

How Caching Distorts MPI Point-to-Point Communication

Niklas Bartelheimer, Jens Domke, Sarah Neuwirth

- Existing Micro Benchmarks execute MPI routines in tight loops

- Branch prediction always right
- Hot data / instruction cache

⇒ Produces idealized performance results

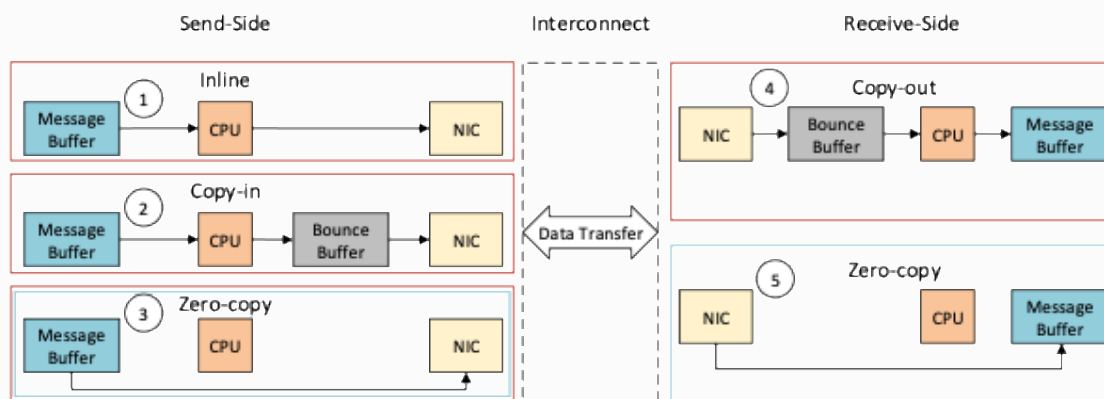
- Real-world applications perform computational tasks before / after executing MPI routines

- Branch prediction not always right
- Cache-misses more likely

⇒ What's the real-world performance of MPI?

- Cold-Cache MPI Benchmark Suite (ccMPI)
 - Investigation of caching-effects on MPI point-to-point communication under various conditions
- Systematic performance evaluation on three distinct HPC systems
 - Fugaku
 - Mogon II
 - Mogon-NHR
- Based on evaluation results
 - What speedup can you expect from a cached bounce-buffer?
 - Interceptor library for UCX bounce-buffers
 - Performance evaluation for various message sizes

Point-to-Point Data Transfer Concepts in MPI



1. Inline: Data is encapsulated into message descriptor and passed to the NIC. (small messages, eager-protocol)
2. Copy-in: Data is copied into pre-registered bounce-buffer. DMA <-> NIC (medium sized messages, eager-protocol)
3. Zero-copy: Buffer is registered for DMA access. No additional copy. (large messages, eager / rendezvous protocol)
4. Copy-out: Data is received in pre-registered and pre-posted set of bounce-buffers and copied back into message buffer. (eager-protocol)
5. Zero-copy: Message buffer is registered for DMA. Data is received directly. (rendezvous-protocol)

- Q1: What is the overall performance degradation for cold-cache settings?
 - Case 1: Cache-Level Flush
 - Selectively flushing individual cache-levels starting from L1 down to LLC before
 - Flush-level -> communicate
- Q2: What is the performance impact of a cached message buffer?
 - Case 2: Cached Message Buffer
 - Inspired by the MESI protocol; we consider cached message buffers either in Exclusive (read access) or Modified (write access) state
 - Flush LLC -> read / write message buffer -> communicate
- Q3: What is the performance impact if everything is cached except the message buffer?
 - Case 3: Cached MPI Data Structures
 - Flush LLC -> communicate (tmp message buffer) -> communicate (real message buffer)

