Heterogeneous Computing Systems for Complex Scientific Discovery Workflows

Christoph Hagleitner, Dionysios Diamantopoulos, Burkhard Ringlein
IBM Research - Europe
Ruschlikon, Switzerland
{hle,did,ngl}@zurich.ibm.com

Constantinos Evangelinos, Charles Johns, Rong N. Chang, Bruce D'Amora, James A. Kahle, James Sexton
IBM Research
Yorktown Heights, USA
{cevange, crjohns, rong, damora, jakahle, sextonjc}
@us.ibm.com

Michael Johnston
IBM Research - Europe
Dublin, Ireland
michaelj@ie.ibm.com

Edward Pyzer-Knapp, Chris Ward
IBM Research – Europe
{Warrington, Hursley} UK
{EPyzerK3,tjcw}@uk.ibm.com

***Abstract*— With Moore's law progressively running out of steam, heterogeneous computing architectures have been powering the top supercomputers in the world for many years and are now finding broader adoption across the industry. The trend towards sustainable computing also requires domain-specific heterogeneous hardware architectures, which promise further gains in energy efficiency. At the same time, today's high performance computing applications have evolved from monolithic simulations in a single domain to multidisciplinary complex workflows. In this paper, we explore how these trends affect system design decisions and what this means for future computing system architectures.**

Keywords*— *heterogeneous computing, high-performance computing, accelerators, workflow, disaggregation

I. Introduction

For the past decades, high-performance computing (HPC) systems have been following Moore's law all the way from the GFlop systems of the early nineties to the recent PFlop systems. Today, technology scaling and architectural changes to homogeneous computing systems do not provide sufficient performance and energy efficiency gains to continue the roadmap beyond the ExaFlop landmark in an economically and ecologically sustainable way. Hence progress towards sustainable HPC in the exascale era relies on architectural innovations. These innovations range from incremental micro-architectural innovations in programmable accelerators like GPUs and FPGAs, through domain specific ASICS for machine-learning (ML) and artificial intelligence (AI) applications, to the emergence of entirely new computing paradigms like quantum computing.

Beyond technology and architecture, there are two trends which are also influencing the evolution of HPC systems:

1. HPC systems are no longer restricted to running large-scale simulations. They are increasingly used to support every aspect of (scientific) discovery.
2. The trend towards a cloud deployment and cloud operating model has arrived in the HPC domain. Future systems will be deployed on hybrid clouds, combining on-prem installations with suitable extensions into the public cloud.

All of these technical innovations and the trend towards complete scientific workflows increase the complexity of developing and deploying applications on HPC systems. This complexity needs to be "hidden" from the user by providing a seamless, integrated development environment that supports users in deploying complex workflows on heterogeneous computing systems.

In this paper, we review the current state of the art and propose a direction towards continued performance gains for sustainable HPC systems. The paper is organized as follows: Section II introduces scientific discovery workflows that typify the new complex computational problems being deployed on today's supercomputers, and describes the difference of their performance characteristics when compared to classic HPC applications. Section III looks at the evolution of HPC systems and the emergence of complex scientific discovery workflows. Finally, Section IV looks at the programming models and runtime systems, which tie the two components together.

II. Scientific Discovery Workflows

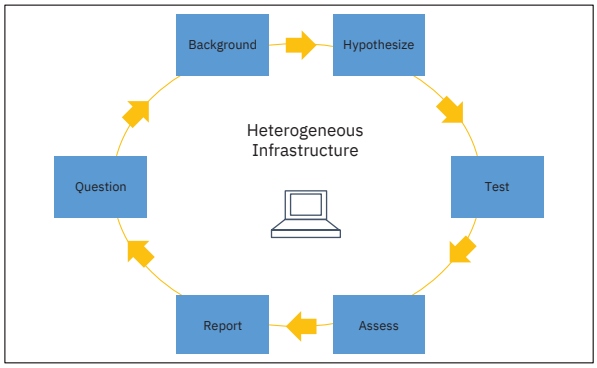

Fig. 1. The cycle of scientific discovery

HPC has transformed the way we attempt scientific discovery throughout the past decades by unleashing ever more compute power to simulate complex materials, run extended weather forecasts, or advance the understanding of molecular dynamics. Until recently, the use of computing for scientific discovery was limited to large-scale simulations. In the cycle of scientific discovery depicted in Fig. 1, this corresponds to the "Test" and "Assessment" phases. The advances in disciplines like natural language understanding, large-scale data analytics, knowledge management (incl. knowledge graphs), and AI (e.g., for generative models) have expanded the application domain of computing to literally all elements of scientific discovery. Furthermore, the ability to build large models has helped to assess and exploit huge datasets, which were not accessible before. As a consequence, contemporary HPC systems are increasingly used to run complex workflows across several steps in the cycle, instead of just running a simulation to confirm a hypothesis.

