MVAPICH
MPI, PGAS and Hybrid MPI+PGAS Library

HiBD
High-Performance
Big Data

HiDL
High-Performance
Deep Learning

Using BlueField-3 SmartNICs to Offload Vector Operations in Krylov Subspace Methods

Kaushik Kandadi Suresh, Benjamin Michalowicz, Nick Contini,
Bharath Ramesh, Mustafa Abduljabbar, Aamir Shafi, Hari Subramoni,
Dhabaleswar K (DK) Panda

{kandadisuresh.1, michalowicz.2, contini.26, ramesh.113, shafi.16,
subramoni.1, panda.2}@osu.edu

Presented initially @ HiPC 2024

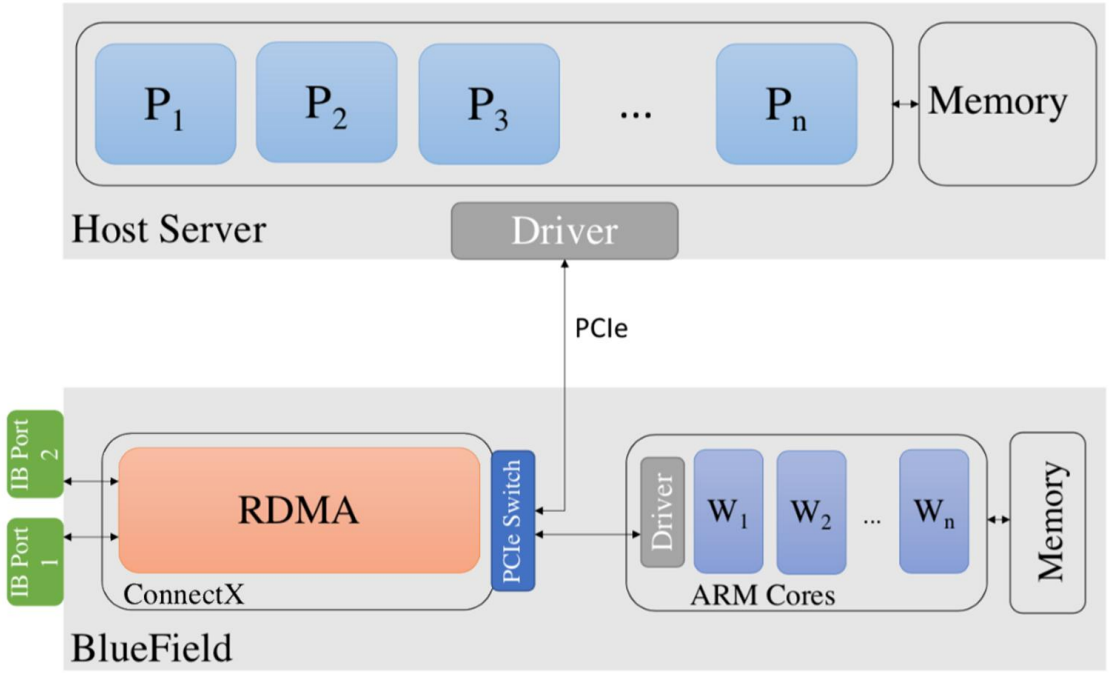
Department of Computer Science and Engineering
The Ohio State University

Outline

- Introduction
- Motivation
- Contributions
- Design
- Evaluation Results
- Conclusion and Future Work

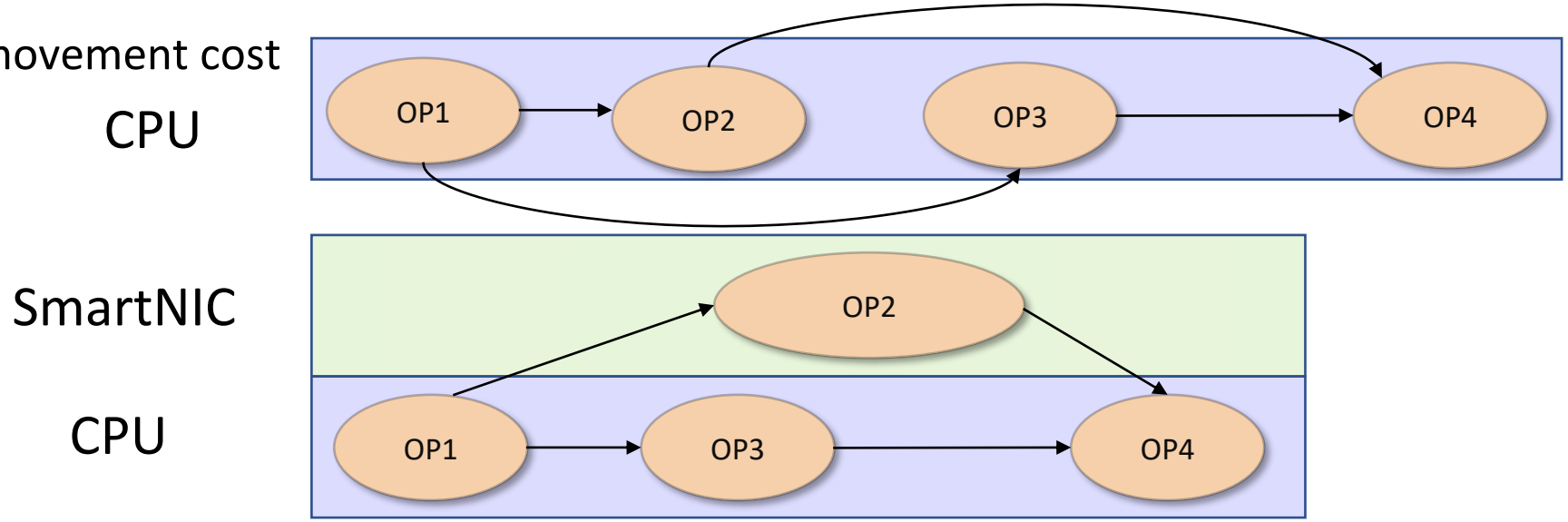
Overview of BlueField-3 DPU/Smart NIC

- InfiniBand network adapter with up to 400Gbps speed
- System-on-chip containing 16 64-bit ARMv8.2 A78 cores with 2.75 GHz each
- 16 GB of memory for the ARM cores



