Pacific Northwest
NATIONAL LABORATORY

PNNL-34901

Navier: Dataflow Architecture for Computation Chemistry

September 2023

Roberto Gioiosa
Eduardo Aprá
Andres Marquez
Ajay Panyala
Rizwan Ashraf
Lenny Guo

U.S. DEPARTMENT OF ENERGY

Prepared for the U.S. Department of Energy
under Contract DE-AC05-76RL01830

DISCLAIMER

PACIFIC NORTHWEST NATIONAL LABORATORY
operated by
BATTELLE
for the
UNITED STATES DEPARTMENT OF ENERGY
under Contract DE-AC05-76RL01830

Printed in the United States of America

Available to DOE and DOE contractors from the Office of Scientific and Technical Information,
P.O. Box 62, Oak Ridge, TN 37831-0062
www.osti.gov
ph: (865) 576-8401
fax: (865) 576-5728
email: reports@osti.gov

Available to the public from the National Technical Information Service
5301 Shawnee Rd., Alexandria, VA 22312
ph: (800) 553-NTIS (6847)
or (703) 605-6000
email: info@ntis.gov
Online ordering: http://www.ntis.gov

PNNL-34901

Navier: Dataflow Architecture for Computation Chemistry

September 2023

Roberto Gioiosa
Eduardo Aprá
Andres Marquez
Ajay Panyala
Rizwan Ashraf
Lenny Guo

Prepared for
the U.S. Department of Energy
under Contract DE-AC05-76RL01830

Pacific Northwest National Laboratory
Richland, Washington 99354

Abstract

Navier's objectives were two evaluate the use of emerging technologies, especially dataflow accelerators, for high-performance computing (HPC) applications, specifically in the domain of chemistry, and to develop a prototype software stack to support such applications. Navier builds on capabilities previously developed by synergistic projects, such as PNNL Data Model Convergence (DMC) LDRD Hardware Advanced Workflows (HAW) and DuOMO, as well as DOE ARIAA.

Throughout its 18 months, the Navier team developed new capabilities and artifacts at all levels of the HW/SW stack, provided a seamless way to integrate novel computing architectures (Sambanova SN10 and Xilinx Versal AI) into an existing software stack, developed chemistry workflows, data analytics tools, and HPC molecular dynamics workflows that leverage the developed stack and PNNL institutional investments in emerging architectures. Navier also explored the use of active learning to accelerate a computational chemistry workflow for organic molecules on PNNL Junction cluster (in collaboration with AMD/Xilinx). Navier developed tools, methodologies, and studies for hardware software co-design and (sparse) dataflow accelerators that are composable and can be used together or separately. These methodologies are now used in other projects, such as DOE AMAIS and HPDA.

This report describes Navier's achievement, the developed tools and methodologies, and the research findings and conclusions.

Summary

This final report describes the research activities performed during the execution of the DMC Navier project. Navier made contributions to the following areas:

- Evaluation of emerging architectures and extremely heterogeneous systems: Navier explored the use of Sambanova SN10, Xilinx AIE, and the PNNL Junction cluster.

- Development of a composable software stack for extremely heterogeneous systems: Navier enhanced COMET and MCL to support emerging architectures, including FPGA and dataflow architectures. Moreover, Navier software stack enables users to leverage multiple heterogeneous accelerators at the same time.

- Development of application workflows (Chemistry CCSD, computational fluid dynamics, data analytics, AI scientific framework), benchmarks, and proxy applications to demonstrate the developed software stack and to act as drivers of HW/SW co-design studies.

- Development of HW/SW co-design methodologies and tools to perform co-design of full applications and architectures using realistic input sets.

All developed code (except code developed under NDA) is/will be released as open-source code on PNNL github account.

Acknowledgments

This research was supported by the Data Model Convergence Initiative, under the Laboratory Directed Research and Development (LDRD) Program at Pacific Northwest National Laboratory (PNNL). PNNL is a multi-program national laboratory operated for the U.S. Department of Energy (DOE) by Battelle Memorial Institute under Contract No. DE-AC05-76RL01830.

Contents

Figures

1.0 Introduction

Navier addresses the challenges and shortcomings of current dataflow architectures and software stacks for high-performance computing (HPC). It aims at developing the required hardware and software capabilities to enable domain scientists to fully leverage recent advantages in processing and memory architectures and provide a clear path to migrate and port scientific applications to future supercomputers. Specifically, Navier set three main objectives:

1. Develop a flexible and composable system software stack that allows scientists to leverage emerging (dataflow) architectures.

2. Explore novel hardware concepts for sparse dataflow architectures to support current and future scientific applications and workflows.

3. Develop proxy applications and full methods derived from NWChemEX using the proposed high-level language and system software.

Navier's objectives were two evaluate the use of emerging technologies, especially dataflow accelerators, for high-performance computing (HPC) applications, specifically in the domain of chemistry, and to develop a prototype software stack to support such applications. Navier builds on capabilities previously developed by synergistic projects, such as PNNL Data Model Convergence (DMC) LDRD Hardware Advanced Workflows (HAW) and DuOMO, as well as DOE ARIAA.

Navier overarching objective was to enable HW/SW co-design of full HPC, data analytics, and AI applications on complex heterogeneous systems, such as PNNL Junction, which consists of 48 compute nodes equipped with AMD CPU, AMD GPU, Xilinx Versal ACAP (FPGA + AI Engine), and Xilinx SmartNICs. All the activities performed were aimed at developing capabilities and methodologies that allow domains scientists to execute scientific workflows on heterogeneous systems and, at the same time, computer scientists to perform HW/SW co-design using the same (unmodified) applications and data sets.

Throughout its 18 months, the Navier team developed new capabilities and artifacts at all levels of the HW/SW stack, provided a seamless way to integrate novel computing architectures (Sambanova SN10 and Xilinx Versal AI) into an existing software stack, developed chemistry workflows, data analytics tools, and HPC molecular dynamics workflows that leverage the developed stack and PNNL institutional investments in emerging architectures. Navier also explored the use of active learning to accelerate a computational chemistry workflow for organic molecules on PNNL Junction cluster (in collaboration with AMD/Xilinx). Navier developed tools, methodologies, and studies for hardware software co-design and (sparse) dataflow accelerators that are composable and can be used together or separately. These methodologies are now used in other projects, such as DOE AMAIS and HPDA.

This report describes the main technical and research activities performed during the execution of the projects and provides some initial results, technical insights, and conclusions of the research.

2.0 Hardware Architectures and Testbeds

Navier heavily relied on the availability of emerging architectures at PNNL. One of the main objectives was to provide a feasible way to domain scientists to leverage PNNL investment in emerging computer architectures and to understand the feasibility of using such architectures for scientific applications and data analytics.