Such a workflow was used to, e.g., model RAS initiation pathways for cancer on a large supercomputer [1]. Using AI and knowledge graph technology, we can take the workflows one step further and also integrate the background knowledge as well as the ability to explore a hypothesis within a workflow: In the case of material discovery, we first ingest a body of knowledge into the system to build a knowledge graph representing the global knowledge base. After querying the knowledge base and identifying gaps, the system can determine a set of virtual experiments to close the gaps in the knowledge base and then derive a set of candidate materials, which are sufficiently promising to advance to real-world synthesis and testing. In the following paragraphs, we describe the essential components required to compose these workflows in more detail.

A. *Workflow Management and Virtual Experiments*

The need to define and orchestrate complex pipelines of programs, or workflows, is recognized across a range of science and engineering domains and has led to a multitude of workflow management systems and tools evolving to address this need (e.g., [12,13]). This reflects the desire to compose flexible workflows from existing complex applications and, hence, reliably automate processes. This approach stands in opposition to extending and enriching the features of existing standalone programs, which leads to ever larger monoliths with overlapping feature sets.

Workflows can express *virtual experiments*, a sequence of computational steps, that perform an in-silico analogue of a real-world experiment. These can then be directly vended to end-users as shown in Fig. 2. This provides substantial efficiency gains and disrupts the usual approach to scientific discovery, where the experimentalist describes the problem to the computational group coding and implementing the workflow. This approach has been demonstrated in a number of proof-of-technology engagements with industry partners working in the formulation chemistry domain [13].

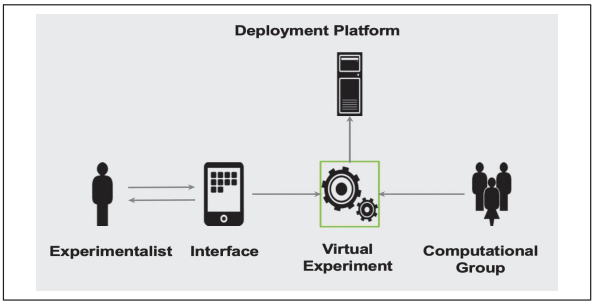

Fig. 2. *Virtual experiments: A new model for consuming computational methods*

B. *Enhancing Virtual Experiments Through ML and AI*

The ability to increase the size, accuracy, and complexity (e.g., through multi-physics) of simulations remains a key driving factor behind the demand for more performance and energy efficiency. Beyond the use of accelerators, there are a number of ways in which the efficiency of a modelling and simulation workflow can be improved through the addition of ML and AI. In some cases, it is possible to replace an entire simulation with an inference from a suitably trained model. Recent examples of this approach include quantum-chemical simulations [4], the prediction of chemical reaction outcomes [6] or the prediction of protein folding [7]. Given that inference is a relatively low energy operation, especially when compared to the execution of an entire HPC workflow, it is clear that there is the potential for significant gains here. However, there are some caveats to this statement: (i) It is well documented that for some high-performing deep learning models, the energy cost of training is very large, and so it would require a large number of inferences to recoup the initial energy outlay of training the model. (ii) Without an uncertainty estimator, an inference determined by a deep learning model cannot support high quality decision making, because there is no warning mechanism. (iii) The goal of a simulation is often not only to obtain the final regressable outcome, but more about providing insight into the mechanism targeted by the simulation. For this purpose, a single black-box model is unsuitable.

Whilst ML and AI provide many opportunities for executing simulation workflows faster, a key observation is that «you cannot go faster (or greener) than the workflow you don't have to run, because you are smart enough to avoid it». This can be accomplished by removing the duplication of effort across a user base. A powerful method for this is known as memoization. Here, we treat the workflow as a graph, and before execution of the workflow occurs, the graph is unrolled and matching sub-graphs from previously run workflows are identified and recycled where appropriate.