Offloading Computation to the SmartNIC

- SmartNICs :
  - Have specialized hardware units for encryption, compression, etc
  - Also have general purpose programmable cores. Eg: NVIDIA's BlueField-3 (BF3) DPU
- SmartNIC cores are not as powerful as CPUs
  - Unlike GPUs, can't offload all the operations. Only beneficial to offload the Op2 in the figure below.
  - Offloading involves data movement cost

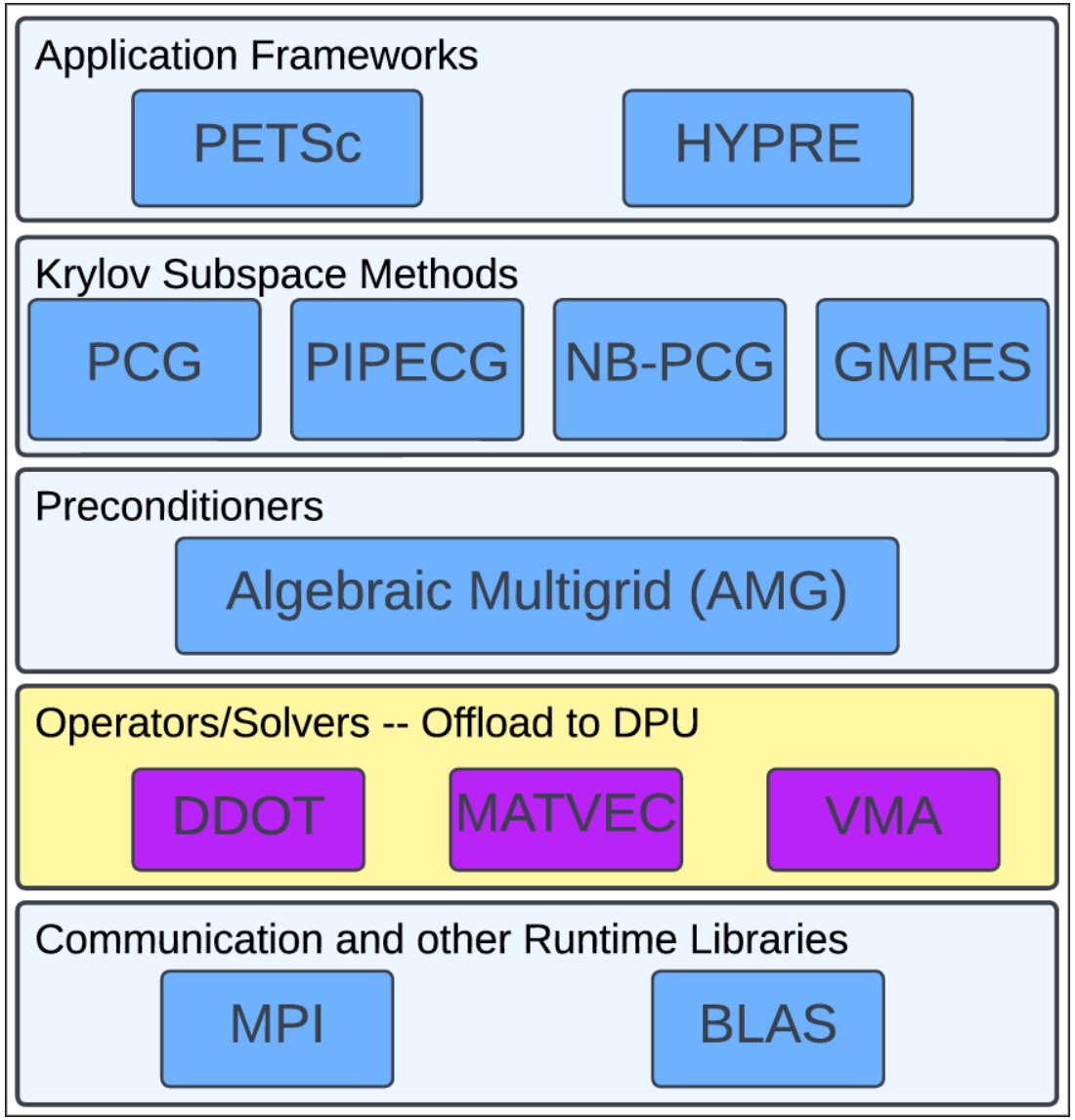


- How to efficiently select and offload the following operations in Krylov Solvers ?
  - Vector-Multiply-Add (VMA), Distributed DOT Product (DDOT), Matrix-Vector Multiplication

Krylov Offload Framework

- Krylov Solvers
  - Numerical methods to solve large sparse linear systems of equations
  - Widely used in HPC applications
  - Eg: Pre-conditioned Conjugate Gradient (PCG)

- Commonly used operations:
  - DDOT: Distributed Dot Product
  - MATVEC : Matrix-Vector multiplication
  - VMA: Vector Multiply Add (AXPY)



