SAND2020-13290PE

Intelligent Networks for High-Performance Computing

Whit Schonbein

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Introduction

- Resurgence of interest in “SmartNICs”

- Expectations regarding what these systems can do
  - Free host resources
  - Accelerate application execution
  - Failure recovery
  - Data staging
  - Checkpointing
  - And more

Grant, R.E., Schonbein, W., Levy, S. (2020) ‘RaDD Runtimes: Radical and Different Distributed Runtimes with SmartNICs’, *Fourth Annual Workshop on Emerging Parallel and Distributed Runtime Systems and Middleware (IPDRM)*, forthcoming.

Introduction

- Contributions of this work
  - Benchmarks for assessing overheads of multithreaded communication
  - In-Network Compute Assistance (INCA)
  - Assessment of application speedups afforded by INCA
  - Demonstration that INCA can enable `adaptive’ networks

Introduction

- What is a SmartNIC?

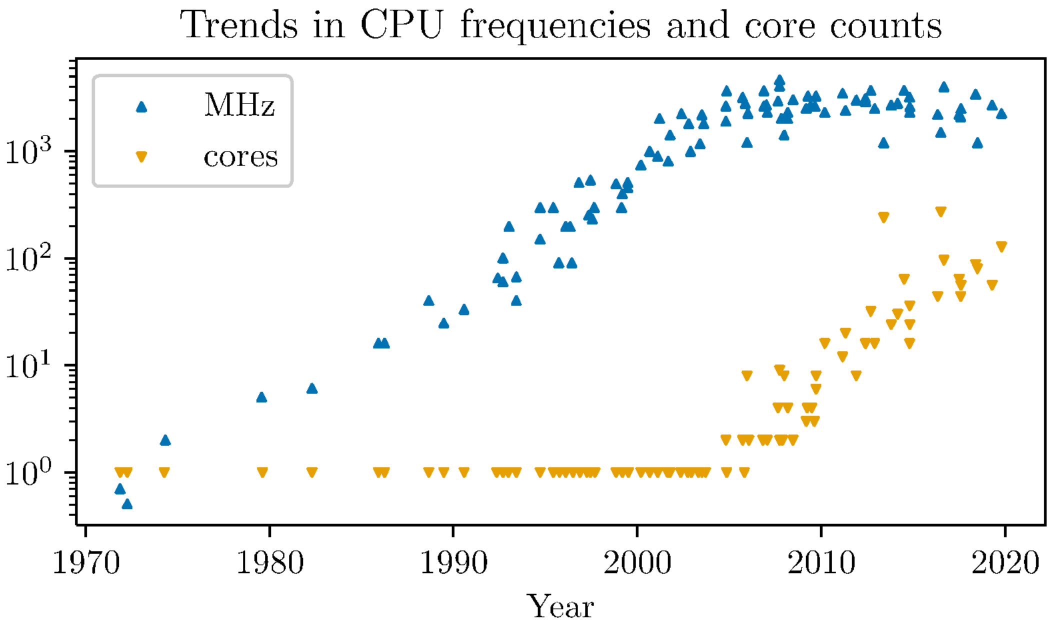
- A SmartNIC is a NIC that offloads core network applications

- In HPC:
  - Offloaded collectives
  - Offloaded message matching

From SmartNIC to ReallySmartNIC

"Our scheme is designed with the idea that as much processing as possible should be done by the host processor." (Buntinas et al. 2000, p. 1).

- Plateauing host resources
- Increasing demands on host resourc
  - Internet
  - Cellular
  - Big data/machine learning/AI
    - SC2018: 1 paper session on ML
    - SC2020: 6 paper sessions on ML



CPU data from: https://github.com/karlrupp/microprocessor-trend-data

From SmartNIC to ReallySmartNIC

- Next generation SmartNICs
  - Netronome Agilio
  - NVIDIA Mellanox Bluefield
  - Broadcom Stingray
  - Microsoft Catapult
  - Xylinx Versal
  - Stream Processing in-Network (SPiN)
- Flexibility
  - FPGAs, CPUs
  - Execute arbitrary kernels for manipulating network data

From SmartNIC to ReallySmartNIC

- Novel types of offloaded applications
  - Core network applications
  - Parts of host applications (data packing, consensus algorithms, CNN layers, object detection)
  - Fully independent applications (Catapult, microservices e.g. Amazon Lambda)

- Revised understanding of `SmartNIC'
  - NICs with FPGAs (Hanford et al. 2018)
  - NICs with CPUs (Liu et al. 2019)
  - Catapult: "beyond SmartNICs" (Caufield, 2016)

Introduction

- Contributions of this work
  - Benchmarks for assessing overheads of multithreaded communication
  - In-Network Compute Assistance (INCA)
  - Assessment of application speedups afforded by INCA
  - Demonstration that INCA can enable `adaptive’ networks

Benchmarking Multithreaded Message Matching

Multithreaded Message Matching

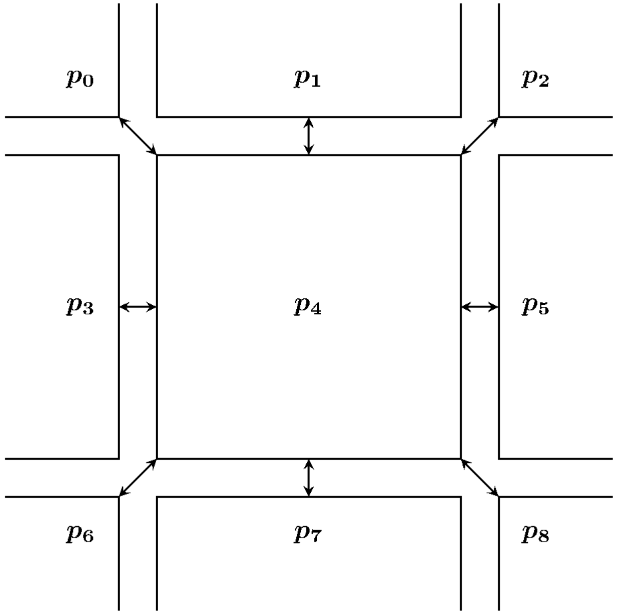
- Schonbein, W., Levy, S, Marts, W.P., Dosanjh, M.G.F., Grant, R.E. (2020) ‘Low-Cost MPI Multithreaded Message Matching Benchmarking’, *High Performance Computing and Communications (HPCC 2020)*, forthcoming.

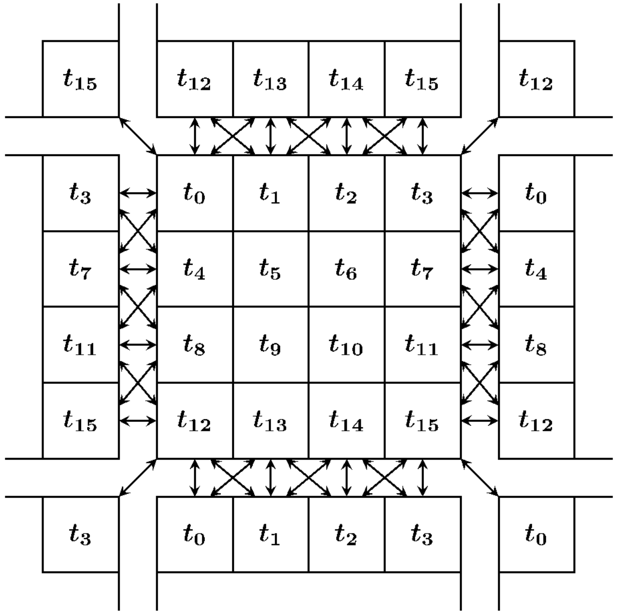
- Levy, S., Ferreira, K.B., Schonbein, W., Grant, R.E., Dosanjh, M.G.F., (2019) ‘Using Simulation to Examine the Effect of MPI Message Matching Costs on Application Performance’, *Parallel Computing*, Volume 84, May 2019, pp. 63-74.

- Schonbein, W., Dosanjh, M.G.F., Grant, R.E., Bridges, P.G. (2018) ‘Measuring multi-threaded message matching misery’, *Euro-Par 2018: Parallel Processing*, Turin, Italy, pp. 480-491.

Multithreaded Message Matching

- Halo exchange communication
- MPI_THREAD_MULTIPLE
- Multithreaded halo exchanges
- Question: What will this do to message processing overhead?

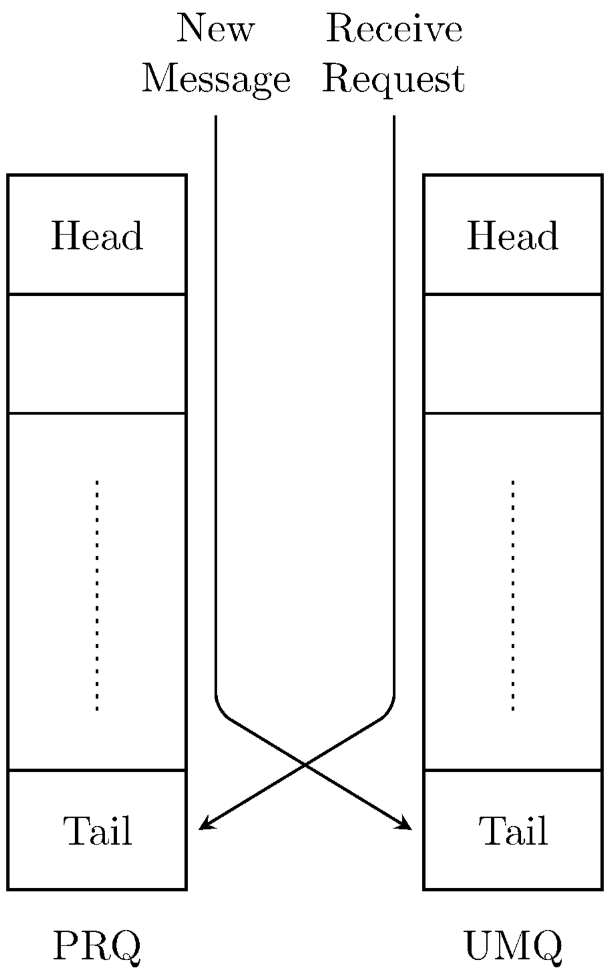




Multithreaded Message Matching

- MPI message processing overview
  - Posted Receive Queue (PRQ)
  - Unexpected Message Queue (UMQ)

- Hypothesis: MPI message processing overhead will increase
  - More messages
  - Non-determinism