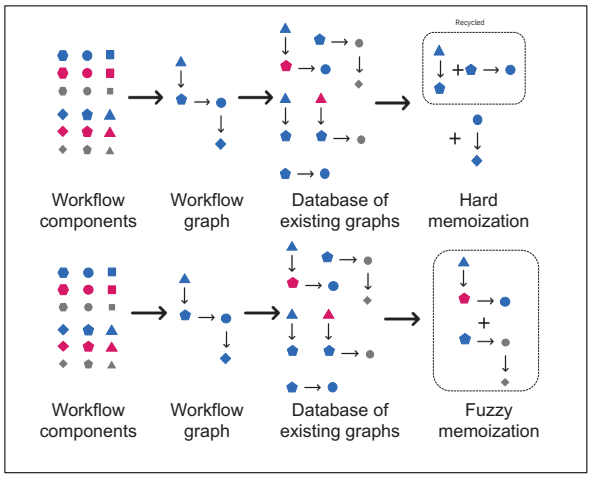

Fig. 3. Schematic representing hard and fuzzy memoization

We expand the more traditional 'hard' memoization, where exact subgraph matches are required, to 'fuzzy' memoization, where near matches are also considered. This drives significant sustainability gains in modelling and simulation as there are often multiple, equally appropriate, routes to a solution and fuzzy memoization allows flexibility in the path taken to maximize reuse. We believe that AI can be used to identify these alternate routes in an increasingly automated manner, but also has applications beyond memorization: We had significant success using a technique called Bayesian optimization to balance the double pressures of exploitation of current knowledge and exploration of new knowledge [5]. By using a Bayesian model to form what is known as an acquisition function – a score which rates how important a data point is to collect – and iteratively updating this model as data flows in, we are able to reduce the number of data points collected by a significant margin – often an order of magnitude.

C. *Deployment of Scientific Discovery Workflows in Hybrid Cloud Environments*

Cloud native programming platforms offer some distinct advantages over traditional conventional on-premise clusters. Deploying containerized applications on Kubernetes-based platforms such as Red Hat® OpenShift® offer the developer and end-user the benefit of improved portability, dependency management and runtime isolation. On a services-based platform like OpenShift, complex workflows consisting of services and applications can be deployed with high reliability, because Kubernetes is based on a declarative state model that constantly monitors and restores to the user-requested state with the correct resources required for the workflow to execute optimally. The platform scales the application resources up or down depending on resource requests and availability over time.

In the domain of simulations and virtual experiments, many applications rely on the message-passing-interface (MPI) standard to scale-up and scale-out across large clusters. IBM Research developed two additions to run MPI-applications on the OpenShift platform. Fig. 4 shows a schematic of the components that were developed to enhance OpenShift with MPI support: (1) an *MPI development and runtime environment* and (2) an *MPI Controller*.

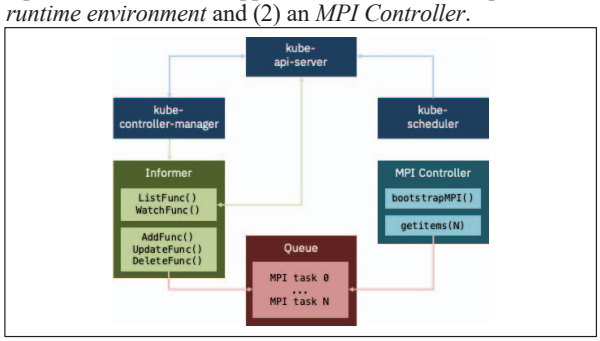

Fig. 4. Kubernetes MPI Integration

The goal of the two additions was to provide support for developers and users who want to migrate an existing MPI application to the Red Hat OpenShift platform. The developers would first need to containerize their application code. Since we are focusing on MPI applications, a base docker image that included an MPI development and runtime environment would be required. The base recipe for creating the image consisted of a Dockerfile and several scripts for building the image for user specified project names (namespaces). The second addition was the MPI controller. The MPI controller is packaged as a separate Docker image and is responsible for launching the distributed application tasks across the OpenShift cluster. It is essentially a Kubernetes controller as it interfaces with the Kubernetes API server to query information about the application that contain the MPI application.