Proposed Offload Framework

Outline

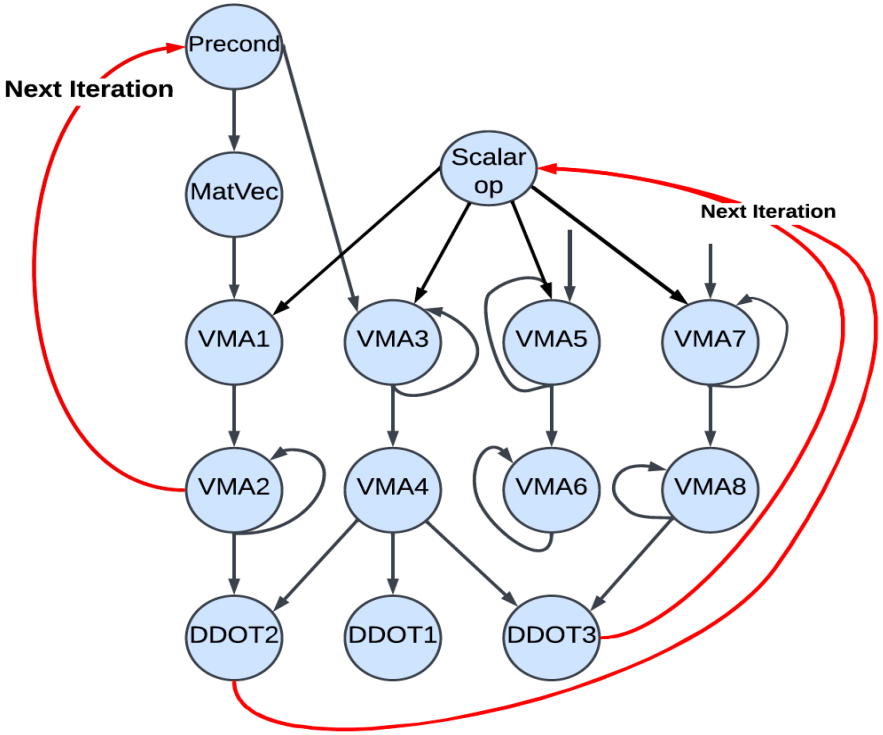
- Introduction
- Motivation
- Contributions
- Design
- Evaluation Results
- Conclusion and Future Work

Motivation

Problems in past works on DPU offload:

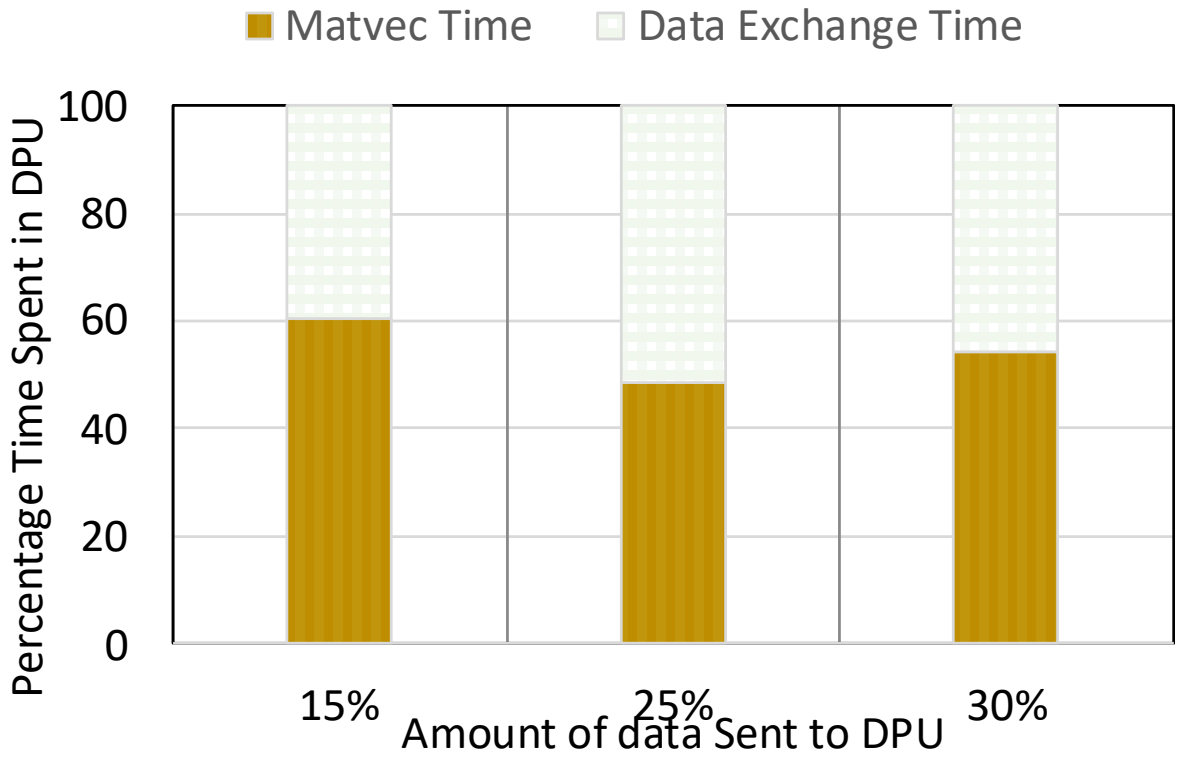
1. Manually identification of tasks to be offloaded
2. Host-to-DPU data movement bottleneck not addressed

- Different PCG algorithms have different types of data dependencies
- Manual identification is tedious and inefficient



(1) Data flow graph for PIPEPCG Algorithm

- Offloading to DPU involves moving vectors to DPU memory
- Data movement overhead can consume at-least 30% of the total time



(2) Data movement cost for moving data to BF3 DPU for AMG Matrix-Vector Multiplication

Outline

- Introduction
- Motivation
- Contributions
- Design
- Evaluation Results
- Conclusion and Future Work

