Pacific Northwest
NATIONAL LABORATORY

Hardware Accelerators: Use, Misuse, and Co-Design

Roberto Gioiosa, PNNL
Research Scientist and Team Lead

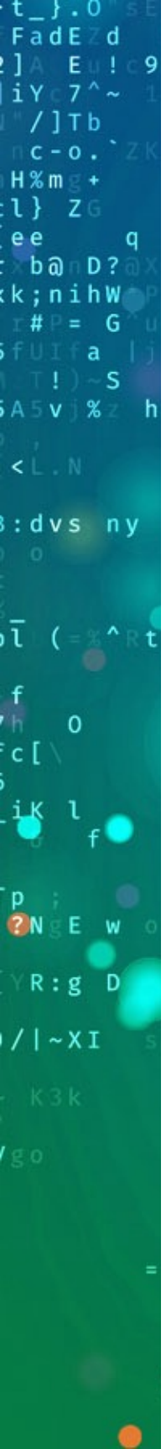
U.S. DEPARTMENT OF ENERGY BATTELLE

PNNL is operated by Battelle for the U.S. Department of Energy

Pacific Northwest
NATIONAL LABORATORY

Outline

- Introduction
- PNNL Converged SW Stack
  - Compiler
  - Runtime
  - CFD with in-situ 3D visualization
- HW/SW Co-Design
  - Hardware simulation
  - FPGA synthesis flow
  - SoC emulation
- Conclusions



Pacific Northwest
NATIONAL LABORATORY

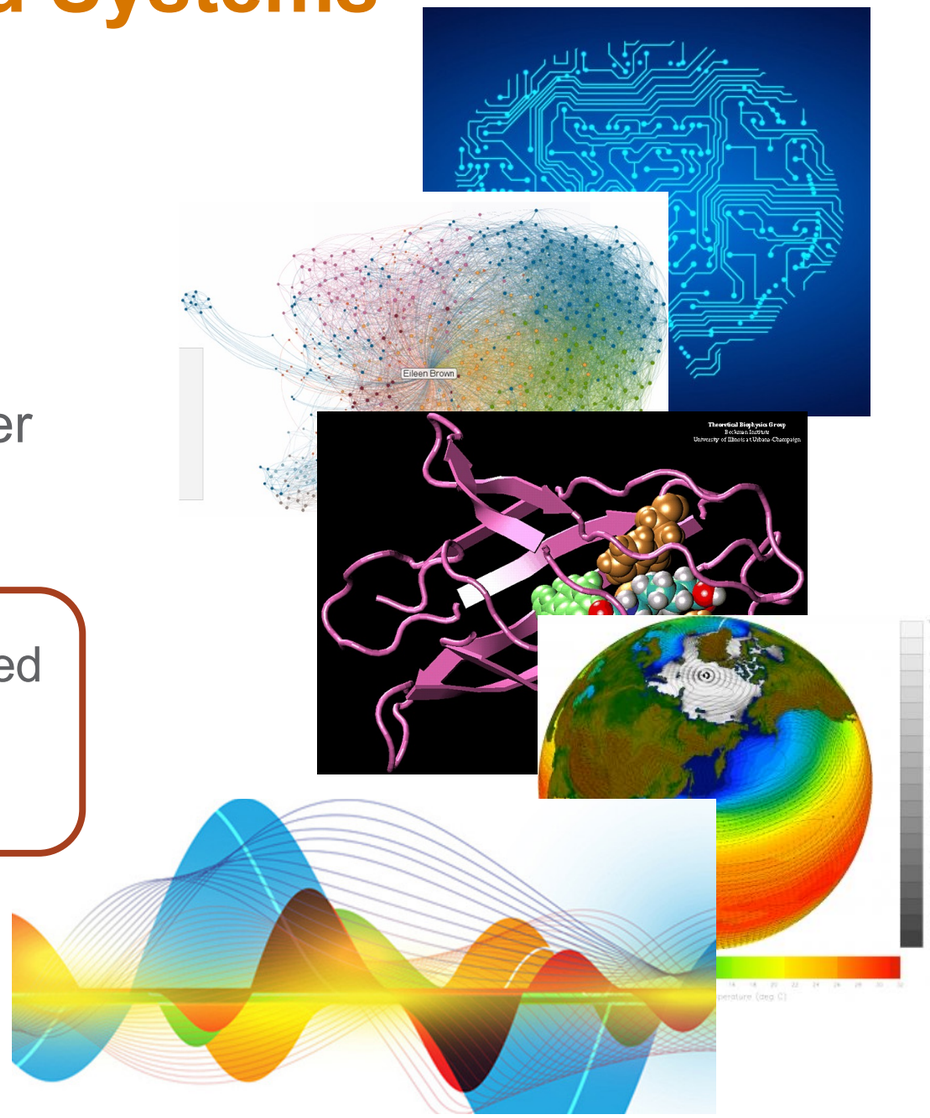
Introduction

Converged Applications and Systems

- New challenges
  - Emerging applications in different domains (scientific simulations, AI/ML, data analytics, signal processing, etc.)
  - Unprecedented amount of **data** to be processed under strict real-time, power, and trust constraints

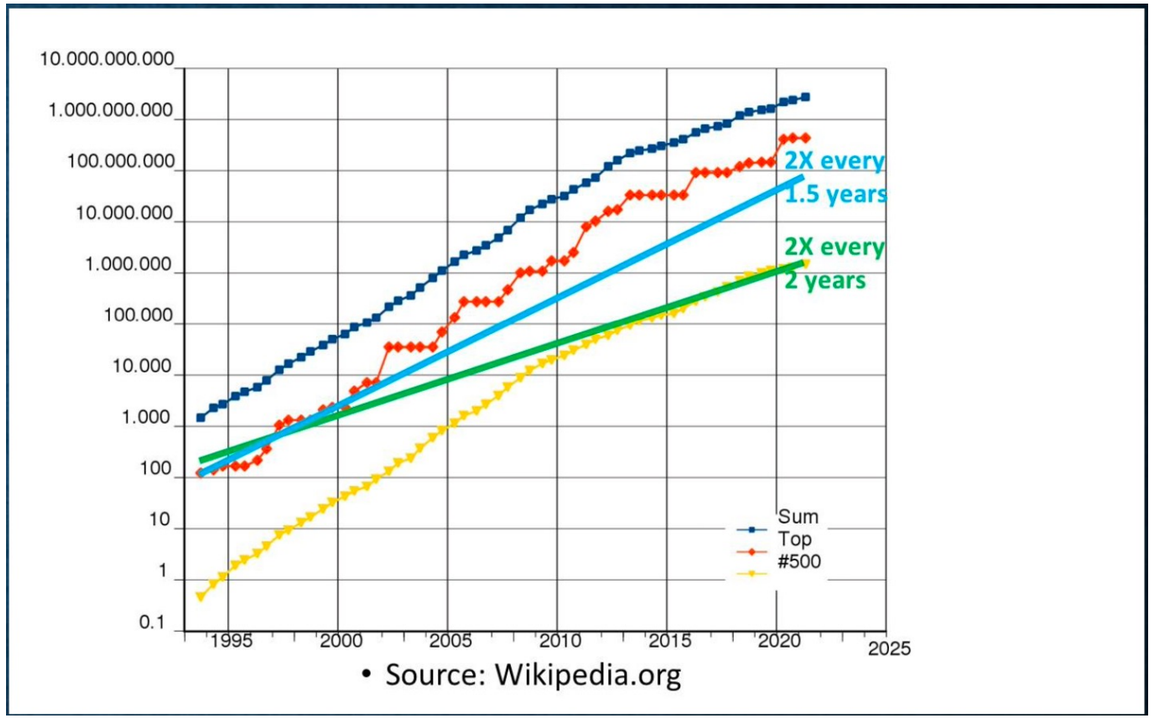
Data centers' electricity consumption in 2026 is projected to reach 1,000 terawatts, roughly Japan's total consumption

