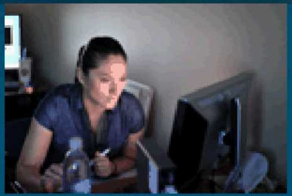
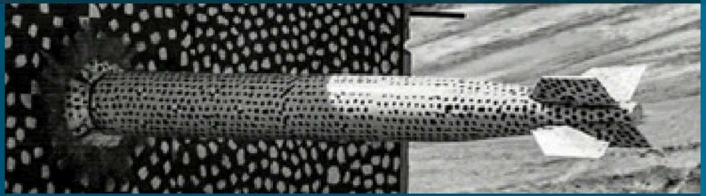
SAND2019-10160C

MPI Tag Matching Performance on ConnectX and ARM





PRESENTED BY

W. Pepper Marts

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.

Co-Authors

Matthew G. F. Dosanjh

- Center for Computational Research, Sandia National Laboratories

Whit Schonbein

- Center for Computational Research, Sandia National Laboratories
- Department of Computer Science, University of New Mexico

Ryan E. Grant

- Center for Computational Research, Sandia National Laboratories
- Department of Computer Science, University of New Mexico

Patrick G. Bridges

- Department of Computer Science, University of New Mexico

Introduction

With exascale, message matching could become a major factor of HPC application performance

Network vendors have made significant improvements to MPI tag matching performance

- Leveraging both software and hardware

The performance characteristics of these approaches are often not well-studied

- More marketing material than published information

In this paper, we quantify the impact of a new vendor matching scheme

Background

Point-to-point send-receive is an important component of many applications

MPI Message Matching will potentially be a problem for exascale applications

- Exascale developers expect to use MPI in a multithreaded manner
  - Bernholdt, David E., et al. "A survey of MPI usage in the US exascale computing project." *Concurrency and Computation: Practice and Experience* (2017): e4851.
- Multithreaded MPI creates problematic matching behavior
  - Schonbein, Whit, et al. "Measuring multithreaded message matching misery." *European Conference on Parallel Processing*. Springer, Cham, 2018.

Background

Traditional MPI implementations have used a pair of linked lists

- Posted Receive Queue (PRQ)
- Unexpected Message Queue (UMQ)

Recent optimizations have used a hashed-binning solution

- Intel's PSM2 driver
- Mellanox's ConnectX-5 driver

Methodology – Experimental Platform (Hardware)

Astra Research Cluster

- Worlds first petascale ARM cluster
- 36 compute racks, each containing 18 HPE Apollo 70 chassis
- 4 compute nodes per chassis
- Two 28-core Cavium ThunderX2 CN9775 chips operating at 2GHz per node
- 4x EDR (100Gb/s) Infiniband network with Mellanox ConnectX-5
- Three-level fat tree topology

Methodology – Experimental Platform (Matching)

Matches based on a single 64-bit tag that contains a full set of matching data including
- MPI tag
- MPI rank
- MPI context ID

There is both a hardware and a software matching layer
- Hardware matching is only used for larger messages
- Hardware matching is effective when the time it takes to match is less than the time spent moving data

Software layer is open source as part of Open UCX
- Hash binning system with 1021 bins
- Hash function that XOR's the upper and lower 32-bits of the UCX tag modulo 1021

Hardware Flags and Thresholds
- UCX_RC_MLX5_TM_ENABLE and UCX_DC_MLX5_TM_ENABLE
- UCX_TM_THRESH (default threshold is 1KiB)

Methodology – Basic Microbenchmarks

Microbenchmark results were generated using a modified version of the OSU benchmark suite

Modifications were made to allow better analysis of the effects of the message matching queue depth and ConnectX-5's tag binning system

Changes to help evaluate the effect of the matching queue:

- Clear cache between iterations
- Prepost all receives, including a configurable number of unmatched receives
  - This allows us to experiment with the amount of time spent in the matching engine
  - We will refer to the number of these extra unmatched receives as the queue depth

Methodology – Microbenchmark Extensions

Changes to help test the specifics of the ConnectX-5 and UCX's binning system:

- We added a configurable binning collision rate (0%, 1%, 10%, 100%)
- We know that the bin assigned to each receive is chosen based on its tag modulo 1021
- When we create the receives for the configurable message queue, we choose tags such that there is a certain percent chance of it being placed in the same bin as the matched receives
- Wildcard Tests (Transient or Permanent)

We also created a modified version to test how the hardware matching layer handles wildcards

- "Transient" test where an MPI_ANY_TAG receive is posted and matched before any timing is done for the benchmark
- "Permanent" test where an MPI_ANY_TAG receive is posted and matched after all timing is done for the benchmark

Methodology – Microbenchmark Setup

Configuration Parameters for Modified OSU Benchmarks

- Collision Rates : 0%, 1%, 10%, and 100%
- Message Size : 1B to 1MiB by powers of two
- Queue Depth : 1 to 32k by powers of two
- Hardware Matching : enabled or disabled
- Separate runs for each wild card

Data points presented are the mean and standard deviation of twenty runs

Methodology – Applications and Proxies

We also ran applications to evaluate the effects of the hardware matching on a more realistic, optimized communication pattern