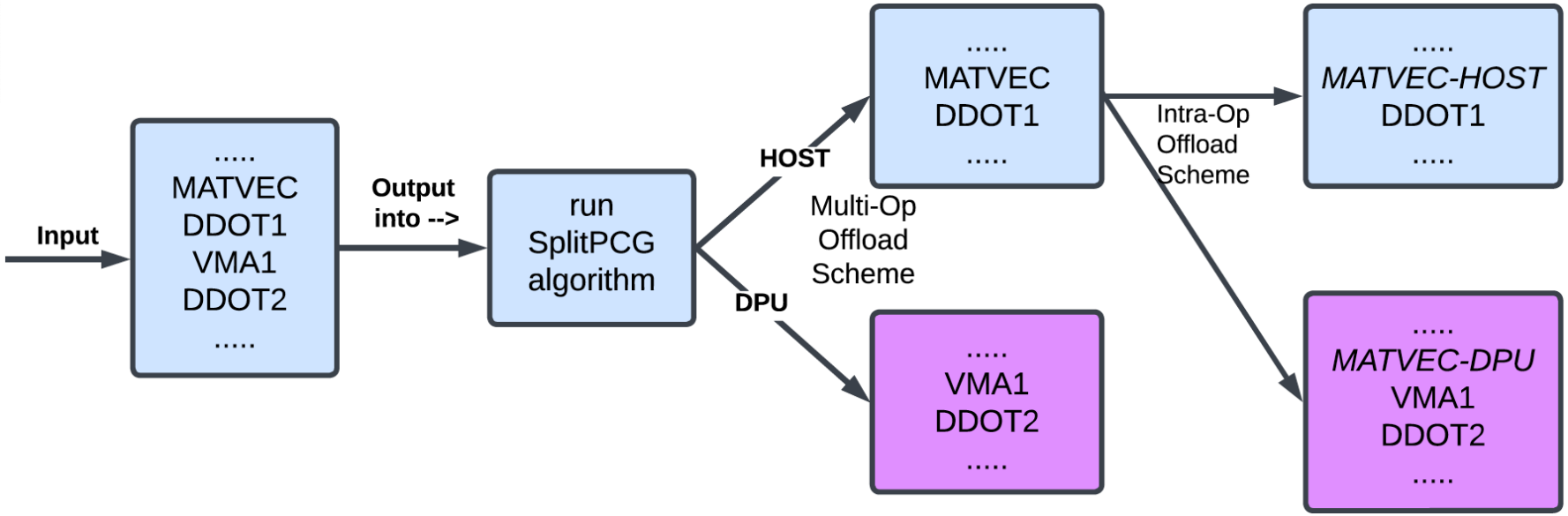
Contributions

- We propose a generalized splitting scheme that can be applied to offload any PCG algorithm for generating a set of host and DPU operations to be offloaded
- We reduce the data transfer overhead by designing and implementing the proposed onloading scheme.
- We show the impact of our designs on 1) AMG-PCG benchmark and 2) PETSc KSPSolve benchmark
- Demonstrate the efficacy of the proposed designs on two testbeds
  - Testbed-1 : Broadwell+BF3
  - Testbed-2 : Sapphire Rapids (SPR) + BF3

Outline

- Introduction
- Motivation
- **Implementation**
- **Evaluation Results**
- **Conclusion and Future Work**

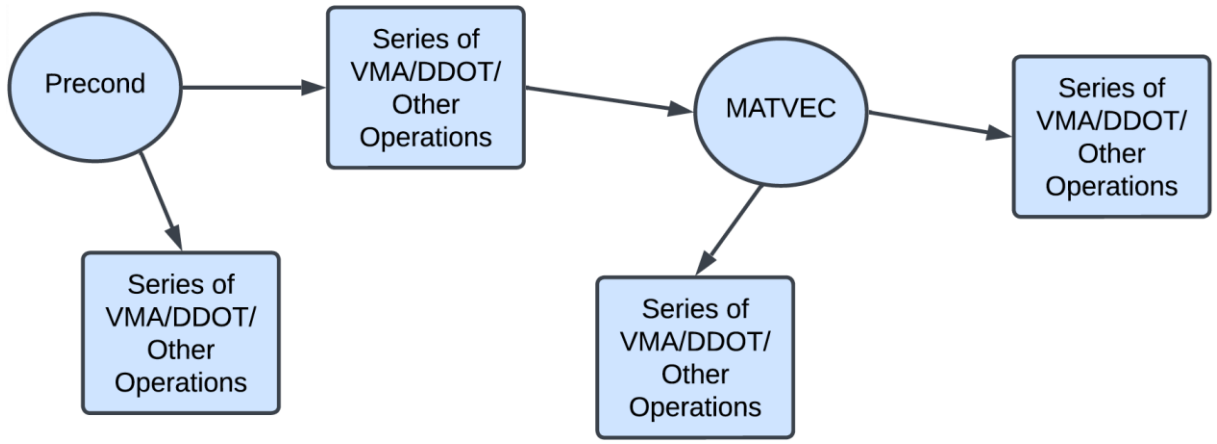
Design Framework



- Splitting Scheme
  - Program that outputs the order of operations to be performed on host and DPU
- Multi-Op Offload
  - Offload a set of VMA/DDOT operations to the DPU
- Intra-Op Offload
  - Offload Matvec operation to the DPU

Designing the Splitting Scheme

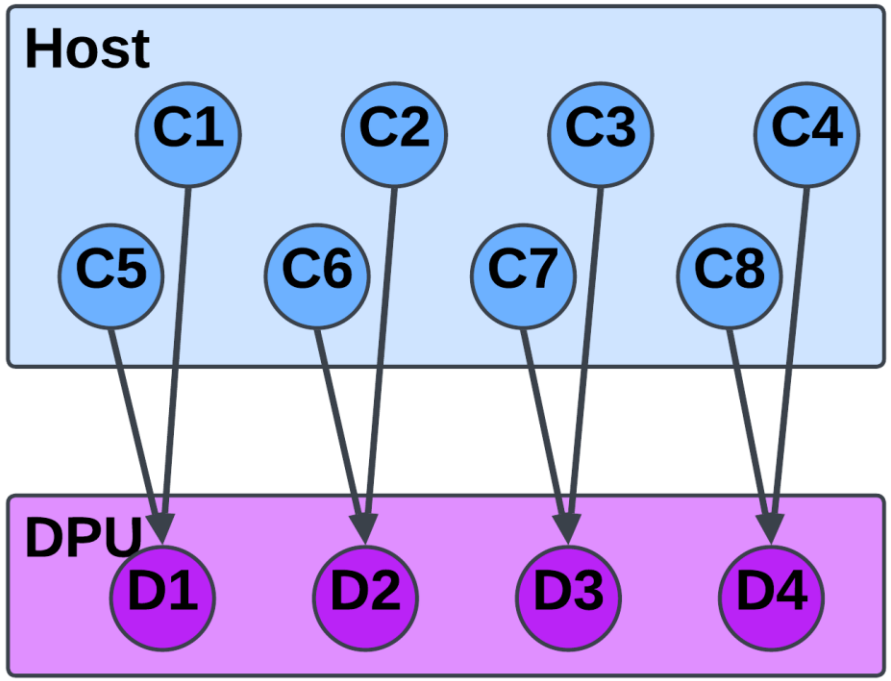
- Construct Data-Flow graph
- Assign relative weights to each node
- Find Dominant path between PC and Matvec nodes
  - Path with maximum weight
  - Scheduled on the host
- Print the topologically sorted order of dominant and non-dominant operations