- Providing high-performance, scalable, and versatile solutions becomes a fundamental requirement



The (Human) Intelligence Factor

- Vectors (Cray)
- Microprocessors (Beowulf)
- Multicore, multithread (x86/ Power)
- Massive parallelism (Blue Gene)
- Heterogeneity (GPUs) – continuing…
- Memory-Coupled Computing (next?)
  - 2.5D
  - PIM
  - Tightly-Coupled Memory-Compute

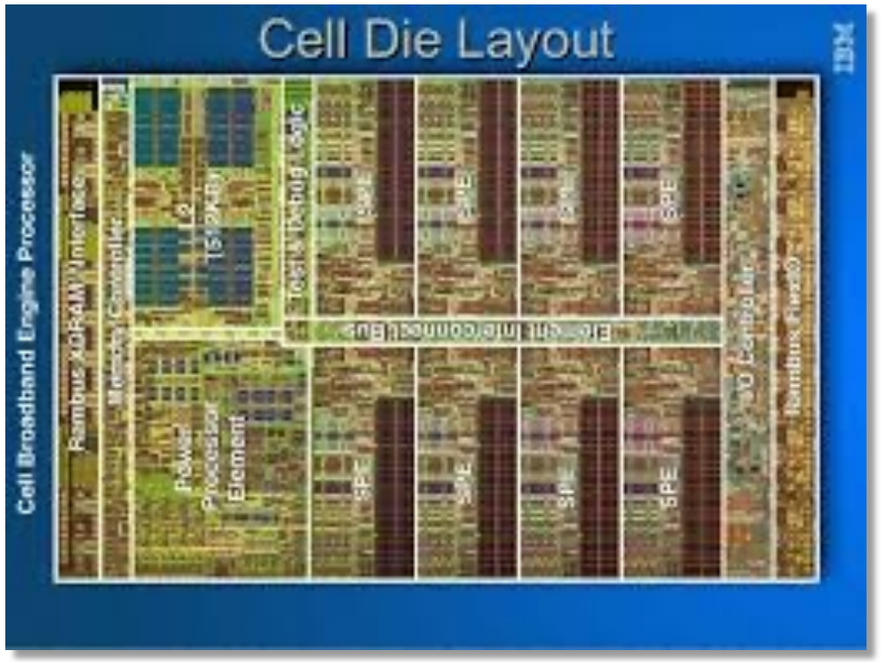


Specialization to the Rescue…. Or not?

- Specialization has become a fundamental pillar for the design of future high-end computer systems:
  - In HPC: GP-GPUs, FPGAs,…;
  - In Military: ASICs, DSPs, …
  - Specialized hardware for machine- and deep-learning (Tensor Cores, NVDLA, SambaNova, Cerebras,…).
- High level of specialization results in extremely heterogeneous systems that are complicated to design, test, validate, and program

My Favorite Processor….

- The IBM Cell BE processor was a major step towards using specialized hardware for commodity systems:
  - Power Processing Elements (PPE)
  - 8 Synergistic Processing Elements (SPE)
  - Element Interconnect Bus (EIB)
  - Used in game consoles **and** supercomputers!
- But…
  - It was difficult to program
  - Eventually replaced by BlueGene in HPC…
  - … then GPUs
  - … then ?

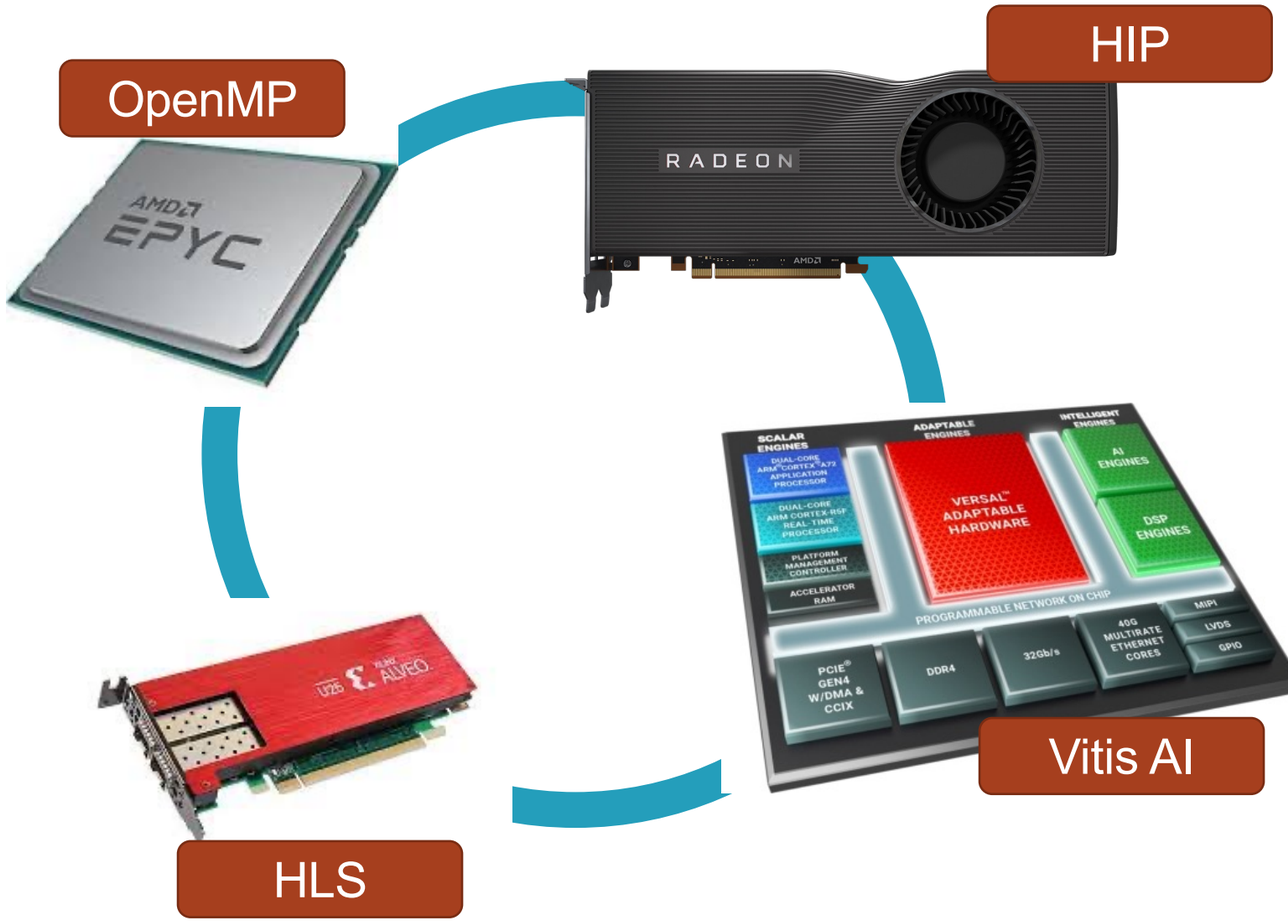


Lesson Learned: *Accelerators are only useful if they are accessible…*

Pacific Northwest NATIONAL LABORATORY

What Have We Learned?

