MENPS: A Decentralized Distributed Shared Memory Exploiting RDMA

Wataru Endo[1], Shigeyuki Sato, and Kenjiro Taura

Graduate School of Information Science and Technology
The University of Tokyo

November 13, 2020
@ IPDRM 2020

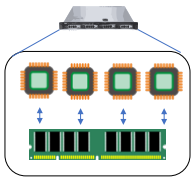
[1] This affiliation is based on when this work was done.

Summary

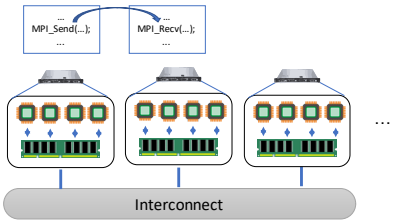
- Background: **RDMA**-capable interconnects provide new design options for middleware
  - We particularly focused on utilizing RDMA for **distributed shared memory (DSM)**
- Motivation: **existing RDMA-based DSM cannot fully exploit the performance of RDMA**
- Contributions:
  - Implemented a DSM library "**MENPS**" ("MENPS is Not a PGAS System")
    - Runs OpenMP programs in C/C++ w/ minimal modifications
  - Propose two changes to the DSM coherence protocol for exploiting RDMA:
    - **Floating home-based protocol** to accelerate write operations
    - **Hybrid invalidation** to accelerate read operations
- Evaluation: NAS Parallel Benchmarks [Bailey et al. '91]
  - MENPS accelerated two of five OpenMP applications using multiple nodes
  - MENPS performed better than an RDMA-based DSM system Argo [Kaxiras et al. '15]

Introduction (1/4): Two memory models

- **Shared memory**
  - All of the cores share the address space
  - Implicit communications by the underlying memory system
  - Pros ☺: easy to program, similarity to sequential programming
  - Cons ☹: not available with many cores



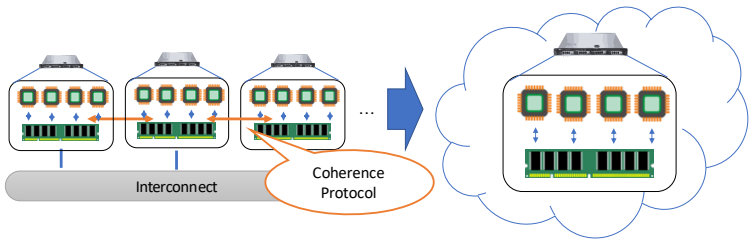
- **Distributed memory**
  - Separate address space for each node
  - Usually programmed w/ MPI
  - Pros ☺: high scalability
  - Cons ☹: poor application productivity



Introduction (2/4): DSM

- **Distributed Shared Memory (DSM)**
  - Physically distributed, logically shared
  - All of the cores share the same address space (as in shared memory)
  - Synchronizes caches with coherence protocols

  - Pros ☺: high application productivity as in shared memory
  - Cons ☹: often difficult to scale due to inter-node latency



Introduction (3/4): Trends of shared-memory systems

- **Hardware** shared-memory systems
  - Multi-core processors with increasing core counts
  - Success of cache-coherent NUMA architectures (≈ hardware DSM)

AMD Opteron[2]

Intel Xeon Phi[3] (max. 72 cores)

IBM Power AC922[4] (Main component of Summit)

Fujitsu A64FX[5] (Fugaku's CPU, 48 cores/node)

- **Software** shared-memory systems
  - - 1990s: Many researchers have contributed to DSM
    - "The lasting impact of these systems has not been high" [Ramesh et al. '11]
  - 2000s -: HPC community switched to **Partitioned Global Address Space (PGAS)**
    - Pros ☺: better scaling / Cons ☹: limited productivity due to the lack of caches

[2] https://upload.wikimedia.org/wikipedia/commons/9/91/AMD_Opteron_2212_IMGP1795.jpg
[3] https://www.intel.co.jp/content/www/jp/ja/architecture-and-technology/many-integrated-core/intel-many-integrated-core-architecture.html
[4] https://www.ibm.com/products/power-systems-ac922
[5] https://www.fujitsu.com/global/Images/supercomputer-fugaku.pdf

Introduction (4/4): Motivation

- We focused on the utilization of **remote direct memory access (RDMA)**:
  - Inter-node communication acceleration of interconnects
  - Pros ☺: low latency, high BW, kernel bypass
  - Cons ☹: specific restrictions of interfaces
- Today's RDMA latency $\approx 1\mu s$
  - Only several times slower than inter-socket latency