III. Heterogeneous Computing Infrastructure

In section II, we described the evolution from HPC simulations towards complex workflows executed on HPC infrastructure. In this section, we are going to explore how this is reflected in the architecture of current and future HPC computing systems. The new workflows include (i) large knowledge graphs, which are memory and memory bandwidth intensive, (ii) training of and inference on AI models, which access large data-sets and have high compute intensity, and (iii) the execution of classic simulation codes, which are either memory or compute bound. Beyond the capability to support a broad set of workflows, the future HPC systems must continue to scale their compute performance in line with comparable efficiency gains to avoid excessive energy consumption. In times of diminishing energy efficiency gains provided by technology scaling, these performance and efficiency gains can only be obtained through the incorporation of specialized accelerators.

A. *Accelerated Systems: GPUs, FPGAs, ASICs and beyond*

Many of the recent advances in compute performance and efficiency were enabled by throughput-oriented compute devices – in practice mainly GPUs. The performance gains result from their very wide single-instruction-multiple-data/thread (SIMD / SIMT) units, their latency-tolerance via threading and their increased memory bandwidth. While earlier generations of GPU architectures were driven by the needs of graphics users (mainly gamers), more recent architectures have been adopting features suited for general purpose computations and especially HPC and Deep Learning (DL). DL relies on tensor operations. Thus, we are witnessing the addition of specialized compute arrays for matrix-matrix multiplication operations (necessary for tensor contractions), and support for lower arithmetic precision (IEEE half and bfloat16) with a range of mantissa and exponent options. GPUs are also beginning to offer integer operand support in the matrix-multiplication units when the application does not require floating point calculations. As DL applications exhibit tolerance for lower precisions, tricks such as transparent rounding to intermediate precisions [23] for matrix multiplications allow further speedups out of the same number of logic gates. As a result of such developments, GPUs have further increased the achievable compute density from the 10 TOPs range for double precision floating point arithmetic into the 1 POPs range for low-precision integer operations.

These trends have influenced architectural design beyond GPUs: The *Fugaku* supercomputer [24], a recent #1 entry on the Top500 list is no exception to this trend. While it is a homogeneous system in the sense that it is built from a single ARM® ISA based CPU chip, it employs two very wide SIMD units capable of operating on different precision operands, including half precision FP and integers of different sizes, and has on-package high-bandwidth memory (HBM). Both elements are also present in GPUs targeted at HPC applications.

DL workflows – especially workflows involving inference – have such a specialized structure that they can be well served by architectures based on systolic arrays of functional units. Early examples are Google®'s TPU family, which forgoes generality of computation for increased execution efficiency. The prospects for a huge DL inference market and – to a lesser extent – DL training in a broad range of scientific and enterprise applications has given rise to a Cambrian explosion of specialized accelerators and new technologies [11]. Examples include Graphcore® [9], Cerebras® [8], IBM® [10], and AWS® Inferentia. Similar to the trend for GPUs, standard CPUs have started to incorporate in-core DL accelerators to improve the performance of DL applications without requiring separate accelerator boards. Initially this was the case for CPUs targeted at mobile applications, but VIA's Centaur Technologies have incorporated such a specialized unit in their latest x86_64 CPUs and Apple®'s new

laptop/desktop M1 processors also offer such capabilities. Both, Intel® and IBM, have also introduced a form of matrix multiplication primitive instructions in the ISAs of their next generation processors.

The trend towards more diverse workflows has raised the interest in different architectural elements, which enable performance and efficiency gains beyond CPUs and SIMD/Tensor operation accelerators. In particular, tasks like data preparation, data-analytics on text [25] and image data and low-precision ML inference tasks can be mapped well to spatial architecture with fine-grained reconfigurable elements, e.g, FPGAs. While FPGAs are already widely deployed for applications in some areas, e.g., genome sequencing and data analytics, they still lack the full development environment and run-time support required for broad adoption in scientific discovery workflows. This will be addressed in section IV.

The trend towards more specialized architectures (domain specific as well as different architectural elements) does not stop there. The requirements of weather codes (and other PDE solvers) for efficient stencil operations are resulting in the emergence of stencil accelerators [26]. Specialized ASICs for molecular dynamics/cosmology/many body calculations have been around since the 1990s (GRAPE-* family) and have evolved into specialized complete machines [2]. Numerous accelerators based on coarse-grained reconfigurable arrays are targeting ML/AI applications, complemented by some recent accelerators based on in-memory computing. In the long term, entirely new paradigms like quantum computing [27] are going to further increase the number of architectural options and will continue to advance the performance and efficiency of HPC. In order for those architectures to make an impact for the new, complex workflows in scientific discovery applications, it will be essential to seamlessly integrate them into the evolving development and runtime environments for HPC.