- PNNL Junction is a prototype of future heterogeneous systems
  - 48 nodes
  - Each node consists of:
    - ✓ 2x AMD CPU
    - ✓ 1x Xilinx Versal (FPGA + AIE)
    - ✓ 1x AMD GPU
- Programming each device requires the use of specific languages and compilers
- Portability programming models and framework (e.g., OpenMP, Kokkos, Raja) have emerged to mitigate the problem

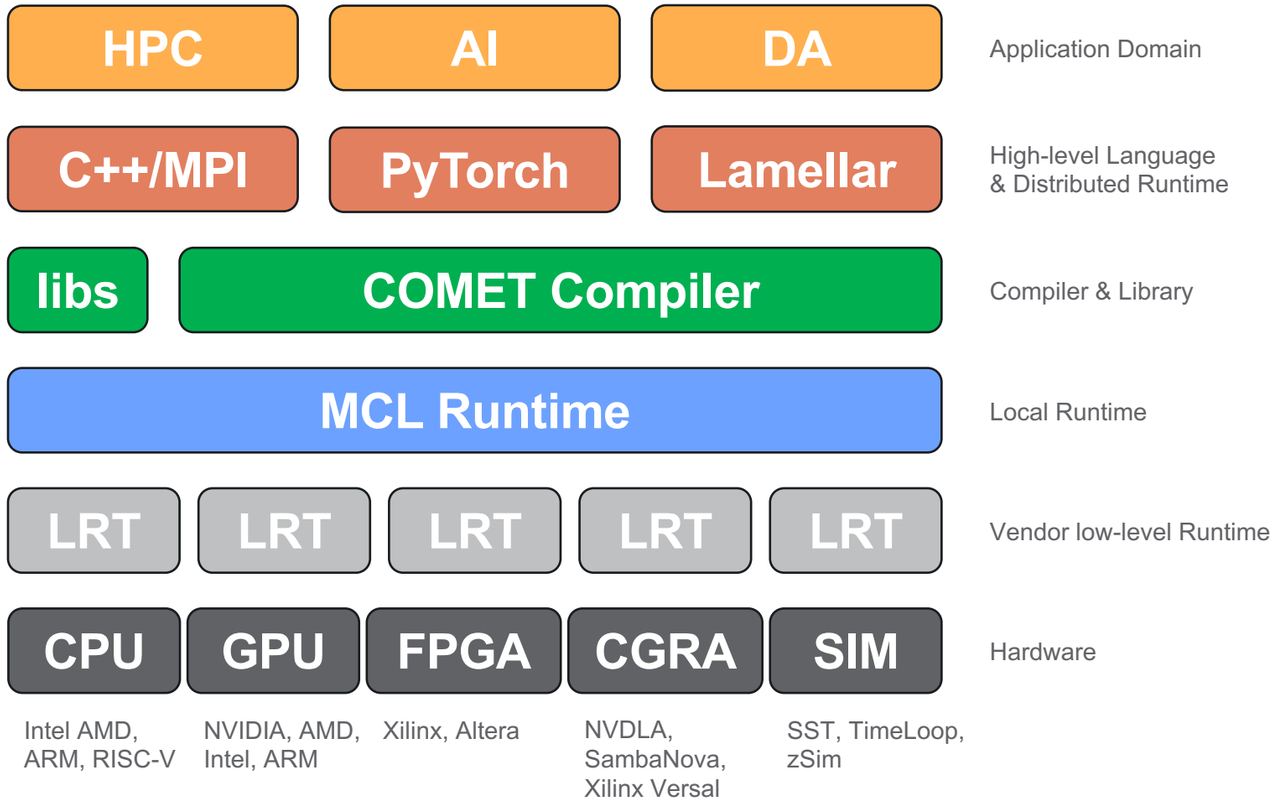


Pacific Northwest
NATIONAL LABORATORY

PNNL Converged SW Stack

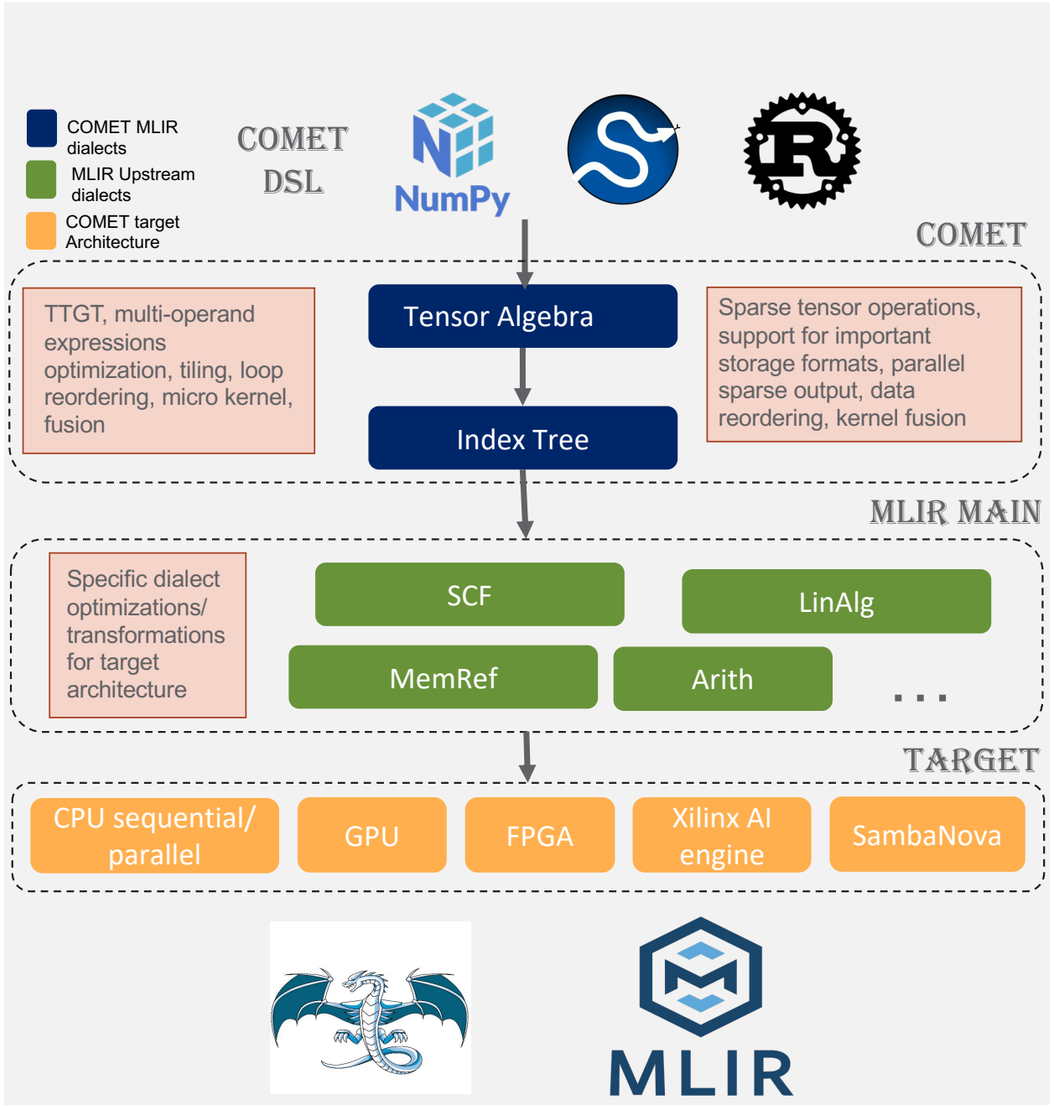
Converged Software Stack

- PNNL integrated and composable software stack aims at deploying application workflows on converged heterogeneous systems
- It supports HPC, AI, and data analytics
- Executes workflows on complex heterogeneous systems
- It provides a path to integrate emerging architectures into existing software eco-systems
- It provides a vehicle for HW/SW co-design



COMET Domain Specific Compiler: Overview

- COMET is a compiler infrastructure that focuses on computational chemistry, graph analytics and AI application domains
- COMET supports various frontends, including DSL, numPy, SciPy and RUST
- COMET compiler infrastructure
  - Abstraction for dense/sparse storage formats to support wide range range of sparse storage formats
  - **Multi-level** compilation pipeline, progressive lowering to leverage domain specific information at each stage
  - Domain specific optimizations for better performance efficiency
    - ✓ Reformulate a tensor contraction operation via Transpose-transpose-GEMM-transpose
    - ✓ Optimal loop-permutation and tiling
    - ✓ Semiring/masking support to reduce unnecessary computation