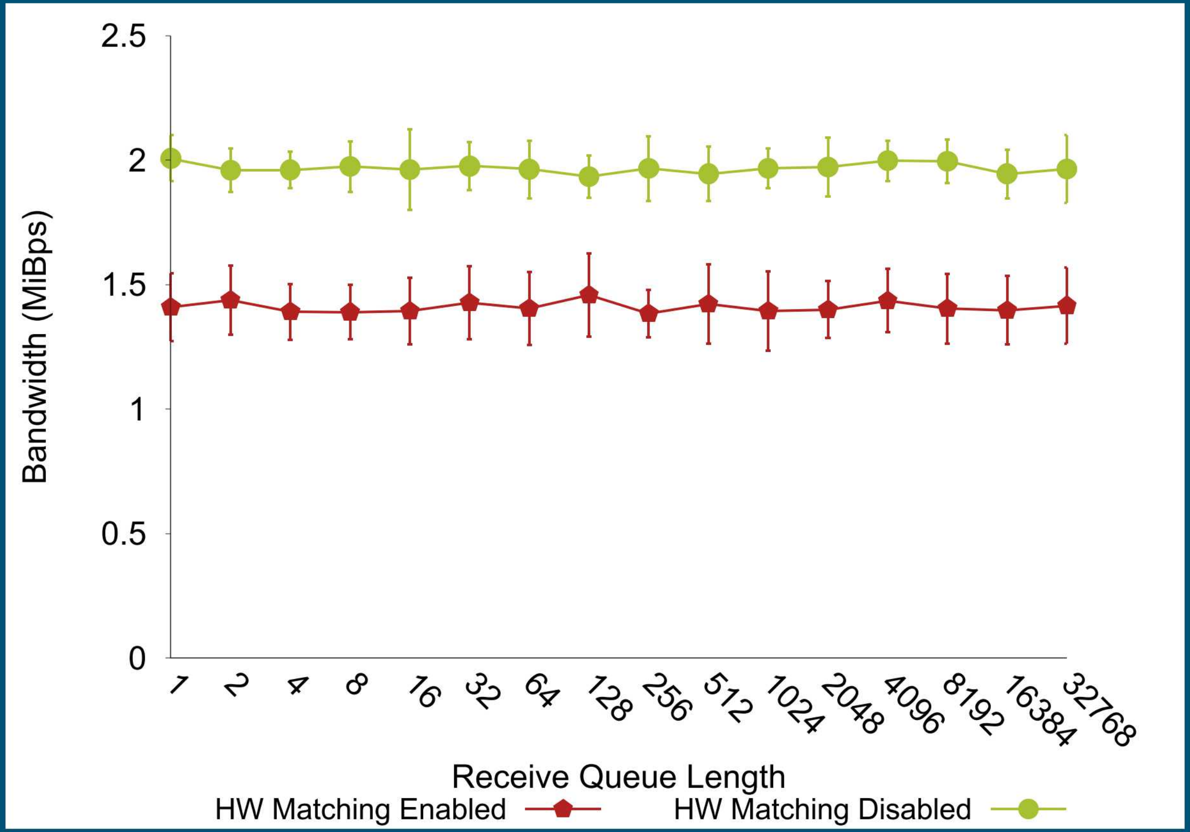
LULESH is a hydrodynamics proxy application from Lawrence Livermore

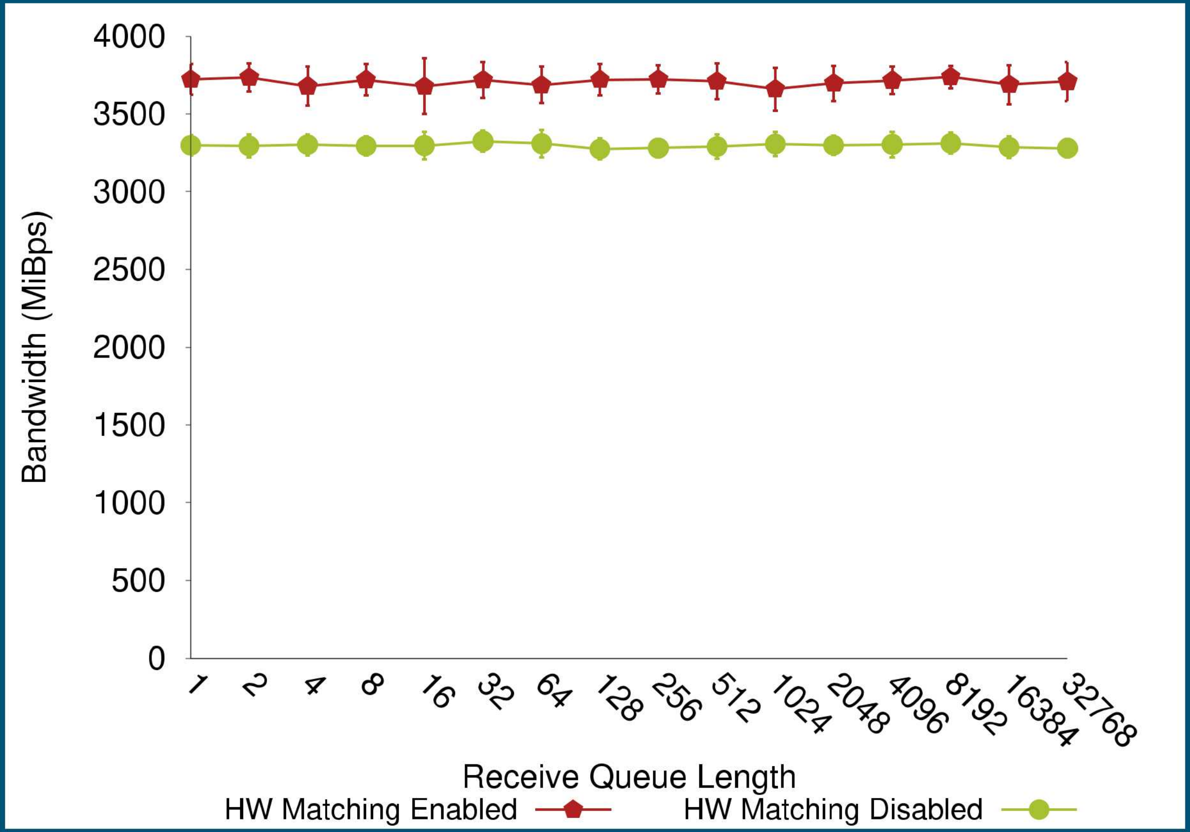
FDS is a fire dynamics simulator from NIST

These were run with three different matching techniques
- Software only
- Software with hardware
- Software with one bucket (linked list)