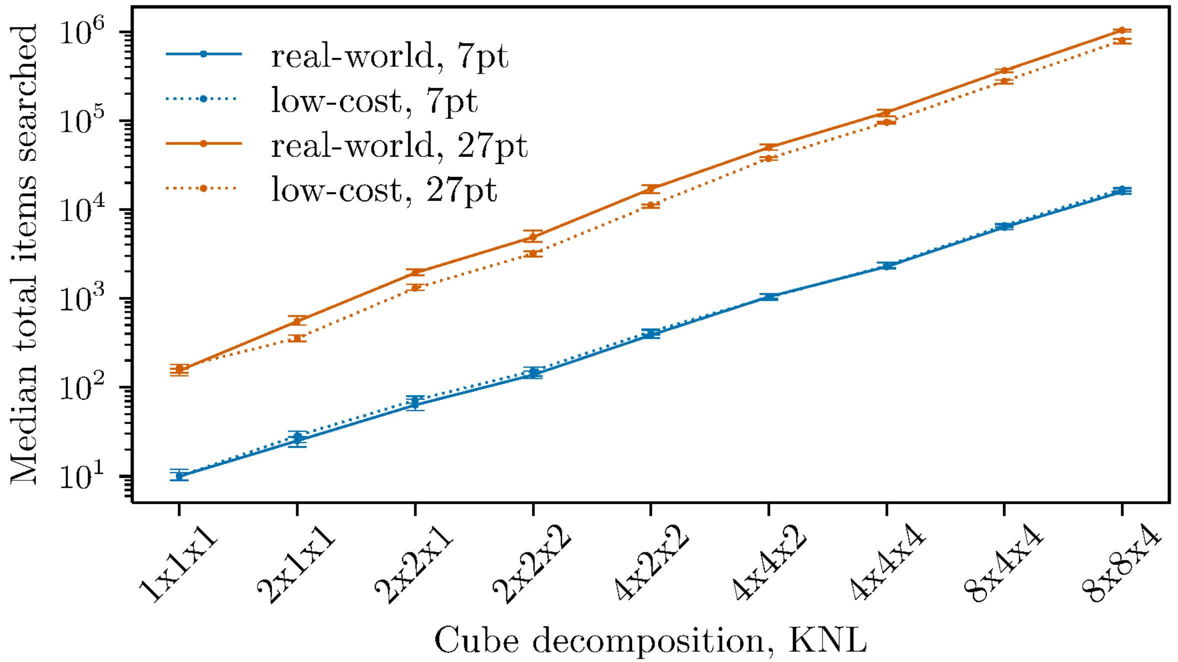


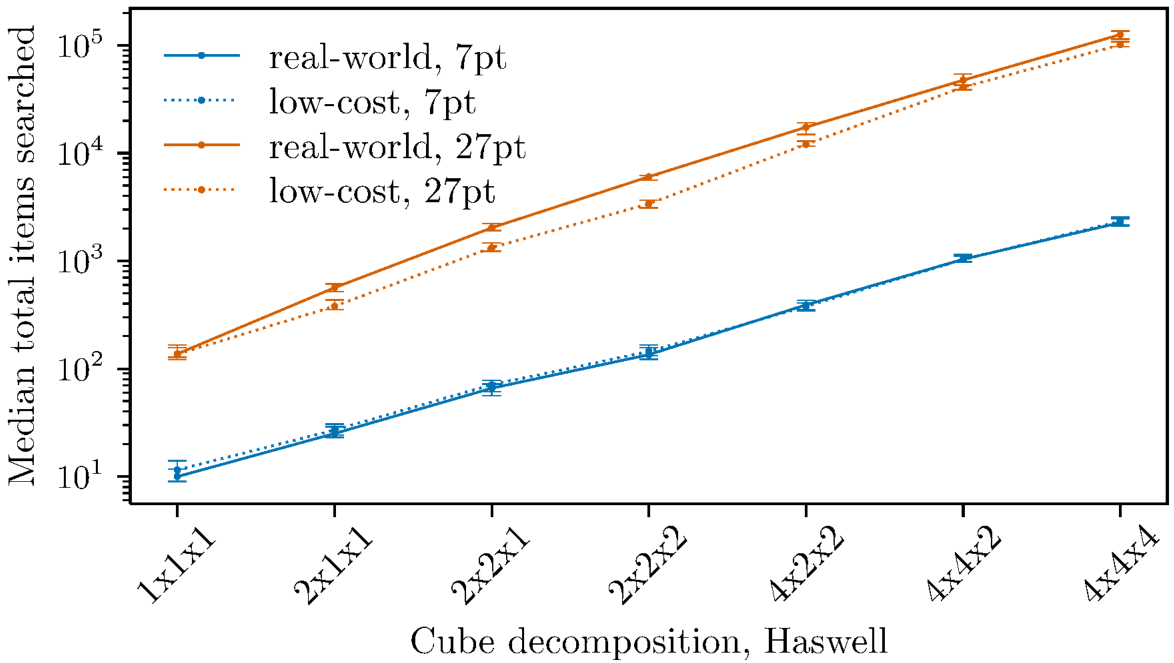
Multithreaded Message Matching

- Two benchmarks:
  - `Real-world’
  - `Low-cost’

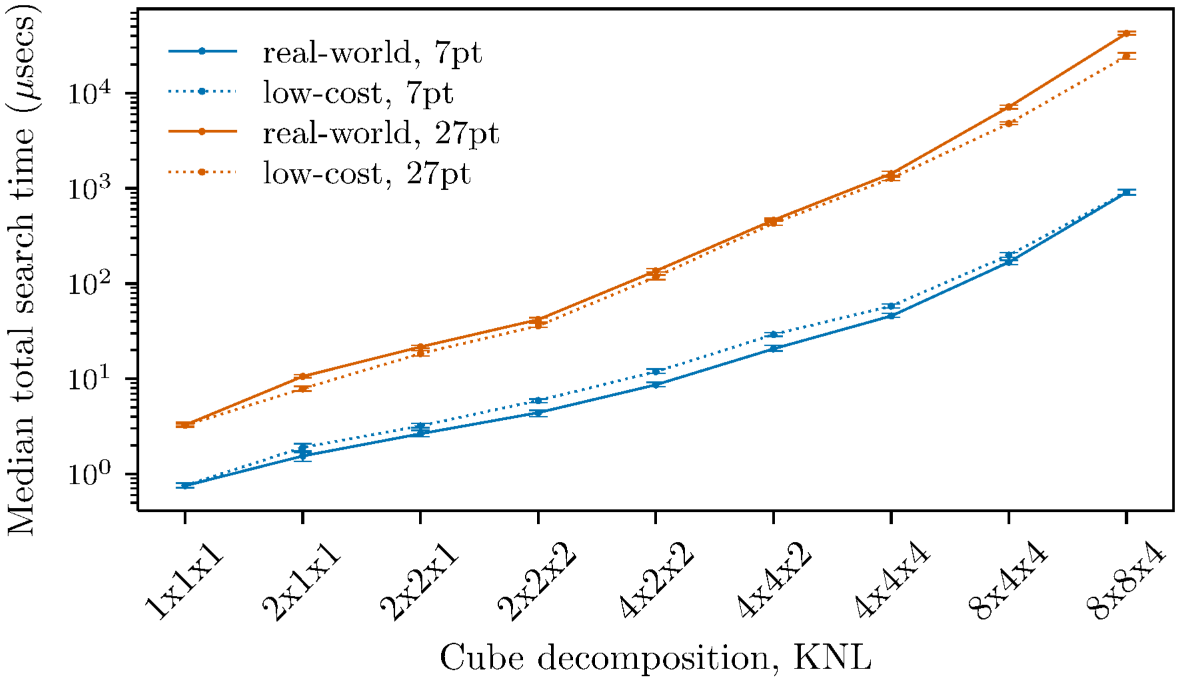
- Executed both benchmarks on two architectures:
  - Intel Xeon (Haswell)
  - Intel Xeon Phi (KNL)
  - Recorded number of items searched and time spent searching

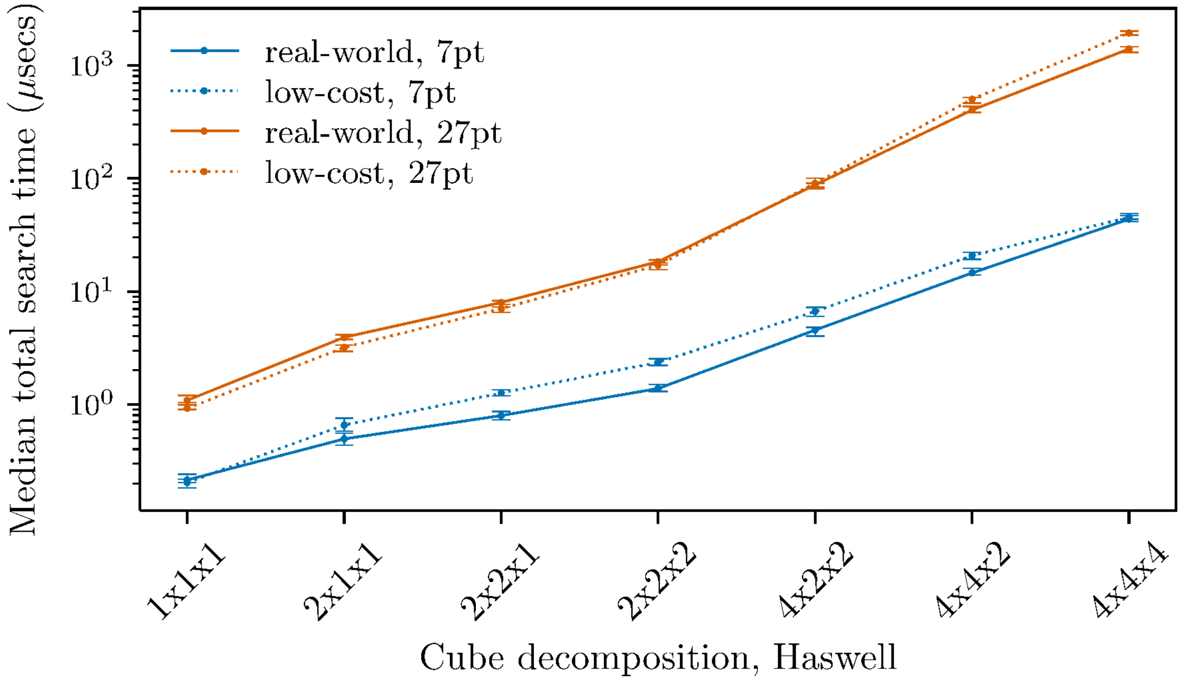
Multithreaded Message Matching: Items searched