2.1 Xilinx Versal ACAP AI Engine

Xilinx Versal devices contain a mix of building blocks designed for efficient acceleration of compute problems. They contain an adaptive programmable logic fabric that allows the composition of application-specific circuits. This fabric has been shown to be capable of very high-throughput computation with flexible data types. Many devices in the Versal family also contain a dense mesh of hundreds of Vector-VLIW processors (AI Engines) combined with distributed memory blocks and associated DMAs. These coarse-grained reconfigurable resources allow efficient and flexible computation of dense tensor operations using fixed- and floating-point data types. The device's compute-oriented fabrics are complemented with an on-die ARM processor subsystem and are connected to external memories using a high-bandwidth network- on-chip interconnect. The combination of these specialized fabrics within a single device enables optimal mapping of the constituent parts of an application to area and power-efficient implementation methodology. Efficient implementations leverage dataflow implementation styles by using explicitly scheduled data transfers to orchestrate overlapping of data transfer and computation. Xilinx has addressed the challenge of programming this heterogeneous mix of fabrics in two ways. The first is through vendor tools that capture compute problems in a DSL (e.g., TensorFlow or Pytorch for AI, or P4 for packet processing) and compiles them to an efficient implementation on a Versal device. Xilinx has also championed a "white-box" open compilation flow using MLIR. This approach exposes the capabilities of the Xilinx fabrics at progressively lower abstraction levels and welcomes integration with third-party tools at all abstraction layers, which makes Xilinx Versal a promising and attractive target for integration with our proposed compiler.

2.2 PNNL Junction Institutional Cluster

PNNL Junction is a distributed heterogeneous cluster that consists of 48 computer nodes featuring several computing devices:

- 2x AMD EPYC 7543 32-Core Processor
- 256GB DDR4-3200 memory
- 1x AMD MI100 Instinct GPU
- 1x Xilinx VCK5000 FPGA
- 1x Xilinx SN1000 dual port 100Gb/s SmartNIC
- 1x Mellanox HDR-100 ConnectX-6 InfinBand HBA
- 1x 960GB SSD

From Navier's perspective, Junction is an ideal testbed to explore 1) the effectiveness of Xilinx FPGA and AIE included in the VCK5000 boards for scientific computing and data analytics; 2) the concurrent use of CPU, GPU, FPGA, and AIE and to develop novel scheduling and mapping algorithms in the runtime; 3) HW/SW co-design through architectural emulation using FPGA; and 5) code generation for Xilinx AIE.

2.3 Sambanova Cardinal SN10

The SambaNova Reconfigurable Dataflow Architecture (RDA) is a computing architecture designed to efficiently run applications that expose dataflow patterns, which range from traditional scientific simulations to recent AI methods. With SambaNova RDA, application-specific computational graphs are mapped to a set of reconfigurable processing, memory, and switch units for optimal execution on the available hardware. The RDA provides a flexible, dataflow execution model that pipelines operations, enables programmable data access patterns, and minimizes excess data movement. The architecture enables a broad set of highly parallelizable patterns contained within dataflow graphs to be efficiently programmed as a combination of compute, memory, and communication networks. This system has been designed to ease the task of running AI applications and allow users to leverage high-level programming frameworks, such as PyTorch and TensorFlow. The SambaNova compiler extracts dataflow information from the algorithmic representations and produces the computational graph that is mapped to the hardware by the runtime. While it is possible to produce computational graphs from other sources, other than PyTorch and TensorFlow, this option generally requires experience with mapping algorithms to spatial accelerators.

3.0 Software Stack for HW/SW Co-Design

In true spirit with the main DMC objectives, rhe Navier team developed an integrated and composable full software stack for converged applications and architectures. As explained above, Navier builds on capabilities previously developed in other synergistic projects, thigh them together, and integrate/extend them to provide a full execution environment of scientific applications, data analytics, and AI workflows on current and emerging heterogeneous architectures. Figure 1 shows a graphical representation of the software stack. Applications from various domains can be developed using domain-specific languages (e.g., PyTorch for AI or Lamellar for data analytics). The user code is parsed by the COMET compiler (Section 3.1) which generates 1) the object code for the target heterogenous device and 2) the MCL host code that drives the devices. The MCL runtime (Section 3.2) encapsulates kernels into tasks, map tasks to available heterogeneous devices, and orchestrates data movement to/from devices, leveraging data locality whenever possible. Through MCL and COMET, user applications can leverage current (e.g., CPU and GPU) and emerging (e.g., FPGA and dataflow) architectures. Moreover, MCL provides a vehicle to perform HW/SW co-design of full applications and architectures (Sections 3.3 and 3.4).

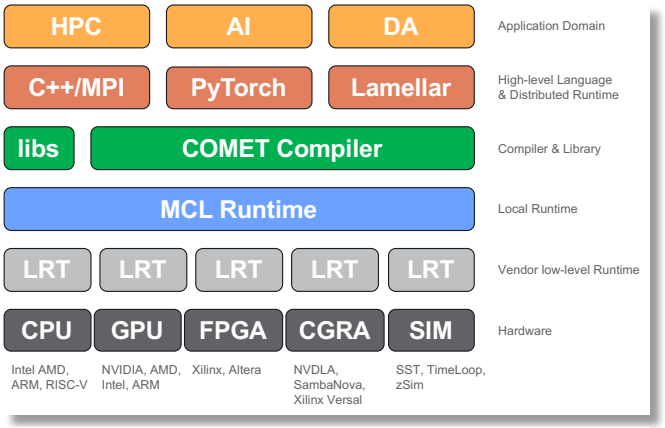

Figure 1: PNNL Software Stack for Extremely Heterogeneous Distributed Systems

3.1 COMET Compiler

The COMET (COMpiler for Extreme Targets) [(COMET n.d., Erdal Mutlu 2020, Ruiqin Tian 2021)] compiler consists of a Domain Specific Language (DSL) for sparse and dense tensor algebra computations and, a progressive lowering process to map high-level operations to low-level architectural resources. During the lowering process, a series of optimizations are performed, and various intermediate representation (IR) dialects are used to represent key concepts, operations, and types at each level of the multi-level IR. COMET is built using Multi-Level IR (MLIR) Compiler Framework. Drawing motivation from MLIR, the COMET compiler performs different optimizations and code transformations at each level of the IR stack. Domain-specific, hardware-agnostic optimizations that rely on high-level semantic information are applied at high-level IRs. These include reformulation of high-level operations in a form that is amenable for execution on heterogeneous devices (e.g., rewriting Tensor contraction operations as Transpose-Transpose-GEMM-Transpose) and automatic parallelization of high-level primitives (e.g., tiling for thread- and task-level parallelism). Hardware-specific transformations are applied at low-level IRs, either within COMET/MLIR or through vendor backends. At this time, COMET supports execution on CPUs, GPUs, and FPGAs (which was added as part of Navier).