    - ✓ **Kernel Fusion** to avoid temporaries and redundant computation
    - ✓ Data layout optimizations to exchange data locality
  - Targets heterogeneous architectures, including CPUs, GPU, FPGAs, emerging dataflow architectures (e.g., SambaNova)



https://github.com/pnnl/COMET

Pacific Northwest NATIONAL LABORATORY

The Minos Computing Library (MCL)

- Framework for programming extremely heterogeneous systems
  - Programming model and programming model runtime
  - Abstract low-level architecture details from programmers
  - **Dynamic** scheduling of work onto available resources
- Key programming features:
  - Applications factored into tasks
  - **Asynchronous** execution
  - Devices are managed by the scheduler
  - Co-schedule **independent applications**
  - Simplified APIs and programming model
  - Flexibility:
  - Scheduling framework
  - Multiple scheduling algorithms co-exist
  - Code portability
  - Resources allocated at the last moment

Supported Architectures

- CPU (x86, Arm, RISC-V, etc.)
- GPU (NVIDIA, Intel, AMD, Arm)
- NVDLA
- Sambanova
- Micron Devices
- Xilinx FPGA
- Xilinx AIE
- SST, GEM5, etc.

https://minos-computing.github.io

Accelerating Computational Fluid Dynamics

- Computation Fluid Dynamics 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 condition
- Lattice Boltzman Method (LBM):
  - a fluid density on a lattice is simulated with streaming and collision (relaxation) processes. This method is more flexible than traditional Navier-Stokes equations
- Graphics rendering is separated but performed in-situ with the CFD simulation
  - Sharing data (communication)
  - Sharing resources (computation)
- MCL orchestrate data movement and resource sharing