Multithreaded Message Matching: Time searching

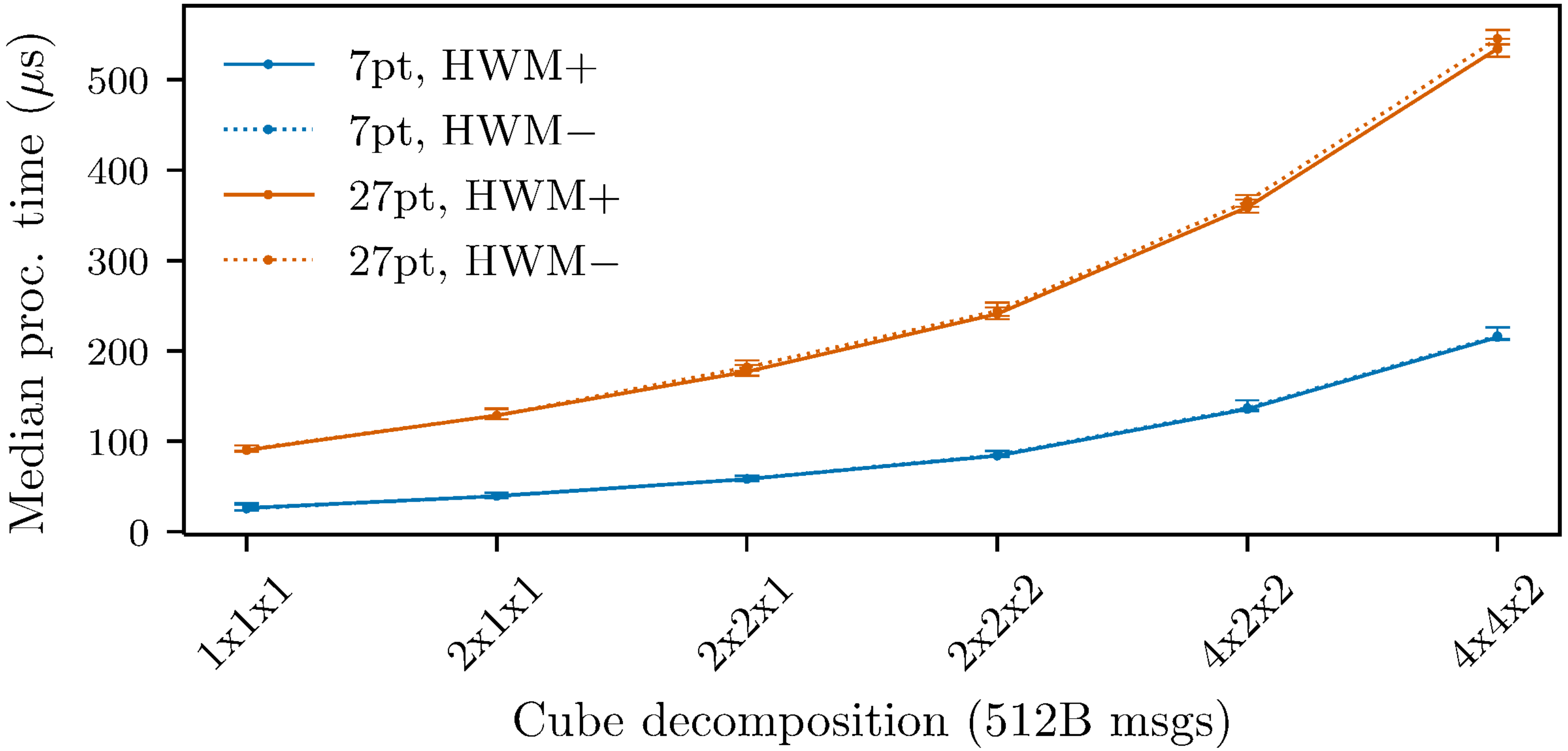




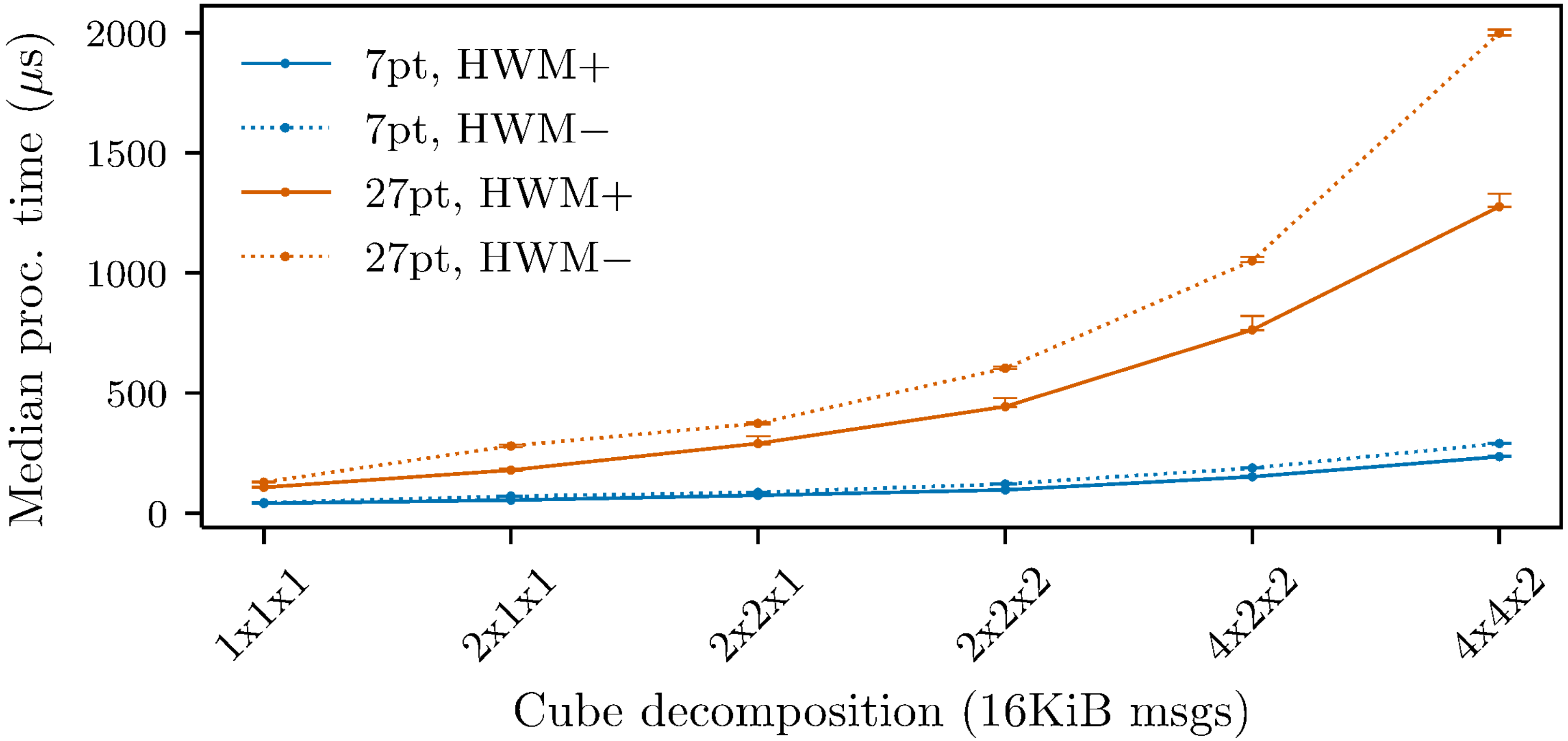
Multithreaded Message Matching: Application to SmartNIC

- Smart HPC NICs: offloaded MPI message matching
- Executed low-cost benchmark on system with this feature
  - Two conditions: off and on
  - On: offloading active for messages over 1024B threshold
- Instrumented benchmark to get time spent processing messages

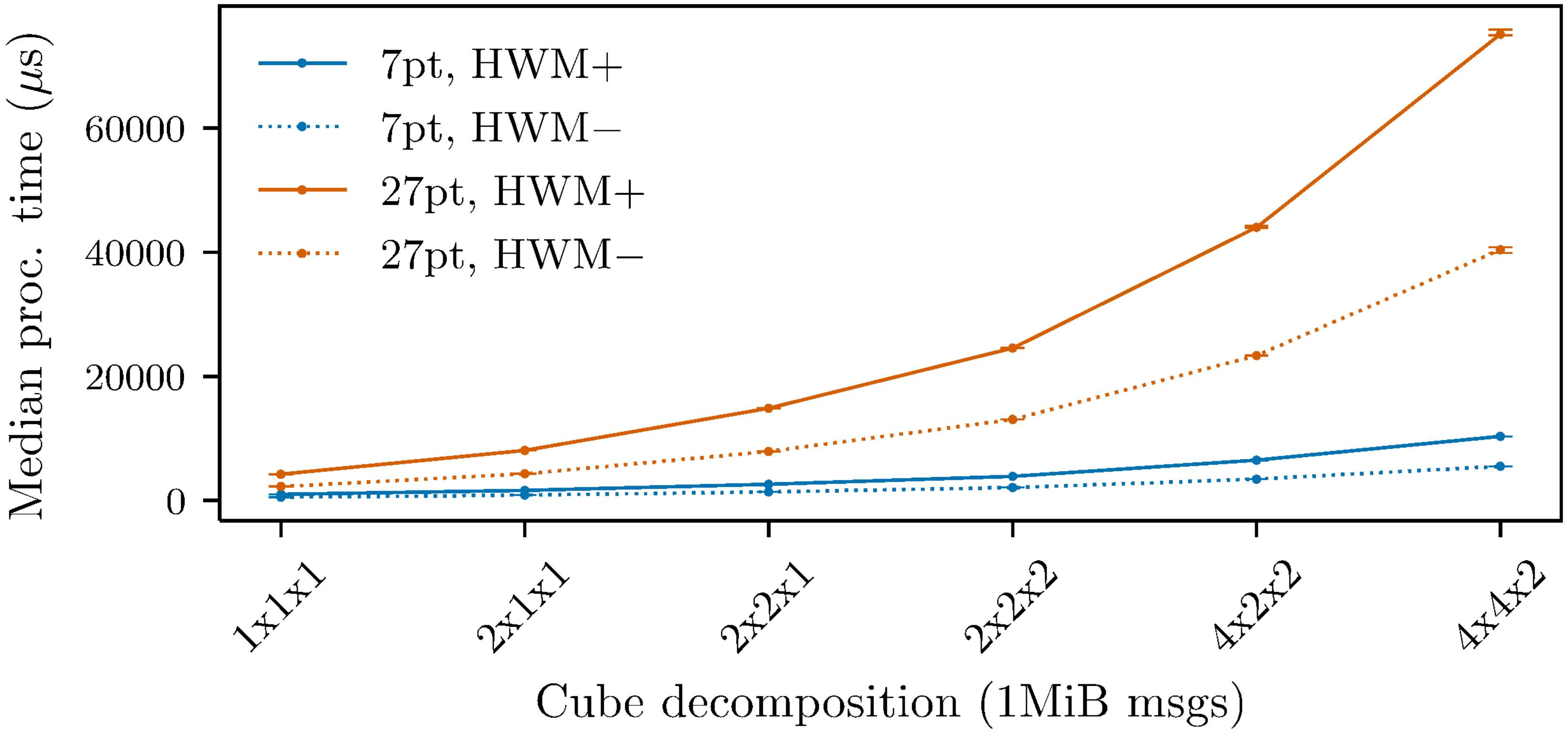
Multithreaded Message Matching: Application to SmartNIC