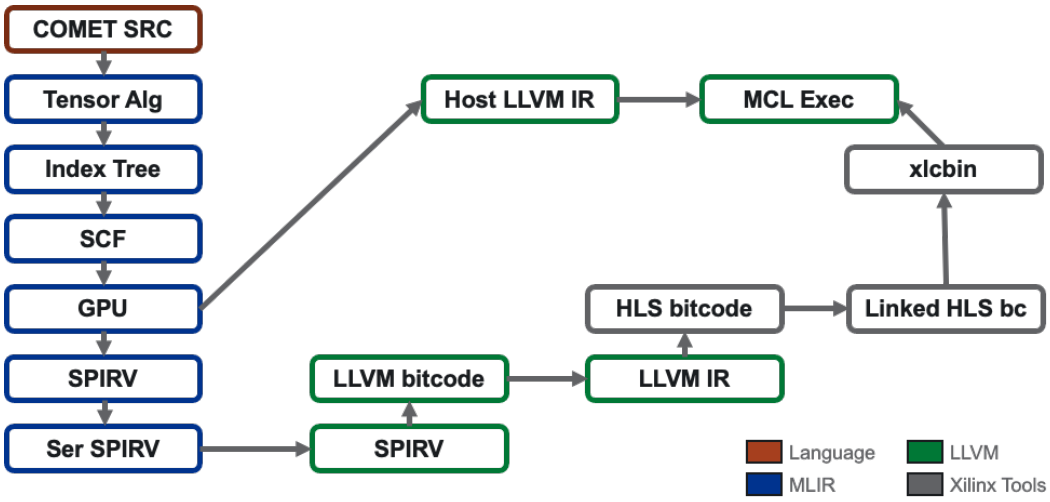

Figure 2: COMET code generation pipeline for FPGA.

COMET was developed in the DMC DuOMO project. In Navier, the team added support to automatically generate bitstreams for Xilinx FPGA starting from COMET DSL. Figure 2 shows the lowering steps and the correlation with other open-source and Xilinx tools. The code generation process follows the traditional high-level steps used in COMET for most computation, hence it benefits from many implemented optimizations. At the GPU dialect levels, the kernels are extracted from loop constructs and separated from the host code. At this point, there are two code generation paths: on one side, COMET produces the host program, which is based on the MCL runtime to execute tasks on heterogeneous devices. On the other side, COMET lowers the kernel code into a SPIR-V dialect (the team modified the original MLIR SPIR-V dialect to support computing kernels) and into proper SPIR-V binaries. Next the SPIR-V binary is translated into synthesizable LLVM IR. The choice of using a SPIR-V intermediate representation reduces the incompatibilities among different LLVM IR versions (Xilinx HLS tools require LLVM 7.0 while COMET is based on LLVM 14.x) and ensure that the generated IR is synthesizable. The last steps consist of adding the necessary libraries and IPs for the VCK500 and producing the final bitstream.

3.2 MCL Runtime

MCL (MCL n.d., Roberto Gioiosa 2020, A. V. Kamatar 2020) is a modern task-based, asynchronous programming model for extremely heterogeneous systems. MCL consists of a single scheduler process and multiple, independent, multi-threaded MCL applications executed concurrently on the same compute node. For example, MCL can seamlessly support the execution of multiple Message Passing Interface (MPI) ranks within a single node as well independent applications programmed with separate programming models and runtimes, e.g., an OpenMP application next to a PyTorch application. Users need not be aware of other applications executed on the same compute node, as the MCL scheduler coordinates access to shared computing resources. MCL aims at abstracting the low-level hardware details of a system, supporting the execution of complex workflows that consists of multiple, independent applications (e.g., scientific simulation coupled with in-situ analysis or AI frameworks that analyze the results of a physic simulation), and performing efficient and asynchronous execution of computation tasks. MCL is not meant to be the programming model employed by domain scientists to implement their algorithms, but rather to support several high-level Domain-Specific Language (DSL)s and programming model runtimes.

MCL was originally develop in Cosmic Castle as part of the DARPA Domain-Specific System-On-Chip (DSSoC) program in the context of Software-Defined Radio (SDR). Next, it was used in the DOE ARIAA project to support execution of scientific workflows on dataflow architectures. In DMC HAW, MCL was extended with support for HW/SW co-design (Proteus). In Navier, MCL has been extended to support execution on FPGA and Xilinx AIE, enhanced mapping of computational tasks to heterogeneous devices, and as a vehicle for HW/SW co-design. The choice of MCL as heterogeneous runtime was also dictated by the necessity of supporting complex workflows and emerging architectures.

3.3 HW/SW Co-Design Using Proteus

Proteus (Gioiosa 2022) device models a data-parallel device with multiple compute units. Each compute unit contains multiple processing elements (PEs) as obtained through the pre-hardware simulators. The Proteus device is emulated on a multi-core CPU and implemented using the POCL framework. The POCL framework provides all the basic functionality to realize a new OpenCL device. It handles all the boiler-plate code to realize the OpenCL standard. For example, the ability to check the legitimacy of arguments provided to an OpenCL function call and return appropriate code to the caller. Using this framework, a new device is implemented to realize the Proteus device. In Navier, the team used Proteus to model candidate HW concepts and perform HW/SW co-design of chemistry applications and hardware accelerators through the MCL runtime (see Section 4.1 for details). The methodology developed allows computer architects and domains scientists to perform different kinds of HW/SW co-design studies, depending on the target metric and design constraints. For example, a typical tradeoff between existing devices and novel hardware concepts can be explored using MCL/Proteus maintaining the same power envelop (iso-power studies). E.g., one could ask what the benefit would be of replacing an existing GPU with a novel hardware accelerator that consumes the same power. Importantly, the approach followed allows users to perform HW/SW co-design studies of whole applications and system, not just kernels and new device. This means that while some kernel may be running on the architecture under study, others and the driver application may still run on CPU or GPU. This approach provides much more accurate results and the impact of novel accelerators on the entire applications, not just a kernel in isolation.