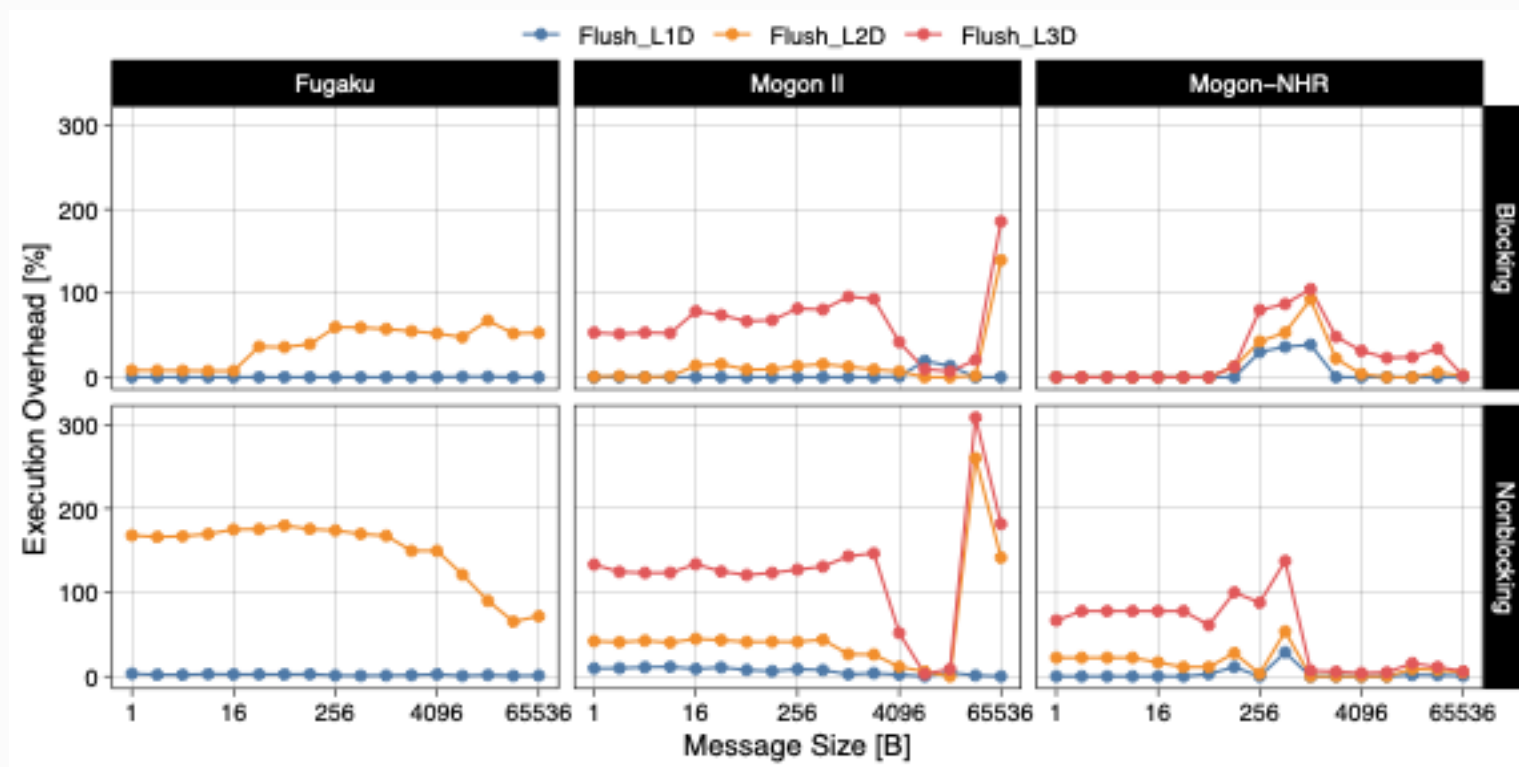
- One-way message transfer instead of typical ping-pong pattern
 - Allows to measure Send and Receive completion time independently
- Receive side benchmarks also consider separate experiments for expected and unexpected messages
- PAPI is used to record CPU cycles and performance counter values
 - Counter value analysis out of scope of this work
- Latency is measured as the CPU cycles needed to finish data transfers
 - Blocking: $\text{Latency}_{\text{Total}} = \text{Latency}_{\text{Send / Recv}}$
 - Nonblocking: $\text{Latency}_{\text{Total}} = \text{Latency}_{\text{Isend / Irecv}} + \text{Latency}_{\text{Wait}}$
- Performance is reported as the percentage overhead relative to the hot-cache baseline performance (median values):