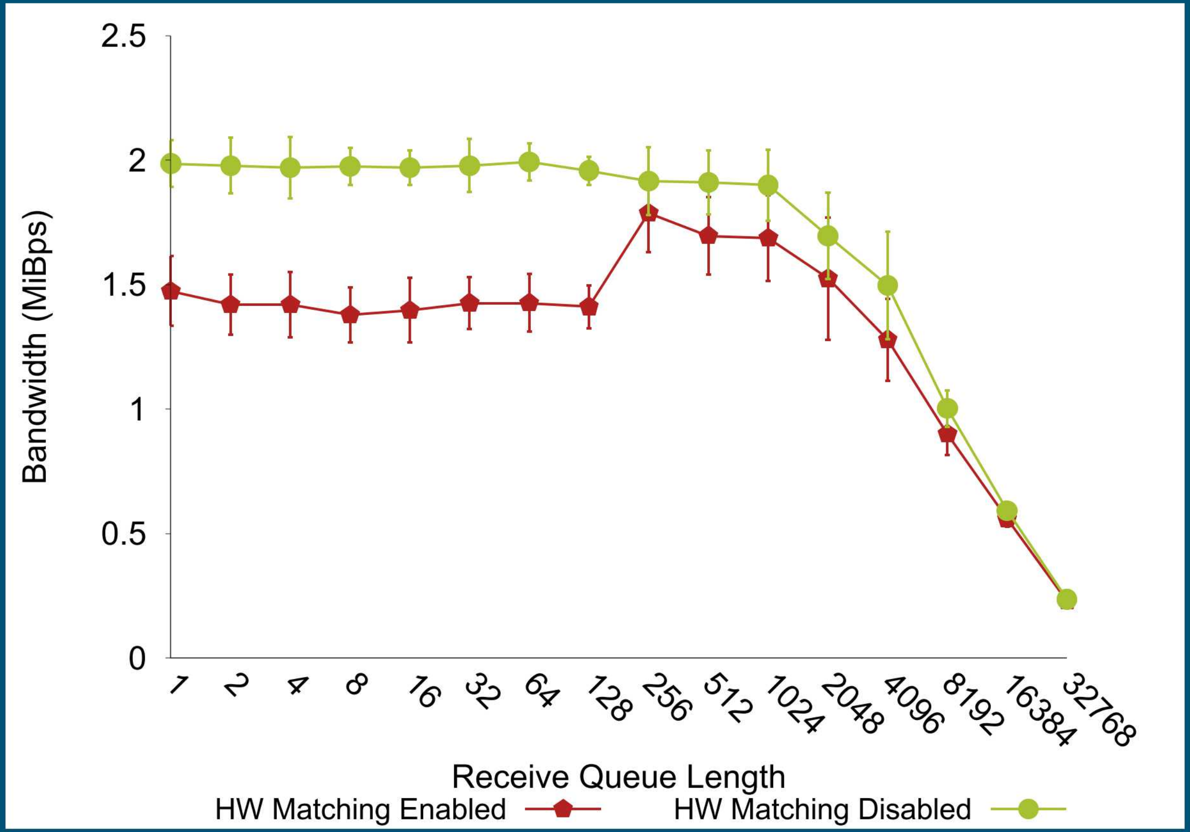
Data points presented are the mean and standard deviation of all 5 runs

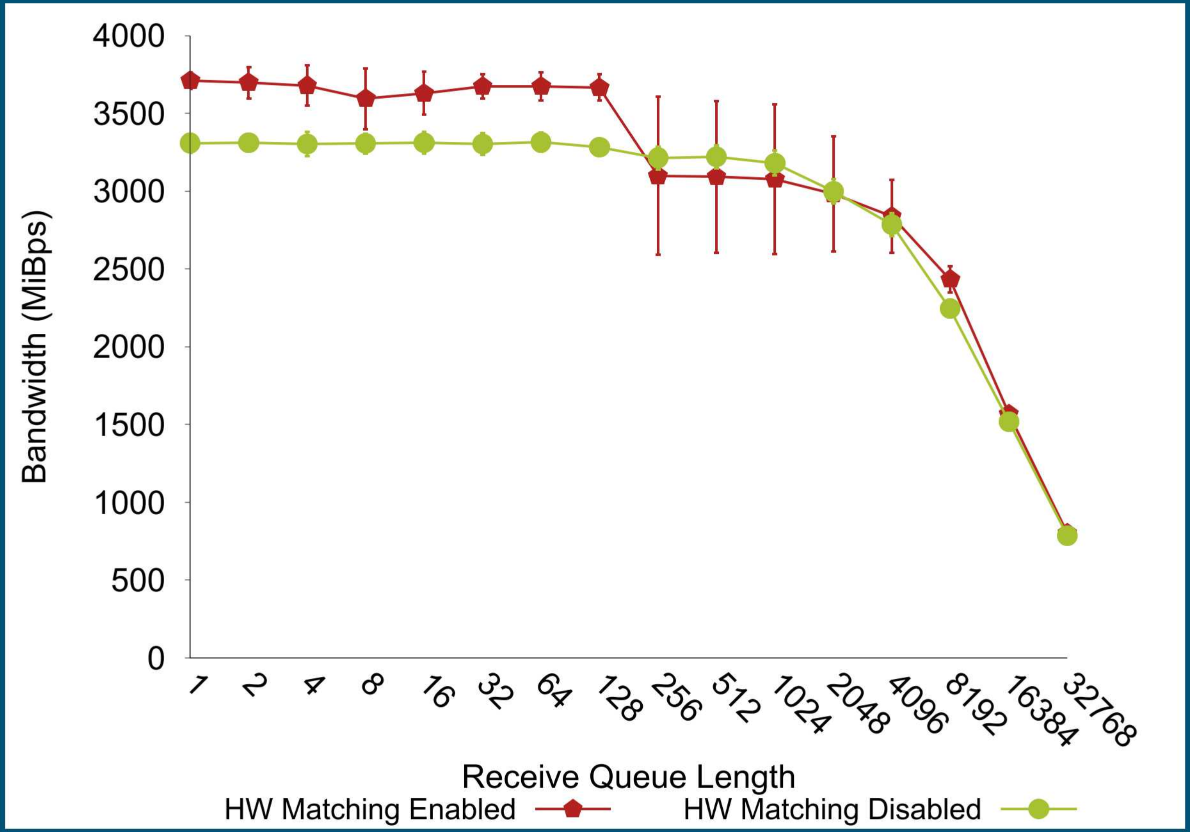
Results – 0% Collision Rate for 1B and 4KiB Messages