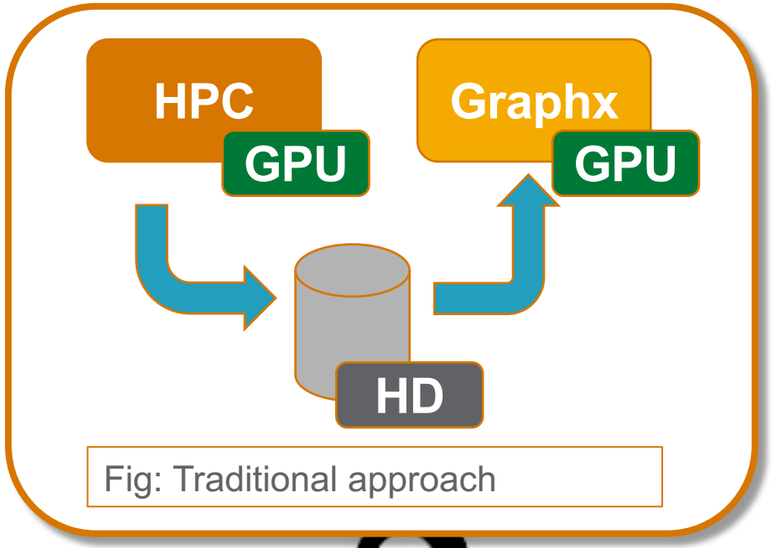


Fig: Traditional approach

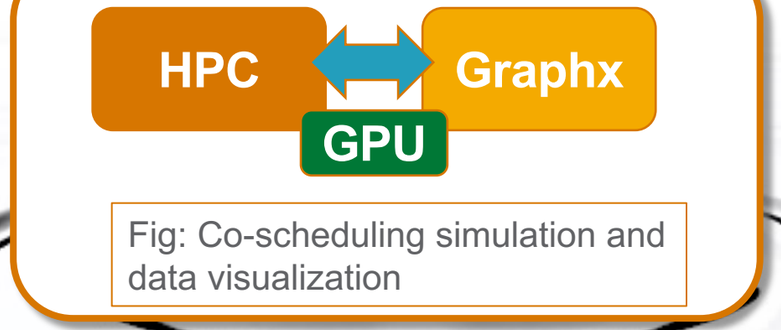
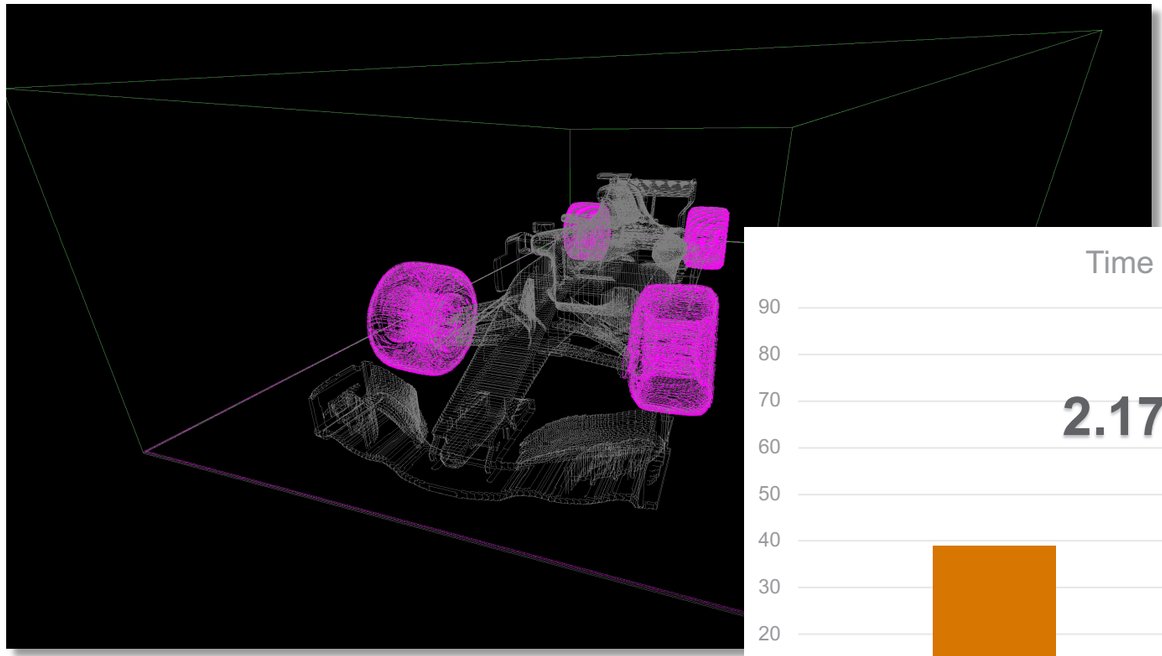
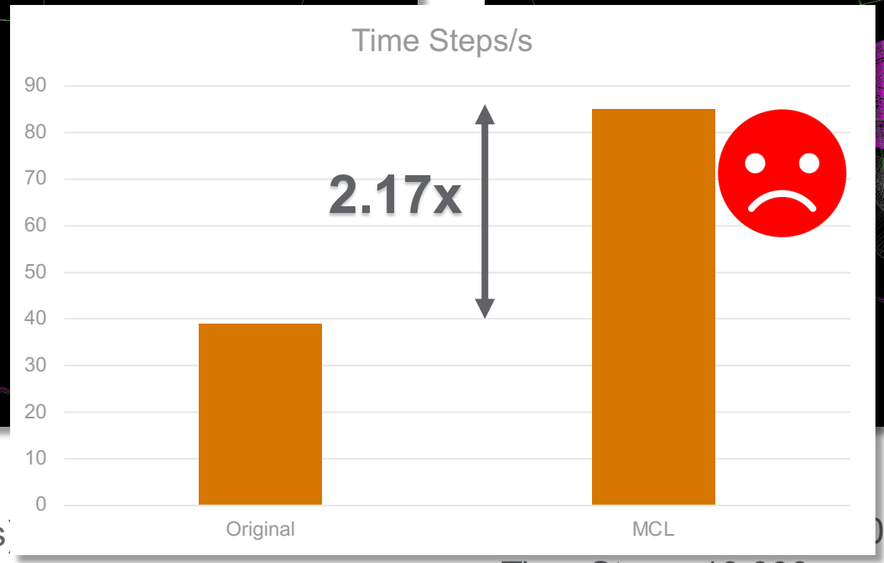
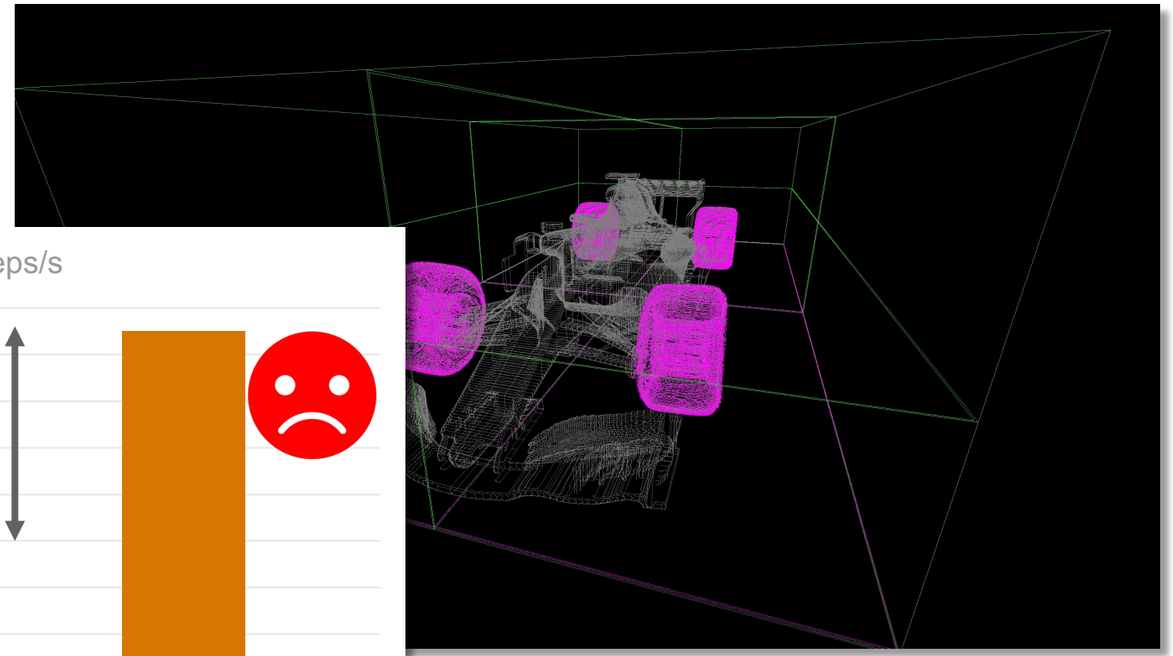