InfiniBand products of Mellanox[6]

- Problem: the existing RDMA-based DSM protocols are not capable for fully exploiting RDMA
  1. They depend on remote diff merging or remote interrupts ("trilemma")
  2. They depend on centralized directory structures

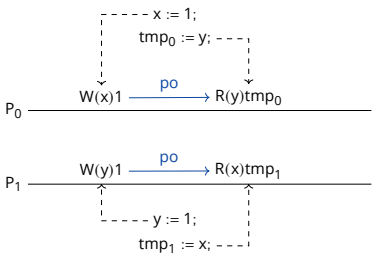
[6] https://www.mellanox.com/products

Contributions

- Implemented an experimental RDMA-based DSM “MENPS”
  - Runs OpenMP programs in C/C++ with minimal modifications
  - Based on release consistency
- Two proposed changes to the DSM protocol to exploit RDMA
  1. **Floating home-based** protocol
  2. Hybrid invalidation using **write notices & logical leases**
- Evaluation using NAS Parallel Benchmarks [Bailey et al. ’91]

Background: Release Consistency (1/3)

- **Release Consistency (RC)**
  - One of the major consistency models for DSM systems
  - Pros ☺: enables reordering of reads/writes, relatively easy to understand
- Requirements of release consistency (a) **program order** $\xrightarrow{po}$
  - Reads & writes should follow the order specified by the program

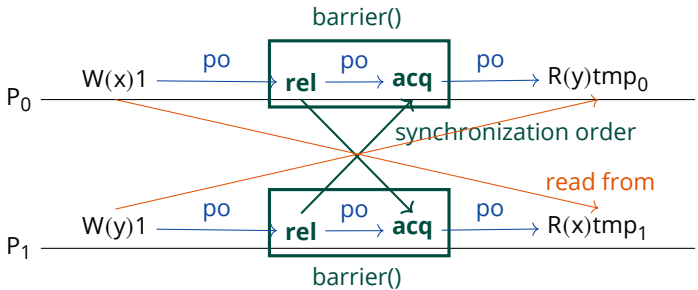


W(v)c = a write of the value c to the variable v

R(v)c = a read of v resulting in c

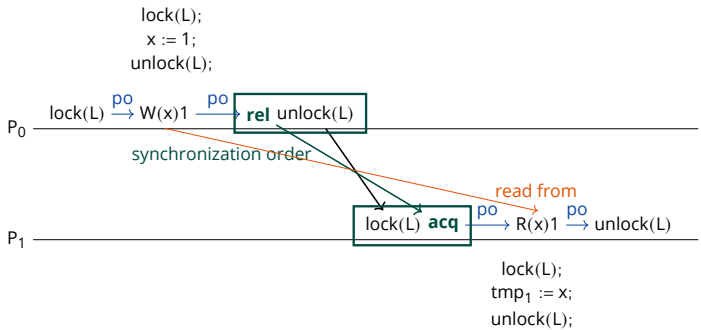
Background: Release Consistency (2/3)

- Requirements of release consistency (b)
  release-acquire synchronization order
  - Programmers need to specify synchronizations along with reads/writes



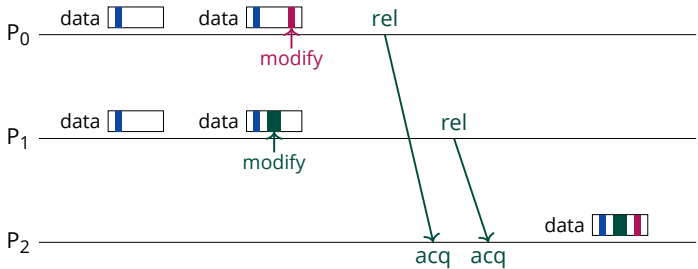
Background: Release Consistency (3/3)

- Why do we think of release-acquire synchronizations?
  - Because they can be generalized not only for barriers but also for mutexes
  - A release fence corresponds to applying writes,
    and an acquire fence corresponds to cache invalidation