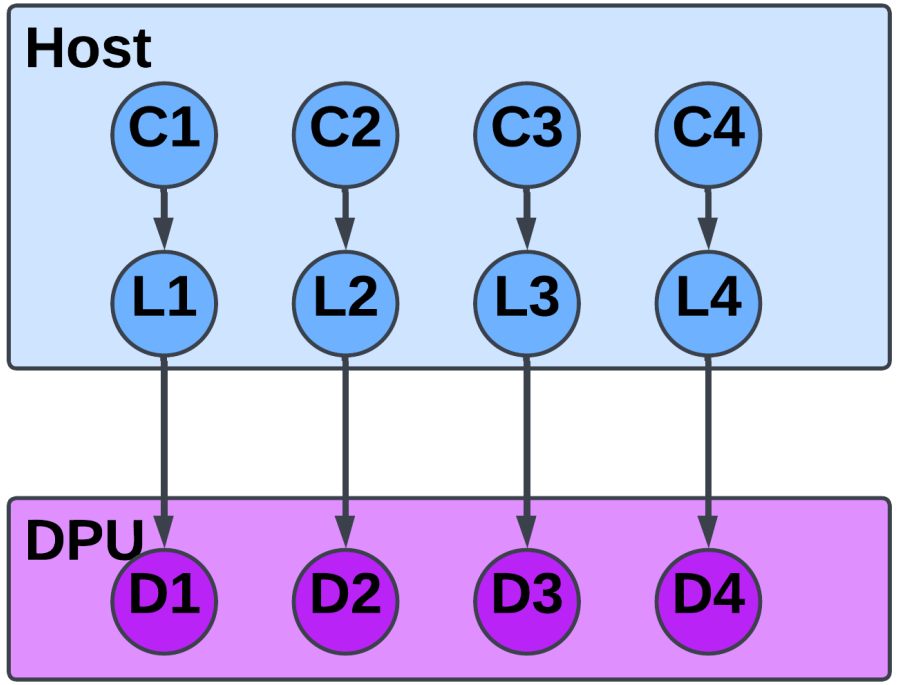

```
1  Function splitPCGAlgo(opList):
2      opGraph ← buildOpGraph(opList)
3      pcOp ← findPCop(opList)
4      mvOp ← findMVECop(opList)
5      pcmvPaths ← findAllSimplePaths(pcOp, mvOp)
6      hostSplit.add(findDominantPaths(pcmvPaths))
7      dpuSplit.add(findOtherNodes(opGraph, hostSplit))
8      (sHostSplt, sDpuSplit) ← topoSort(opGraph,
         hostSplit, dpuSplit)
9      (sHostSplt, sDpuSplit) ←
         addDataExchOps(opGraph, sHostSplit, sDpuSplit)
```

Onloading Scheme

- Default Offloading scheme
  - All host processes offload to DPU workers



- Proposed Onloading scheme
  - Leader host processes offload more work to DPU workers
  - Non-Leader host processes onload work to leader host processes



Outline

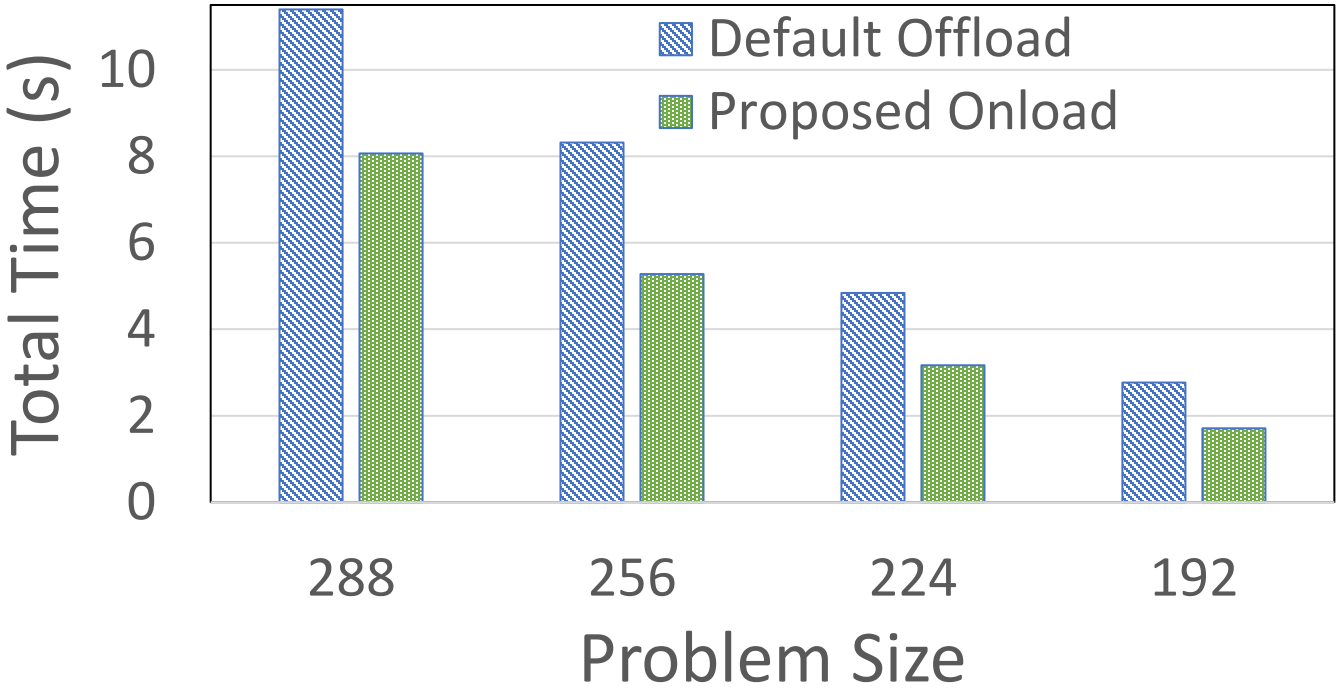
- Introduction
- Motivation
- Contributions
- Design
- Evaluation Results
- Conclusion and Future Work