Using MCL/Proteus, the team analyzed the potential impact of sparse dataflow accelerators and compare it to mainstream (dense) dataflow accelerators using a sparse matrix-sparse matrix kernel. On one side, sparse dataflow accelerators provide higher performance than their dense counterparts (assuming the inputs and outputs are stored in some sparse format such as CSR). On the other side, most sparse dataflow accelerators available in the literature require a custom sparse storage format and format conversion to/from the host if the accelerator and host format are not the same, which is typically the case. For this study, the team compared the Sambanova SN10 and TimeLoop dense accelerators against SpareLoop and Sigma. The results show that the overhead of data conversion overshadow the benefits of faster computation on sparse dataflow accelerators. This leaves two alternatives: 1) use dense dataflow accelerators to perform sparse computation and let the compiler generate efficient code to map sparse computation to dense hardware. This is akin to using GPUs to perform sparse computation, even though sparse computation is not completely amenable to GPU architectures because of branches, indirect memory accesses, and poor data locality. 2) Leverage data movers to perform data conversion from one format to another when moving data form the host to the accelerator and vice-versa and overlap computation and data conversion. The Xilinx ACAP VCK5000 is a possible test case for the second option, where the programmable logic can be used to program a data mover that converts CSR format to AIE format and back.

3.4 HW/SW Co-Design Using FPGA Emulation

HW/SW co-design using FPGA emulation follows a more traditional co-design approach in which a novel hardware concept is emulated using programmable logic. However, as for the previous case, it is important to contextualize the execution of kernels on the new accelerators and relate that to the rest of the application. Thanks to MCL and PNNL Junction cluster, the team was able to develop and execute a proxy of the CCSD chemistry method and explore the concurrent use of CPU, GPU, and FPGA (see Section 4.6 for more details). Also, by using COMET to generate the FPGA bitstream and the MCL host program, the entire co-design process is completely automated, flexible, and agile.

4.0 Driver Applications

Key to perform HW/SW co-design of novel hardware accelerators and applications/algorithms is the availability of realistic applications and data sets. True to the spirit of DMC, Navier developed converged applications in the domains of HPC (chemistry and molecular dynamics), Data analytics, and AI for Science, as well as workflows that combine two or more of those domain applications. The following describes the various applications and their execution models.

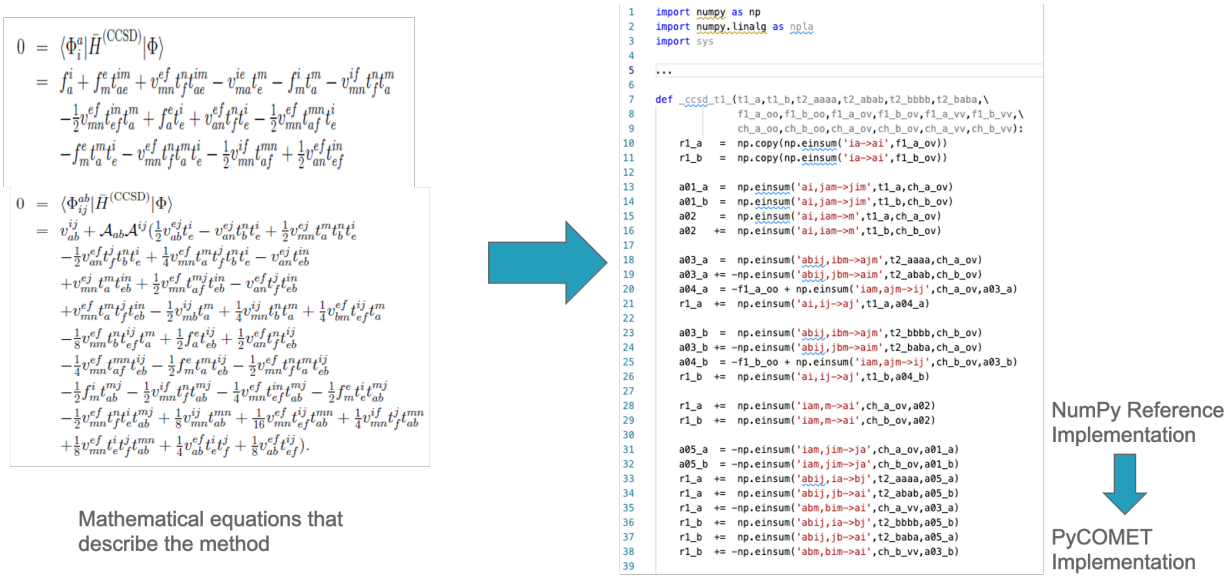

Figure 3: CCSD method mathematical formulation and NumPy implementation (extract)

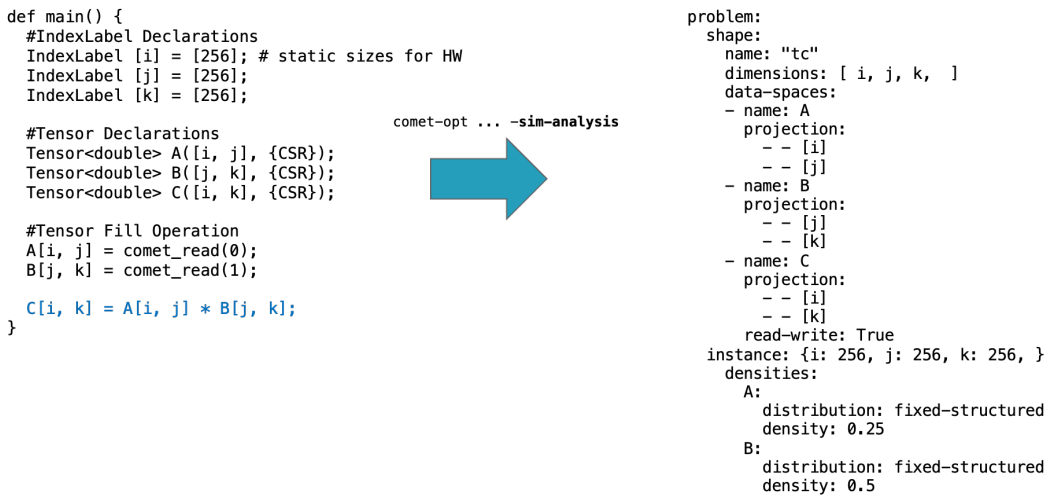

Figure 4: Automatic generation of SparseLoop accelerator description from COMET DSL.

4.1 Couple-Cluster Method

The first application developed is a Couple-Cluster Method (CCSD) which is integral part of a chemistry pipeline that also includes Self-Consistent Field (SCF) and Density Functional Theory (DFT) computation. All three components and their interconnections have been developed/ported, but the team eventually decided to focus on the CCSD method for the HW/SW co-design studies. The CCSD method has been developed using a block-sparse approach and initially implemented in NumPy (baseline). Figure 3 shows the mathematical formulation of the method and an extract of its counterpart NumPy implementation. Next, the CCSD NumPy code has been ported to CometPy, a NumPy-like python frontend that can be parsed by the COMET compiler. Being able to use COMET to compile the CSSD method has provided several advantages: 1) COMET generated code performs better than Python or LLVM code on CPUs; 2) COMET can generate object codes for GPUs, FPGAs, and dataflow accelerators Xilinx AIE and Sambanova SN10 (work-in-progress); 3) finally, COMET provides a way to performs HW/SW co-design. In this case, we used COMET to automatically generate input for TimeLoop and SparseLoop and performed HW/SW co-design studies answering questions such as "What would be the performance of the CCSD method if executed on a (sparse) dataflow accelerator that is iso-power with a state-of-the-art GPU?". The entire HW/SW co-design pipeline is automated and does not require any code modification. Figure 4 shows an example of COMET DSL code which performs a sparse matrix-sparse matrix multiplication and the resulting SparseLoop code generated by COMET with the new analysis and code generation path developed in Navier. Using the developed COMET passes, the team was able to perform various co-design studies (performance, iso-power with GPU, SoC scenarios, etc.). Figure 6 shows experimental results of a HW/SW co-design study in which a GPU is replaced with a dataflow accelerator modeled with Time/SpareLoop, which description is automatically generated by COMET from the code listed in Figure 3. As the results suggest, dataflow accelerators show great potentiality and reduced execution time.

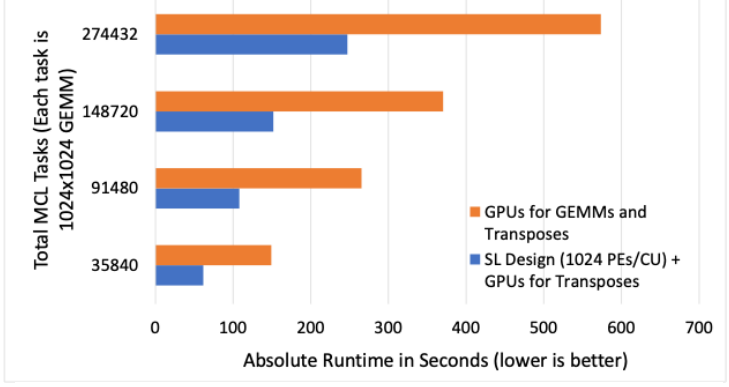

Figure 6: Co-Design of CCSD (computational chemistry) and dataflow accelerators (Time/SparseLoop)

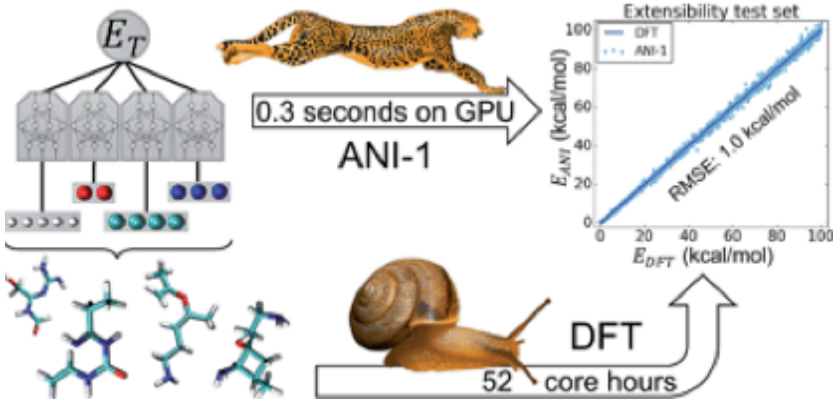

Figure 5: Leveraging fast AI surrogate models to infer the energy or large organic molecules from a database of small organic molecules.

4.2 Active Learning Chemistry Workflow

One of the main objectives of Navier was to explore the use of AI surrogate models to speedup computational chemistry and then explore the use of specialized AI accelerators to further speedup the cost of inference, hence leverage the multiplicative combined effect of the two novel technologies. Consider the quantum chemistry application shown in Figure 5: the objective of this system is to infer the energy of large molecules based on the knowledge of small molecules and the learning obtained while studying new molecules. The energy of the molecule under study is inferred from an ensemble of AI models and the uncertainty of the prediction is compared against a certain threshold. If the prediction is below the threshold, the correct energy is computed using traditional quantum mechanics methods, the new information is added to the database of molecules, and the AI models are trained again using the new knowledge. This workflow is representative of modern HPC scenario. In fact, while in the past HPC applications mostly leveraged MPI, OpenMP, or other languages for GPU programming, current workflows combine scientific simulation with in-situ HPDA or AI-based analysis, pair simulations with data obtained from instruments (digital twins) or employ AI surrogate models to speed up part of the scientific simulation. Figure 7 shows the workflow for the problem described in Figure 5. The system uses active learning and train refined models every time a new data point is added because of the misprediction of the energy cost of a large organic molecule. Transfer learning is employed to infer the energy cost of large molecules from smaller ones.

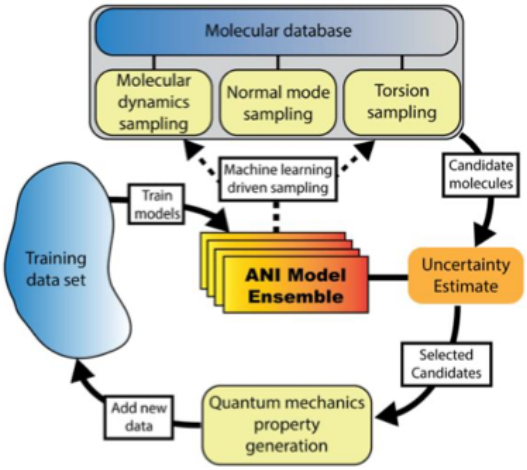

Figure 7: ANI Quantum Chemistry Workflow. The various components of the workflows (traditional quantum chemistry methods, active learning training, and AI inference) map to different computing devices (CPU, GPU, and AI Engine, respectively).

In Navier, the team mapped this workflow on PNNL Junction. Specifically, training is mapped on the GPU, inference on the AI engines, and the quantum mechanics methods on CPU. The team modified the original ANI workload to use PNNL NWChemEx (instead of the original PySCF) when computing the energy cost of molecules with traditional quantum methods. Initial experiments verified the correctness of the workload and allowed the team to drive the following conclusions about Xilinx AIE:

- With some effort, it possible to achieve performance close to GPU (within 10%).
- The precision of the AIEs is not adequate for scientific computation. Generally, the team has observed errors in the range or above 10%, which is too large for iterative methods and results to divergence of the methods and incorrect results. This is due to the 8-bit quantization employed in the current version of Xilinx AIE, which introduces excessive errors for FP23 and FP64 values commonly used in HPC applications such as NWChem.

This work has led to a collaboration with AMD/Xilinx and potential new projects.

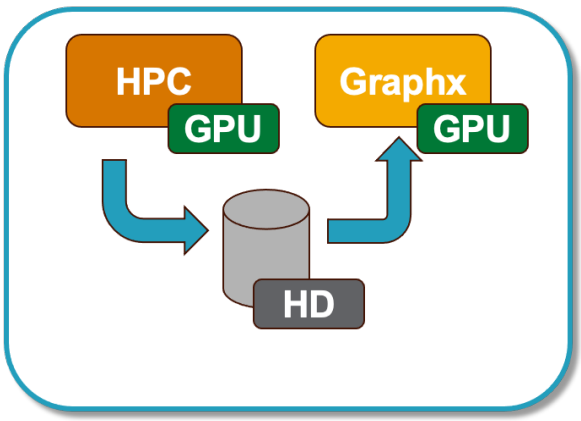

Figure 8: Traditional Approach

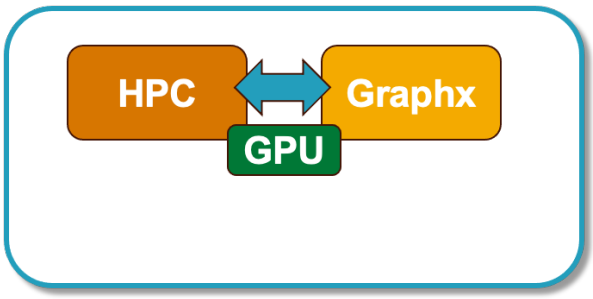

Figure 9: Co-scheduling simulation and data visualization

4.3 Computational Fluid Dynamics

Computation Fluid Dynamics (CFD) models attempt to simulate the interaction of liquids and gases where the surfaces are defined by boundary conditions. These models employ the principles of the Navier-Stokes equations. Simulations are then conducted by solving the equations iteratively as either a steady-state or transient conditions. Compared to solving traditional Navier-Stokes equations, with LBM methods a fluid density on a lattice is simulated with streaming and collision (relaxation) processes. Besides the MD simulation, graphics rendering is often necessary to produce human-readable outputs and for real-time analytes. Graphics rendering is separated from the CFD simulation but requires similar computational resources. Figure 8 shows the traditional workflow execution approach in which the results of the computation performed on a GPU is stored in DRAM (or disk) just to be reloaded immediately after on (potentially the same) GPU for graphics rendering. This incurs in unnecessarily data movement and waste of energy. On the contrary, Figure 9 shows the approach taken in Navier in which the data stays on the GPU where the CFD computation was performed and the graphics rendering task on the GPU on which the data is already stored, avoiding data movement, increasing performance, and reducing energy consumption. We ported FluidX3D, a CFD application that employ Lattice Boltzman Method (LBM) to MCL (mclCFD) to 1) leverage complex heterogeneous systems and multi-GPU system and 2) perform HW/SW studies. Besides CFD computation, FluidX3D performs in-situ graphics rendering and visualization and takes advantage of data locality on the heterogeneous devices. In the MCL implementation, the MCL schedule tracks data locality and co-locates producer-consumer tasks on the same GPU. In the original implementation, the user needs to manually partition the grid. There are, thus, two types of resources sharing between the CFD simulation and graphics rendering and visualization: data sharing (communication) and computing resource sharing.

The results of the experimental campaign performed showed that Navier has achieved its goals:

- mclCFD is highly portable and can run on multiple systems without any code modification. As a proof-of-concepts, the team successfully executed mclCFD on CENATE DGX-1 V100 (8x NVIDIA Volta GPUs), PNNL Junction (AMD GPU), and an Apple MacBook Pro (1 Intel GPU + 1 AMD GPU). Figure 10 and Figure 11 show a frame extracted from the CFD simulation video for a Ferrari SF71H race car and a StarWars X-Wing figther performed on CENATE DGX-1 V100 and PNNL Junction, respectively. The team was also

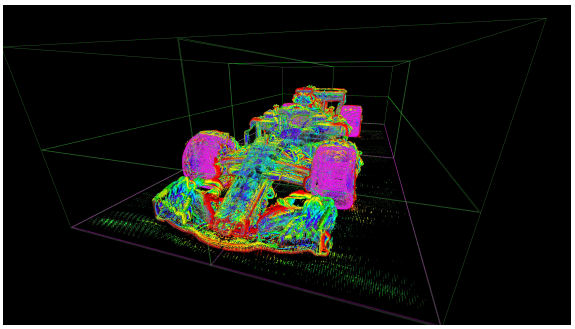

Figure 10: CFP Simulation of a SF71H Ferrari F1 race car

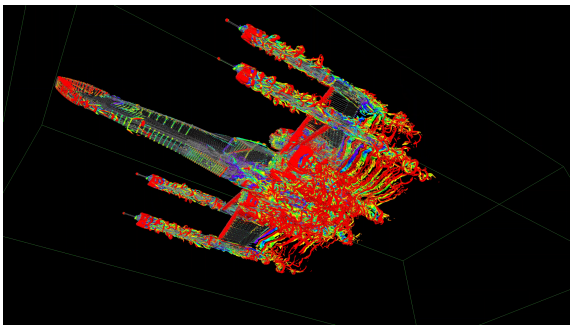

Figure 11: CFD Simulation of Star Wars X-Wing fighter

able to show a live demo during the last DMC review using the Apple MacBook Pro system and performing real-time analysis.

- mclCFD shows great scaling. We performed both strong and weak scaling analysis on the CENATE DGX-1 V100 system. First, we increased the number of GPUs from 1 to 8 keeping the problem size constant (strong scaling). We use the model of the Ferrari SF71H with a resolution of 512x1024x256 (134 million points) for the grid. Results show a speedup of 2.7x when using 8 GPUs compared to a single GPU. These results are below expectation and are due to underutilization of the devices (there is not enough data to feed all GPUs). Second, we increased the problem size when moving from 1 (512x1024x256 pints) to 8 GPUs (1024x2048x1024 --- 2.15 billion points) and obtained a super-linear speedup of 16.04x on 8 GPUs. These results exceeded the expectations and show strong evidence that the capabilities developed in Navier can lead to new Science and discoveries by enabling scientists to perform simulations of problems that could not be solved with traditional methodologies.