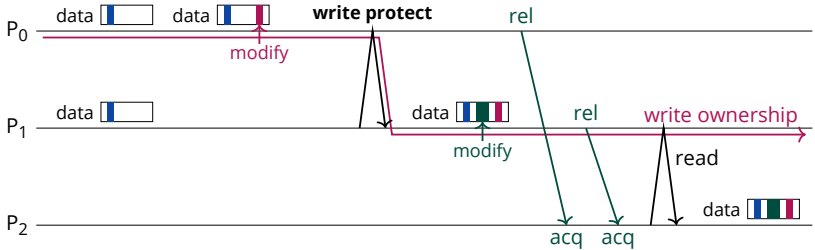
Background: False sharing in DSM systems

- Reordering is not accomplished only with relaxing consistency
  - Real shared-memory systems process caches as **blocks**
- **False sharing**
  - Multiple processes writing on the same cache block
  - Memory systems must preserve correct semantics



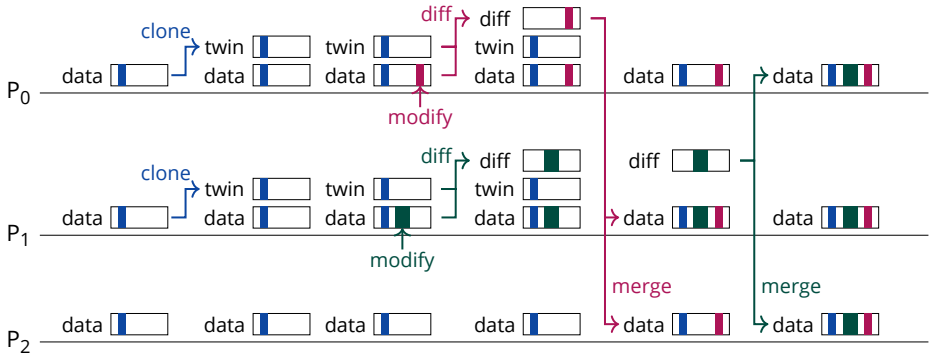
Background: Single-Writer DSM

- **Single-writer** protocols
  - Only a single writer process can write on the block at a time
- Impossible to implement Single-writer DSM purely with RDMA
  - Due to the system calls for protecting remote memory



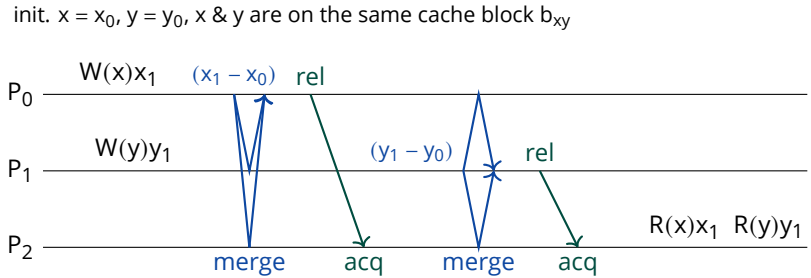
Background: Multiple-writer DSM

- **Multiple-writer** (MW) protocols [Carter et al. '91]
  - Generate diffs by comparing twins before & after writes
  - Allow multiple processes to concurrently write on the same cache block
  - Mitigate the performance degradation of false sharing



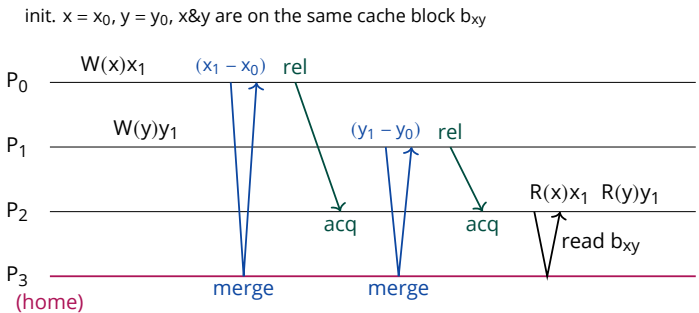
Background: Multiple-Writer + Release Consistency

- A naive example of implementing multiple-writer release-consistent DSM
  - Pros ☺: communications for writes can be delayed until the next fence
  - Cons ☹: diffs must be applied in all of the processes



Background: Home-based Multiple-Writer

- **Home-based** Multiple-Writer DSM [Zhou et al. '96]
  - Aggregate diffs to a home node
  - Pros ☺: only one application for each diff
  - Cons ☹: a mismatch between the home and writer increases the latency for merging



Background: Restrictions of RDMA programming

- Only two available types of RDMA operations:
  - One-sided writes/reads (RDMA READ/WRITE)
  - Atomic operations (RDMA compare-and-swap, fetch-and-add)