Fig: Co-scheduling simulation and data visualization

Simulation of Turbulence of Ferrari SF71H







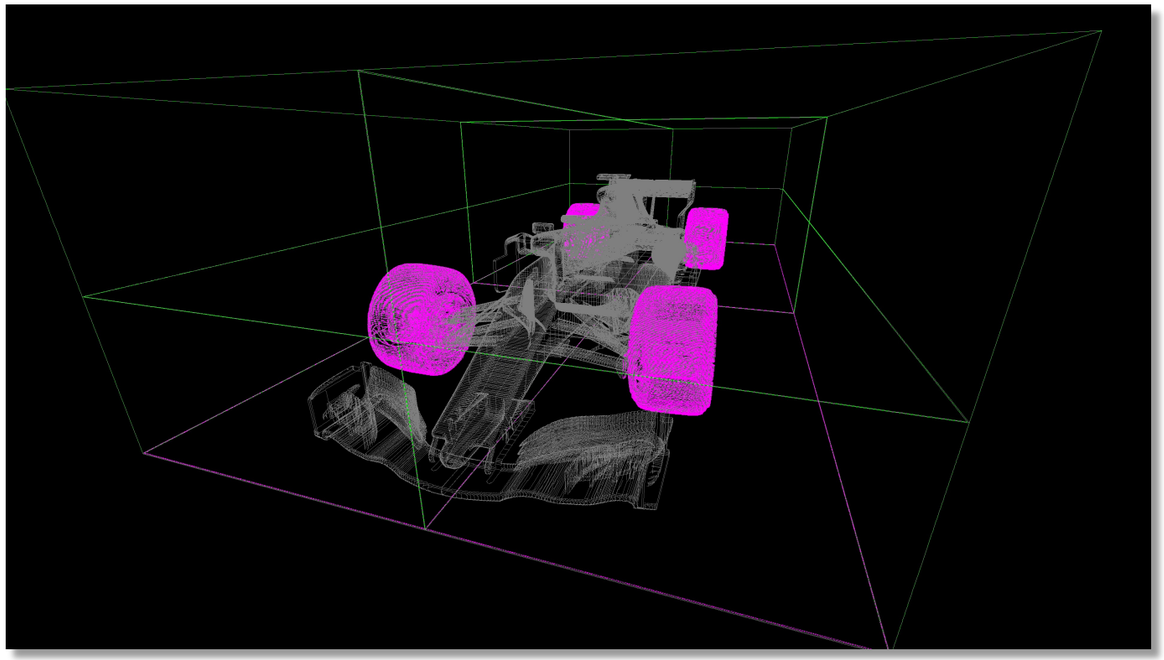
Model: Ferrari SF71H
Grid Resolution: 512x1024x256 (134M points)
Time Steps: 10,000
System 1: DGX-V100 (CEANTE)
Execution time: 298 sec
Time Steps/sec: 39

024x256 (134M points)
Time Steps: 10,000
System 1: DGX-V100 (CEANTE)
Execution time: 197 sec
Time Steps/sec: 85

Scaling up CFD Simulation

- If we have more resources (computing and memory) we don't necessarily run simulations faster (strong scaling)
- We attempt to solve larger problems (weak scaling)

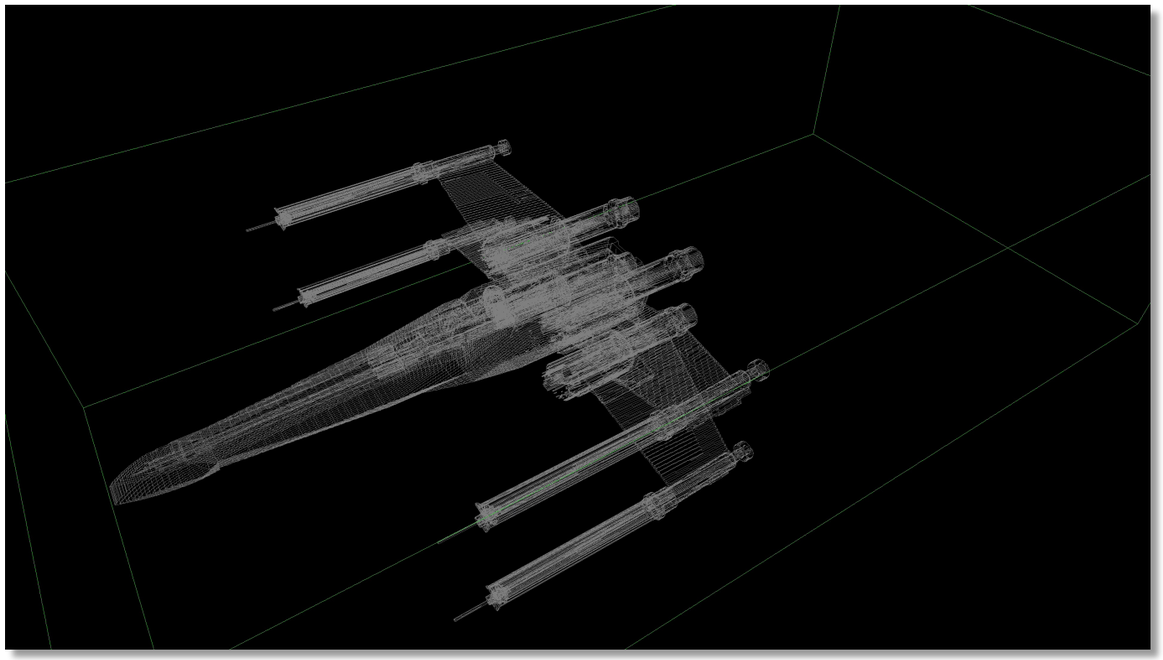
The main objective of developing more capable supercomputers is to enable *new science*



Model: Ferrari SF71H
Grid Resolution: 1024x2048x1024 (2.15B points)
Time Steps: 50,000
System 1: DGX-V100 (CEANTE)
Execution time: 3,232 sec
Time Steps/sec: 19

16.04x

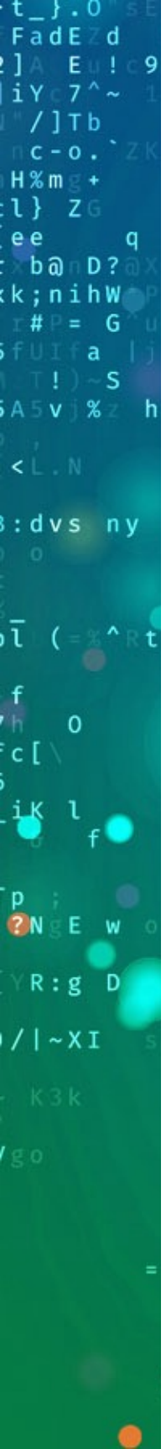
Simulation of Turbulence of Star Wars X-Wing



Model: Star Wars X-Wing
Grid Resolution: 768x1536x384 (453M points)
Time Steps: 3,000
System 2: PNNL Junction (GPU)
Execution time: 261 sec
Time Steps/sec: 13

- Simulation shows that the X-Wing is not as aerodynamic as the SF71H
- Cannons and wings pose resistance to the flow

Lesson learned: *George Lucas should have hired Mattia Binotto to design the X-Wing*