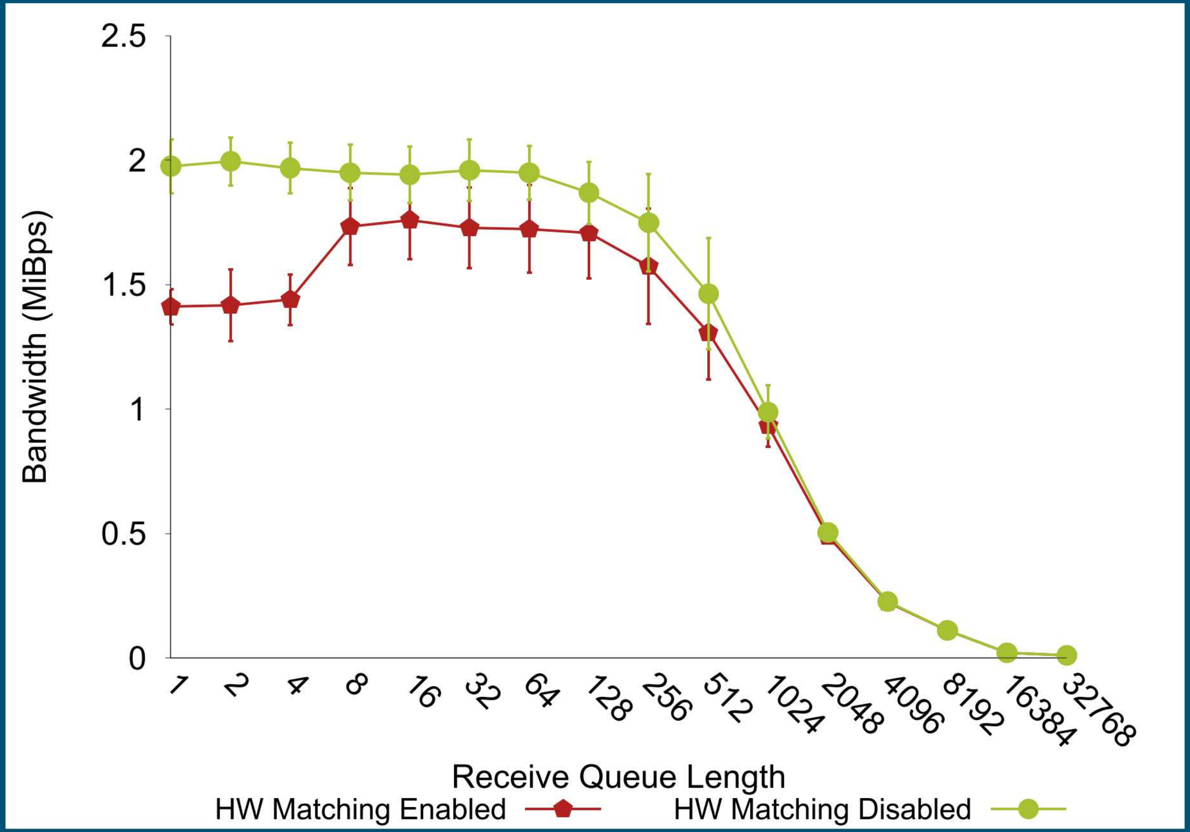


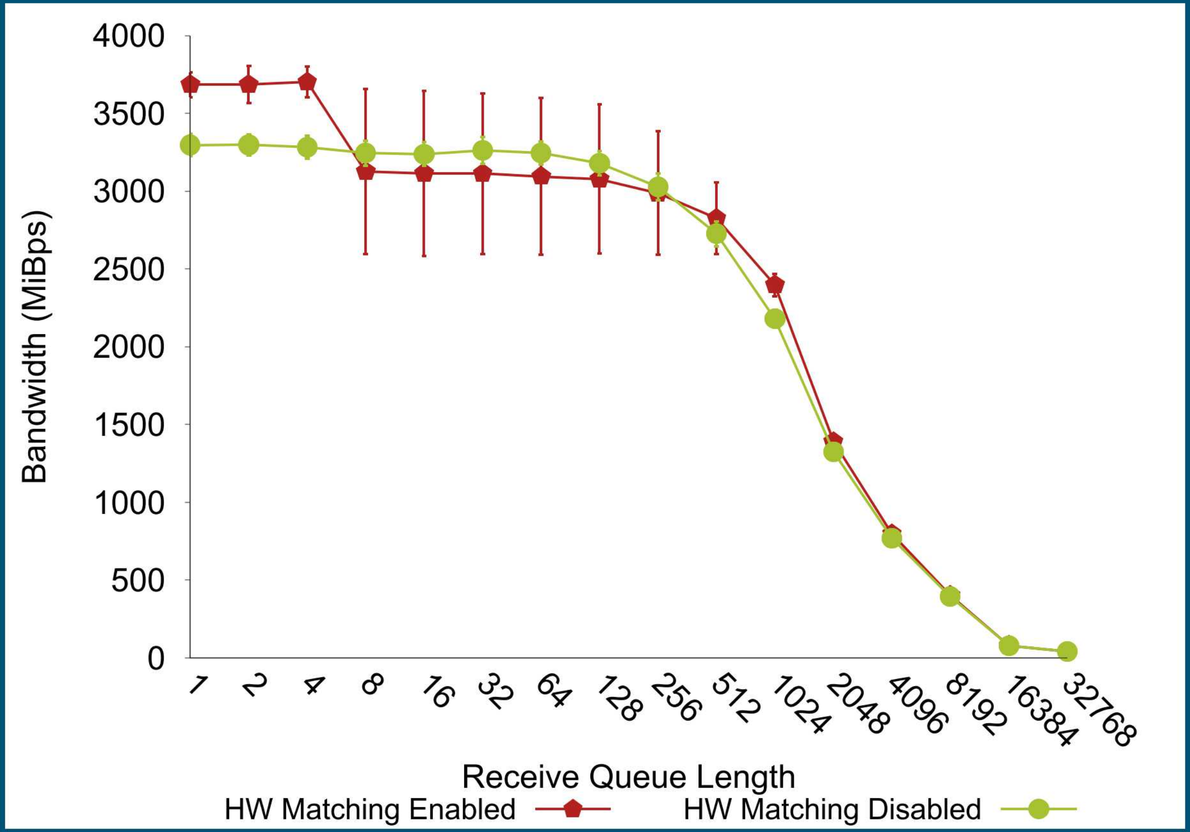
Results – 1% Collision Rate for 1B and 4KiB Messages