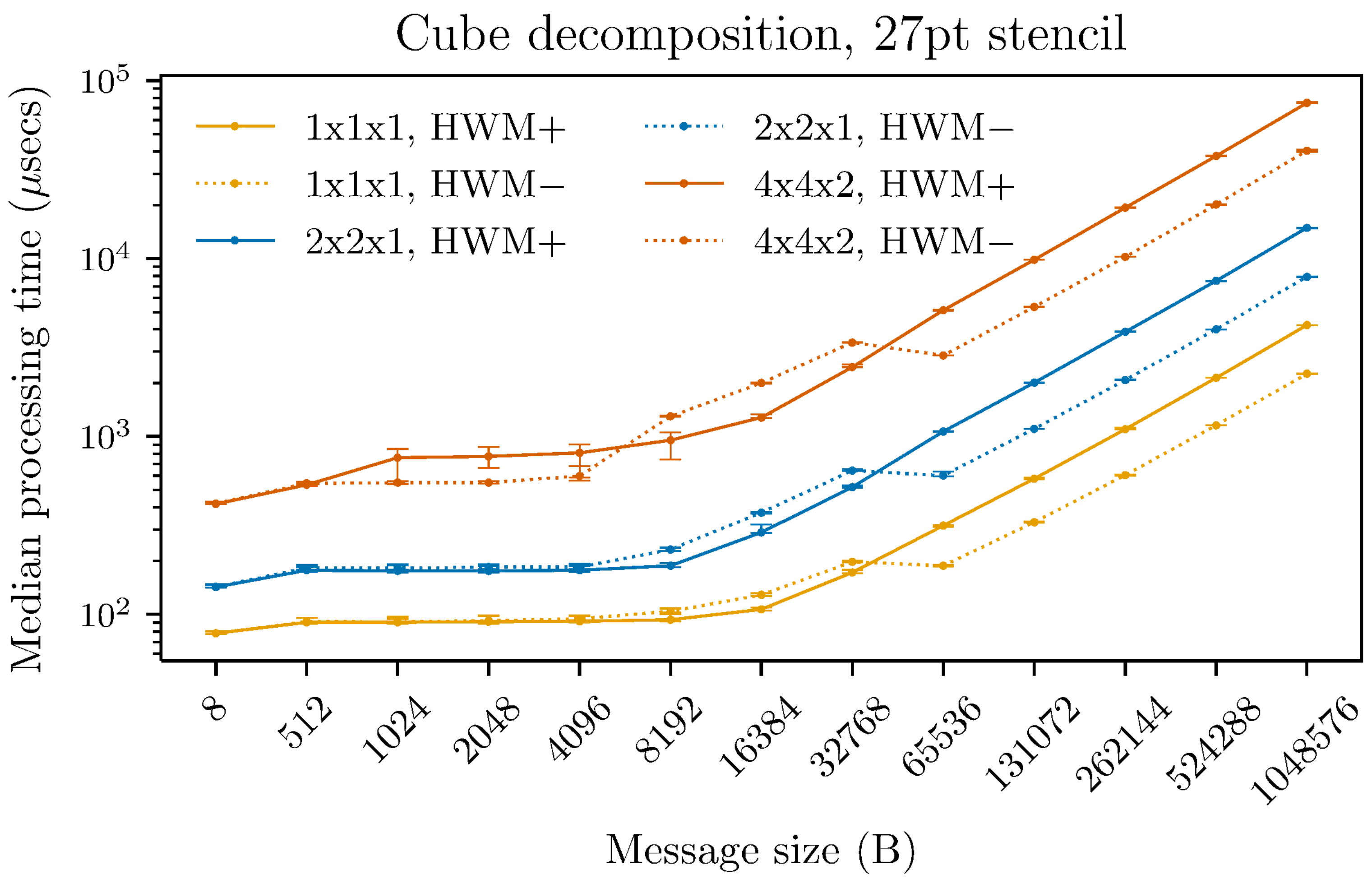
Multithreaded Message Matching: Application to SmartNIC



Multithreaded Message Matching: Application to SmartNIC



Multithreaded Message Matching: Application to SmartNIC



Multithreaded Message Matching: Conclusions

- Benchmarks to assess the overheads of multithreaded message matching
- Enabled assessment of a contemporary `smart’ offloaded application
- Revealed:
  - The offloaded capability does not fully mitigate overhead
  - And it may exacerbate the problem for larger message sizes

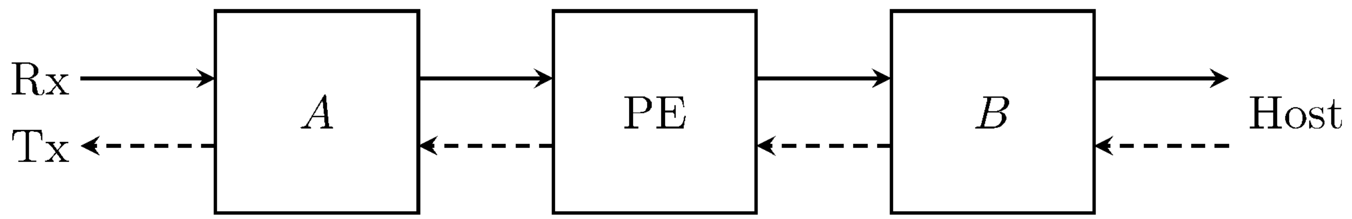
INCA: Recap

INCA

- W. Schonbein, R. E. Grant, M. G. F. Dosanjh, and D. Arnold, “INCA: in-network compute assistance,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Denver, Colorado, Nov. 2019, pp. 1–13