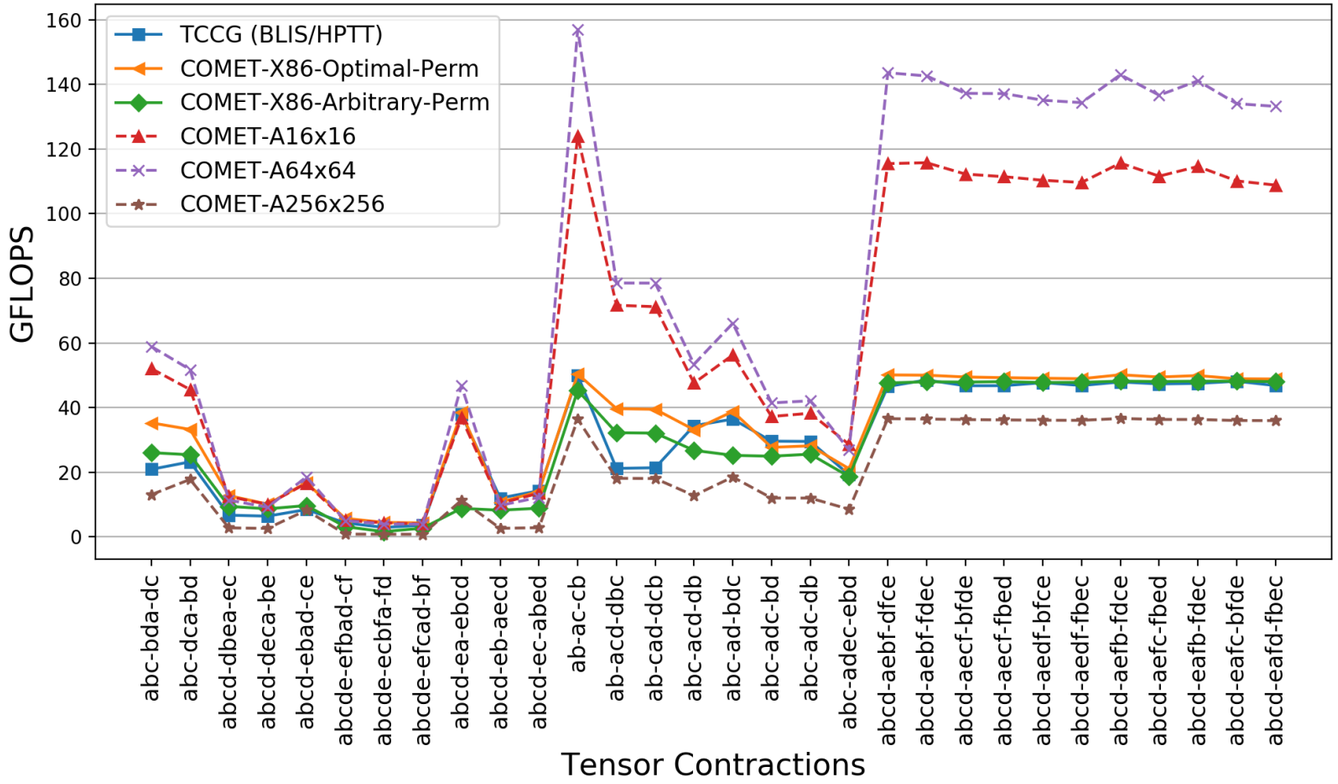


Pacific Northwest
NATIONAL LABORATORY

HW/SW Co-Design

Accelerating Tensor Contraction Operations

- Tensor contractions are common in many domains (chemistry, AI, powergrid, etc.)
  - Tensor contractions are often refactored as TTGT
- COMET generates code that is faster than hand-optimized
- Integrate with Harvard Aladdin to evaluate GEMM accelerators



Lesson Learned: *Bigger is (not always) better!*

Integration of Application-Specific Accelerators

- COMET automated code generation produces RTL descriptions of application-specific accelerators from high-level languages
- RTL descriptions can then be
  - Synthesized for hardware emulation
  - Tapeout
  - Successfully embedded into larger SoC designs (e.g., LBL MoSAIC architecture)

```
def main() {
#IndexLabel Declarations
IndexLabel [i] = [8];
IndexLabel [j] = [4];
IndexLabel [k] = [2];

#Tensor Declarations
Tensor<double> A([i, j], {Dense});
Tensor<double> B([j, k], {Dense});
Tensor<double> C([i, k], {Dense});

#Tensor Fill Operation
A[i, j] = 2.2;
B[j, k] = 3.4;
C[i, k] = 0.0;

C[i, k] = A[i, j] * B[j, k];
print(C);
}
```

Figure: COMET DSL GEMM source code

```
llvm.func @mcl_init(i64, i64) -> i32
llvm.mlir.global internal constant @gemm.xclbin("gemm.xclbin\00" : i8)
llvm.mlir.global internal constant @c_opts("" : i8)
llvm.func @mcl_prg_load(!llvm.ptr<i8>, !llvm.ptr<array<0 x i8>>, i64) ->
i32
llvm.func @mcl_task_create() -> !llvm.ptr<struct<(i64, i32, i64, i64,
i32)>>
llvm.func @mcl_task_set_kernel(!llvm.ptr<struct<(i64, i32, i64, i64,
i32)>>, !llvm.ptr<i8>, i64) -> i32
llvm.mlir.global internal constant @kernel_name("main_kernel\00" : i8)
llvm.func @mcl_task_set_arg(!llvm.ptr<struct<(i64, i32, i64, i64, i32)>>,
i64, !llvm.ptr<i8>, i64, i64) -> i32
. . .
llvm.func @mcl_exec(!llvm.ptr<struct<(i64, i32, i64, i64, i32)>>,
!llvm.ptr<array<3 x i64>>, !llvm.ptr<array<3 x
i64>>, i64) -> i32
llvm.func @mcl_wait(!llvm.ptr<struct<(i64, i32, i64, i64, i32)>>) -> i32
func private @comet_print_memref_f64(memref<*xf64>)
```

Figure: COMET-generated host program (MLIR)

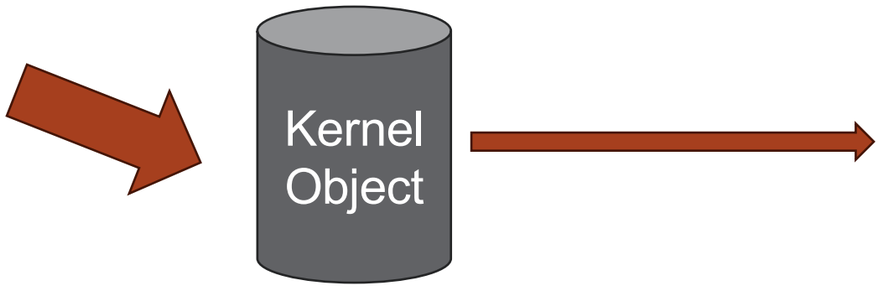


Figure: COMET-generated kernel object (xclbin)

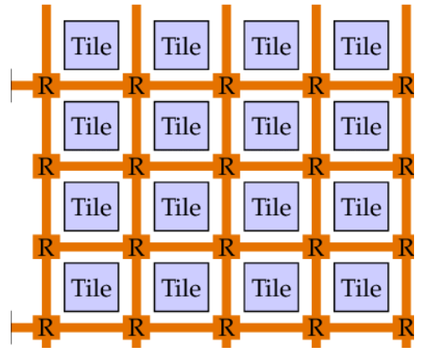


Figure: LBL MOSAIC architecture

GNN Acceleration to Meet Energy Efficiency

GNNs are known for their ability to learn structural information. Usually, nodes with similar features or properties are connected to each other (e.g., social media)

- Transformation step: features vector *X* is transformed through an affine operator $F(x_i) = W_i * x_i$, where x_i is the vector of features of vertex *i* and *W* is the transformation operator
- Aggregate step: the information from all neighbors of each node, multiply $\bar{A}$ by H, $Y = \bar{A} * H = \bar{A} * W * X$.
- Repeat process for each layer *l*, using H^{l-1} as input of layer *l*, i.e., $H^l = \bar{A} * W * H^{l-1}$

- Two main kernels: DGEMM and SPMM

```
1  # Graph Neural NetwoRK (GNN)
2
3  def main() {
4      #IndexLabel Declarations
5      IndexLabel [i] = [?];
6      IndexLabel [k] = [?];
7      IndexLabel [j] = [4];
8      IndexLabel [h] = [4];
9
10     #Tensor Declarations
11     Tensor<double> B([i, k], {CSR});
12     Tensor<double> C([k, h], {Dense});
13     Tensor<double> D([h, j], {Dense});
14     Tensor<double> A([i, j], {Dense});
15     Tensor<double> T([i, h], {Dense});
16
17     #Tensor Data Initialization
18     B[i, k] = comet_read(0);
19     C[k, h] = 1.2;
20     D[h, j] = 3.4;
21     A[i, j] = 0.0;
22     T[i, h] = 0.0;
23
24     T[i, h] = B[i,k] * C[k,h];
25     A[i, j] = T[i, h] * D[h, j];
26     print(A);
27 }
```

Figure: COMET DSL implementation of GNN. The COMET DSL allows users to express their computation with familiar notations. The COMET compiler can distinguish the type of operator (e.g., dense-dense matrix multiplication vs sparse-dense matrix multiplication) from the format of the inputs. The outputs are directly stored in sparse format, avoiding the "densification problem".