Experimental Setup

	Testbed-1	Testbed-2
CPU	2x 20-cores Intel Xeon Gold 6138	2x 48-core Intel Xeon Platinum 8468 (Sapphire Rapids)
Memory	256 GB DDR4 DIMMs	256 GB DDR5 DIMMs
NIC	NVIDIA ConnectX-6 HDR100	NVIDIA ConnectX-6 HDR100
SmartNIC	NVIDIA BF3 SmartNIC	NVIDIA BF3 SmartNIC
Node Count	16	1

Impact of the Onloading Scheme for Multi-Op Offload

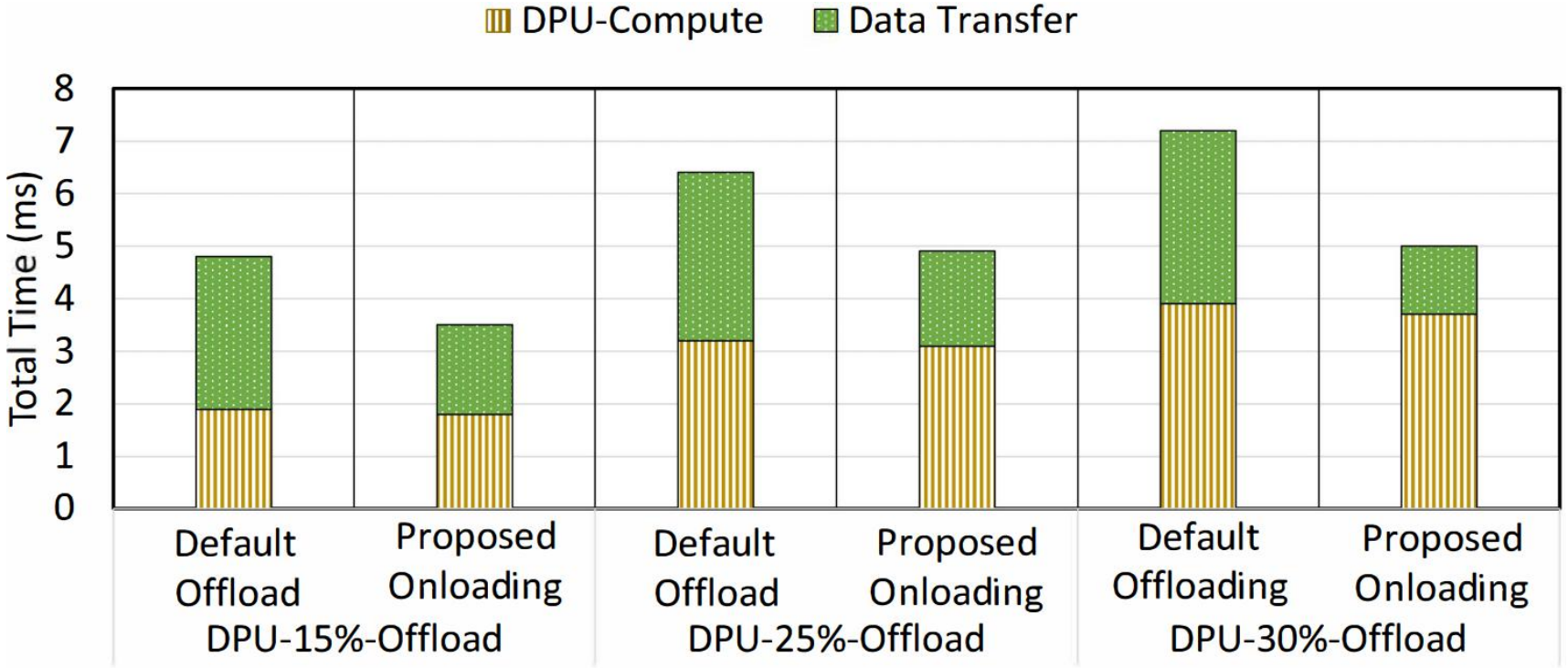
- PIPECG MultiOp Offload with PETSc benchmark
- Upto 38% improvement compared to the default offload scheme
  - Only 48 processes offload to the DPU in the proposed-scheme
  - This leads to Reduced Communication Overhead -> improved performance



1 Node, 96 PPN, 16 DPN, Testbed-2

Impact of the Onloading Scheme for Intra-Op Offload

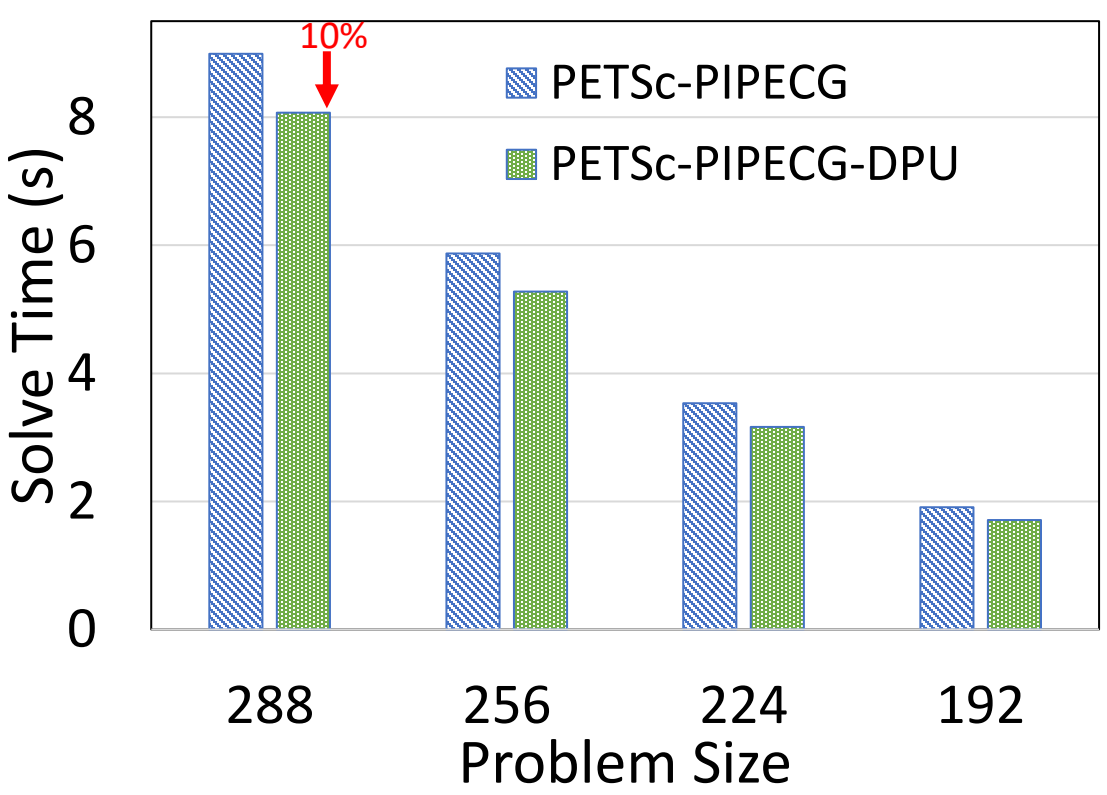
- Intra-Op Offload for Matvec Operation in AMG-PCG
- Maximum benefits at 15% offload
- At least 2X improvement in data-transfer time