Faced with this cornucopia of choices for executing individual workflow tasks, the emerging problem on the system side is to design an architecture that is open to incorporating diverse computing elements but at the same time also able to use them efficiently and connect them at high bandwidth and low latency. This is where we see the utility of an architecture that uses a high performance fabric and an intelligent infrastructure to compose the right system for the right task.

B. *Composable Systems with High-performance Fabric*

HPC Systems are typically constructed from traditional monolithic servers, where each server contains a fixed set of resources (CPUs, Memory, Acceleration, and Storage). As the diversity of workloads increases, and knowledge-based solutions are introduced, providing the right mix of configurations in the data center without a tremendous waste of resources becomes increasingly difficult. The growing number of heterogeneous system components (accelerators, NV memory, etc.) further multiplies the number of possible system configurations. To address this challenge, our goal is to replace the traditional monolithic servers with pools of resources (CPUs, Memory Acceleration, and Storage). Servers are then dynamically composed from these pools of resources into servers with just the right set of resources to meet the workloads requirements. The composable server not only makes better use of the resources but also provides greater power efficiency by allowing any unused resources to be powered down.

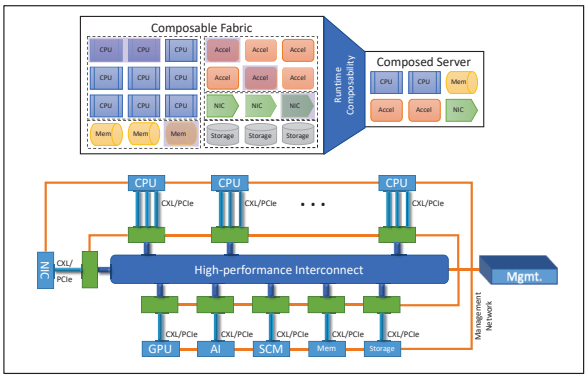

Fig. 5. High-performance fabric for composable systems

The concept of a composable system is not new. The idea has been discussed by system architects for well over a decade with technology being the major hurdle preventing its realization. Today, there are examples of partially disaggregated systems using PCI-Express switches (I/O only disaggregation). With the introduction of CXL, the technology needed to realize a true composable system (I/O, Memory, and Storage disaggregation) is starting to emerge. Our goal is to leverage the CXL standard to provide a high-performance, coherent fabric for creating a composable system architecture. With the introduction of 32 Gb/s and 64 Gb/s signaling, PCIe Gen 5 and PCIe Gen 6 respectively, the CXL protocol will provide the latency and bandwidth needed for the disaggregation of system resources, making composable system a reality.

C. *Disaggregated, Standalone Accelerators using FPGAs*

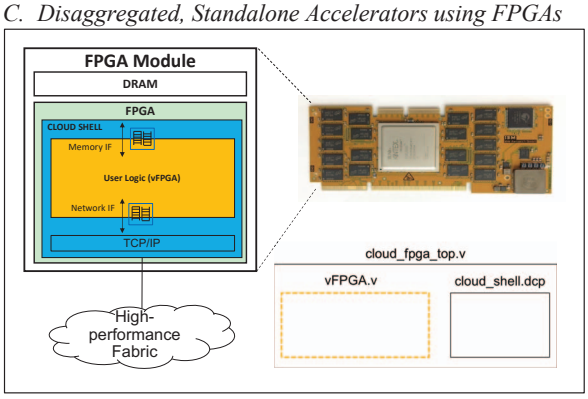

Fig. 6. *cloudFPGA*, a disaggregated, standalone FPGA accelerator platform

Depending on the application (monolithic application vs. microservices) and the programming model, a disaggregated accelerator can become a part of a composed server or provide a standalone microservice. In the latter configuration, the accelerator implements all functions to offer a complete microservice, e.g., via a REST API or an MPI rank (see section IV-C). An example of a disaggregated accelerator platform is *cloudFPGA* (see Fig. 6, [22]), which is based on disaggregated standalone FPGAs. The platform introduces a shell-role architecture, which allows the developer to focus entirely on implementing his FPGA-accelerated kernel in the "role", while everything else (incl. the integration with the data-center's control-plane) is abstracted away through the "shell" provided by the platform provider. This offers ultimate efficiency not only through the accelerated kernel, but also by minimizing the substantial platform overhead of common

accelerated servers: There is no CPU/memory with its operating system overhead and there is no need for an additional NIC with its network integration overhead.