$$O = \frac{P_e - P_b}{P_b}$$

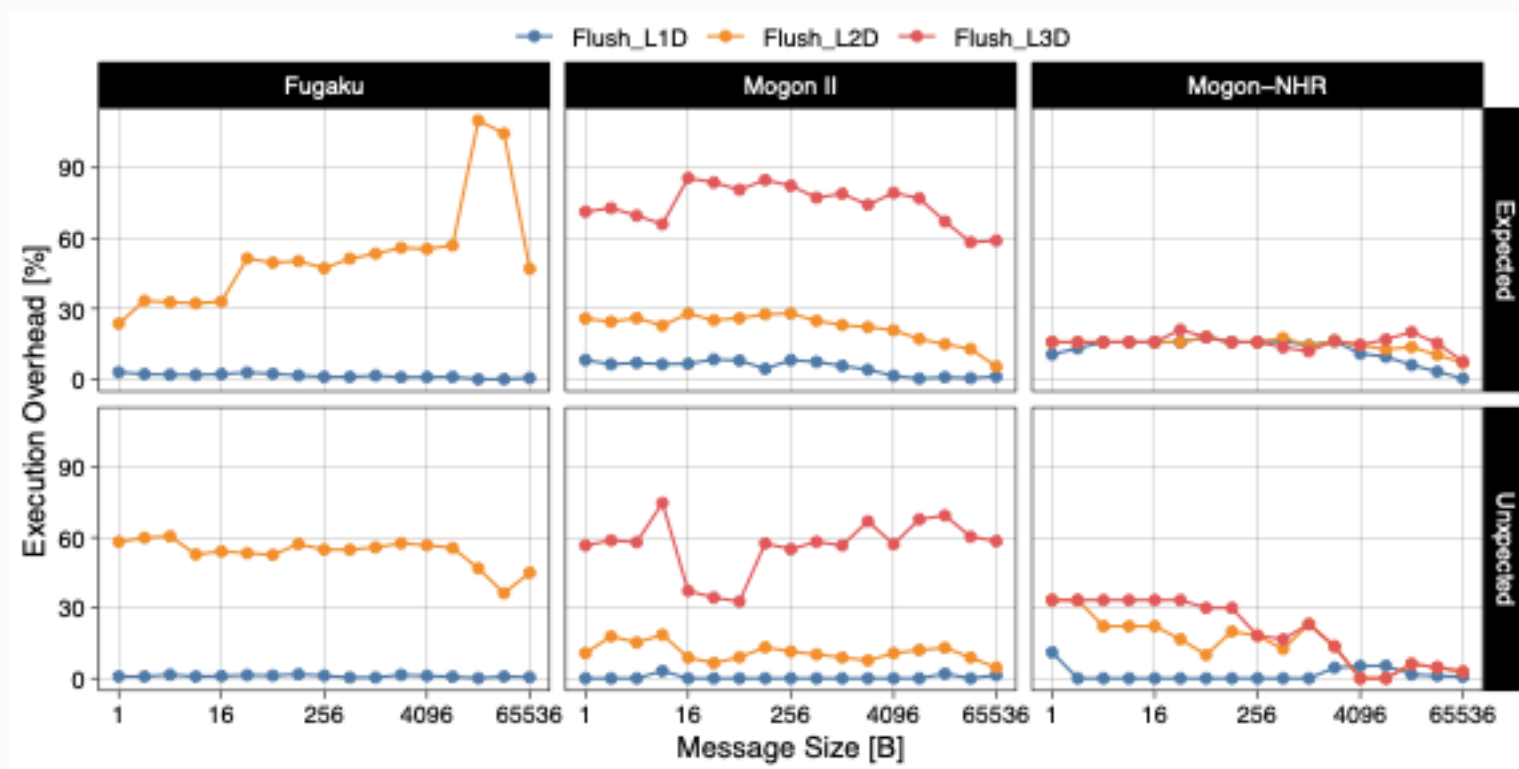
	Fugaku [26]	Mogon II [17]	Mogon-NHR [18]
Number of Nodes	158,976	576	590
Number of Sockets	1	2	2
Cores per Socket	48 + 2	16	64
Manufacturer	Fujitsu	Intel	AMD
CPU	A64FX	Xeon 2630v4	EPYC 7713
Clock Frequency	2.2 GHz	2.2 GHz	2.0 GHz
L1 Instruction	64 KiB	32 KiB	32 KiB
L1 Data	64 KiB	32 KiB	32 KiB
L2	8 MiB	256 KiB	512 KiB
L3	-	25 MiB	32 MiB
Line Size	256 B	64 B	64 B
Main Memory	32 GiB HBM2	64 GiB DRAM	256 GiB DRAM
Hyperthreading	-	Off	Off
Frequency Scaling	Off	On	Off
ASLR	Off	On	Off
OS	RHEL 8.10	AlmaLinux 8.5	AlmaLinux 8.10
Kernel Version	4.18.0-553.22.1	4.18.0-348.12.2	4.18.0-553.77.1
Interconnect	TofuD	Omni-Path	InfiniBand HDR
MPI	FujitsuMPI ¹ 4.12.1	IntelMPI 2021.13	OpenMPI 5.0.3
Network API	libtofu	PSM2@11.2	UCX@1.16
Compiler	fj@4.12.1	icp@2024.2	gcc@13.3.0