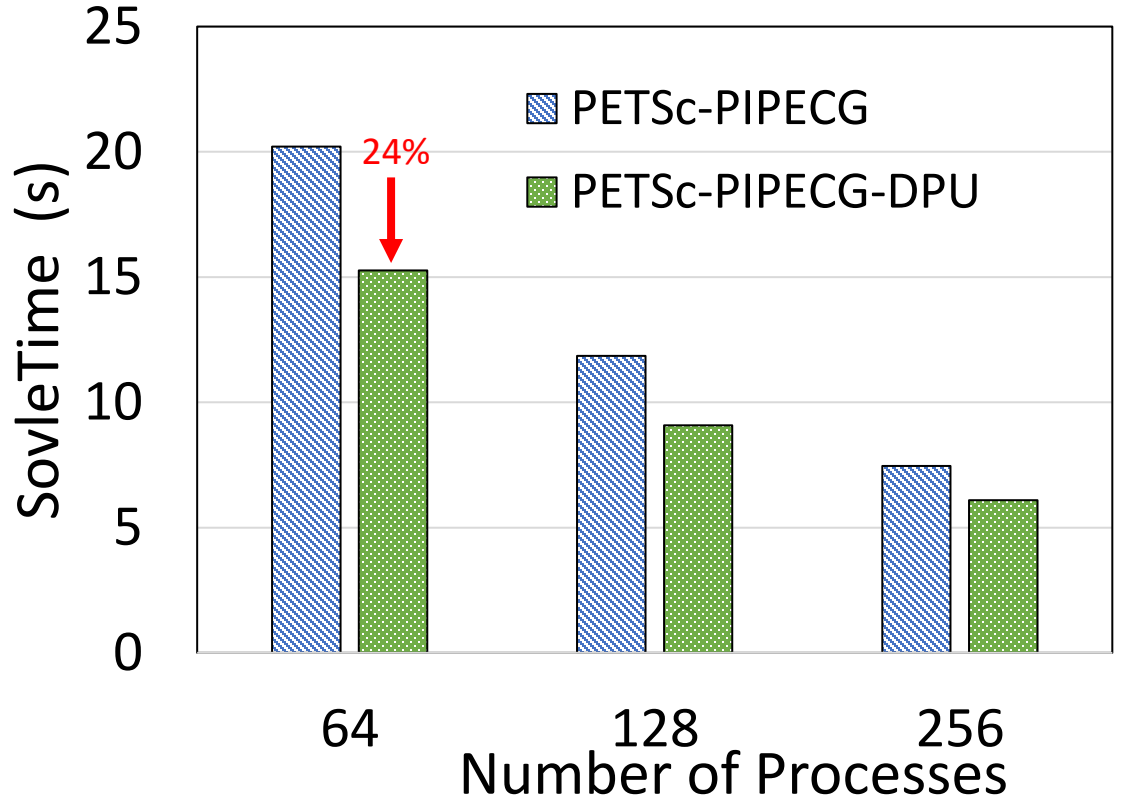
2 Nodes, 32 PPN, 16 DPN on Testbed-1

PIPECG Offload Results

- PETSc Benchmark: Solves 3D Laplacian with 27-point finite difference stencil with PIPECG solver.
- Improvements up-to 24% and 10% on Testbeds 1 and 2 respectively
  - Reduced improvement on Testbed-2 since the host-CPU is faster
  - Benefits are primarily due to :
    - 1) overlap of selected VMA,DDOT operations with other operations and 2) Onloading scheme



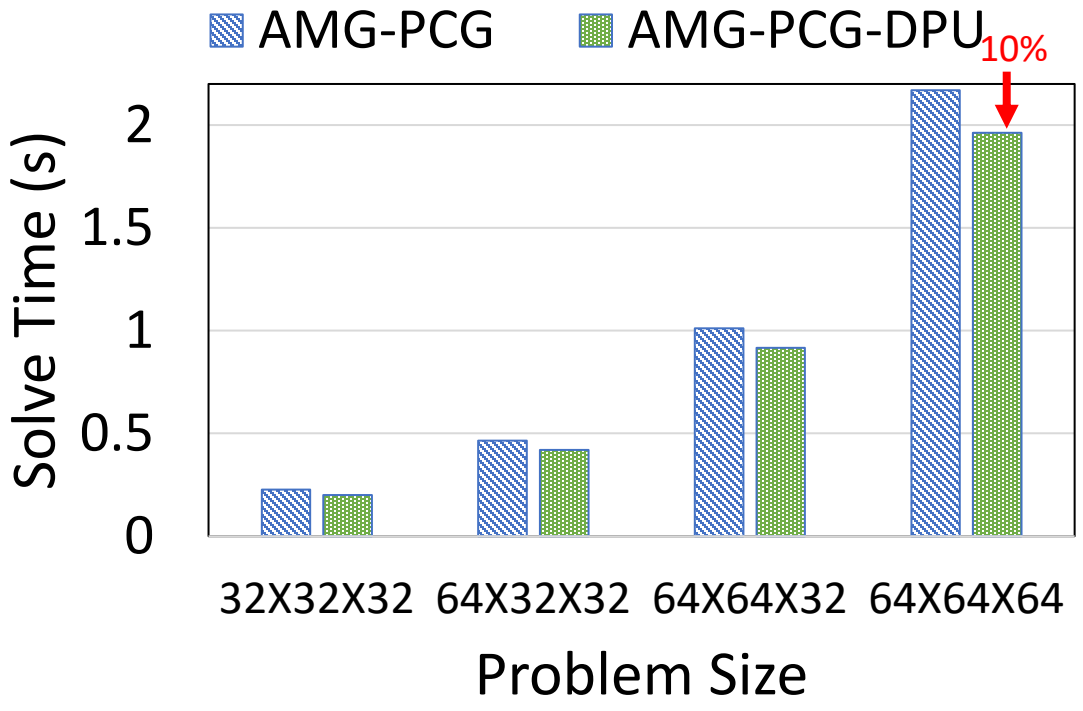
1-Node 96 PPN, Testbed-2 (SPR+BF3)