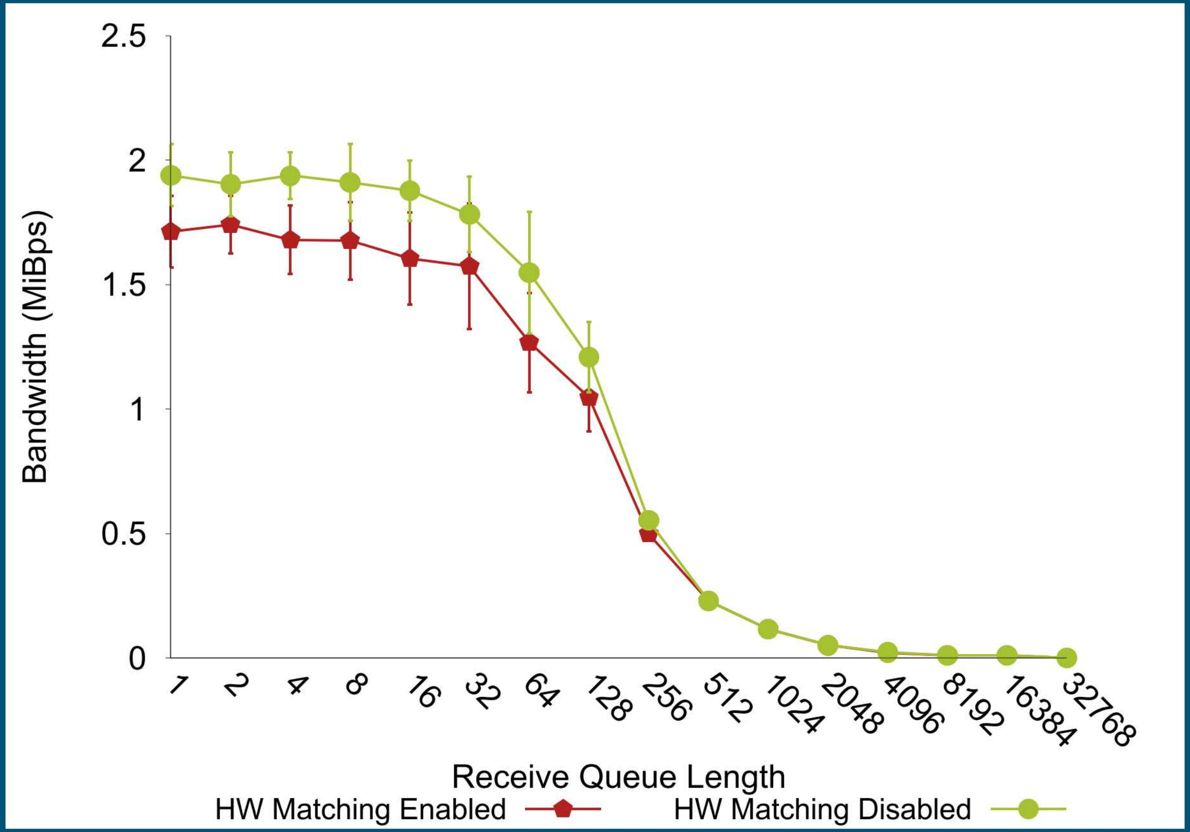


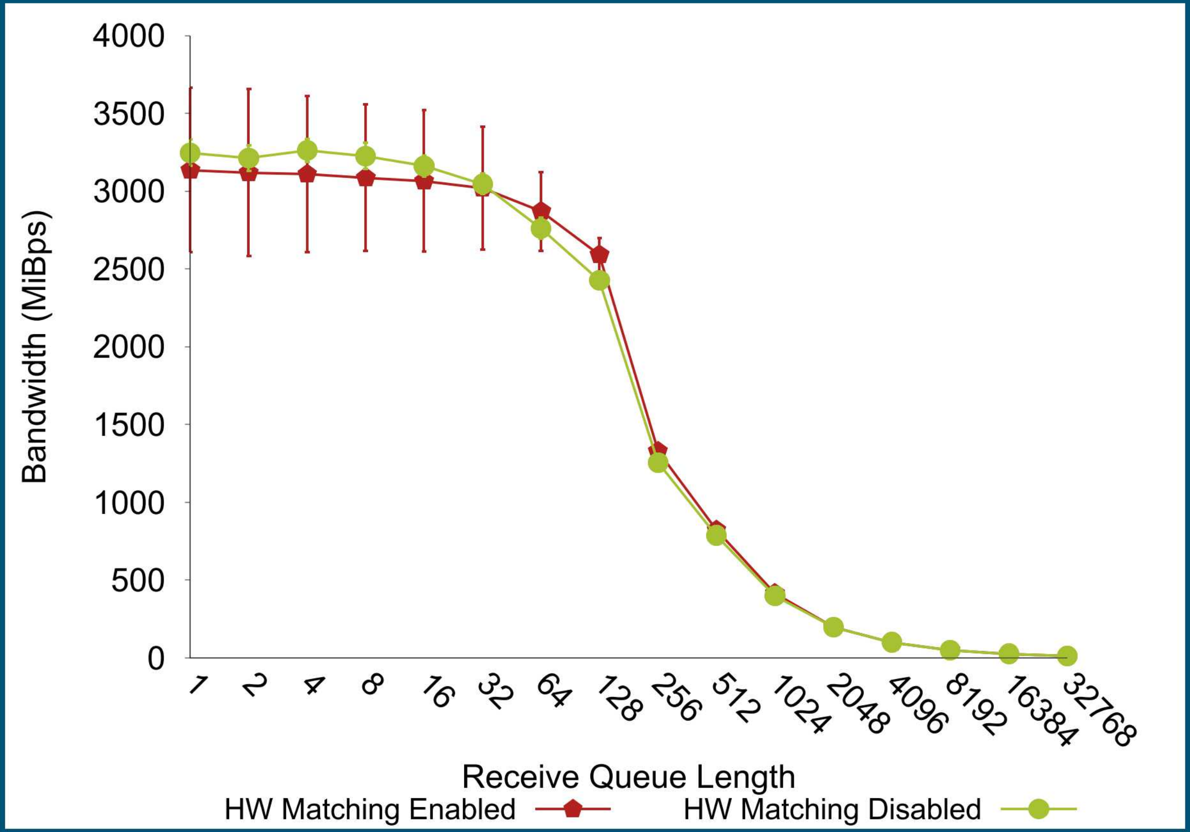
Results – 10% Collision Rate for 1B and 4KiB Messages



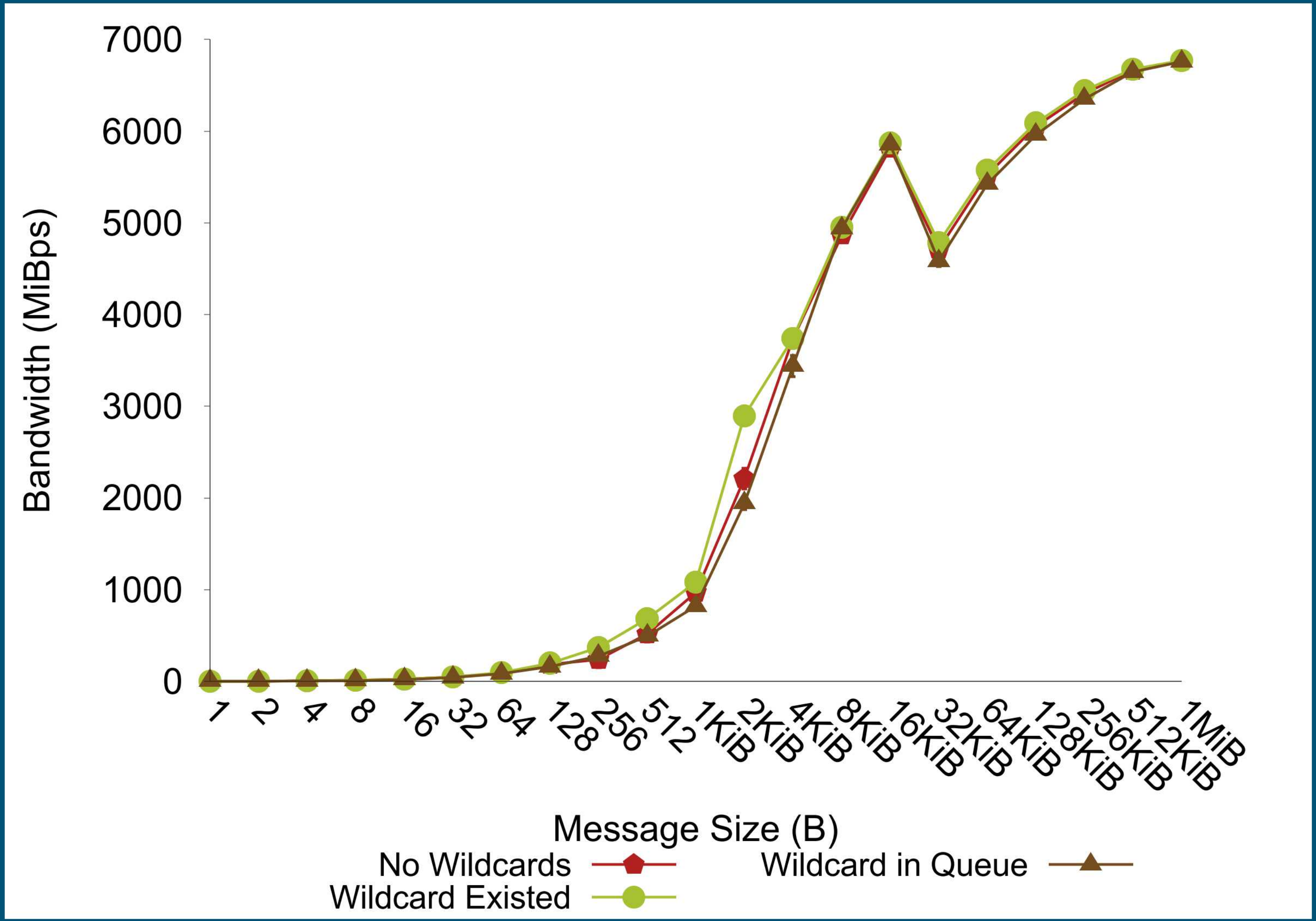


Results – 100% Collision Rate for 1B and 4KiB Messages

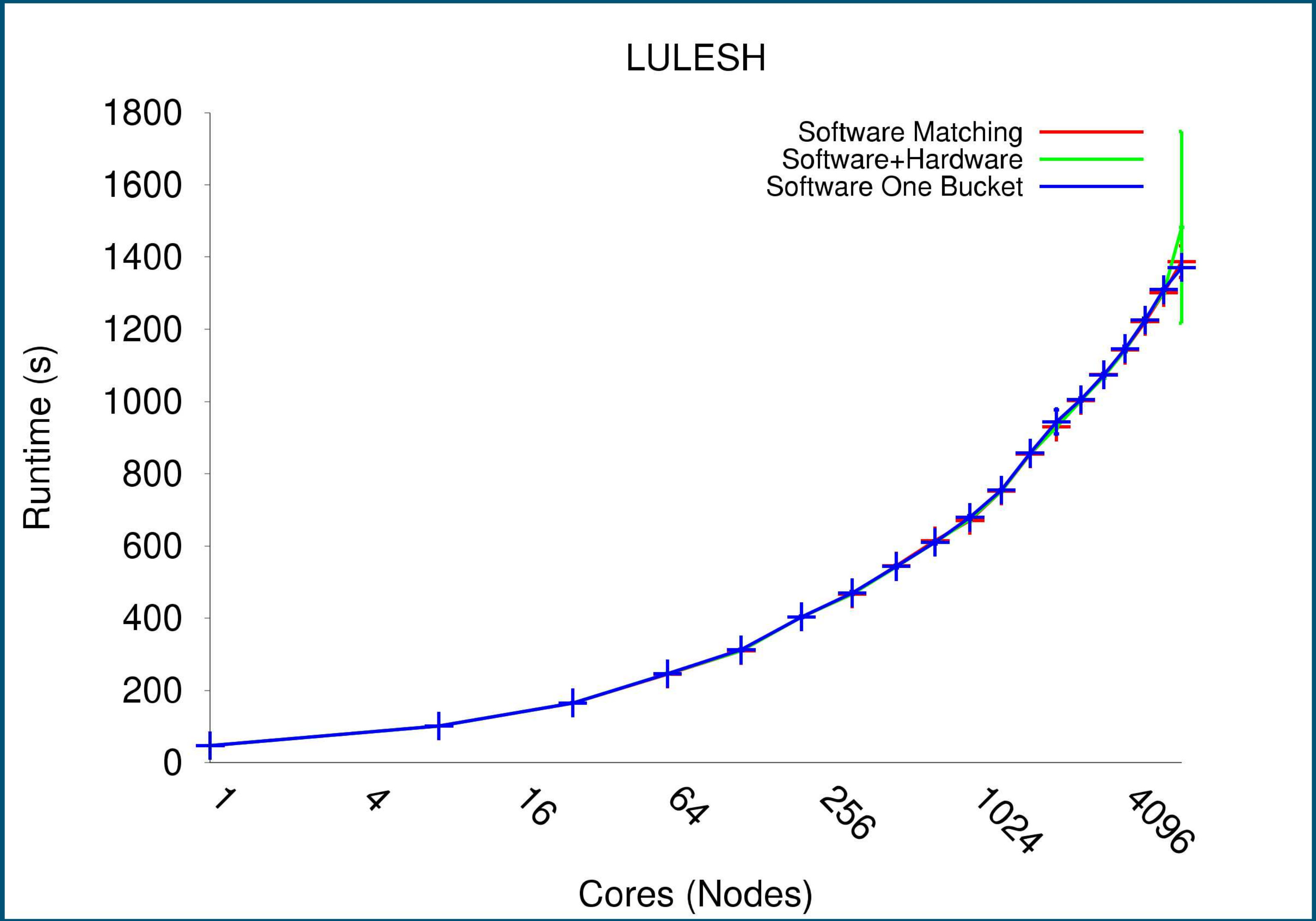




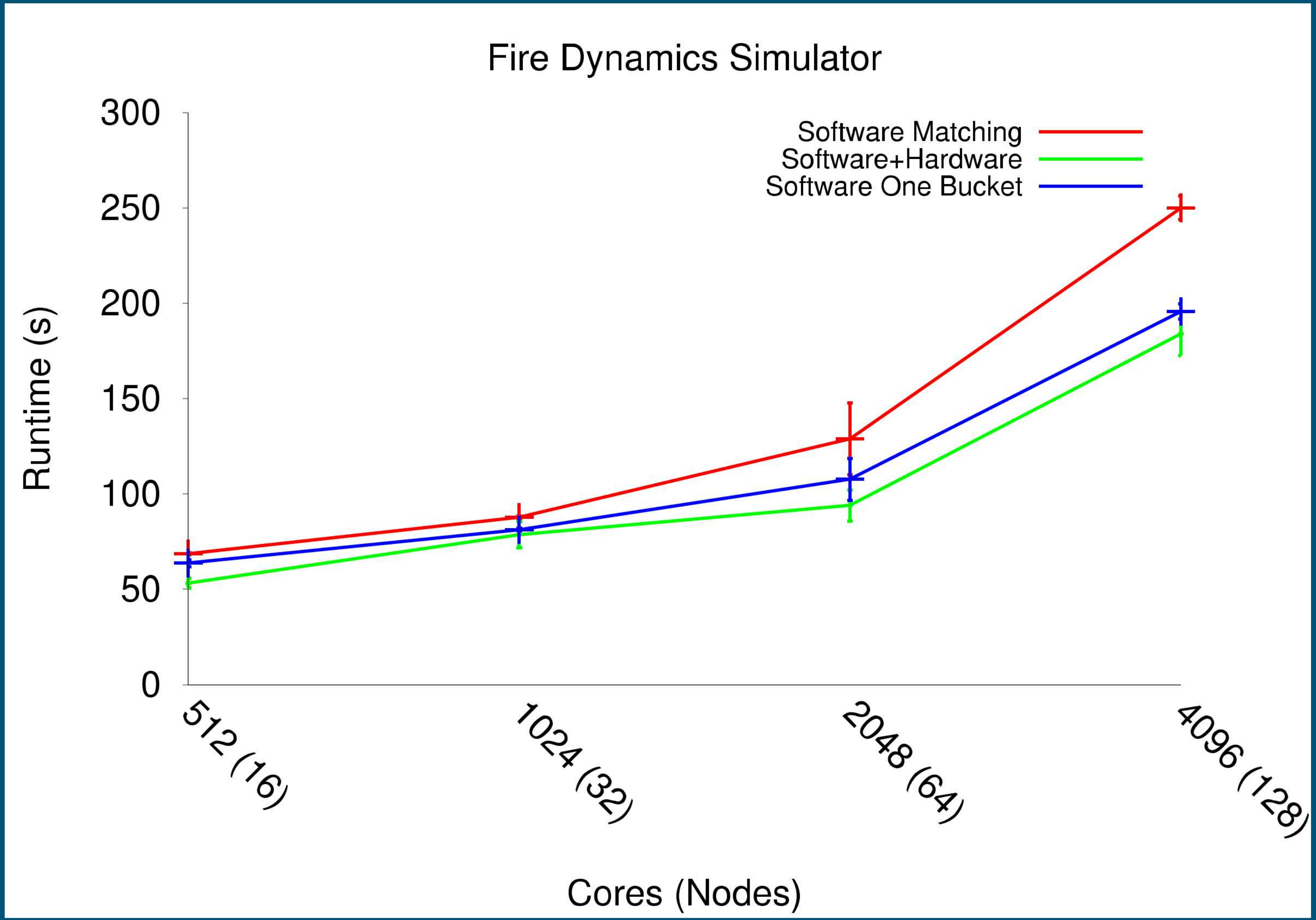
Results – Wildcard Test



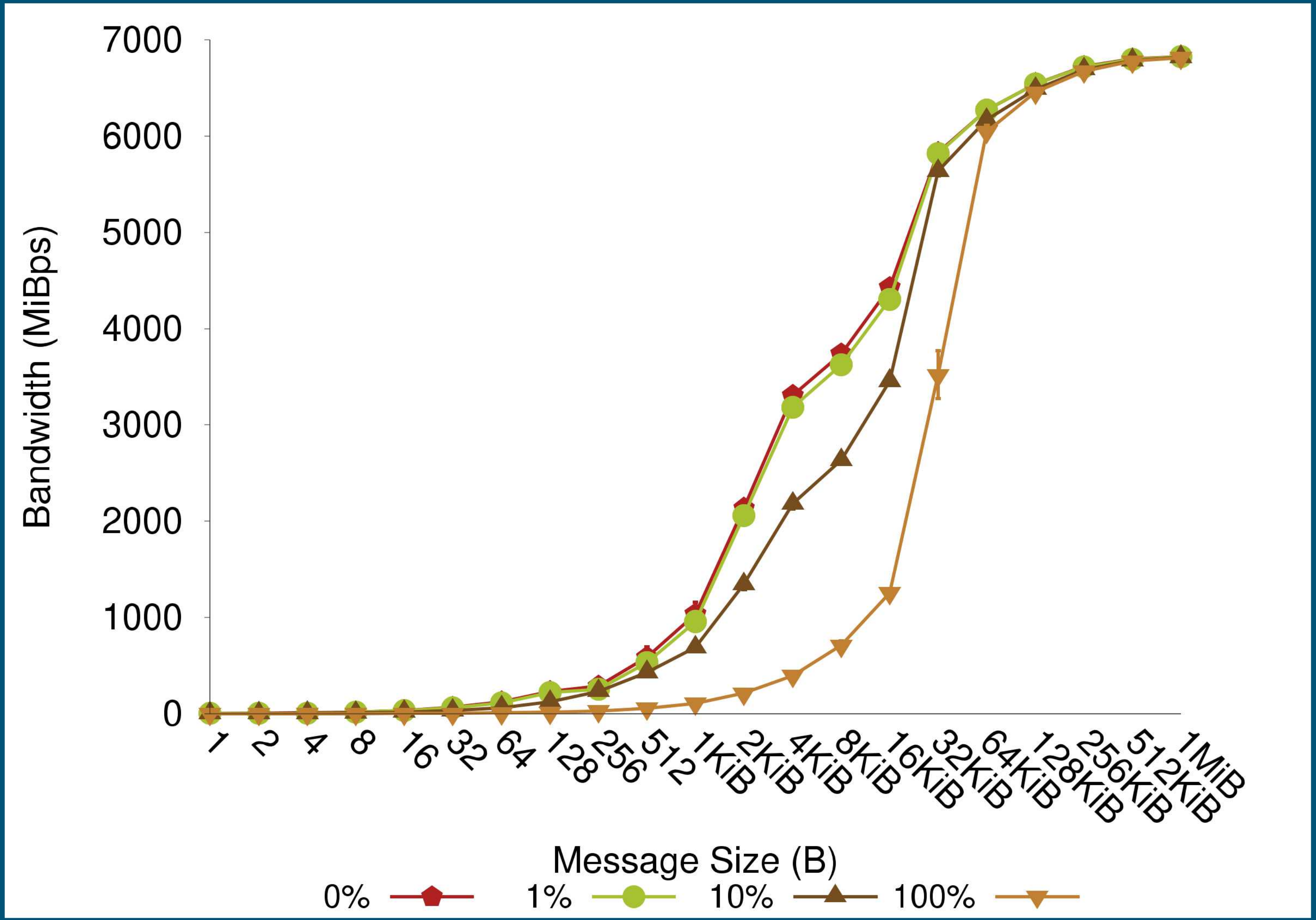
Results – LULESH



Results – FDS


Fire Dynamics Simulator
Software Matching
Software+Hardware
Software One Bucket
Runtime (s)
300
250
200
150
100
50
0
512 (16)
1024 (32)
2048 (64)
4096 (128)
Cores (Nodes)

Results – Bandwidth vs. Message Size (Hardware Off)



Results – Bandwidth vs. Message Size (Hardware On)