IV. Development Environment for Scientific Discovery Workflows

A. *Development for heterogeneous computing systems using Jupyter workbooks*

The transition to complex workflows and heterogeneous systems implies a significantly more challenging programming and development environment. Most domain scientists (e.g., a researcher in systems biology or physics) do not have the knowledge to configure and tune complex workflows on heterogeneous systems. The knowledge intersection among domain scientists, computational scientists and computing infrastructure professionals as shown in Fig. 7a is small and the development environment has to bridge the gaps and provide "ease-of-use". We address this by using Jupyter Notebooks technology [14] as an interactive way of programming. The technical details of the computing workflow execution are delegated to the Workflow Management System.

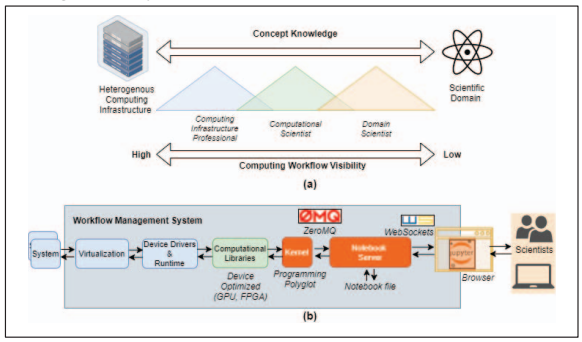

Fig. 7. Jupyter Notebooks as a scientific gateway tool for advanced heterogenous computing systems

Notebooks are documents produced by the Jupyter Notebook App, which contain both computer code (e.g. python) and rich text elements (paragraphs, equations, figures, links, etc…). Notebook documents are both human-readable documents containing the analysis description and the results (figures, tables, etc.) as well as executable documents that can be run to perform data analysis. The code contained in a Notebook document is executed by a kernel. A notebook kernel is a "computational engine" for the associated programming language and is communicating with the Notebook server through websockets and the ZeroMQ library [15,16], which provides the low-level transport layer over which the messages are sent. As of today, the most popular languages, including Python, Fortran, R, C/C++, Javascript, Scala, Go, etc. are supported and provide a programming environment. While the standard Notebooks distribution allows the serving of a notebook by a single Kernel, Script of Scripts (SoS) [17] is an example of Jupyter Notebook extensions with support for multiple Kernels per notebook that share variables among the kernels/languages. We propose the extension of Notebook Kernels with the support of libraries optimized for the emerging heterogeneous components, e.g., GPUs and FPGAs. Today, there are point solutions integrating GPUs [18] and FPGAs [19] in Notebooks, but in the future we will need open standards to enable the combination of different components at the design-time or run-time in order to improve the overall system efficiency [20,21]. We believe the method used in the Jupyter Notebooks can serve as a blueprint for the interaction with heterogenous computing systems on which scientific workflows are deployed with openness and interoperability.

B. *Service-oriented Heterogeneous Computing Architecture for Scientific Discovery on Hybrid Cloud*

The complexity of building, deploying, and managing scientific discovery workflows on heterogeneous computing systems, while providing self-service and quality assurances and fulfilling security policies, has been increasing exponentially. Existing monolithic HPC applications have caused various development and deployment issues for the users. Moreover, conventional HPC application development and deployment practices do not facilitate decomposing an HPC application in terms of heterogeneous infrastructure capabilities, nor do they facilitate composing independently developed scientific discovery workflows with quality assurance, which is key to enterprise computing.

We have designed and implemented a service-oriented heterogeneous computing architecture [3] with the following design principles: (1) individual administration, (2) client-facing separation of control and data planes, (3) eventual consistency of service states, and (4) individually administered composite services. In a hybrid-cloud based computing environment, every container-based OpenShift microservice has its own DevOps and operations management processes so that functional capabilities and non-functional properties can be defined, measured, and managed at its service endpoint, e.g., a REST API endpoint. From the viewpoint of service client applications, the service API endpoint can be different from that used for transferring input/output data, which can be large and result in significant operations management costs and serviceability issues. As per the CAP theorem [28], we adopt various state-of-the-art eventual consistency based distributed computing technologies (e.g., OpenShift, MongoDB, etc.) and do not attempt to assure state consistency across the whole system all the time. Finally, when composing hybrid cloud service APIs, manageability of the consumption relations between the composite service and the constituent services is assured such that the composite service can be consumed as one individually administered service.