¹ Based on OpenMPI 4.0.1

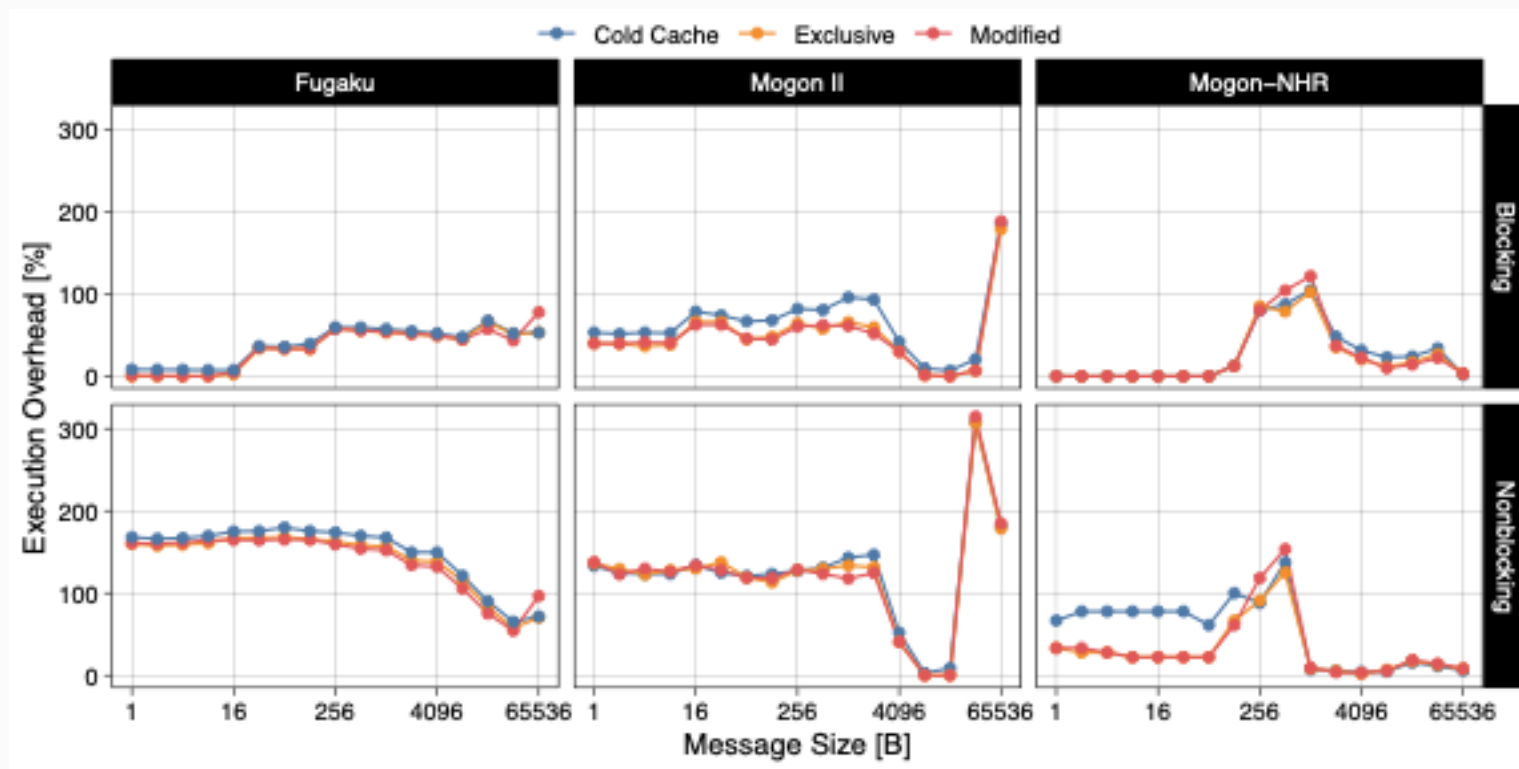
Case 1: Send Results



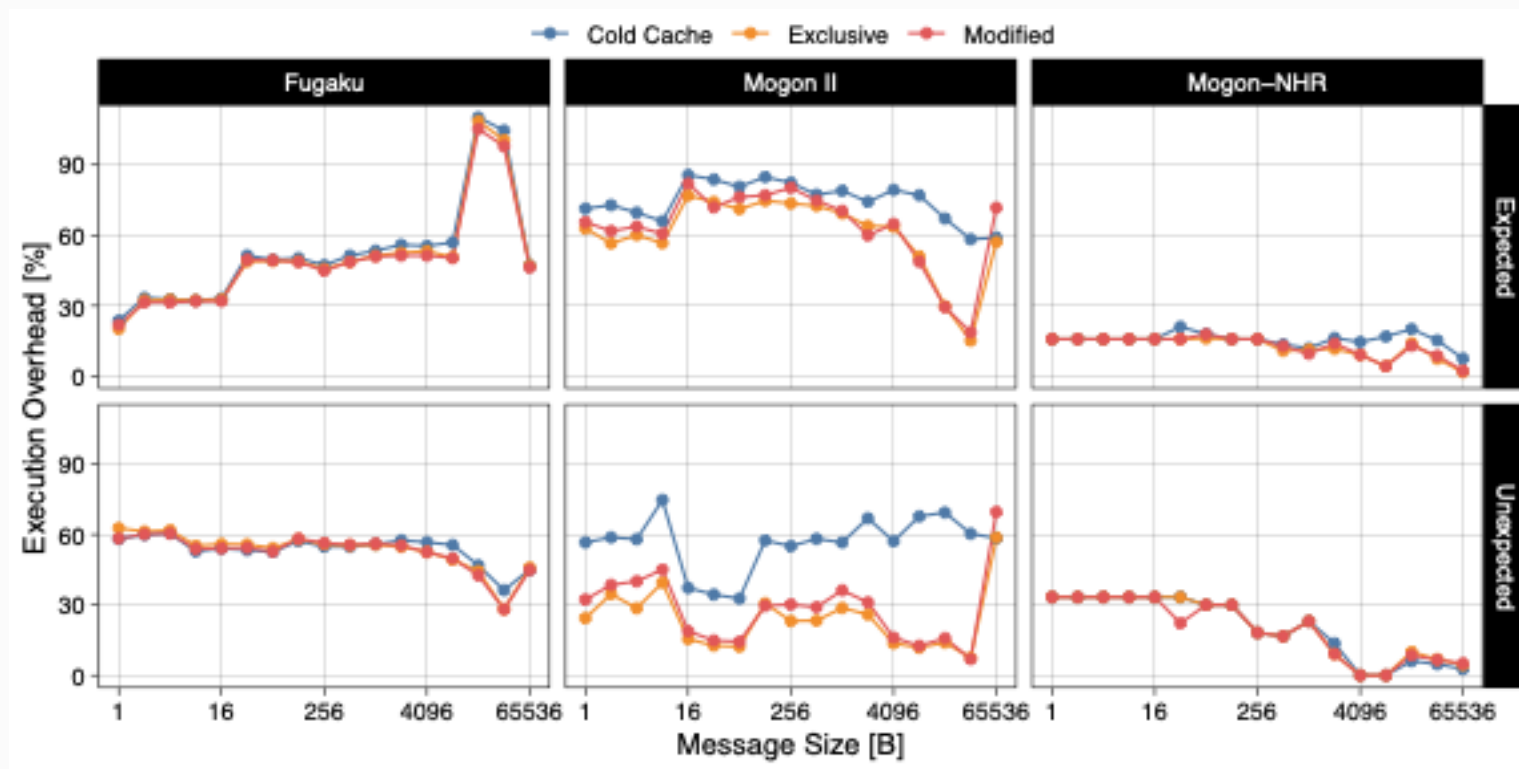
Case 1: Receive Results



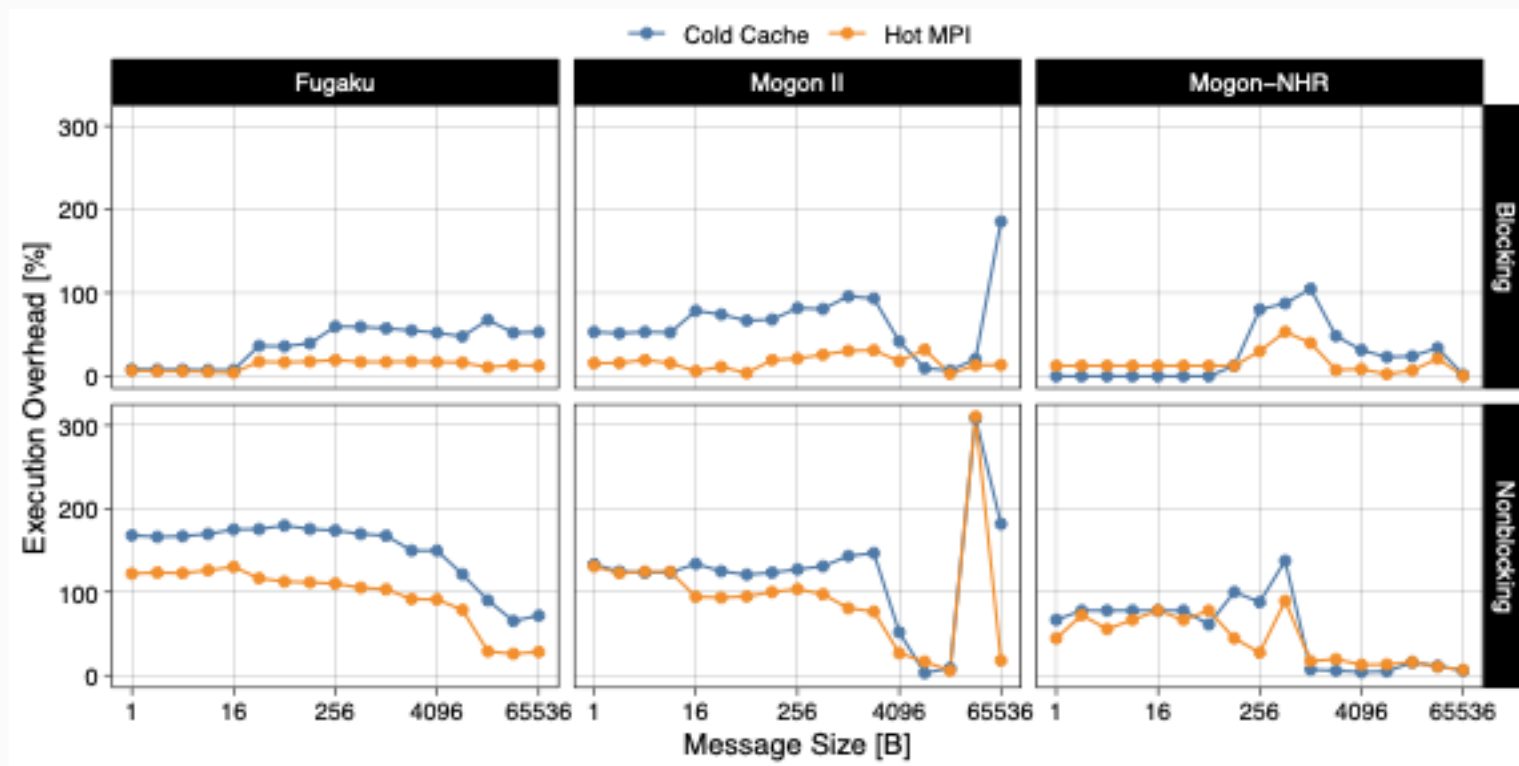
Case 2: Send Results



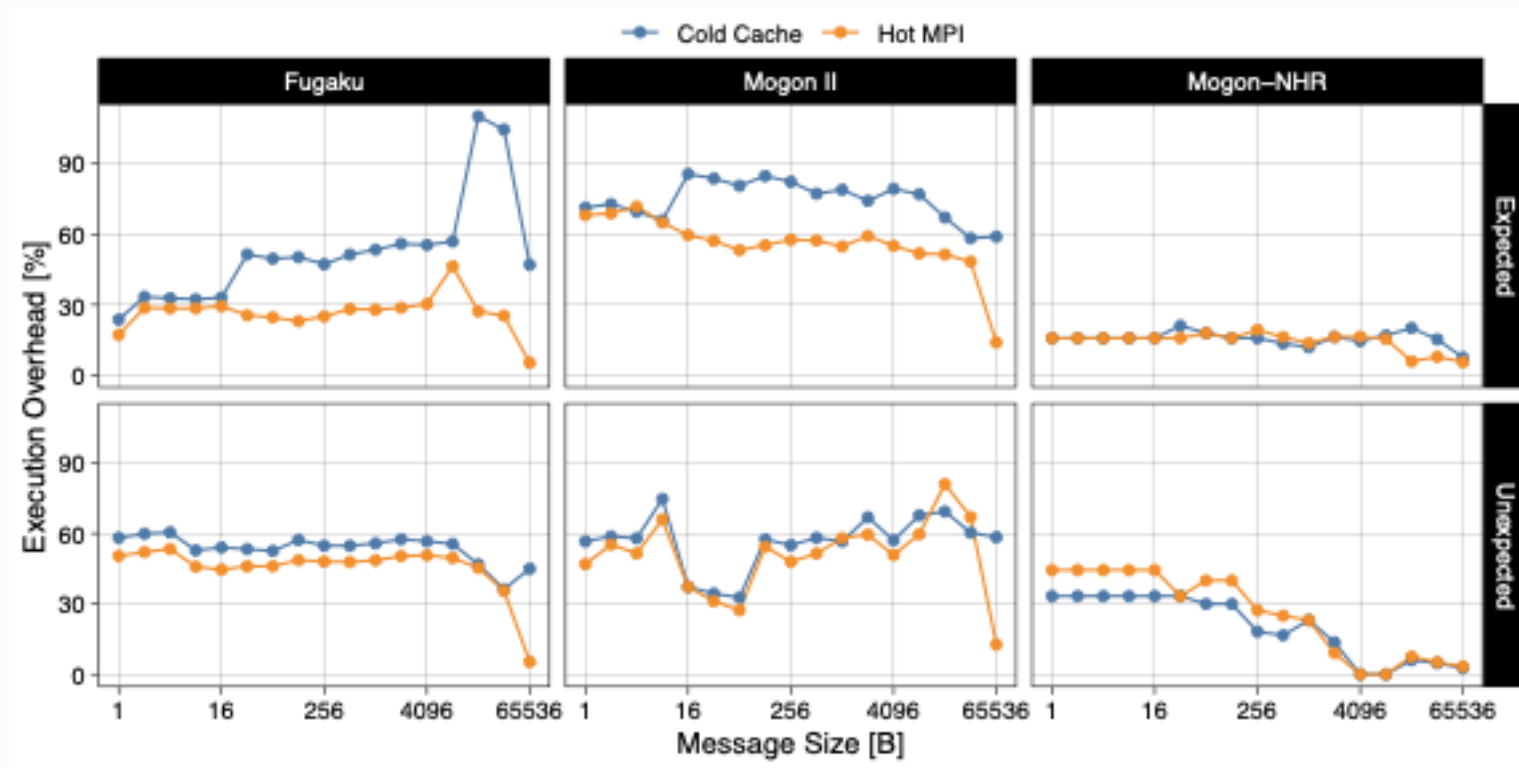
Case 2: Receive Results



Case 3: Send Results



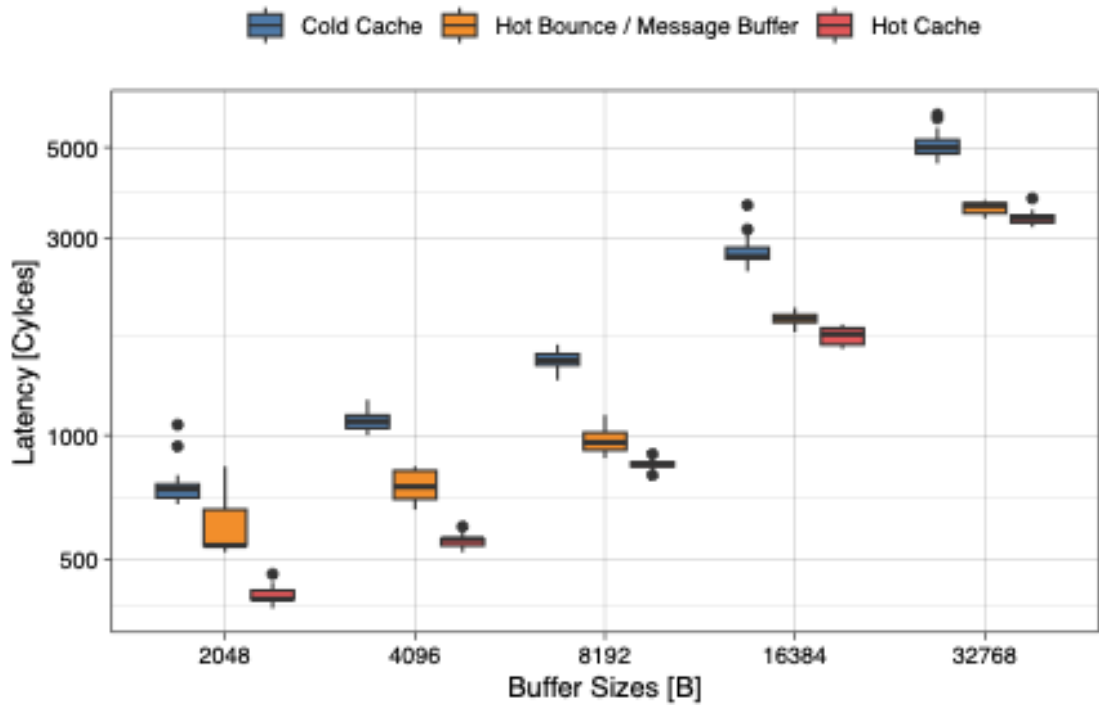
Case 3: Receive Results



- Observable caching effects vary between CPU micro architectures and network technologies
 - Omni-Path uses on-loading model (CPU used to process parts of the network protocol)
 - Tends to be more sensitive
 - TofuD, InfiniBand HCAs (NVIDIA / Mellanox) use off-loading model