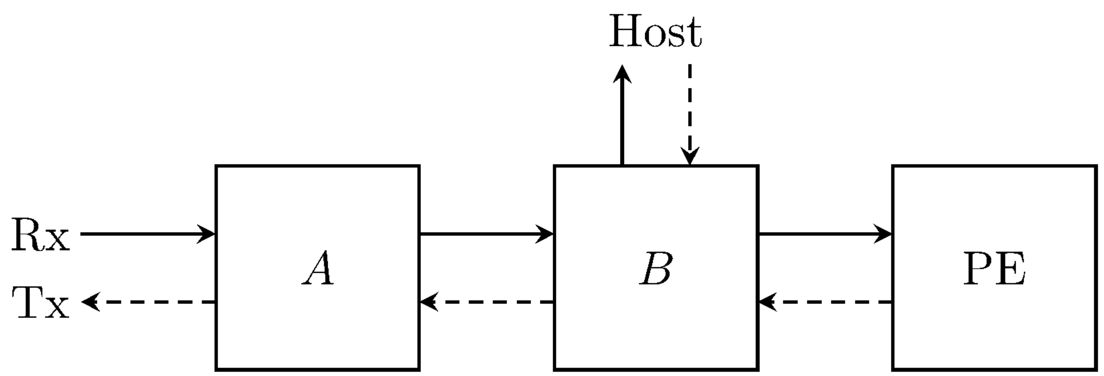
INCA: Recap: SmartNIC Architectures

On-Path



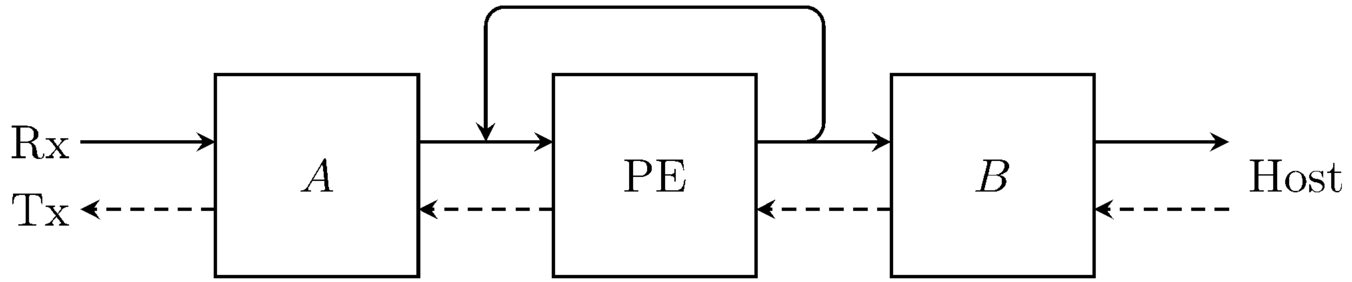
Deadlines

Off-Path



Deadline-free

In-Path

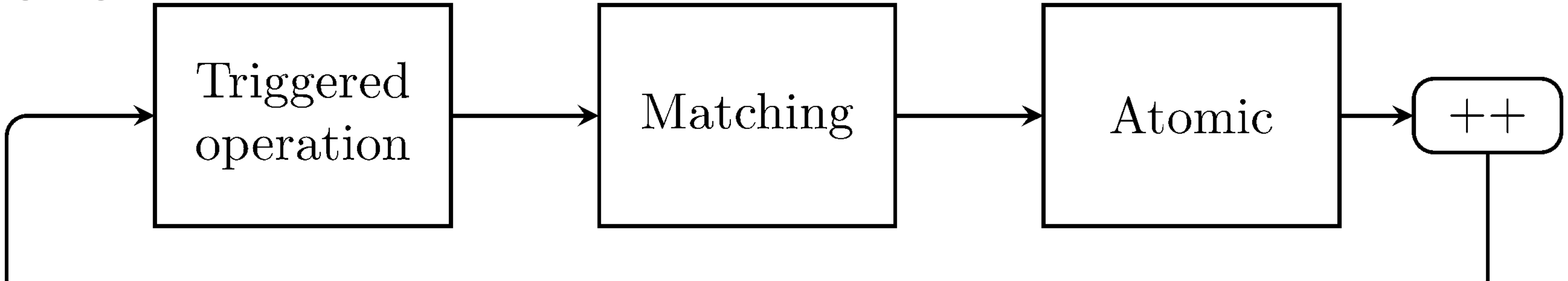


Deadline-free

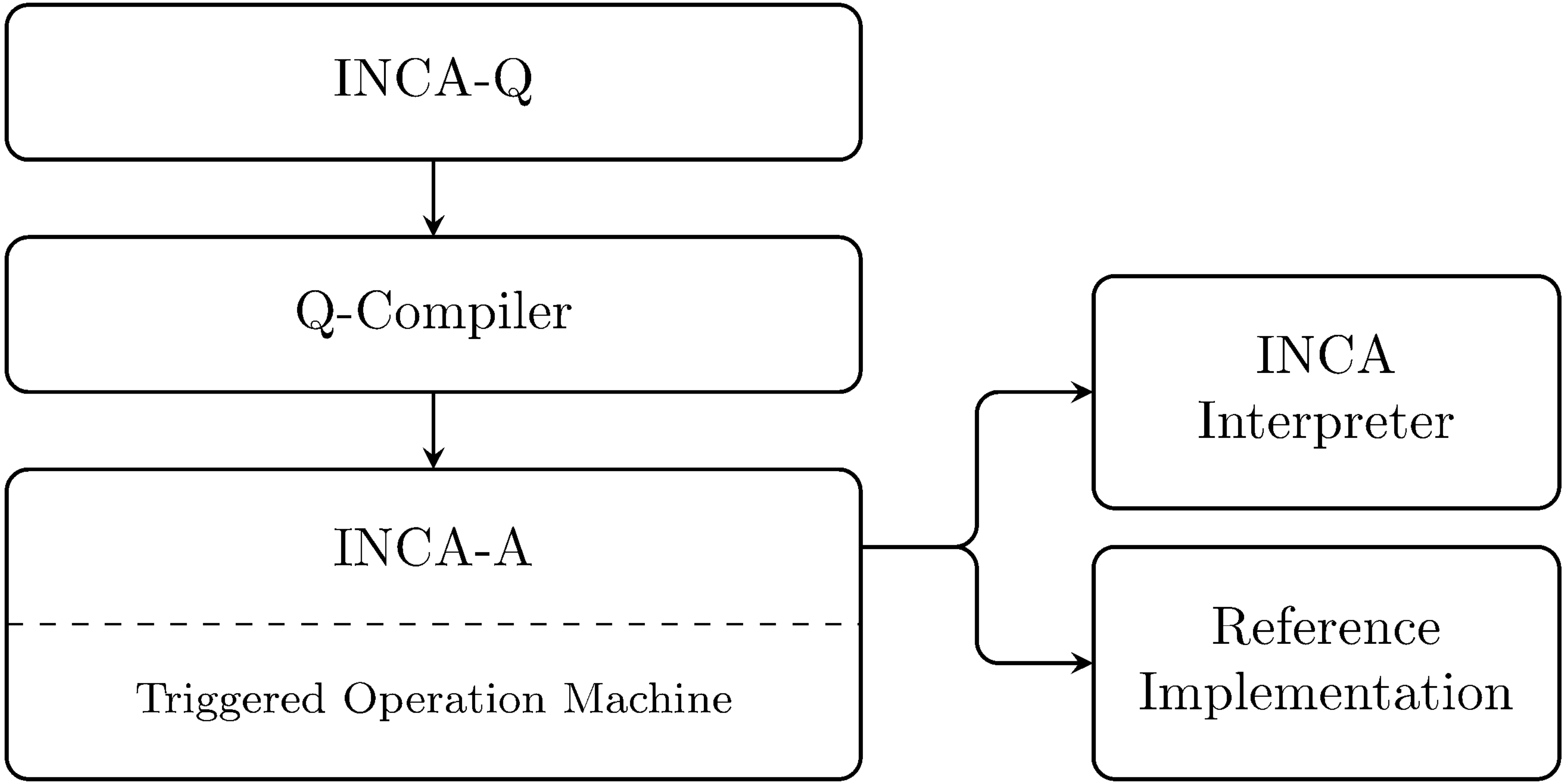
INCA: Recap: Program Execution

- HPC NICs often offload
  - Message matching
  - Atomic operations
  - Triggered operations

- These can be coordinated into a Turing complete model of computation



INCA: Recap: Ecosystem



INCA: Kernel Runtimes

- INCAsim: simulator for estimating INCA kernel runtimes
- With software and hardware optimizations (SIMD), in some cases kernel runtimes are comparable with contemporary CPUs
- Application speedups up to 30%

INCA: NIC-as-Coprocessor

INCA: NIC-as-Coprocessor

- Quiescent applications: applications with idle networks
- Opportunity:
  - Harvest idle NIC resources
- Ray tracing application for enabling over-the-horizon radar
- One hour cadence; idle 56 minutes/hour
- Resolution proportional to time available to process data

INCA: NIC-as-Coprocessor

- Bottlenecks:
  - **solve_normal_dist**
  - **convolve_normal**

- Two scenarios:
  - No hardware acceleration
  - Hardware acceleration

Network	Packet Size	Memory	Hardware
400 Gb/s	64 B	1 MiB, 1ns	exp: 100 ns conv: 80 usecs

INCA: NIC-as-Coprocessor

Kernel	Time in function (usecs)		
	Original	INCA scratchpad	INCA hardware
solve_normal_dist	78600	69496 (1.13x)	63981 (1.23x)
convolve_normal	28487	28470 (1.0006x)	23628 (1.21x)

INCA: NIC-as-Coprocessor

- Quiescent applications (idle networks)
- Harvest idle NIC resources to assist host applications
- INCA may speedup portions of host applications to free resources

INCA: Enabling Adaptive Networks

INCA: Enabling Adaptive Networks

- Variations in network workloads can impact overall quality of service
  - Elephant vs. mouse flows
  - Ingress problem
  - Network-induced memory contention

- Emerging research suggests techniques from ML have potential
  - Avoid network-induced memory contention (Groves et al. 2018)
  - Energy consumption (Dickov et al. 2014)
  - Dynamic network tuning (Kiran & Chhabra, 2019)

INCA: Enabling Adaptive Networks

- SmartNICs are in a good position help
  - Situated between the network and the host
  - If the NIC can predict incoming traffic (e.g., expected data over time):
    - Intelligently schedule DMA transfers
    - Pre-emptively adjust credits to senders
    - Adjust power usage
    - Schedule processors (SPiN) to adjust allocated time

INCA: Enabling Adaptive Networks

- Is it feasible to offload ML kernels to an INCA-enabled SmartNIC?
- Do these kernels generate accurate predictions?
- How well do these kernels address the issues raised above?

Yes

Constraints on INCA ML Kernels

- INCA is Turing-complete
- INCA is deadline-free
- Constraint: lower memory requirements are better
- Constraint: faster execution is better

Applications and Data Collection

Applications
HPCG
LAMMPS-lj
LAMMPS-rhodo
Lulesh
MILC
MiniAMR
MiniFE
MiniMD

Applications and Data Collection

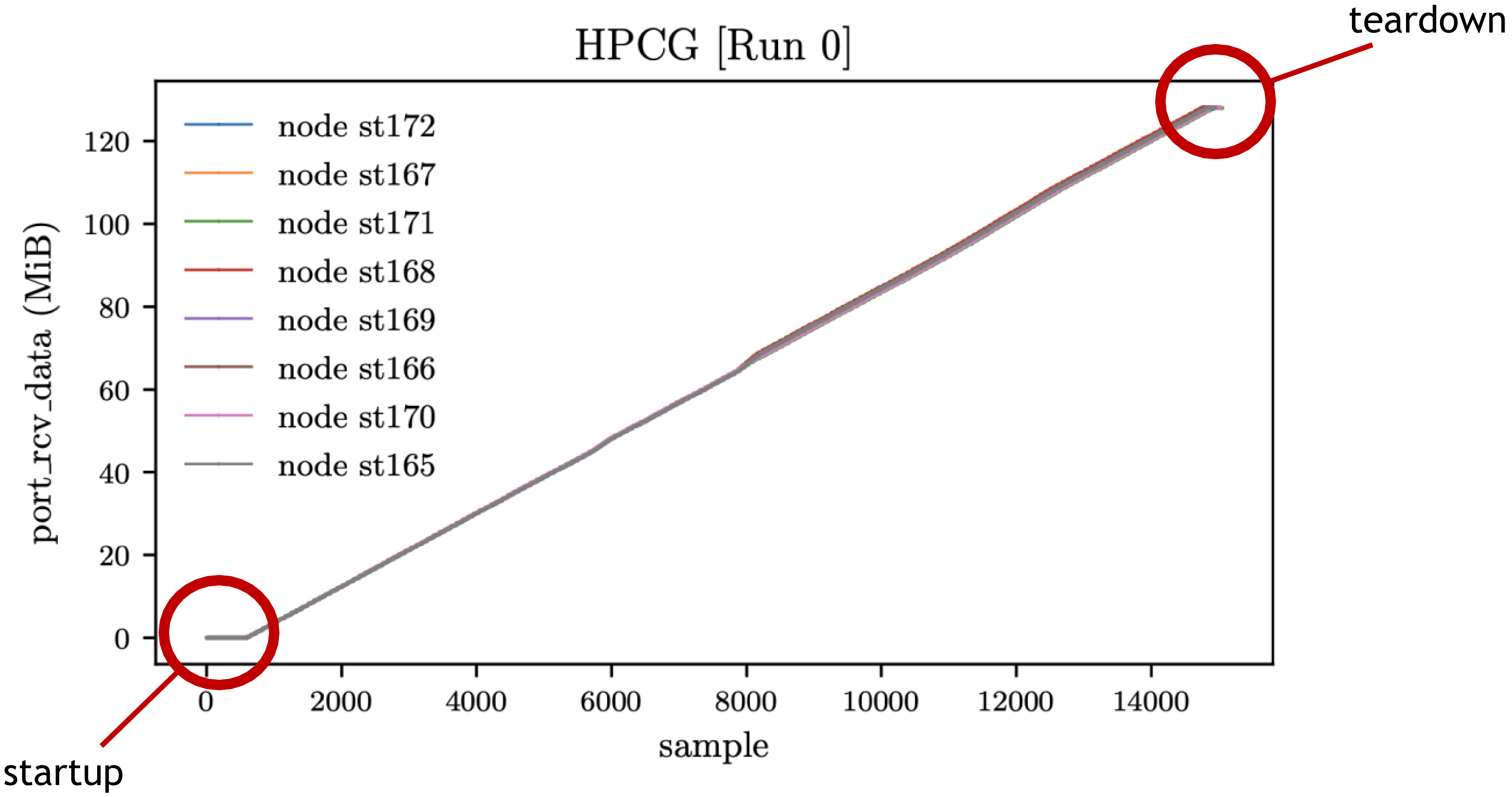
- System:
  - ARM
  - 2 sockets/node
  - Each socket has a 28-core Cavium ThunderX2 ARM CPU @ 2 GHz
  - NVIDIA Mellanox ConnectX-5 NICs