C. *Programming model and scaling for heterogeneous, disaggregated computing systems*

In general, there are two approaches for programming models: Shared Memory and Distributed Memory. The first is used by languages like OpenCL, OpenMP and CUDA and assumes that all processes or nodes of a program can access the same (global) memory and consequently, programmers can access the memory of other nodes/processes directly, with implicit synchronization. The latter is used by the MPI standard and requires an explicit message interchange, done by the programmer, to synchronize data. While MPI is built on the distributed memory approach, its optimized implementations include optimizations to utilize shared memory (where available) for performance gains.

Many HPC applications combine the two approaches by using OpenMP to scale applications within a core or multi-core processor and MPI to scale beyond. For systems including accelerators, where shared memory is usually not available, the distributed memory approach is used. Consequently, we believe that MPI should be extended to

include heterogeneous nodes beyond the available GPUs and developed a proof of concept for the disaggregated FPGAs introduced in section III-D.

ZRLMPI [29] is a framework that allows the deployment of MPI applications on a cluster of FPGA and CPU nodes without code modifications (see Fig. 8). Using ZRLMPI for a simple 2D stencil application on 8 nodes with optimized tree communications, the CPU-FPGA cluster achieves a speedup of 19 over a CPU-only cluster implementation.

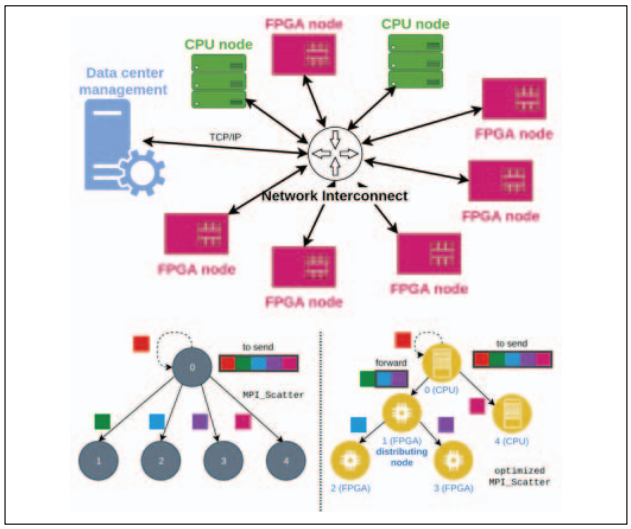

Fig. 8. ZRLMPI proof of concept with MPI_scatter optimization

V. Summary

The shift from "classic" HPC towards complex workflows for scientific discovery running on heterogeneous hybrid cloud infrastructure demands changes not only to the system architecture of future computing systems, but also to the software architecture and development environment. We show that novel workflow management techniques are needed to deploy workflows on a co-designed HW/SW platform with a microservices-inspired SW-architecture. Composable HW-systems built on top of pools of disaggregated system components offer an interesting paradigm to tackle the challenges ahead. Finally, a multi-tier high-performance fabric based on open standards is required to efficiently connect all system components across the data center and between cloud installations.

References

[1] F. Di Natale, et.al., "A massively parallel infrastructure for adaptive multiscale simulations: modeling RAS initiation pathway for cancer," *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (*SC '19*). ACM, New York, NY, USA, 2020, Article 57, pp. 1–16.

[2] David E. Shaw, et.al., "Anton 2: Raising the Bar for Performance and Programmability in a Special-Purpose Molecular Dynamics Supercomputer," *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC14)*, Piscataway, NJ: IEEE, 2014, pp. 41–53.

[3] Rong N. Chang, Kumar Bhaskaran, Prasenjit Dey, Hsianghan Hsu, Seiji Takeda, Toshiyuki Hama, "Realizing A Composable Enterprise Microservices Fabric with AI-Accelerated Material Discovery API Services," *Proceedings of the IEEE International Conference on Cloud Computing (CLOUD)*, IEEE, 2020, pp. 313–320.

[4] E.O. Pyzer-Knapp, K. Li, A. Aspuru-Guzik, "Learning from the Harvard Clean Energy Project: The Use of Neural Networks to Accelerate Materials Discovery". *Adv. Funct. Mater*., 2015, 25, pp. 6495-6502.