4.4 Arkouda (Data Analytics)

Arkouda is a data analytics framework developed by our sponsors to scale data analytics to a cluster of distributed compute nodes and solve larger problems. Arkouda attempts to merge the need of data scientists with traditional HPC infrastructures. HPC developers generally develop code and submit experiments to a job queue for batch execution and analyze the results once the experiment is over. The job manager (e.g., Slurm) takes care of executing the experiments when adequate resources are available. Eventually, the developer changes the code and/or inputs for the next experiments. Data scientists, on the other hand, prefer an interactive environment in which they can visualize and analyze the results as they develop the code. For this reason, data analytics is generally performed on single workstations using a Python environment. However, as the need of analyzing larger and larger data set increases, single workstations are not capable of storing and analyzing such large data sets, hence data scientists have moved to distributed systems. Arkouda provides a Python-like interface backed up by a distributed cluster to users. Data scientists can develop their code and analyze the results in an interactive way, while the execution of the code is performed on a cluster of distributed compute nodes. Arkouda originally uses Cray Chapel to distribute computation across the compute nodes. In Navier, we developed a new high-performance back end for Arkouda based on Lamellar (Lamellar - Rust HPC runtime n.d.), which has been developed in the HPDA program. The objectives of this work are three-fold:

1. Provide a high-performance back end for Arkouda that can seamlessly execute application in shared memory and distributed systems without code modification.

2. Provide a mechanism to execute Arkouda code on heterogeneous systems, both traditional GPUs (from various vendors) and emerging architectures (e.g., Sambanova and Xilinx FPGA).

3. Provide a vehicle to perform HW/SW co-design using Arkouda applications and realistic inputs sets.

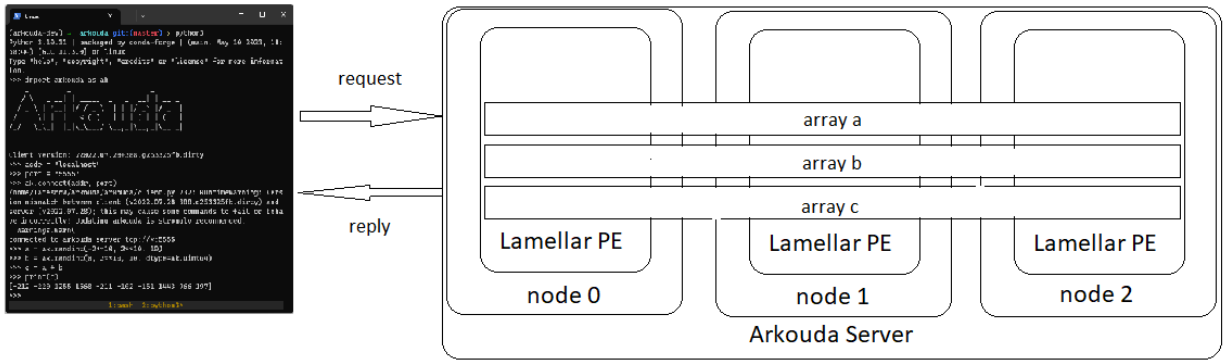

Figure 12: Software architecture of the high-performance Arkouda back end implemented in Lamellar.

Figure 12 shows the software architecture of the Arkouda high-performance back end developed with Lamellar to replace Chapel. As the figure shows, shared memory abstractions, such as array, are distributed across multiple compute nodes automatically through a LamellarArray data structure. Similarly, Lamellar orchestrates and synchronizes the access to those data structures through Distributed Atomic Reference Counters (DARCs). Distributed data structures management is hidden to the users, who interfaces with the system through the standard Arkouda interface (also show in the figure) or through a Jupiter notebook.

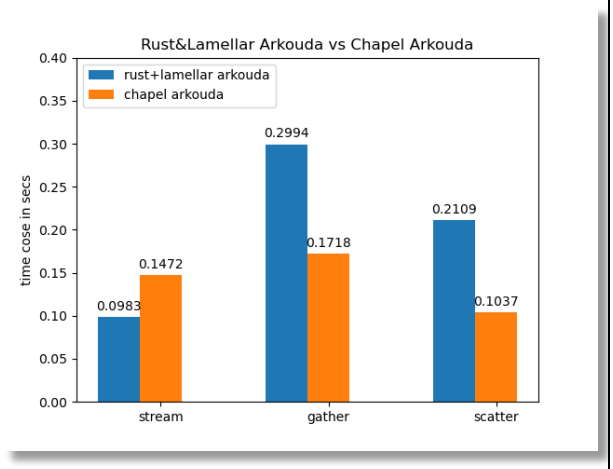

Figure 13: Arkouda performance comparison: Chapel vs Lamellar

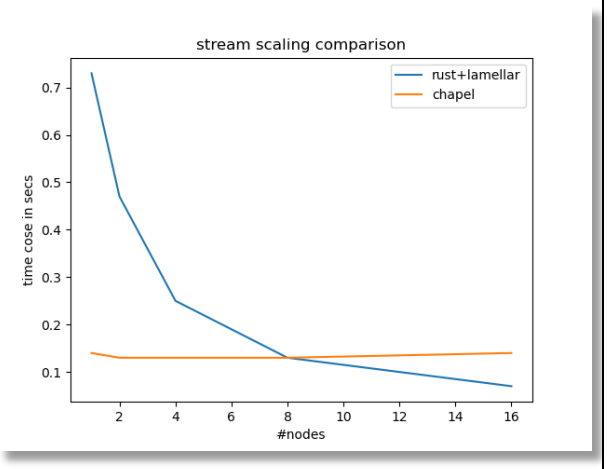

Figure 14: Performance scalability of the Stream benchmark (Chapel vs Lamellar)

Figure 13 and Figure 14 show some initial performance comparison results comparing Chapel and Lamellar as back end for Arkouda. As the figures show, in some case Lamellar is faster than Chapel, at least beyond some number of processing elements, but the current implementation is not yet capable to outperform Chapel in all cases. This work has been very instrumental to the Lamellar team to highlight bugs and performance issues that had not been discovered to date. has discovered. The use of a full-fledge application instead of specific benchmarks has drastically increased code and data transfer patterns exercised. The Lamellar team has been very proactive in solving some of the discovered issues.

Lamellar supports heterogeneous devices through MCL and COMET (through the Rust eDSL). This means that any code, including Arkouda, that is implemented using Lamellar/MCL/COMET can be executed not only on various GPUs but also on emerging architectures such as Sambanova and Xilinx AIE or FPGA. The team developed a proof-of-concept of a simple Arkouda vector addition benchmark that is executed on Sambanova SN10 through MCL. Considering that there is limited support in Chapel even for NVIDIA GPUs, this is potentially a very important development to accelerate data analytics in Arkouda using hardware accelerators.

Finally, Lamellar supports HW/SW co-design through MCL and Protheus. Again, as for actual hardware accelerators, there is not much modification required to perform co-design studies using MCL Proteus and connecting the entire Arkouda workflow to simulators, emulators, FPGAs, or analytical models. The team is setting up proof-of-concept experiment to showcase HW/SW co-design opportunities, such as the ones showed in Section 4.1.