- NIC hardware performance counters
  - Bytes received

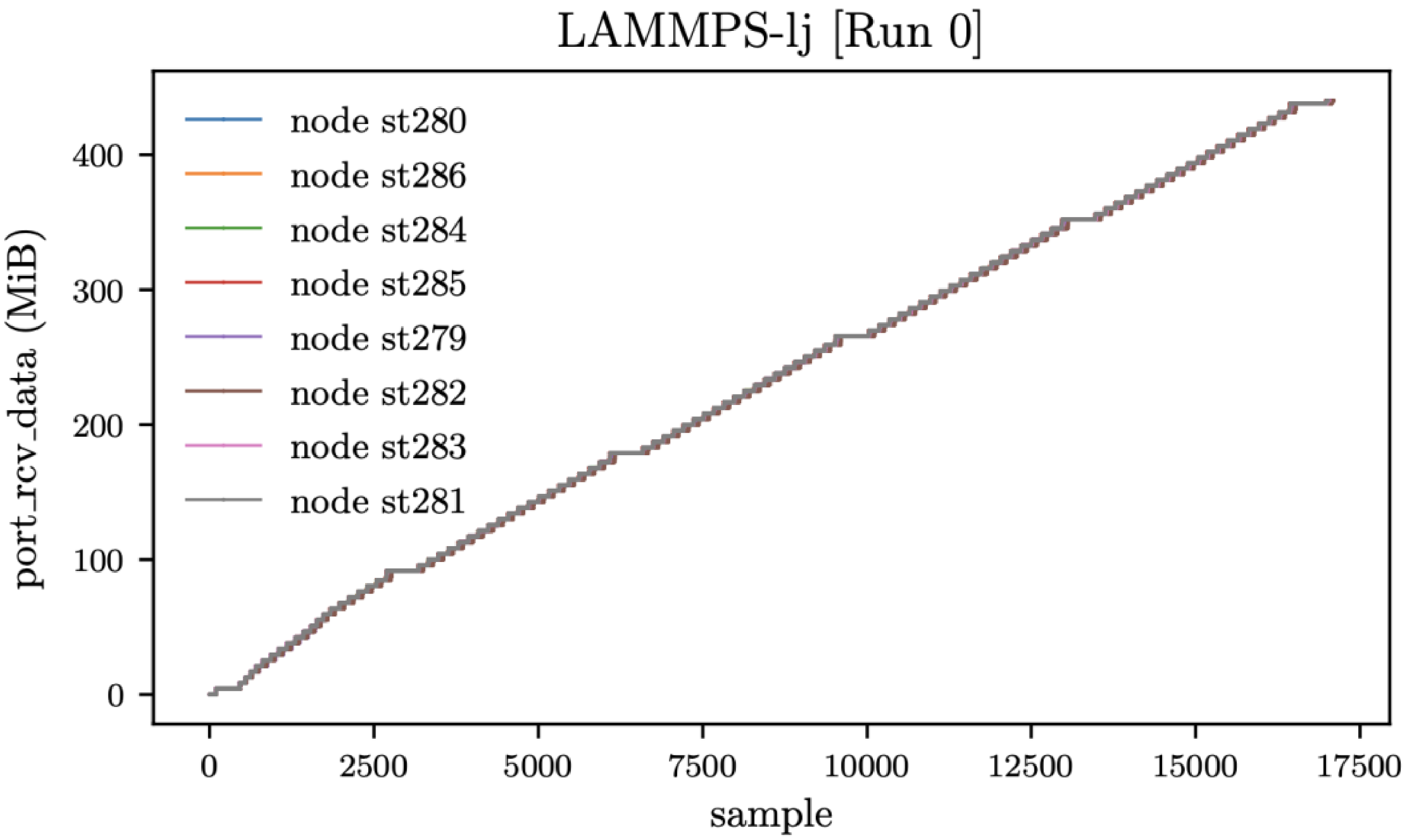
Applications and Data Collection

- Performance counter data collection
  - Filesystem interface
  - Sampling rate 20 Hz (period = 50 ms)
  - Tool is pinned to one socket, application MPI process pinned other socket.
  - Each application executed 11 times, 8 MPI processes, 1 process per node

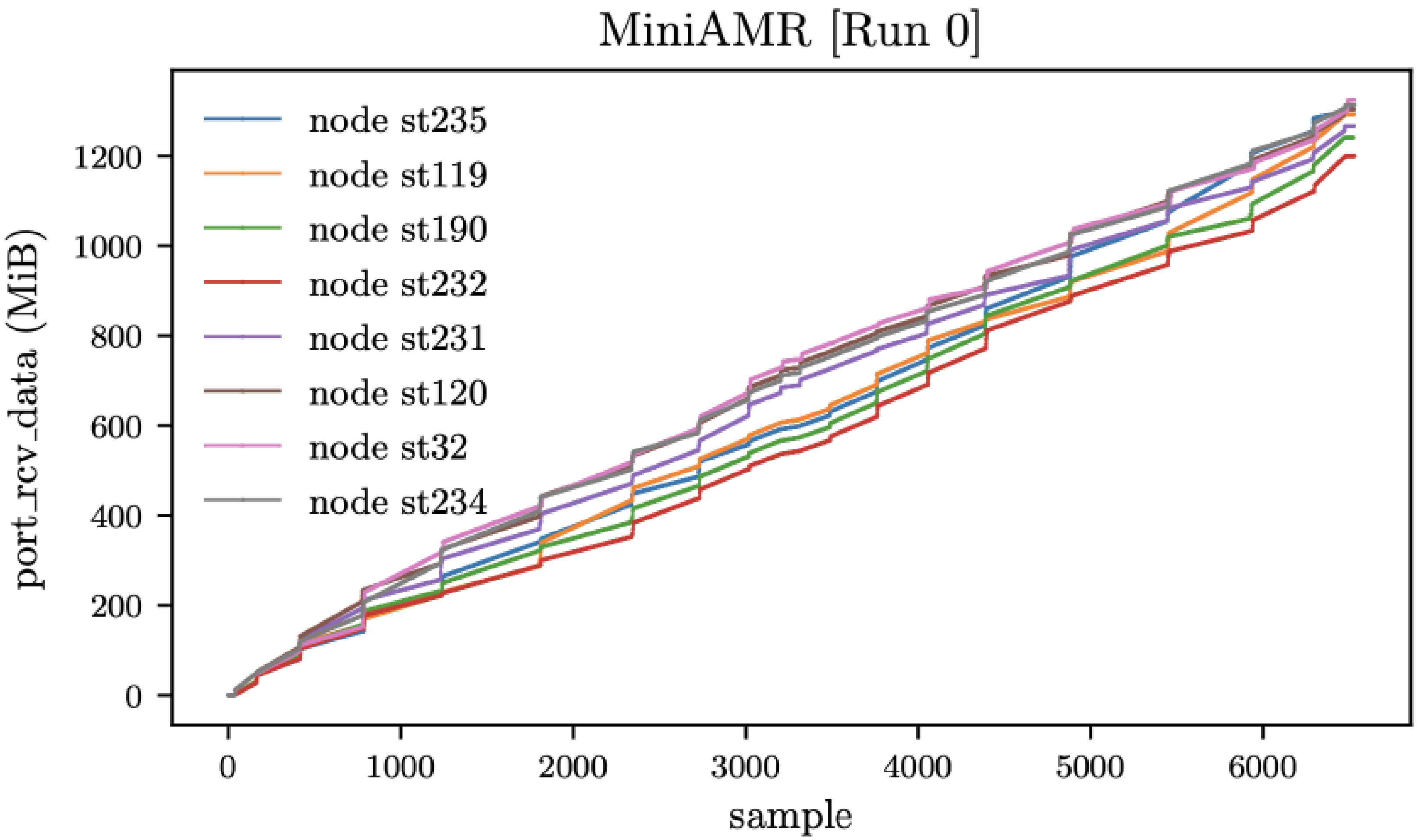
Data and ML Methods



Data and ML Methods



Data and ML Methods



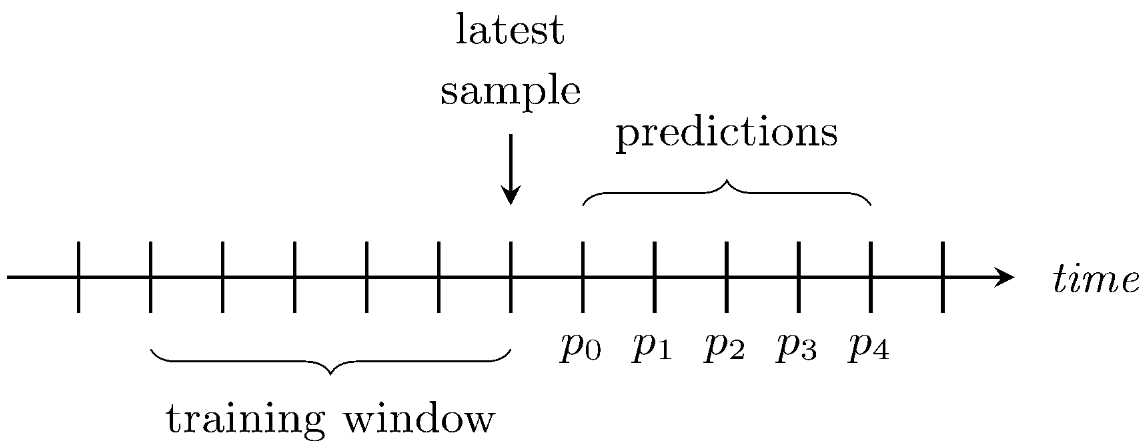
Data and ML Methods

- Given the general shape of the data
- Constraints on INCA kernels
- We selected linear regression as a method

Data and ML Methods

- Ordinary LR
  - Train model on known data set, and apply to subsequent runs of the same application
  - *Static*: model does not dynamically adapt to incoming data

- Rolling LR



Data and ML Methods

- Ordinary LR
  - Train model on known data set, and apply to subsequent runs of the same application
  - *Static*: model does not dynamically adapt to incoming data

- Rolling LR
  - As each data point arrives, train model on the points in the training window, and generate predicted values
  - *Dynamic*: model adapts to incoming data

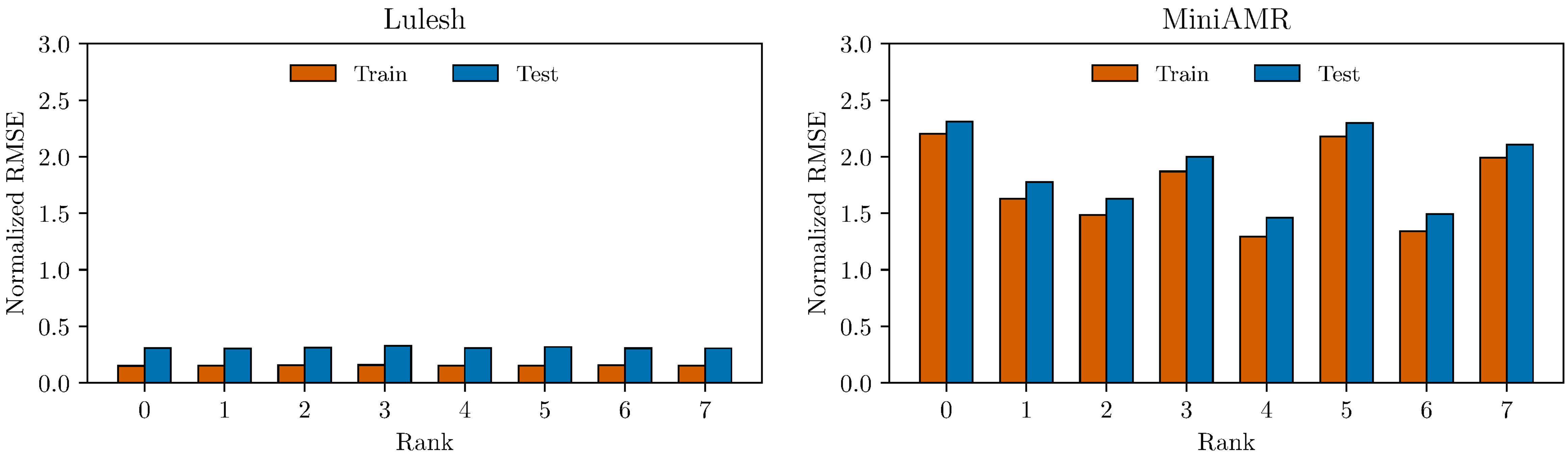
Data and ML Methods

- Rolling LR
  - Ordinary
  - Weighted
    - Exponential increasing (exp-inc)
    - Exponential decreasing (exp-dec)

Results: Ordinary LR

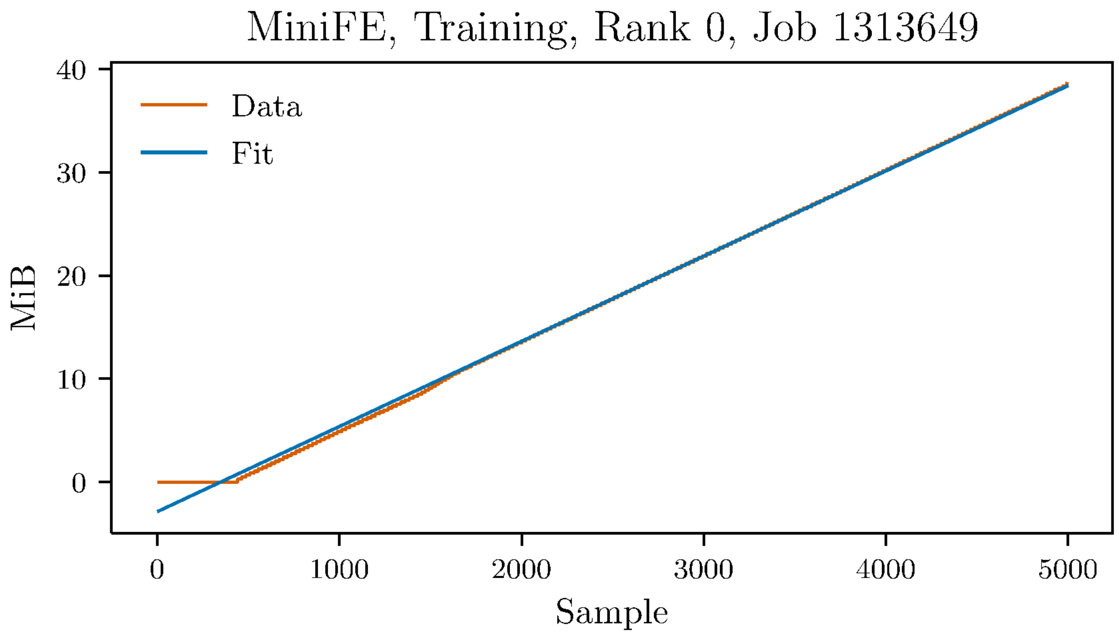
- For each run, train and then test against the remaining 10 runs
  - Report normalized RMSE (NRMSE)
  - For training, averaged across 11 runs
  - For testing, averaged across 110 runs
  - Error bars are standard deviations

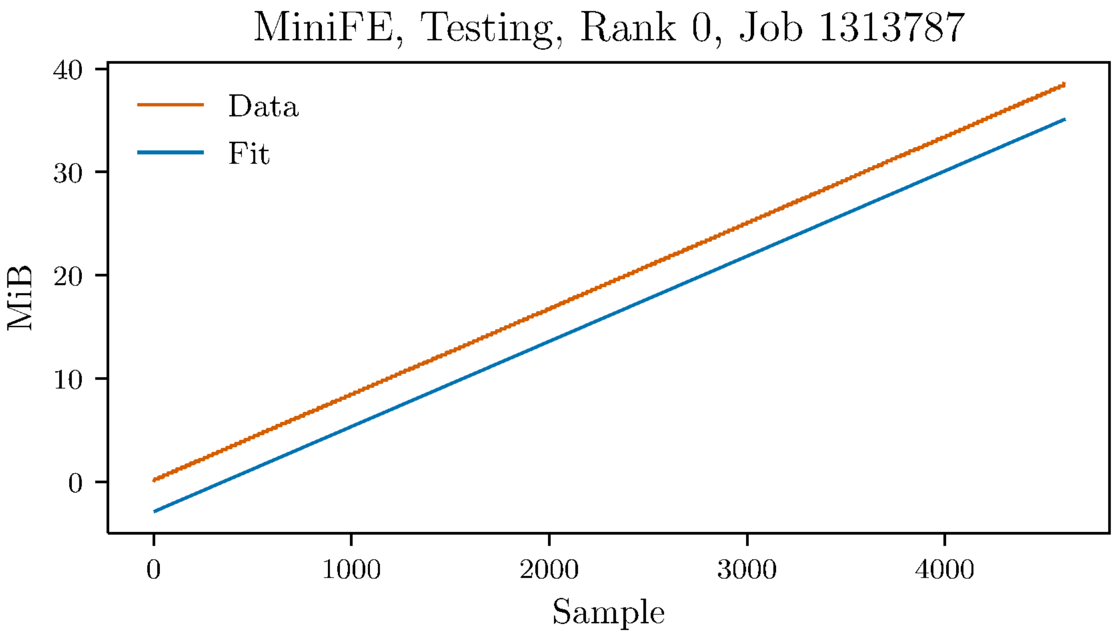
Results: Ordinary LR



Best (Lulesh) and worst (MiniAMR) performing ordinary LR applications

Results: Ordinary LR





Overfitting due to startup phase

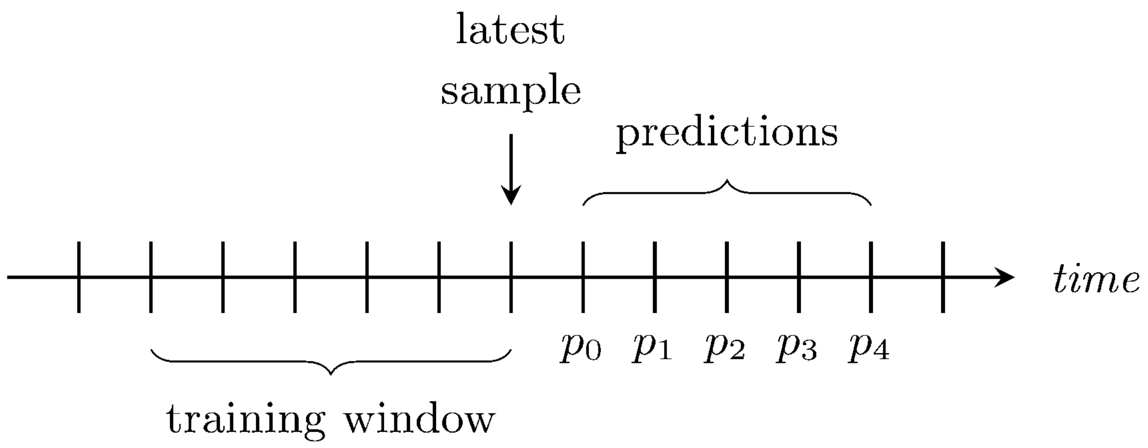
Results: Ordinary LR

```
PUTL output, b0, f
PUTL _R1, b1, f
MUL _R1, _R1, x, f
ADD output, output, _R1, f
END
```

INCA-A ordinary LR inference kernel

Memory: < 64 B (64b operands)
Runtime @ 200 Gb/s: 26.48 ns
Runtime @ 400 Gb/s: 16.24 ns
(64 B packets, scratchpad)

Results: Rolling LR



- How large should the training window be?
- Same for all prediction points?

