because the ENN architecture makes PCS vendor-agnostic.

V. CONCLUSIONS

In this paper, we presented a productive computational science (PCS) platform aimed at simplifying the mapping

and programming of scientific workloads on emerging and future accelerator-based heterogeneous systems at cluster-scale. To facilitate the deployment of scientific applications on PCS, we proposed a new programming environment that abstracts individual FPGA accelerators in a cluster and provides a simplified interface for latency-aware, throughput-optimized dataflow computing.

The research contributions of this paper will not only be vital in realizing the principal motivations behind PCS (as discussed in Section II-A), but by extension, provide new insights on programming models, compilers and tools for large-scale scientific computing with FPGAs and other similar accelerators within a heterogeneous computing environment. Other indirect benefits that can be derived from this research include ecosystem support for domain-specific language (DSL) research and development, and integration with science gateways for more widespread community involvement [18].

REFERENCES

[1] Guo, Liucheng & Cross, Andreea-Ingrid & B. Thomas, David & Fu, Haohuan & Luk, Wayne. (2016). Parallel Genetic Algorithms on Multiple FPGAs. ACM SIGARCH Computer Architecture News.

[2] [2] Accelerating Persistent Neural Networks at Datacenter Scale Microsoft, http://isfpga.org/fpga2018/slides/FPGA18_Panel_Talk_Eric_Chung.pdf

[3] T. v. Eicken, D. E. Culler, S. C. Goldstein and K. E. Schauser, "Active Messages: A Mechanism for Integrated Communication and Computation," *[1992] Proceedings the 19th Annual International Symposium on Computer Architecture*, Gold Coast, Australia, 1992

[4] T. Cheatham, A. Fahmy, D. C. Stefanescu and L. G. Valiant, "Bulk synchronous parallel computing-a paradigm for transportable software," Proceedings of the Twenty- Eighth Annual Hawaii International Conference on System Sciences, Wailea, HI, USA, 1995

[5] R. Matsumiya and T. Endo, "PGAS Communication Runtime for Extreme Large Data Computation," 2016 Second International Workshop on Extreme Scale Programming Models and Middlewar (ESPM2), Salt Lake City, UT, 2016Dan Bonachea, Paul H. Hargrove, "GASNet-EX: A High-Performance, Portable Communication Library for Exascale", Languages and Compilers for Parallel …

[6] [6] www.gasnet.lbl.gov

[7] M. M. Deneroff *et al*., "Anton: A specialized ASIC for molecular dynamics," *2008 IEEE Hot Chips 20 Symposium (HCS)*, Stanford, CA, 2008

[8] https://upc.lbl.gov

[9] Y. Zheng, A. Kamil, M. B. Driscoll, H. Shan and K. Yelick, "UPC++: A PGAS Extension for C++," *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, Phoenix, AZ, 2014

[10] June 2017 Summit on Extreme Heterogeneity. https://orau.gov/exheterogeneity2018/2018-Extreme-Heterogeneity-BRN-report-final.pdf

[11] J. J. Willcock, T. Hoefler, N. G. Edmonds and A. Lumsdaine, "AM++: A generalized active message framework," *2010 19th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Vienna, 2010.

[12] Ruediger Willenberg and Paul Chow. 2014. A Heterogeneous GASNet Implementation for FPGA- accelerated Computing. In Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models (PGAS '14). ACM, New York, NY, US

[13] Filip Blagojević, Paul Hargrove, Costin Iancu, and Katherine Yelick. 2010. Hybrid PGAS runtime support for multicore nodes. In Proceedings of the Fourth Conference on Partitioned Global Address Space Programming Model (PGAS '10). ACM, New York, NY, USA

[14] MLPerf. https://mlperf.org

[15] June 2017 Summit on Extreme Heterogeneity. https://orau.gov/exheterogeneity2018/2018-Extreme- Heterogeneity-BRN-report-final.pdf

[16] E. Colmenares, P. Andersen and B. Wei, "An Overlap Study for Cluster Computing," 2015 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, 2015.

[17] Anton. https://www.psc.edu/resources/computing/anton

[18] D. Ojika, H. Lam, A. Gordon-Ross, B. Patel, "SCAIGATE: Science Gateway for Scientific Computing with Artificial Intelligence and Reconfigurable Architectures", Gateways 2018, Austin, TX

[19] R. Diwan, A. Rizvi, S. Bhattacharyya, D. Katramatos and K. G. Yager, "Application of Analysis on the Wire to Streaming NSLS-II Beamline Data," 2018 New York Scientific Data Summit (NYSDS), New York, NY, 2018

[20] R. C. Sah, "The Advanced Light Source," in *IEEE Transactions on Nuclear Science*, vol. 30, no. 4, pp. 3100-3102, Aug. 1983.

[21] Aimone, James. (2019). Neural algorithms and computing beyond Moore's law. Communications of the ACM.

[22] S. A. Jacobs, N. Dryden, T. Moon, B. Van Essen, S. He, J. Allen, "Scaling Deep Learning for Cancer Drug Discovery on HPC Systems", The 7th International Workshop on Parallel and Distributed Computing for Large Scale Machine Learning and Big Data Analytics Vancouver, British Columbia , Canada May 21, 2018

[23] Krizhevsky A, Sutskever I, Hinton GE. ImageNet Classification with Deep Convolutional Neural Networks. Communications of the ACM. 2017

[24] ImageNet: A large-scale hierarchical image database. 2009 IEEE Conference on Computer Vision and Pattern Recognition, Computer Vision and Pattern Recognition, 2009 CVPR 2009 IEEE Conference on. 2009

[25] PipeCNN: An OpenCL-based open-source FPGA accelerator for convolution neural networks. 2017 International Conference on Field Programmable Technology (ICFPT), Field Programmable Technology (ICFPT), 2017 International Conference on. 2017