MENPS: A Decentralized Distributed Shared Memory Exploiting RDMA

1st Wataru Endo
Graduate School of Information Science and Technology
The University of Tokyo[1]
Tokyo, Japan

2nd Shigeyuki Sato
Graduate School of Information Science and Technology
The University of Tokyo
Tokyo, Japan
sato.shigeyuki@mi.u-tokyo.ac.jp

3rd Kenjiro Taura
Graduate School of Information Science and Technology
The University of Tokyo
Tokyo, Japan
tau@eidos.ic.i.u-tokyo.ac.jp

***Abstract*—The spread of RDMA-capable interconnects on supercomputers has enabled the middleware developers to explore new design options for runtime systems based on efficient communications. Observing low-latency networks and shared-memory infrastructure for multi-core processors, we have focused on extending shared-memory abstraction into multiple nodes exploiting RDMA, i.e., Distributed Shared Memory (DSM). We have found that the traditional protocols of DSM designed for two-sided communications cannot fully exploit the performance of RDMA, which necessitates decentralization and coarse-grained communications. To solve this problem, we introduced two methods for the DSM coherence protocol to exploit RDMA and implemented a DSM library MENPS using this protocol. Our evaluation shows that MENPS could accelerate two of five shared-memory applications with minimal modifications and beat an existing RDMA-based DSM runtime.**

I. Introduction

Most of the modern supercomputers are distributed-memory machines, which are typically programmed with message passing via MPI. MPI enables to implement portable and efficient parallel applications, while MPI programmers have suffered from its low application productivity over decades.

Shared memory, the other memory model, is more productive than MPI since it provides a global memory view closer to that of sequential programs. Distributed Shared Memory (DSM) is a form of shared memory implemented on top of distributed memory. The difficulty of DSM systems is their performance due to the inter-node latency.

Software DSM systems (e.g., [1]–[4]) were largely explored in the 1990s but are generally regarded as prototype implementations due to the poor scalability. Since the 2000s, the research community has switched to Partitioned Global Address Space (PGAS) which is a group of global memory interfaces without coherent caches, such as [5] and [6]. Because PGAS requires us to reduce global memory accesses manually for better performance, it largely offsets the productivity derived from global memory views.

In contrast, hardware DSM systems have widely spread as ccNUMA architectures. Typical supercomputers are composed of nodes equipped with ccNUMA or many-core processors. To exploit the excessive intra-node parallelism it provides, the middleware including the operating systems or compilers has also evolved for efficient multithreading. Following this success, we have started to consider the possibility of extending the current shared-memory infrastructure into multiple nodes with the power of a modern runtime system.

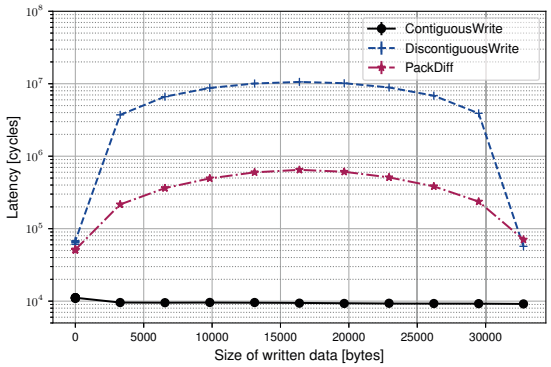


Fig. 1: Microbenchmark result of three methods for diff merging. The block size is set to 32 KiB. The benchmark randomly selects the written addresses on a per-byte basis.

One of the key advances of interconnection networks is the emergence of Remote Direct Memory Access (RDMA), which provides low-latency and high-bandwidth communications utilizing the kernel bypass. Currently, the RDMA latency is roughly 1 µs, only several times slower than the latency between sockets, which shines a light on creating the next level in the existing shared-memory hierarchy.

We note that RDMA does not automatically accelerate runtime systems because it has different interface limitations and performance characteristics [7] from conventional two-sided communications. To fully exploit the performance of RDMA, data should be copied between contiguous memory areas that are registered beforehand. Since remote CPU cores are not aware of one-sided transfers from other nodes, the initiator nodes should solely decide their behaviors. These properties require the runtime systems to change in their protocol level; decentralized protocols with coarse-grained data transfers should be established.

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

978-0-7381-1064-6/20/$31.00 ©2020 IEEE
DOI 10.1109/IPDRM51949.2020.00006

Authorized licensed use limited to: FUDAN UNIVERSITY. Downloaded on March 12,2021 at 08:14:16 UTC from IEEE Xplore. Restrictions apply.