Results: Rolling LR

Application	Weights	Prediction Point									
		p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
HPCG	ord	5	6	6	13	151	150	149	148	148	147
	exp-inc	8	12	14	17	19	247	250	250	250	250
	exp-dec	2	2	5	117	148	147	146	146	145	144
LAMMPS-lj	ord	6	5	5	5	5	6	6	6	6	242
	exp-inc	8	9	8	8	9	10	13	18	26	34
	exp-dec	2	5	4	5	5	6	6	6	7	7
MILC	ord	2	4	5	5	5	5	5	5	5	5
	exp-inc	5	7	7	8	8	7	7	7	7	7
	exp-dec	2	2	2	2	3	3	4	4	4	4

Median best performing window size by prediction point across all ranks and all runs

Data and ML Methods

Application	Pr(data)	Pr(~data)
HPCG	0.15	0.85
LAMMPS-lj	0.03	0.97
LAMMPS-rhodo	0.09	0.91
Lulesh	0.93	0.07
MILC	1.00	0.00
MiniAMR	0.79	0.21
MiniFE	0.14	0.86
MiniMD	0.06	0.94

Probability of data arriving at any given sample (i.e., probability of positive slope) averaged across all ranks of all runs

Results: Rolling LR

Application	Weights	Prediction Point									
		p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
HPCG	ord	5	6	6	13	151	150	149	148	148	147
	exp-inc	8	12	14	17	19	247	250	250	250	250
	exp-dec	2	2	5	117	148	147	146	146	145	144
LAMMPS-lj	ord	6	5	5	5	5	6	6	6	6	242
	exp-inc	8	9	8	8	9	10	13	18	26	34
	exp-dec	2	5	4	5	5	6	6	6	7	7
MILC	ord	2	4	5	5	5	5	5	5	5	5
	exp-inc	5	7	7	8	8	7	7	7	7	7
	exp-dec	2	2	2	2	3	3	4	4	4	4

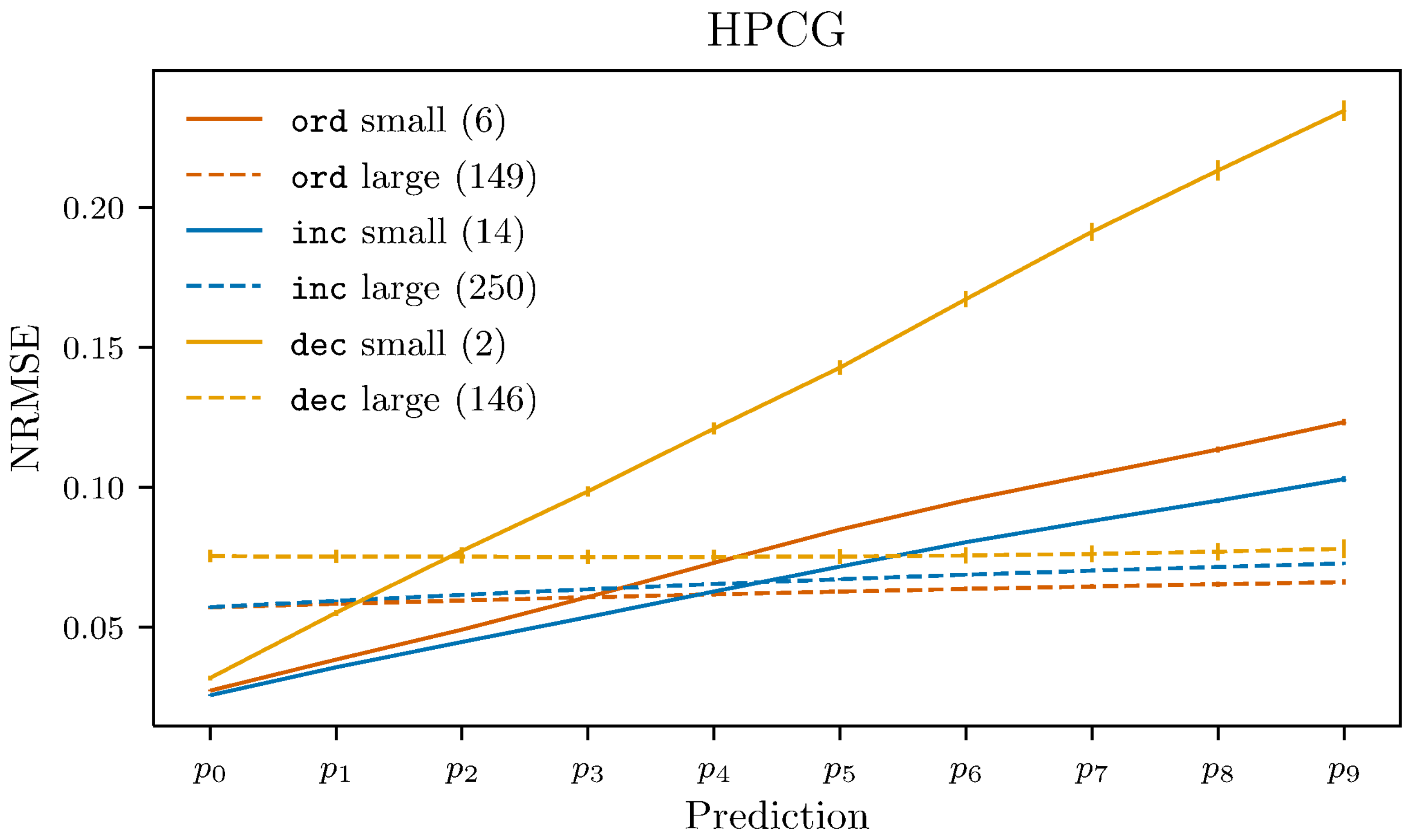
Median best performing window size by prediction point across all ranks and all runs

Results: Rolling LR

- Two-window strategy:
  - Small window for near-term predictions
  - Large window for far-term predictions

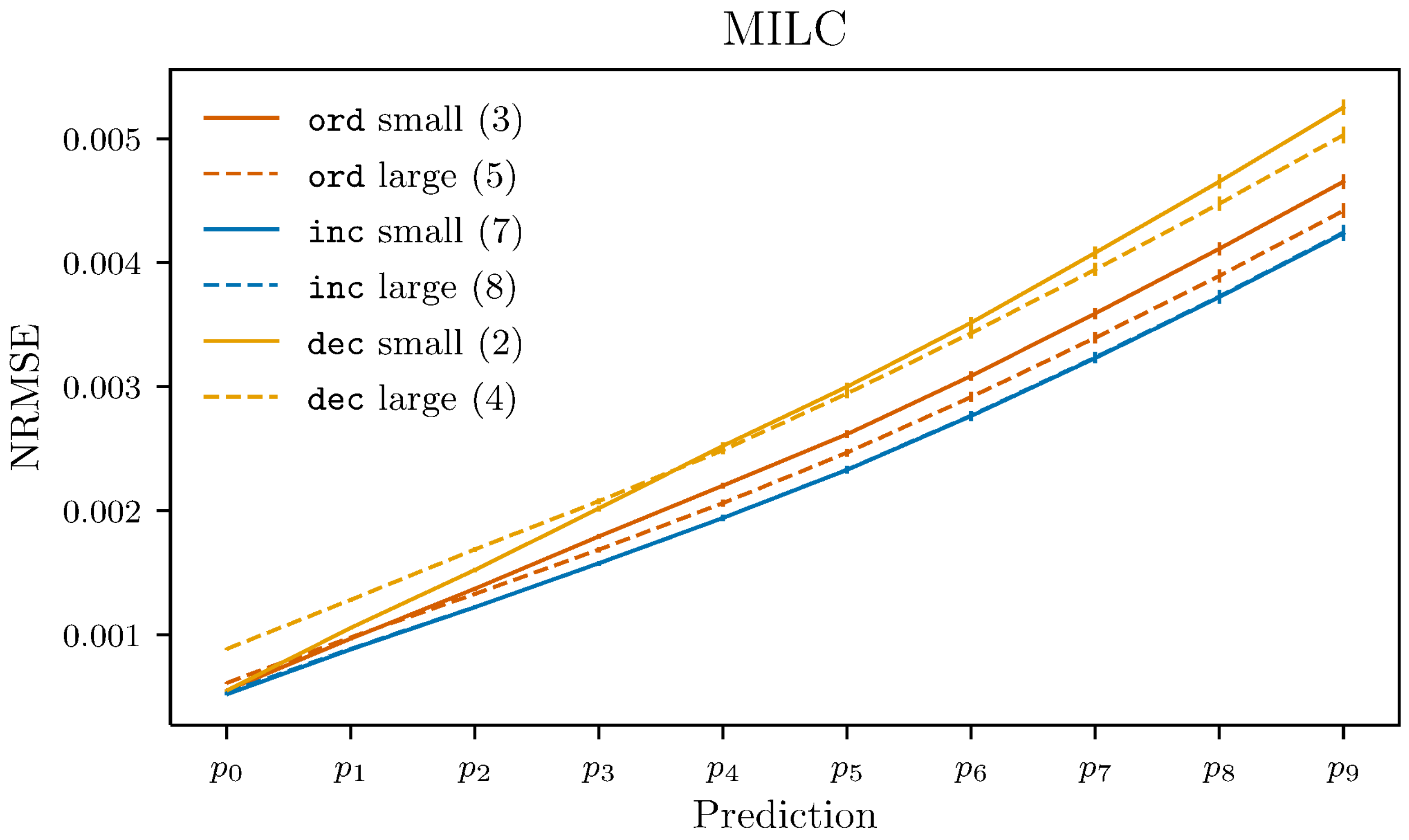
- Bifurcate each set of window sizes, take median of each

Results: Rolling LR



Average NMRSE by prediction point for HPCG, for small and large training window sizes

Results: Rolling LR



Average NMRSE by prediction point for MILC, for small and large training window sizes

Results: Rolling LR

- Rolling methods outperform static ordinary LR in all cases
  - HPCG: Static approaches 1.0%, while best performing rolling methods < 0.1%
  - LAMMPS-lj: Static exceed 1.0%, best performing rolling < 0.3%

Results: Rolling LR

- INCA kernel is more complex
- INCA kernels admit of SIMD parallelization
- Simulator parameters:
  - 200 Gb/s and 400 Gb/s network speeds (64B messages)
  - Small window (5 samples) and large window (250 samples)
  - Vanilla and optimized

Results: Rolling LR

Rolling LR Method	Network Gb/s	Runtime (usecs)		Memory (KiB)	Instructions executed
		5 sample window	250 sample window		
ord	200	0.68	21.87	< 4	101
	400	0.41	13.46		3286
ord-parallel	200	0.28	0.69	< 4	42
	400	0.17	0.34		42
weighted	200	1.17	38.10	< 8	179
	400	0.71	23.22		58124
weighted-parallel	200	0.74	15.72	< 12	112
	400	0.45	9.65		2317

Estimated runtimes and memory requirements for rolling LR methods

INCA: Enabling Adaptive Networks

- Simple ML techniques can generate accurate results
  - NRMSE ranges from 2.5% to 0.1%

- Offloading these ML kernels in INCA is *feasible*
  - Runtimes (200 Gb/s) range from 26.48 ns to 38.10 usecs
  - Memory never exceeds 12 KiB

Conclusion & Future Work

Conclusion

- Contributions of this work
  - Benchmarks for assessing overheads of multithreaded communication
  - In-Network Compute Assistance (INCA)
  - Assessment of application speedups afforded by INCA
  - Demonstration that INCA can enable `adaptive' networks

Future Work

- FPGA prototype
- Portals 5.0
- Follow-up on quiescent network applications
- Distributed applications
- INCA as a mechanism for coordinating heterogeneous on-NIC compute resources
- Other ML techniques

Acknowledgements

Dorian Arnold
Ryan Grant
Trilce Estrada
Jinho Choi

Matthew Dosanjh
Noah Evans
Scott Levy
David DeBonis
Jeremy Benson