[5] J.M. Hernández-Lobato, J. Requeima, E.O. Pyzer-Knapp, A. Aspuru-Guzik, "Parallel and Distributed Thompson Sampling for Large-scale Accelerated Exploration of Chemical Space". *Proceedings Intl. Conference on Machine Learning,* PMLR, 2017, pp. 1470-1479

[6] Philippe Schwaller, Théophile Gaudin, Dávid Lányi, Costas Bekas, Teodoro Laino. "Found in Translation: predicting outcomes of complex organic chemistry reactions using neural sequence-to-sequence models". *Chemical Science* **2018,** *9* (28) , 6091-6098.

[7] Andrew W Senior, et.al., "Improved protein structure prediction using potentials from deep learning". *Nature* 577, 7792 (2020), 706–710. DOI:https://doi.org/10.1038/s41586-019-1923-7

[8] Cerebras wafer scale engine, https://www.cerebras.net/product/, Online, Accessed 4 Dec. 2020

[9] Graphcore IPU, https://www.graphcore.ai/products/ipu, Online, Accessed 4 Dec. 2020

[10] J. Oh et al., "A 3.0 TFLOPS 0.62V Scalable Processor Core for High Compute Utilization AI Training and Inference," *2020 IEEE Symposium on VLSI Circuits*, Honolulu, HI, USA, 2020, pp. 1-2

[11] L. Traversa and M. Di Ventra, "Universal Memcomputing Machines," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 11, pp. 2702-2715, Nov. 2015.

[12] Argo workflows, https://github.com/argoproj/argo, Online, Accessed 4 Dec. 2020

[13] J.L. McDonagh, W.C. Swope, R.L. Anderson, M.A. Johnston, D.J. Bray, "What can digitisation do for formulated product innovation and development?" *Polym Int.,* 2020, https://doi.org/10.1002/ pi.6056

[14] https://jupyter.org, Online, Accessed 11 Nov. 2020

[15] https://www.websocket.org/aboutwebsocket.html, Online, Accessed 11 Nov. 2020

[16] https://zeromq.org , Online, Accessed 11 Nov. 2020

[17] https://vatlab.github.io, Online, Accessed 11 Nov. 2020

[18] https://rapids.ai , Online, Accessed 11 Nov. 2020

[19] https://pynq.readthedocs.io , Online, Accessed 11 Nov. 2020

[20] Roger D. Chamberlain, "Architecturally Truly Diverse Systems: A Review," *Future Generation Computer Systems*, 110:33-44, September 2020. DOI: 10.1016/j.future.2020.03.061.

[21] D. Diamantopoulos and C. Hagleitner, "HelmGemm: Managing GPUs and FPGAs for Transprecision GEMM Workloads in Containerized Environments," *IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, New York, NY, USA, 2019, pp. 71-74, doi: 10.1109/ASAP.2019.00-27.

[22] J. Weerasinghe, F. Abel, C. Hagleitner and A. Herkersdorf, "Disaggregated FPGAs: Network Performance Comparison against Bare-Metal Servers, Virtual Machines and Linux Containers," *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Luxembourg City, 2016, pp. 9-17.

[23] K. Wang, Z. Liu, Y. Lin, J. Lin and S. Han, "HAQ: Hardware-Aware Automated Quantization With Mixed Precision," 2019 *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR),* Long Beach, CA, USA, 2019, pp. 8604-8612.

[24] https://top500.org/system/179807/, Online, Accessed 4 Dec. 2020

[25] R. Polig et al., "Giving Text Analytics a Boost," *IEEE Micro*, vol. 34, no. 4, pp. 6-14, July-Aug. 2014, doi: 10.1109/MM.2014.69.

[26] G. Singh et al., "NERO: A Near High-Bandwidth Memory Stencil Accelerator for Weather Prediction Modeling," *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, Gothenburg, Sweden, 2020, pp. 9-17.

[27] A. D. Córcoles et al., "Challenges and Opportunities of Near-Term Quantum Computing Systems," *Proceedings of the IEEE*, vol. 108, no. 8, pp. 1338-1352, Aug. 2020, doi: 10.1109/JPROC.2019.2954005.

[28] E. Brewer, "CAP twelve years later: How the "rules" have changed," *Computer*, vol. 45, no. 2, pp. 23-29, Feb. 2012, doi: 10.1109/MC.2012.37.

[29] B. Ringlein, F. Abel, A. Ditter, B. Weiss, C. Hagleitner, D. Fey, "Programming Reconfigurable Heterogeneous Computing Clusters Using MPI With Transpilation" , IEEE/ACM International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC)