Implementation and Performance Development of Omni-Path Express

Douglas Fuller, VP Software Engineering

ExaComm ISC 23

CORNELIS NETWORKS

Our Company

Cornelis Networks provides open, intelligent, high-performance networking solutions designed to accelerate the world's most demanding technical computing applications.

CORNELIS™ NETWORKS

A History of Driving Interconnect Innovation

A PROVEN PEDIGREE

InfiniBand + Cray Aries

CORNELIS NETWORKS

Omni-Path Express

MARKET-LEADING TECHNOLOGY
Predictable Performance at Scale

CONTINUING INNOVATION

Driving Fabric Performance Forward

- Roadmap Innovation to 1600Gbps
- Network-based Compute
- CXL and Ethernet Expansion
- Open and Interoperable

"Unlocking the performance of technical computing workloads for over 20 years"

CORNELIS NETWORKS

Omni-Path Express Software Innovation

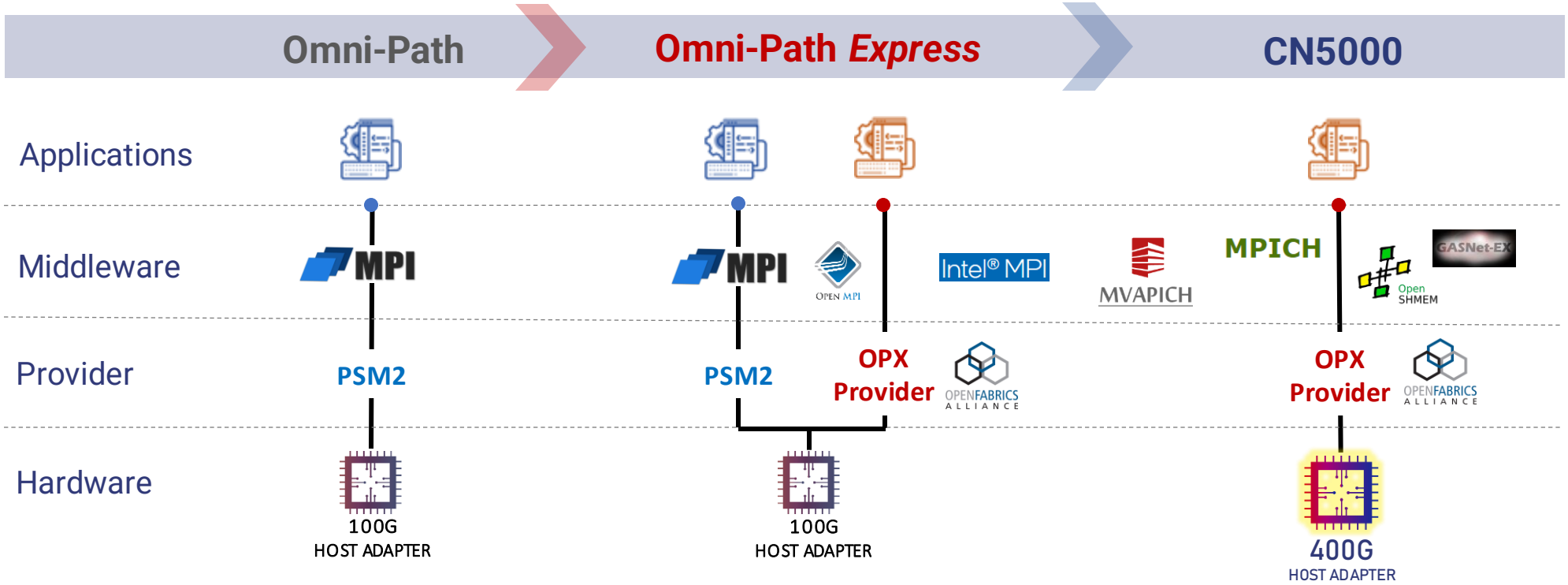
Optimized for high-performance converged infrastructures

- **Host architecture based on OpenFabrics Interfaces**
  - Collaboration with OSU for libfabric development
- Accelerated fabric performance at scale
- Broad support of application-critical technologies
- Foundational for next generation 400 Gbps fabric technology

CORNELIS™ NETWORKS

Omni-Path Evolution

Future-proofing your infrastructure with high-performance software layer



Consistent Software Platform

High performance today

Simple adoption of new hardware

libfabric over Omni-Path Express

New Hardware

Optimized performance via

Premier OFI Adapter

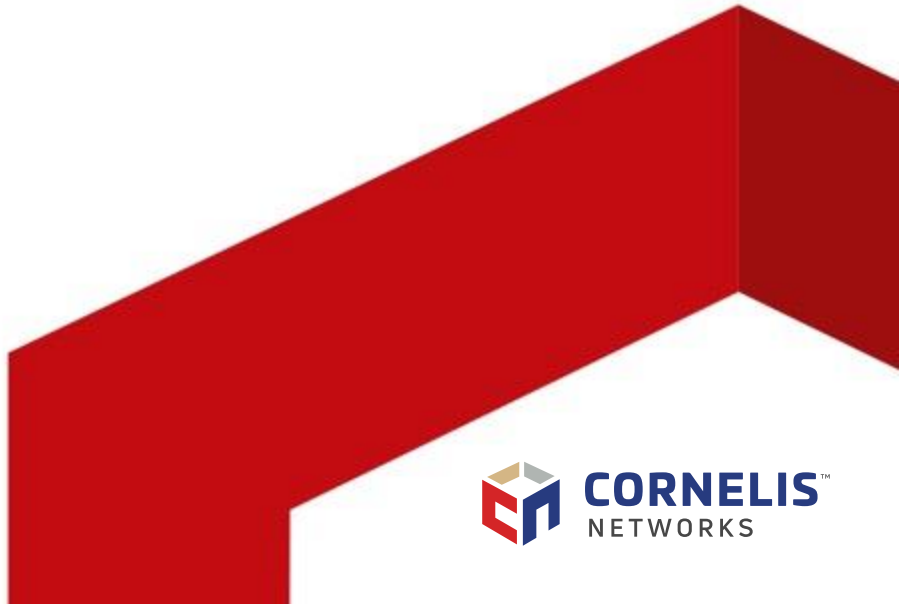
Omni-Path - Broad Ecosystem Engagement

Interoperability Drives Performance and Choice

Software	Ansys, ALTAIR, CONVERGENT SCIENCE, SIEMENS, OpenFOAM®
Storage	BeeGFS, ddn, lustre, PANASAS, VAST, WEKA, daos, IBM Spectrum Scale
Compute (CPU/GPU)	AMD, intel, NVIDIA, AMPERE
Operating Systems	Red Hat Enterprise Linux, Rocky Linux™, SUSE, ubuntu
Communication APIs	Open MPI, Intel® MPI, MVAPICH, MPICH, GASNet-EX, Open SHMEM
AI / ML	TensorFlow, PyTorch

Engineering OPX

- Extensive performance modeling
  - Instruction counts and cache hit rates
  - Intel software development emulator (SDE)
- Not the C that your professor taught you
  - Macro indirections to avoid function calls
  - Code duplication in some instances to avoid branches
- Result: latency reduced 18-24%!



CORNELIS™ NETWORKS

Performance Modeling with SDE

- Ice Lake 2.4GHz
- Intel SDE 9.0.0
- One side instrumented per run
- Examined regularly as development progressed

```
sde64                                \
   -debugtrace                        \
   -start_ssc_mark 200:1               \
   -stop_ssc_mark  210:1               \
   -log:mt                            \
   -log:focus_thread 1                 \
   -dt_out sdelog/sdelog.              \
   -dt_symbols 1                      \
   -dt_flush                          \
   -mix                               \
   -omix   sdelog/mixlog               \
   -footprint                         \
   -ofootprint sdelog/footprint        \
   -length 2000000000                 \
```

4 byte MPI_Send

send.gnu.dynamic.impi_psm:main	7
send.gnu.dynamic.impi_psm:MPI_Send@plt	1
send.gnu.dynamic.impi_psm	8
libmpi.so.12:MPI_Send	31
libmpi.so.12:pthread_self@plt	1
libmpi.so.12	32
libpthread.so.0:pthread_self	3
libpthread.so.0	3
libmpi.so.12:MPI_Send	7
libmpi.so.12:pthread_mutex_lock@plt	1
libmpi.so.12	8
libpthread.so.0:__pthread_mutex_lock	30
libpthread.so.0	30
libmpi.so.12:MPI_Send	208
libmpi.so.12	208
libpsmx2-fi.so:psmx2_tagged_injectdata_no_flag_av_map	27
libpsmx2-fi.so:psm2_mq_send2@plt	1
libpsmx2-fi.so	28
libpsm2.so.2:psm2_mq_send2	11
libpsm2.so.2:psm2_uuid_generate	302
libpsm2.so.2:pthread_spin_lock@plt	1
libpsm2.so.2	314
libpthread.so.0:pthread_spin_lock	5
libpthread.so.0	5
libpsm2.so.2:psm2_uuid_generate	41
libpsm2.so.2:pthread_spin_unlock@plt	1
libpsm2.so.2	42
libpthread.so.0:pthread_spin_unlock	4
libpthread.so.0	4
libpsm2.so.2:psm2_uuid_generate	43
libpsm2.so.2:hfi_get_mylabel	13
libpsm2.so.2:psm2_uuid_generate	74
libpsm2.so.2:psm2_mq_send2	3
libpsm2.so.2	133
libpsmx2-fi.so:psmx2_tagged_injectdata_no_flag_av_map	10
libpsmx2-fi.so	10
libmpi.so.12:MPI_Send	40
libmpi.so.12:pthread_mutex_unlock@plt	1
libmpi.so.12	41
libpthread.so.0:__pthread_mutex_unlock	3
libpthread.so.0:__pthread_mutex_unlock_usercnt	22
libpthread.so.0	25
libmpi.so.12:MPI_Send	11
libmpi.so.12	11
send.gnu.dynamic.impi_psm:main	1
send.gnu.dynamic.impi_psm	1
Totals	
	903

PSM2: 903 instructions

send.gnu.dynamic.opx_user:main	7
send.gnu.dynamic.opx_user:MPI_Send@plt	1
send.gnu.dynamic.opx_user	8
libmpi.so.12:MPI_Send	31
libmpi.so.12:pthread_self@plt	1
libmpi.so.12	32
libpthread.so.0:pthread_self	3
libpthread.so.0	3
libmpi.so.12:MPI_Send	7
libmpi.so.12:pthread_mutex_lock@plt	1
libmpi.so.12	8
libpthread.so.0:__pthread_mutex_lock	30
libpthread.so.0	30
libmpi.so.12:MPI_Send	215
libmpi.so.12	215
libfabric.so.1:fi_opx_tinject_0_FI_AV_MAP_0x0018000000000000ull_OFI_RELIABILITY_KIND_ONLOAD	256
libfabric.so.1	256
libmpi.so.12:MPI_Send	40
libmpi.so.12:pthread_mutex_unlock@plt	1
libmpi.so.12	41
libpthread.so.0:__pthread_mutex_unlock	3
libpthread.so.0:__pthread_mutex_unlock_usercnt	22
libpthread.so.0	25
libmpi.so.12:MPI_Send	11
libmpi.so.12	11
send.gnu.dynamic.opx_user:main	1
send.gnu.dynamic.opx_user	1
Totals	
	630

OPX: 630 instructions!

4 byte MPI_Send

PSM2

#				CACHE LINES				
# PERIOD[ms]	TID	LOAD	STORE	LD+ST	CODE	CD+LD	CD+ST	C+L+S	NEW
# THREAD_START 0									
# THREAD_START 1									
--	0	70	3	27	118	0	0	0	218
--	all	70	3	27	118	0	0	0	218

PSM2: 218 cache line accesses

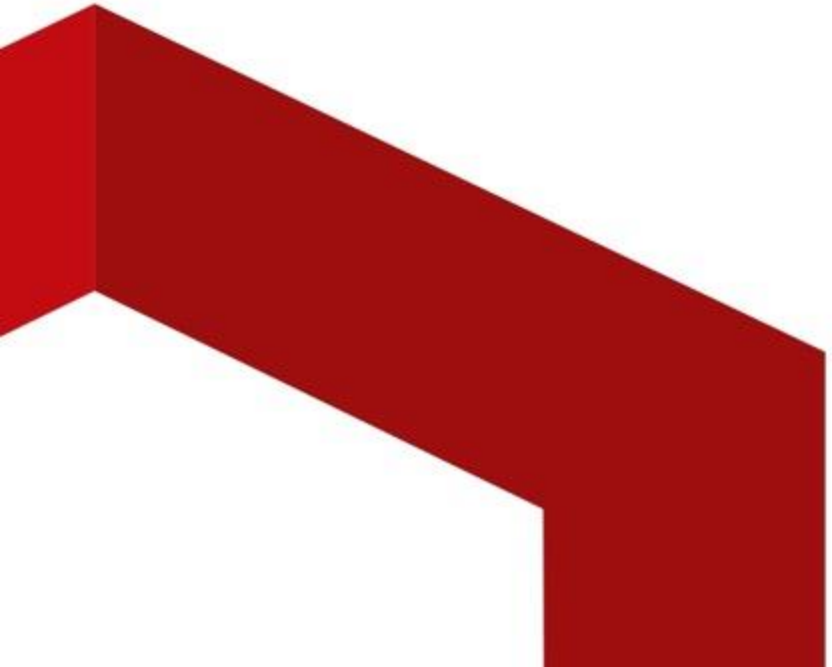
OPX

#				CACHE LINES				
# PERIOD[ms]	TID	LOAD	STORE	LD+ST	CODE	CD+LD	CD+ST	C+L+S	NEW
# THREAD_START 0									
--	0	45	4	17	78	0	0	0	144
--	all	45	4	17	78	0	0	0	144

OPX: 144 cache line accesses!

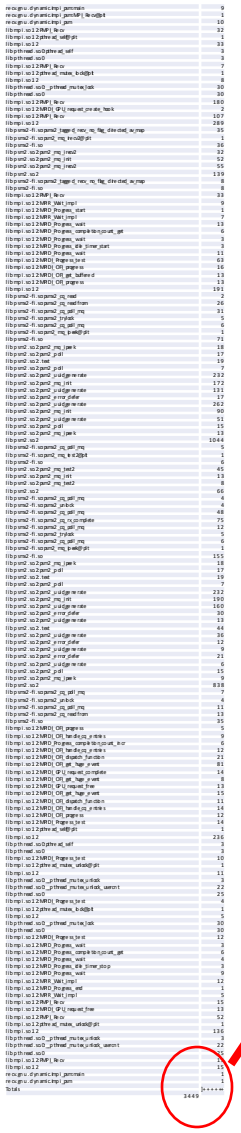
4 byte MPI_Send

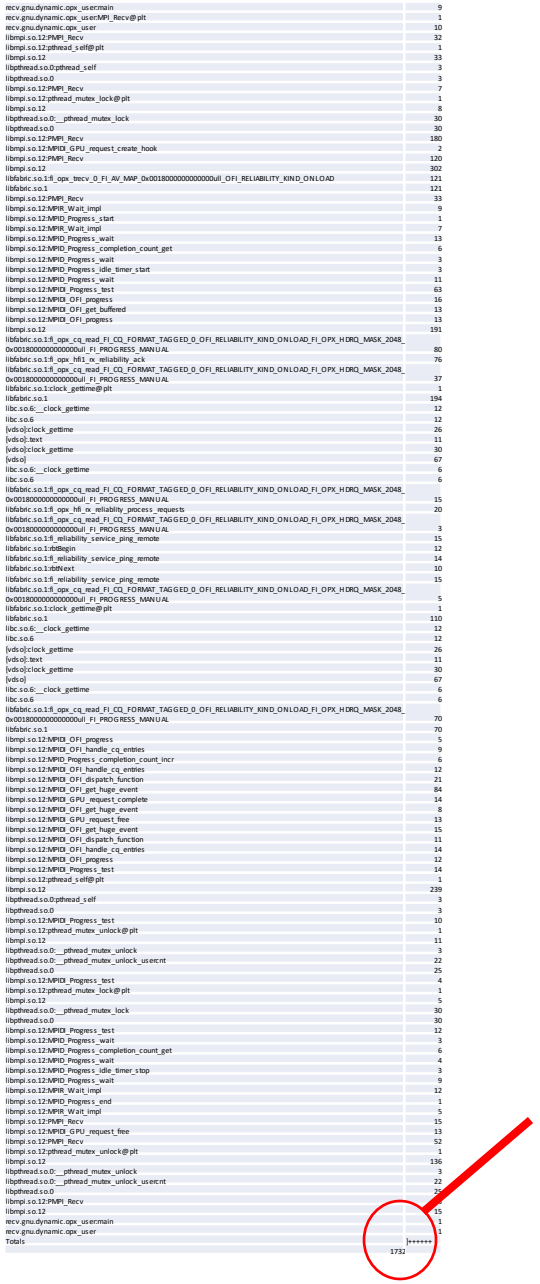
- OPX removed 14 function calls and a lock
- 30% reduced instruction count
- 33% reduced cache line accesses



CORNELIS™ NETWORKS

4 byte MPI_Recv





4 byte MPI_Recv

PSM2

```
#                        ============================================== CACHE LINES ==============================================
#  PERIOD[ms]          TID          LOAD         STORE         LD+ST          CODE         CD+LD         CD+ST         C+L+S          NEW
# THREAD_START 0
# THREAD_START 1
          --            0           104             6            66           290             0             0             0           466
          --          all           104             6            66           290             0             0             0           466
```

PSM2: 466 cache line accesses

OPX

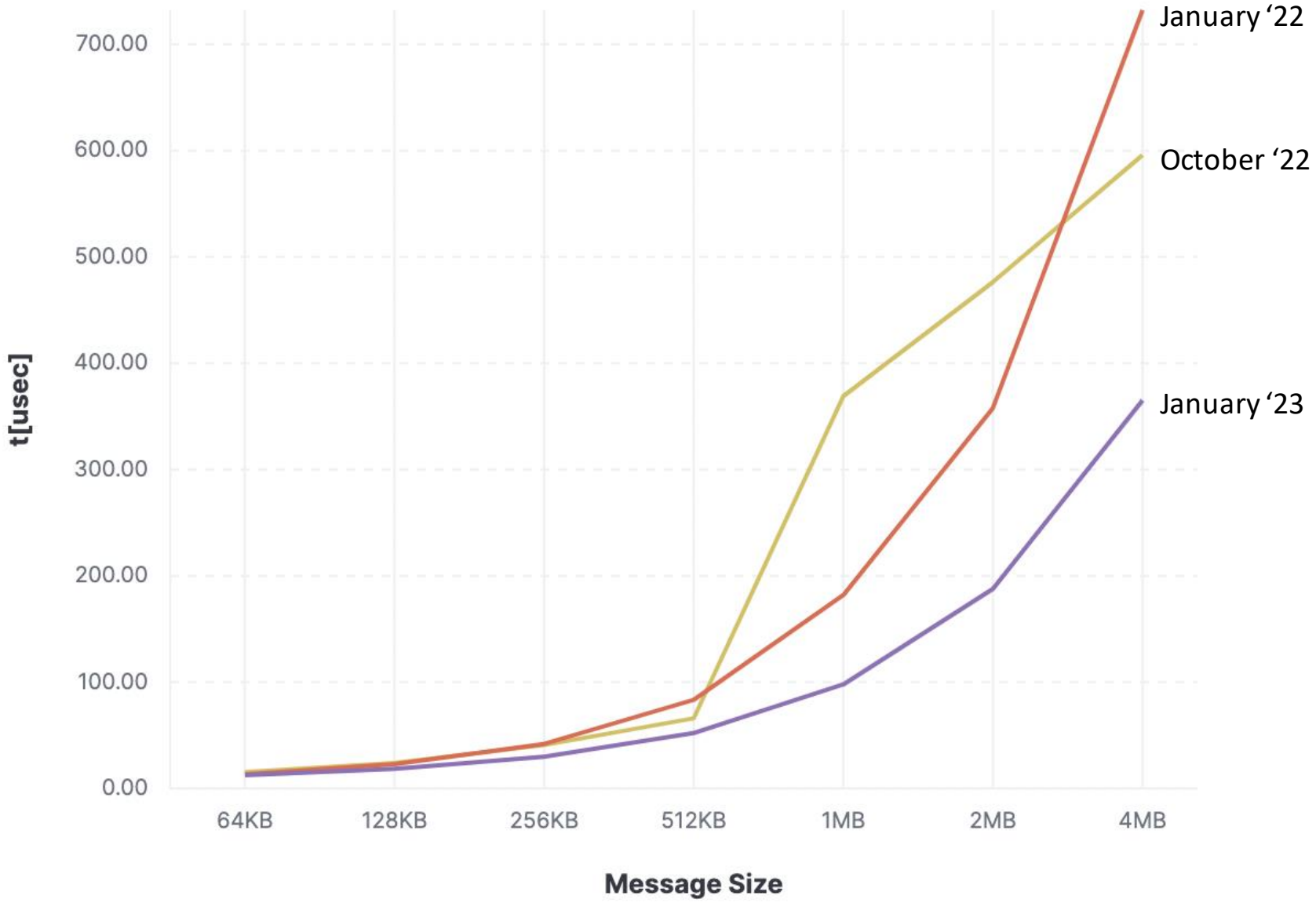
```
#                        ============================================== CACHE LINES ==============================================
#  PERIOD[ms]          TID          LOAD         STORE         LD+ST          CODE         CD+LD         CD+ST         C+L+S          NEW
# THREAD_START 0
          --            0            69             3            39           203             0             0             0           314
          --          all            69             3            39           203             0             0             0           314
```

OPX: 314 cache line accesses!

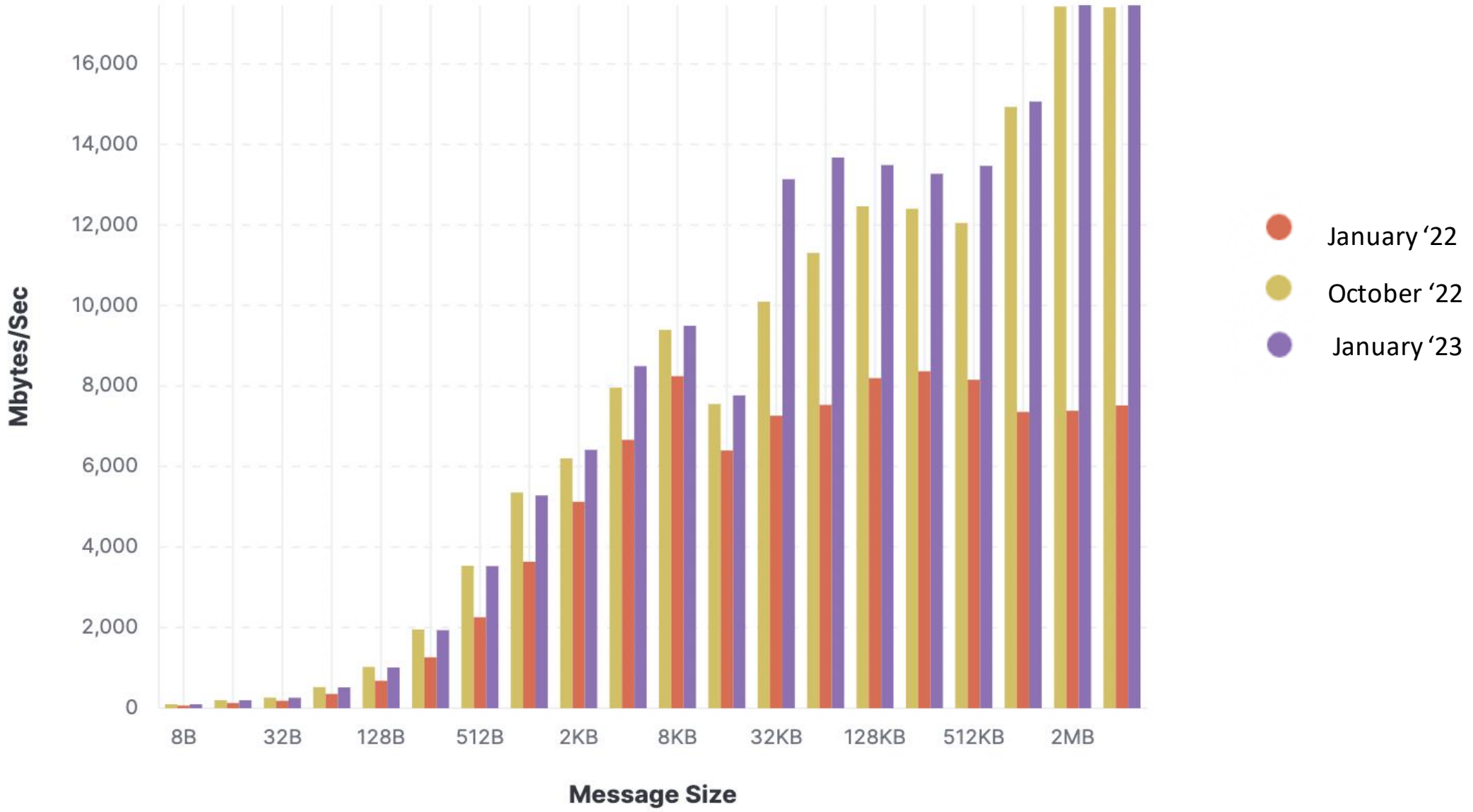
4 byte MPI_Recv

- OPX removed 34 function calls and a lock
- 50% reduced instruction count
- 32% reduced cache line accesses

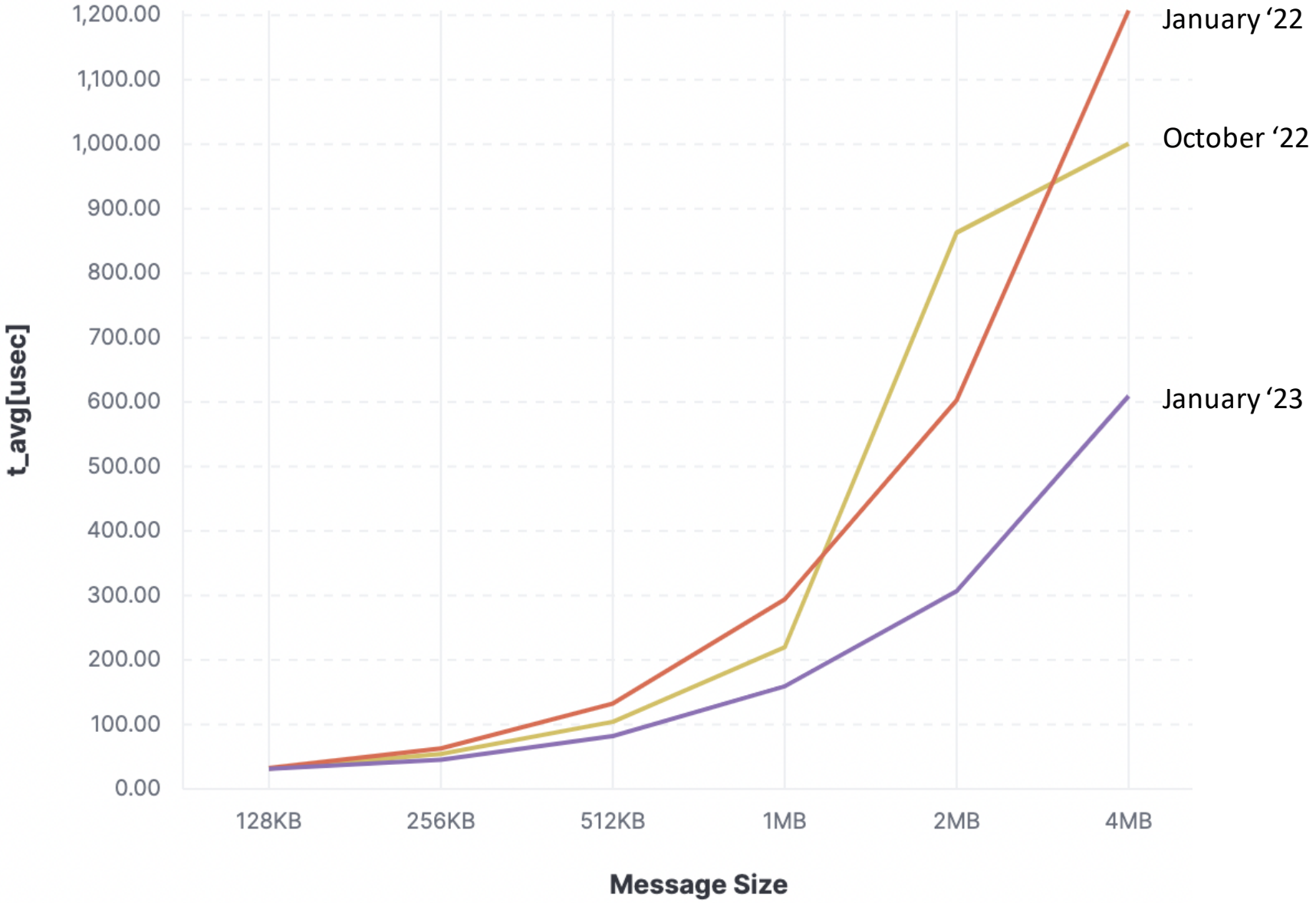
MPI-1 PingPong latency, 1 PPN



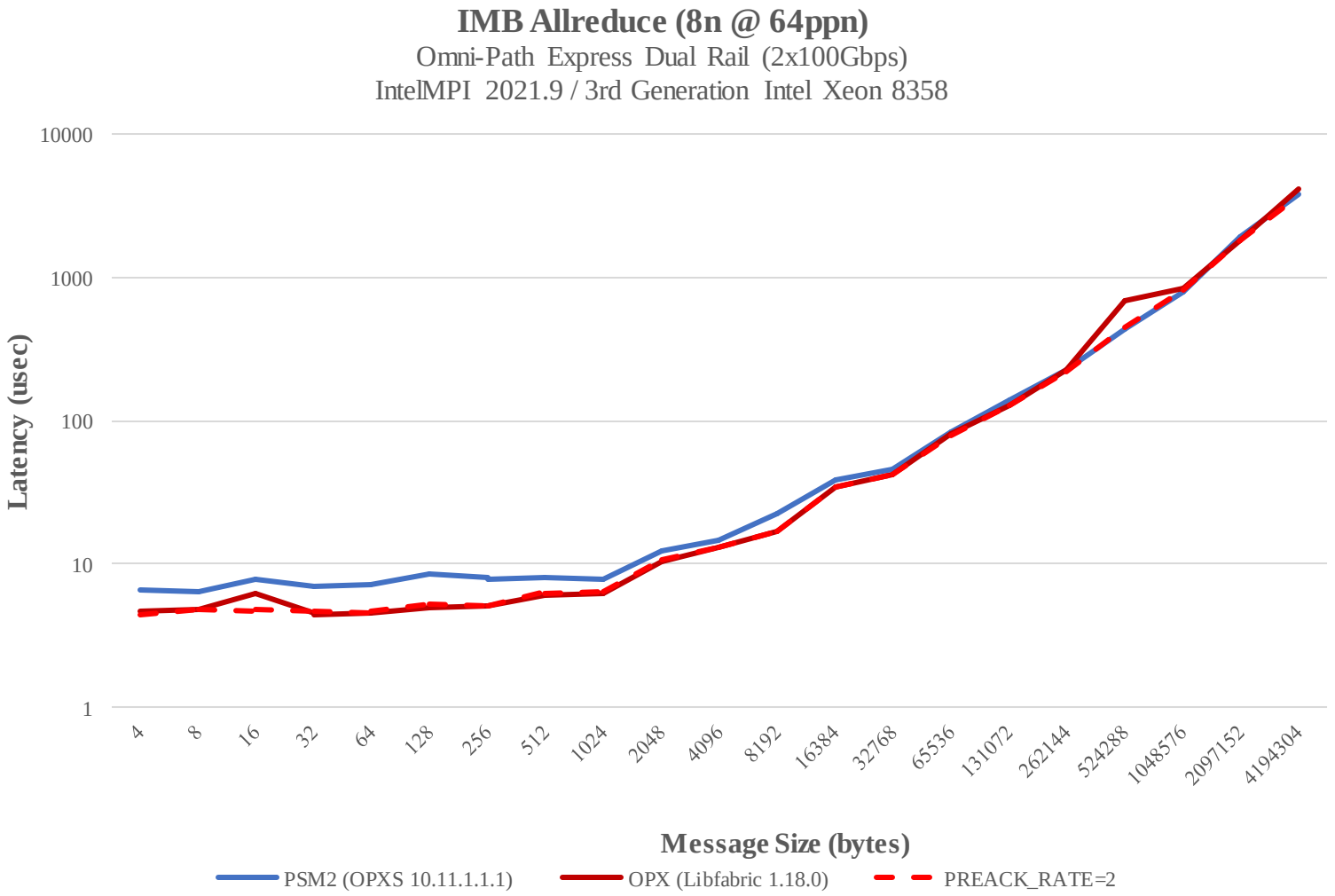
MPI-1 Biband Throughput



MPI-1 Allreduce latency



OPX AllReduce Collective Performance

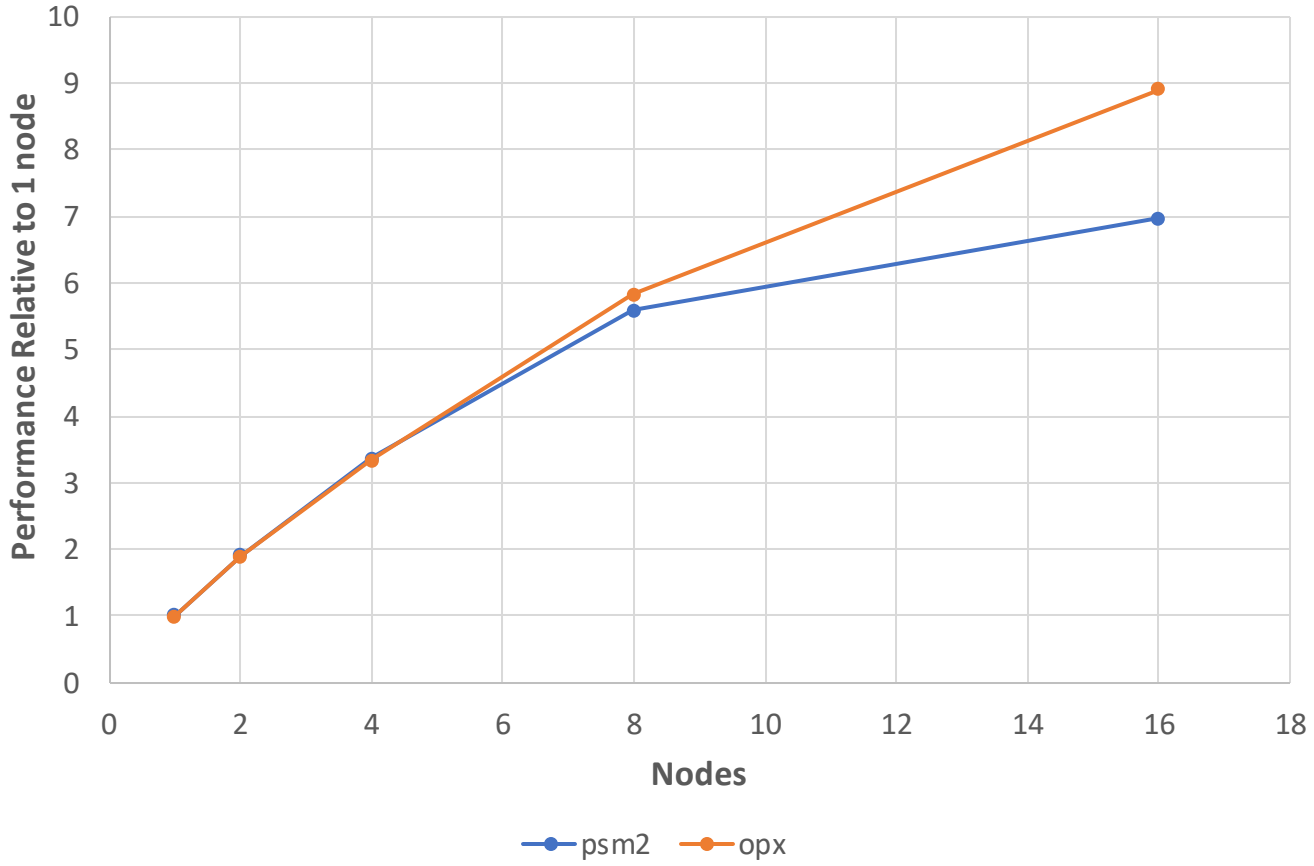


OPX delivers ~12-39% performance improvement over PSM2 up to 8KB message size

Setting `FI_OPX_RELIABILITY_SERVICE_PRE_ACK_RATE=2` provides additional minor performance improvements

OpenRadioss – Taurus 20m (2ms) Performance

3rd Generation Intel® Xeon® Scalable 8358
32 PPN/OMP=2
OpenMPI 4.1.4 & Libfabric 1.18.0



Selection of OPX provider at runtime delivers improved performance scaling for Taurus 20m workload.

28% scaling improvement at 16 nodes

MPI profiling shows dominant collectives are 4b and 80b AllReduce – inside the message size range where OPX delivers improved performance over PSM2

Why is OPX faster with non-blocking pt2pt?

CORNELIS NETWORKS

@--- Aggregate Sent Message Size (top twenty, descending, bytes) ---

Call	Site	Count	Total	Avrg	Sent%
Isend	173	90939438	1.18e+12	1.3e+04	54.18
Isend	249	11402352	2.32e+11	2.03e+04	10.61
Isend	143	52408336	1.77e+11	3.38e+03	8.10
Isend	44	2219533	1.71e+11	7.71e+04	7.84
Isend	287	8039202	1.53e+11	1.9e+04	7.01
Isend	113	21865275	1.07e+11	4.92e+03	.92
Isend	147	2219533	5.13e+10	2.31e+04	
Isend	33	22871424	1.82e+10	795	0.8
Allreduce	70	5147136	1.52e+10	2.94e+03	0.69
Isend	87	25816104	1.19e+10	462	0.55
Isend	353	8473836	1.08e+10	1.28e+03	
Isend	358	8473836	1.08e+10	1.28e+03	0.50
Reduce	139	10240	8.56e+09	8.36e+05	0.39
Ialltoallv	14	53776	6.95e+09	1.29e+05	0.32
Bcast	286	5137083	4.44e+09	864	0.20
Bcast	269	5137083	4.44e+09	864	0.20
Reduce	71	10240	3.64e+09	3.55e+05	0.17
Allreduce	203	5147136	2.43e+09	472	0.11
Bcast	189	261632	1.79e+09	6.85e+03	0.08
Bcast	97	261632	1.79e+09	6.85e+03	0.08

- Average msg sizes for non-blocking sends are 13KB, 20KB, 3.3KB, …
- This is the OPX "sweet spot" compared to psm2

	1ppn IMB-MPI1 Uniband, OpenMPI		
	opx mrate	psm2 mrate	opx/psm2 (%)
0	7864593	5051509	56%
1	7807544	5053984	54%
2	7842650	5046551	55%
4	7813961	5070017	54%
8	7479479	5056195	48%
16	7423408	4559895	63%
32	7398062	4542301	63%
64	7382638	4600275	60%
128	7040281	4532902	55%
256	6362364	4358932	46%
512	5719266	4170307	37%
1024	4633466	3793012	22%
2048	3389946	3185466	6%
4096	2149579	1883231	14%
8192	1185409	645387	84%
16384	590134	206465	186%
32768	350180	360876	-3%
65536	181899	183142	-1%
131072	92706	92971	0%
262144	46792	46899	0%