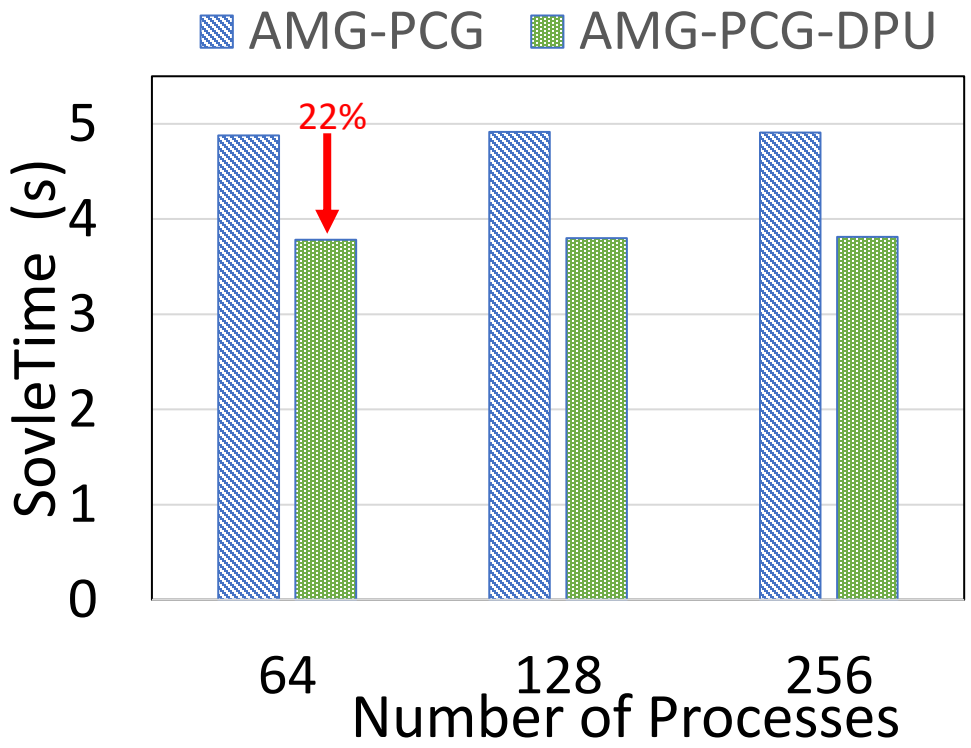
32 PPN, Strong Scaling, Testbed-1 (Broadwell+BF3)

AMG-PCG Offload Results

- AMG benchmark runs PCG solver with Algebraic Multi-Grid Preconditioner
- Improvements up-to 22% and 10% on Testbeds 1 and 2 respectively
  - Matvec operation dominates the total time
  - Onloading Intra-Op scheme contributed to the improvements



1-Node 96 PPN, Testbed-2 (SPR+BF3)



32 PPN, Strong Scaling, Testbed-1 (Broadwell+BF3)

Outline

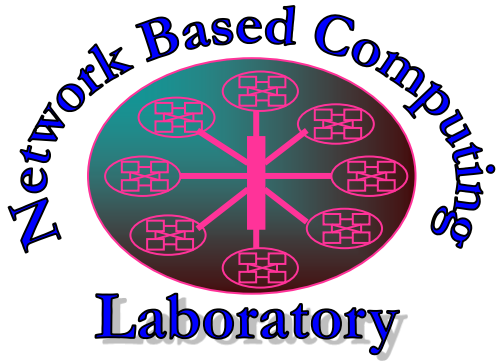
- Introduction
- Motivation
- Contributions
- Design
- Evaluation Results
- **Conclusion and Future Work**

Conclusion and Future Work

- Conclusions:
  - We identified three key operations to offload: DDOT, VMA, and Matrix-Vector multiplications
  - We designed multi-Op and intra-Op schemes to offload these operations to the DPU.
  - In the multi-Op offloading strategy we proposed a generalized splitting scheme to segregate a set of VMA and DDOT operations.
  - We proposed the onloading scheme to reduce the communication time for intra-op and multi-op offloading strategies.
  - We showed up-to 24% and 21% improvements on PETSc PIPECG and AMG-PCG benchmarks respectively for 256 processes.
  - Furthermore, we also showed up-to 10% improvement in a cluster with Sapphire-Rapids CPU and BF3 .

- Future Work:
  - Extend our optimization to other algorithms such as FFTs, QR algorithms for finding Eigenvalues
  - Evaluate our designs with other modern CPUs+BF3 combinations such as AMD Genoa + BF3, Grace+BF3.

Thank You!



Network-Based Computing Laboratory
http://nowlab.cse.ohio-state.edu/

Follow us on
https://twitter.com/mvapich

MVAPICH
MPI, PGAS and Hybrid MPI+PGAS Library

The High-Performance MPI/PGAS Project
http://mvapich.cse.ohio-state.edu/

HiBD
High-Performance
Big Data

The High-Performance Big Data Project
http://hibd.cse.ohio-state.edu/

HiDL
High-Performance
Deep Learning

The High-Performance Deep Learning Project
http://hidl.cse.ohio-state.edu/