- Various restrictions of RDMA programming:
  1. **Unable to notify the remote nodes** (Except for RDMA WRITE with Immediate)
     - Hard to delegate computation to remote CPUs
     - **Decentralized** designs are desirable
  2. Need to place data in **contiguous buffers** for performance
     - Unable to scatter to/gather from remote buffers
  3. Necessary to register the memory before transfers
     - Registration may be slower than communications [Frey et al. '09]
  4. RDMA atomics are not synchronized with processor atomics

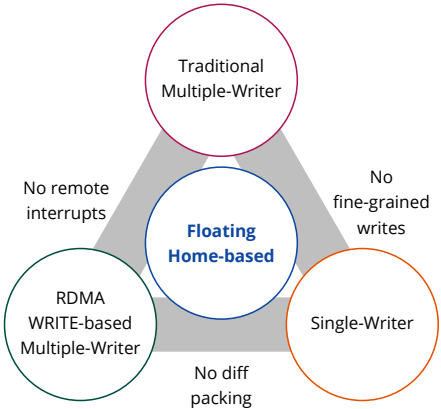
Preliminary Evaluation: RDMA-based DSM

- Three methods for merging diffs to a home node
  - **PackDiff**: Two-sided messaging for transfering packed diffs
  - **DiscontiguousWrite**: One-sided discontiguous RDMA WRITEs
  - **ContiguousWrite**: A single RDMA WRITE of the whole block (≠ diff merge)
- Assuming a 32 KiB cache block
- Microbenchmarking result of latency
  - ContiguousWrite < PackDiff < DiscontiguousWrite (lower is better)

	Latency w/ 50% changes
PackDiff	309 µs
DiscontiguousWrite	5042 µs
ContiguousWrite	4.5 µs

  - The overall latency is dominated by **software overhead** rather than communications

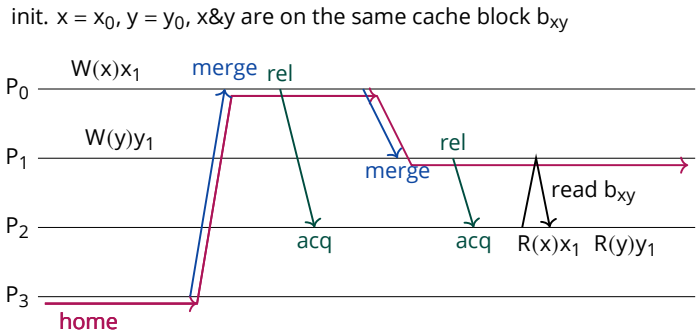
Problems of the existing RDMA-based DSM protocols



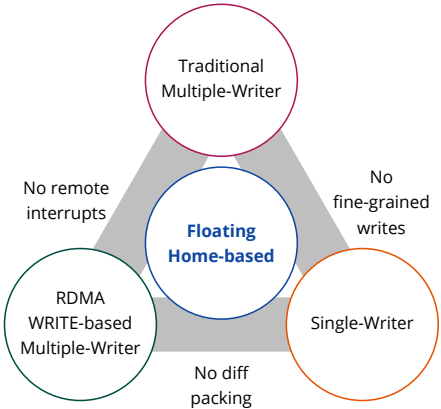
1. Traditional home-based MW (e.g., HLRC [Zhou et al. '96])
   - Software overhead due to packing/unpacking diffs
   - Does not concide with the zero-copy nature
2. RDMA WRITE-based MW (e.g., Argo [Kaxiras et al. '15])
   - Software overhead coming from many small RDMA WRITEs
   - RDMA does not transfer fine-grained messages efficiently
3. Single-Writer (e.g., MAGI [Hong et al. '19])
   - Messaging is needed to protecting writes at remote cores
   - Messaging inherently increases remote CPU overhead

Proposal: Floating home-based DSM (1/2)

- **Floating Home-based DSM**
  - Complete merging by home migration
  - Multiple-writer, but serializes merging in release fences
  - Transfer master versions of cache blocks, not diffs



Proposal: Floating home-based DSM (2/2)

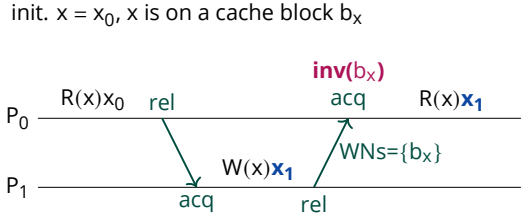


Why does our protocol solve the trilemma?