4.5 NeuroMANCER

Neural Modules with Adaptive Nonlinear Constraints and Efficient Regularizations (NeuroMANCER) is an open-source differentiable programming (DP) library for solving parametric constrained optimization problems, physics-informed system identification, and parametric model-based optimal control. NeuroMANCER is written in PyTorch and allows for systematic integration of machine learning with scientific computing for creating end-to-end differentiable models and algorithms embedded with prior knowledge and physics. In Navier, the

team ported and adapted NeuroMANCER to Sambanova SN10 using SambaFlow. The objective was to assess the suitability of Sambanova dataflow architectures to PNNL scientific workflows and AI for Science. Results show performance comparable to NVIDIA V100 GPU until some size of the input sets, which is dependent on the benchmarks tested. Beyond that point, Sambanova SN10 shows an abrupt performance degradation. The team has been in contact with Sambanova to pinpoint and solve these issues.

4.6 Benchmarks and Proxy Application

To facilitate the development and testing of the software stack and perform analysis of the advanced architectures available at PNNL, the Navier team also developed benchmarks and proxy applications.

- mclCCSD: this proxy applications resembles the computation characteristics of the CCDS method. The application is developed after the code optimization employed by COMET when optimizing tensor contractions and factoring them into transpose-transpose-GEMM-transpose (TTGT). mclCCSD performs N TTGT operations, launching first all the transposes, then the GEMMs, the second round of transposes, and finally a reduction. The resulting computational graph is depicted in Figure 15. In the implementation, runnable tasks are executed asynchronously while the dependencies among tasks are enforced used mcl_wait(hdl) or mcl_test(hdl). We executed this proxy applications on PNNL Junction leveraging CPU, GPU, and FPGA. While the initial mapping of tasks onto devices was for illustration only and mapped Transposes to GPU, GEMMs to FPGA, and reduction on CPU, the team quickly found out that a better mapping was GEMMs to GPU, reduction to CPU, and transposes to FPGA. In fact, GEMM kernels are about 10x faster on GPU than FPGA (thought the FPGA consumes about 10x less energy) while transposes are only 2x faster on GPU than FPGA, but still the FPGA consumes about 10x less energy, thus the overall computation results in better performance efficiency.

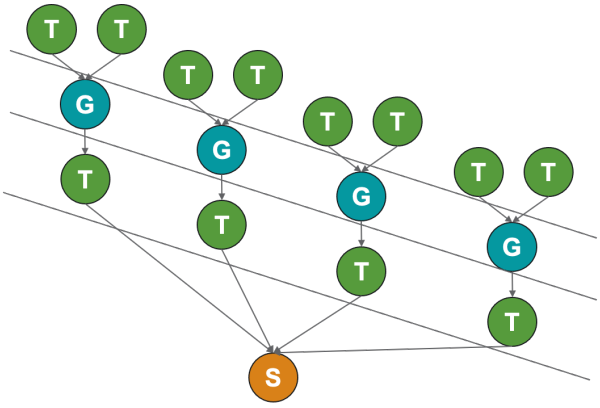

Figure 15: Computational graph of mclCCSD proxy application

- CometMM: This is a simple application that executes a matrix multiplication between two dense matrices. The objective of this benchmark was to test and validate the COMET lowering passes to generate bitstreams for Xilinx FPGAs in Junction. As explained above, COMET passes and MLIR dialects were modified to generate an initial representation of the kernels that can be lowered to bitstream. The team also developed equivalent benchmarks using Vitis HLS and OpenCL for comparison reasons.

5.0 Conclusions and Future Directions

Navier targeted 1) the development a flexible and composable system software stack that allows scientists to leverage emerging (dataflow) architectures; 2) the exploration of novel hardware concepts for sparse dataflow architectures to support current and future scientific applications and workflows; and 3) Develop proxy applications and full methods derived from NWChemEX using the proposed high-level language and system software. Throughout the project execution the team was able to meet the original goals and expand beyond chemistry applications to molecular dynamics, data analytics, and AI. The team made contributions to enhance both COMET and MCL and performed several HW/SW co-design studies of applications and hardware accelerators. Navier leaves a legacy of HW/SW co-design studies and conclusions, a set of applications that leverage DMC technologies, including COMET and MCL/Proteus, a software stack that enables users to leverage PNNL institutional investments in emerging architectures (Sambanova and AMD/Xilinx Junction), and various collaborations with vendors (AMD/Xilinx) and research institutions (Harvard, EPFL, and GT).

Going forward, the team believes that there is need to harden the developed technologies and methodologies, moving them from research prototypes to close-to-production tools. Moreover, the developed software stack can be used to integrate novel computer architectures (e.g., NextSilicon and GraphCore) and to answer important questions about future DOE and other sponsors' systems.

Overall, Navier concluded DMC by 1) developing a software infrastructure to execute converged applications (HPC, data analytics, and AI) on converged systems (CPU, GPU, FPGA, and AI engines) and 2) developing methodologies and tool to perform composable and agile HW/SW co-design of full application workflows and architectures.

6.0 Bibliography

n.d. https://github.com/pnnl/mcl.

A. V. Kamatar, R. D. Friese and R. Gioiosa. 2020. "Locality-Aware Scheduling for Scalable Heterogeneous Environments." *nternational Workshop on Runtime and Operating Systems for Supercomputers.*

n.d. *COMET.* https://github.com/pnnl/COMET.

Erdal Mutlu, Ruiqin Tian, Bin Ren, Sriram Krishnamoorthy, Roberto Gioiosa, Jacques Pienaar & Gokcen Kestor. 2020. "COMET: A Domain-Specific Compilation of High-Performance Computational Chemistry." *Languages and Compilers for Parallel Computing.* Lecture Notes in Computer Science.

Gioiosa, Rizwan Ashraf and Roberto. 2022. "Exploring the Use of Novel Spatial Accelerators in Scientific Applications." *ACM/SPEC International Conference on Performance Engineering.*

n.d. *Lamellar - Rust HPC runtime.* https://github.com/pnnl/lamellar.

Roberto Gioiosa, Burcu O. Mutlu, Seyong Lee, Jeffrey S. Vetter, Giulio Picierro, and Marco Cesati. 2020. "The Minos Computing Library: efficient parallel programming for extremely heterogeneous systems." *Workshop on General Purpose Processing using Graphics Processing Unit .* ACM.

Ruiqin Tian, Luanzheng Guo, Jiajia Li, Bin Ren, & Gokcen Kestor,. 2021. "A High Performance Sparse Tensor Algebra Compiler in MLIR." *Workshop on the LLVM Compiler Infrastructure in HPC.* St. Louis.

Pacific Northwest National Laboratory

902 Battelle Boulevard
P.O. Box 999
Richland, WA 99354

1-888-375-PNNL (7665)

www.pnnl.gov