Fully-Integrated HW/SW Co-Design for GNNs

- Designed a prototype SoC that combines:
  - Third-party RISC-V-based GPGPU (GT Vortex)
  - COMET-generated SPMM and DGEMM kernels
  - Could integrate ARM or RISC-V CPU for complete MPSoC
  - Fully emulated in FPGA
- The SoC can be combined with actual CPU and GPU to:
  - Explore mapping opportunities
  - HW/SW co-design fully integrated with other technologies
  - Explore trade-offs between custom and general-purpose technologies

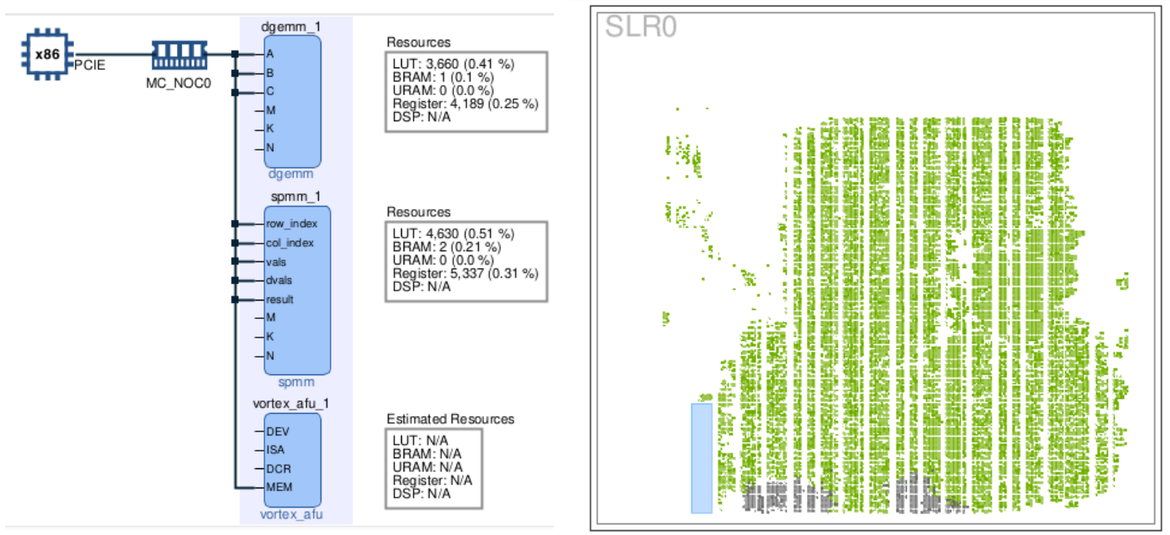


Figure: SoC layout (RISC-V GPU, SpMM, and DGEMM)

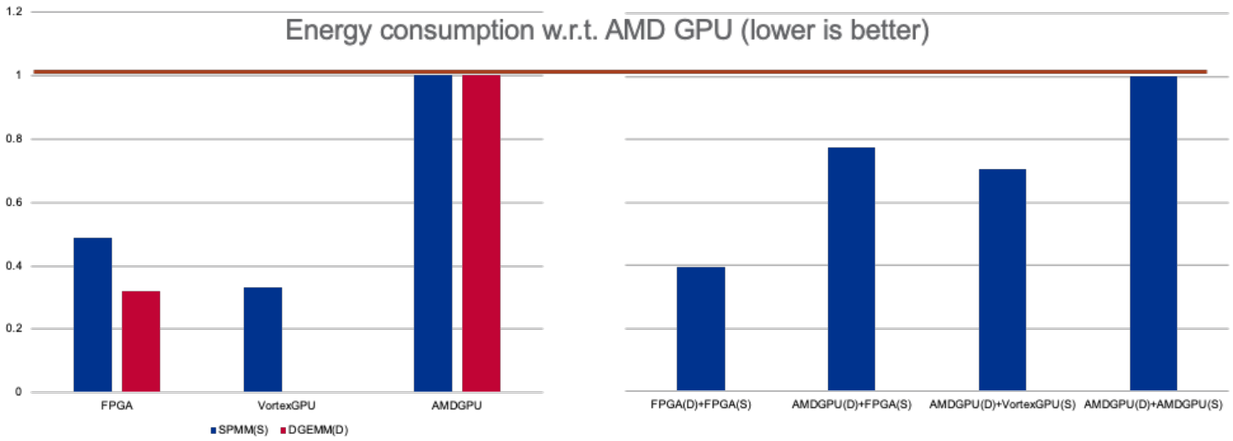


Figure: Energy-to-completion of various configuration for GNN pass

Pacific Northwest NATIONAL LABORATORY

Conclusions

- The current trajectories for AI, data analytics, and science is not sustainable without fundamental innovation
  - Reduce waste
  - Increase efficiency
- Need of converged, composable, integrated, and flexible SW/HW stack
  - Fosters collaborations among researchers
  - Enables holistic HW/SW co-design of applications workflows and novel HW computing and memory technologies
- PNNL SW stack is an attempt to develop a sustainable soluiton:
  - Converged AI, HPC, data analytics workflows
  - Used in various projects with multiple sponsors and collaborators
  - Enables HW/SW co-design from high-level DSLs to RTL
- Reach out if you're interested (of just curious) in collaborating with us!

Pacific Northwest NATIONAL LABORATORY

Collaborators & Sponsors

- PNNL:
  - Rizwan Ashraf
  - Ryan Friese (Lamellar PoC)
  - Lenny Guo
  - Gokcen Kestor (COMET PoC)
  - Zhen Peng
  - Polykarpos Thomadakis

- Current/Former Students:
  - Alok Kamatar
  - Evan Ruttenberg
  - Ruiqin Tian
  - Blaise Tine
  - Tong Zhou

U.S. DEPARTMENT OF ENERGY | Office of Science

CO-DESIGN CENTER FOR ARTIFICIAL INTELLIGENCE-FOCUSED ARCHITECTURES AND ALGORITHMS (ARIAA)

DMC DATA-MODEL CONVERGENCE INITIATIVE @PNNL

Pacific Northwest
NATIONAL LABORATORY

Thank you

Roberto.gioiosa@pnnl.gov