1. No diff packing
   - Only coarse-grained master versions are transferred
2. No fine-grained RDMA WRITEs
   - Home migrations can be implemented w/ coarse-grained **RDMA READ & ATOMICs**
3. No messaging
   - Multiple-writer protocols do not require remote interrupts for write protection

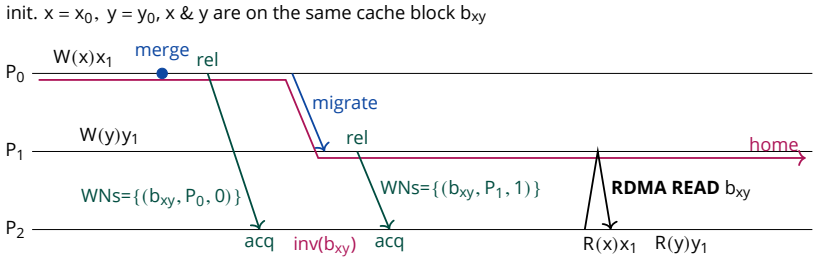
Proposal: Hybrid invalidation (1/4)

- Cache invalidation is orthogonal to processing writes
  - Mainly determines the read performance
  - Calculates which cache blocks must be invalidated at acquire fences
- Our basic idea: **write notices (WNs)** [Keleher et al. '94]
  - Synchronized operations piggybacks a set of the written block IDs
  - This set is gradually broadcast via synchronized operations



Proposal: Hybrid invalidation (2/4)

- Write notices enable us to implement our **"fast read"** method
  - Each write notice carries the process ID of the last releaser
  - In the best case, a read completes w/ a single RDMA READ from the last releaser in the synchronization order
    - ☺ No need to search for the current home node



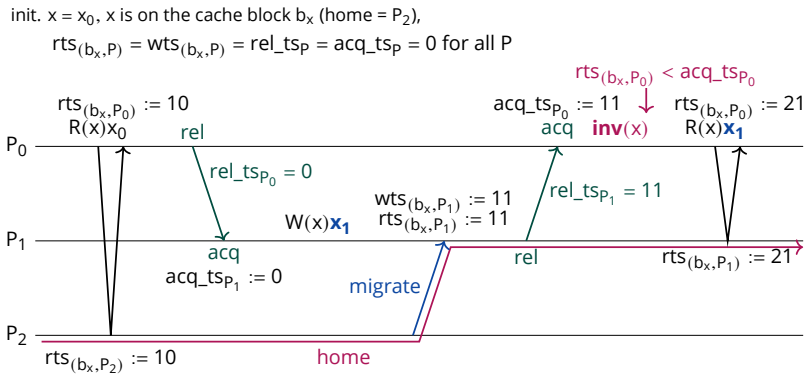
Proposal: Hybrid invalidation (3/4)

- ☺ Write notices enable to accelerate reads (“fast read”)
- ☹ Hard to discard write notices
  - Need to confirm that all of the processes processed the write notice
  - Traditional DSM systems implemented global garbage collection mechanisms (e.g., TreadMarks [Keleher et al. ’94])
    - One of the reasons complicating the design
    - RDMA is not capable of broadcasting

Decentralize the removal of write notices by **Logical leases** [Yu et al. ’15]

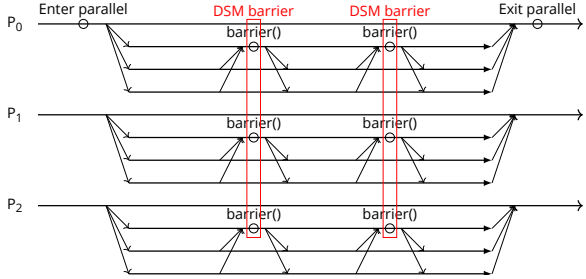
Proposal: Hybrid invalidation (4/4)

- **Logical leases** [Yu et al. '15]
  - Invalidate caches (or WNs) based on logical timestamps
  - Readers increases the read timestamp (= when a cache becomes stale) at the home node
  - Writers increases the write timestamp to invalidate old replicas
  - Both write notices and cache blocks can be discarded based on timestamps