- Cold cache scenarios consistently slow MPI sends and receives across all evaluated systems
- MPI internals contribute substantially (analysis based on Callgrind):
 - Internal buffer copies
 - Copy-in protocol usage on Mogon-NHR
 - PIO usage on Mogon II
 - Polling progress engine
 - Memory registration
- Locality of message buffer has little effect on performance? (old Omni-Path HW excluded)
- Systems tend to be noisy which makes it harder to find explanations for every peak

- Based on performance results from Mogon-NHR (Case 1)
 - What's the speedup of if memcopy between message and bounce buffer happens in cache?
- Interceptor library based on LD_PRELOAD
 - Monitor memory pool creation and usage inside UCX (UCT)
 - Memory pools employ LIFO list of available bounce-buffers
 - First buffer in list is the one we need to cache
- Performance evaluation similar to ccMPI
 - Only Send performance is measured
 - Receive bounce-buffer usage out of scope for now

Prefetch-based Optimization for UCX Bounce-Buffers – Results



Message Size [B]	2,048	4,096	8,192	16,384	32,768
Hot Message / Bounce Buffer	1.37	1.44	1.58	1.41	1.38
Hot Cache	1.85	1.93	1.79	1.54	1.48

- Speedup of hot message and bounce-buffer is substantial
 - Approaches performance of hot-cache scenario with growing message sizes
- Tried to translate speedup into test application