Implementation

- MENPS includes a simple layer for running OpenMP programs
  - Implements the OpenMP ABI functions emitted by the compiler
  - **No code transformation or special compiler required**
    because call stacks are shared (i.e., everything-shared DSM [Costa et al. '06])
  - Minimal modifications to the application are needed
    (e.g., annotations to global variables, avoid using threadprivate)
- Running multiple threads in each process (hybrid parallelization)
  - In detail, both the system and application use user-level threads



Evaluation

- We employed NAS Parallel Benchmarks [Bailey et al. '91] for the evaluation
  - Unofficial version ported to C & OpenMP
  - NAS EP, CG, FT, IS, and BT are experimented
  - NAS LU, SP, and MG are excluded due to the implementation issues
- Compared systems against MENPS
  - Intel **OpenMP runtime**
  - **Argo DSM** [Kaxiras et al. '15]: an RDMA-based DSM system
    - w/ our wrapper library because Argo does not directly support OpenMP
  - **MPI**: the original MPI implementation (differing from the OpenMP version)

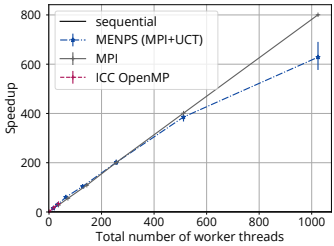
Evaluation environment (Reedbush-H)

CPU	Intel Xeon E5-2695 v4 2.1 GHz (max. 3.3 GHz with Turbo boost) 18 cores × 2 sockets / node
Memory	256GB / node
Interconnect	InfiniBand FDR 4x, 2 links
OS	Red Hat Enterprise Linux 7.2
Compiler	Intel C++ Compiler version 18.1.163
MPI	Intel MPI Library version 2018.1.163

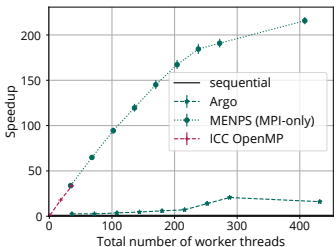
Reedbush[7]
(ITC at the Univ. of Tokyo)

[7] https://www.cc.u-tokyo.ac.jp/supercomputer/reedbush/service/

Evaluation: NAS EP (embarassingly parallel)



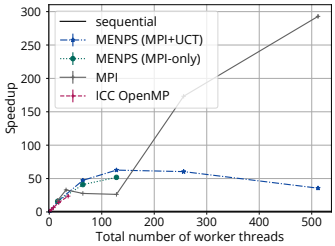
NAS EP (CLASS=D)



NAS EP (CLASS=C)

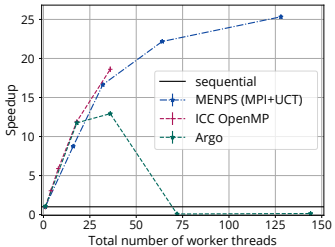
- CLASS means the problem size of the benchmarks
- The scalability results of CLASS=D
  - MENPS scaled with multiple nodes
    - No communications in the main computatin
  - The performance is slightly worse than MPI in 32 nodes
    - Initialization and finalization incur memory accesses to the shared area
- The scalability results of CLASS=C (smaller than CLASS=D)
  - Comparing the performance with Argo DSM [Kaxiras et al. '15]
    - We could not reproduce the good scalability of Argo in NAS EP
  - MENPS scales until 238 cores (= 7 nodes)
    - The speedup saturates due to the reduction phase

Evaluation: NAS CG (conjugate gradient)



NAS CG (CLASS=D)

- The scalability results of CLASS=D
  - MENPS performs better than ICC OpenMP
    - The maximum speedup was 63 times using 128 cores
  - MPI's result exhibits a different trend
    - It uses a different algorithm from the shared-memory version [Kwon et al. '12]



NAS CG (CLASS=C)

- The scalability results of CLASS=C (smaller than CLASS=D)
  - Comparison with Argo again
    - We could not reproduce the good results
  - Possible reasons of the reproduction failure
    1. Problems of our modified benchmarks
    2. Heavy use of Pthreads calls [Gracia '17]
    3. Other design-level issues (e.g., its coherence protocol)

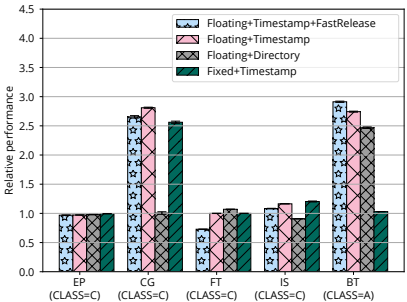
Evaluation: NAS FT, IS, BT

- MENPS could not accelerate other three benchmarks
  - Single-node ICC OpenMP performs better than multi-node MENPS
  - DSM inserts additional overhead in each fence and each read/write fault
  - One possible reason: too small problem sizes for multi-node experiments

Relative performance comparisons between MENPS and ICC OpenMP.
Only the best settings are listed.

	MENPS		ICC OpenMP	
	Speedup	# of threads	Speedup	# of threads
FT	6.80	16	17.55	36
IS	2.94	16	3.74	8
BT	0.996	16	8.63	36

Evaluation: Proposed prototol vs. baseline



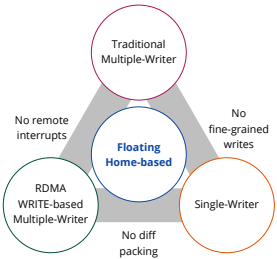
Relative performance improvement of different methods with 64 cores (two nodes) normalized to the results of Fixed+Directory.

- Comparing the proposed methods w/ the baseline ones
  - Fixed home-based (vs. Floating home-based)
    - Merging diffs to the fixed home node
    - Blocks are transferred in a coarse-grained manner (differing from DiscontiguousWrite)
  - Directory-based (vs. Timestamp-based)
    - Home nodes hold directories instead of timestamps
- **Floating+Timestamp** mostly performs the best
  - Timestamp-based method was important for CG
  - Floating home-based method was important for BT

Related Work: DSM

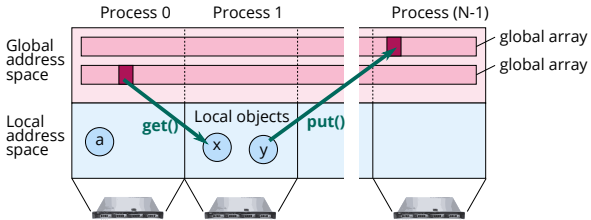
- Software DSM systems
  - The first software DSM system: Ivy [Li et al. '88]
  - Numerous examples of DSM systems in the 1990s:
    e.g., TreadMarks [Keleher et al. '94], JIAJIA [Hu et al. '98]
  - Similar idea to our floating home-based method:
    Moving Home-Based Lazy Release Consistency [Chung et al. '99]
    - Proactively migrates home nodes to accelerate writes

- RDMA-based DSM systems
  - PackDiff + RDMA: e.g., [Iosevich et al. '05]
  - DiscontiguousWrite: e.g., Argo [Kaxiras et al. '15]
  - Single-writer + RDMA: e.g., MAGI [Hong et al. '19]
  - **Home migration + RDMA**: MENPS



Related Work: PGAS

- **Partitioned Global Address Space (PGAS)**
  - **Global address space** + **local address space(s)**
    - Global address space is accessible by all of the nodes
    - Local address spaces are not shared



  - Pros ☺: good scaling results, better productivity than MPI
  - Cons ☹: requires changes to the shared-memory applications (no coherent caches)
  - Many systems being actively developed (e.g., UPC [El-Ghazawi et al. '02], Global Arrays [Nieplocha et al. '06], X10 [Charles et al. '05], Chapel [Chamberlain et al. '07], Co-array Fortran [Numrich et al. '98], XcalableMP [Lee et al. '10], UPC++ [Zheng et al. '14], HPX [Kaiser et al. '14], DASH [Schuchart et al. '18], OpenSHMEM [Chapman et al. '10])

Conclusions

- Developed a DSM library MENPS that exploits the performance of RDMA
  - Runs OpenMP programs in C/C++ w/ minimal modifications
- Proposed two protocol-level changes for MENPS:
  - **Floating home-based protocol** solves the trilemma of diff merging
  - **Hybrid invalidation** enables decentralized coherence
- Evaluated MENPS using NAS Parallel Benchmarks:
  - MENPS accelerated the OpenMP version of NAS EP & CG
  - MENPS performed better than an RDMA-based DSM system Argo

Acknowledgement

We would like to thank:

- Information Technology Center at the University of Tokyo
  - For providing the computing resources for experiments
- Takuya Fukuoka, the University of Tokyo
  - For commenting on the early versions of this paper
- Dr. Sriram Krishnamoorthy, PNNL
- Ilya Zhukov, Julich Supercomputing Centre
- The dissertation committee members of Wataru Endo
  - For valuable discussions