 - 2D-Stencil from PRK repository
 - Adjusted domain size to cover 1,2,4,8 and 16 KiB message sizes
 - Used 2, 4 and 16 nodes (1 PPN)
 - Failed to hide prefetching overhead
 - A lot of consecutive prefetch instructions needed
 - Might cause cache pollution
 - Interfere with HW prefetcher
 - Performance always below baseline

- Study caching effects on MPI collective communication performance
- Profile cache-misses in MPI calls in Proxy / Mini Apps
 - mpiP extension is already existing
- Briefly investigated feedback-driven optimization using DynamoRIO's cache simulator
 - Generated prefetching hints for LLVM
 - No performance gain

Questions?

- Each benchmark case: 10 warm-up and 100 internal iterations
- 30 invocations of “mpirun” per benchmark case
- All measurements collected within the same node allocation (connected to same switch if applicable)
- CPU Frequency scaling: Fugaku (off), Mogon II (pinned to 2.2 GHz), Mogon-NHR (off)

ccMPI: <https://github.com/NeuwirthLab/ccMPI>

ucxmempooltracer: https://github.com/NeuwirthLab/ucx_mpool_tracer

- PIO better latency SDMA (Source DMA) better bandwidth
- Receive Side still PIO
- Blocking Messages: — Intel® Xeon® Processor: PSM2_MQ_EAGER_SDMA_SZ=34000
- Non-Blocking messages: — Intel® Xeon® Processor: PSM2_MQ_EAGER_SDMA_SZ=16000

[1] Intel® PSM2 API—Intel® Omni-Path Fabric