Traditional DSM systems in the 1990s [3] heavily depended on two-sided communications. They relaxed the consistency models (e.g., data-race-free [8]) to delay coherence actions until fences and introduced multiple-writer coherence protocols that merge diffs to aggregate writes until fences and mitigate the performance degradation of multiple writes on the same cache block (i.e., false sharing). While diff merging to remote memory can compress messages and hence reduced the communication cost in the early days, our experiment on Fig. 1 in the current hardware shows that this method (PackDiff) has software overhead for gathering and scattering diffs.

To exploit the RDMA performance for remote diff merging, some of the existing RDMA-based DSM systems (e.g., [9], [10]) employed RDMA WRITE messages to solely put diffs to the home nodes (of the home-based approach [4]). In reality, however, our emulated result in Fig. 1 shows that their method (DiscontiguousWrite) highly degraded the write performance because RDMA basically requires contiguous memory transfers and each message has initiation overhead.

We have intended to propose a coherence protocol that aligns transfers contiguously as in ContiguousWrite of Fig. 1. Our *floating home-based* approach avoids costly remote diff merging by migrating a cache block to the last releaser node. Migrations can be efficiently implemented via RDMA READ and ATOMIC operations and enable local diff merging. This approach works efficiently in a single-writer workload and is more suitable for exploiting RDMA than single-writer RDMA-based DSM (e.g., [11]) because it can eliminate remote message handlers that prevent concurrent writes of another node conservatively thanks to the multiple-writer property.

As stated by the authors of Argo's paper [9], the coherence protocol for RDMA also needs changes in data sharing because most of the existing protocols have depended on cache directories, which require readers to register their references to the centralized data structure and prohibit decentralization.

When adopting release consistency, it is possible to piggyback writer invalidations with release-acquire synchronization, which was proposed as write notice invalidation in [3]. Our central idea is to extend the metadata of write notices for coherence decentralization. First, our approach introduces *fast read*, which combines the writer's information with an invalidation message and completes reading the memory block only with a single RDMA READ in the best-case scenario. Second, we incorporate logical leases [12] into write notices to eliminate broadcast-based garbage collection which does not coincide with RDMA's semantics. These methods realize directory-less cache coherence that self-invalidates cache blocks exploiting RDMA.

Based on these observations, we present the design and implementation of our DSM library MENPS (MENPS is Not a PGAS System). This paper has the following contributions:

- We have implemented an experimental RDMA-based DSM runtime system MENPS, which can transparently execute OpenMP programs on distributed-memory machines with minimal code modifications of the applications. MENPS does not require any special compiler or code transformation.
- We propose two novel methods for DSM to exploit the performance of RDMA. First, we introduce our floating home-based protocol to avoid remote diff merging. Second, we introduce our hybrid invalidation method that combines write notices and logical leases for coherence decentralization.
- We evaluated MENPS using five applications from NAS Parallel Benchmarks [13]. MENPS could accelerate two OpenMP programs using multiple nodes compared to a normal OpenMP runtime running on a single node. MENPS also performed better in two applications than a DSM library Argo [9], one of the existing RDMA-based DSM libraries based on data race freedom.

This paper presents a technical overview of MENPS that may overlap with Endo's dissertation [14]. The source code of MENPS is publicly available online [15].

II. Design of MENPS

A. Memory consistency model of MENPS

Memory consistency models define which value of a write is observed by a read of another core. It is commonly known that Sequential Consistency (SC) is the most strict memory consistency model for parallel computers. Because guaranteeing SC prohibits beneficial optimizations such as memory reordering, DSM designers have extensively tried to relax the consistency models to hide the high communication costs of distributed-memory systems.

MENPS is based on data race freedom (DRF) [8] of target applications because it is adopted in the consistency models of modern languages such as C++11 and Java. Those models guarantee the same behavior as of SC for data-race-free programs (i.e., SC-for-DRF). We specifically employ Release Consistency (RC) [16] for MENPS, which introduces *acquire* and *release* fences as synchronized operations distinguished from ordinary reads and writes. For example, when unlocking a mutex, which implicitly has the effect of the release fence, immediately precedes locking the same mutex by another processor with the effect of the acquire fence, this *release-acquire pair* forms the synchronization order between the processors. We also assume that synchronized operations are sequentially consistent (i.e., RC_{SC} [16]). Because unsynchronized operations are not always ordered by the program order and the synchronization order, in general, memory accesses are partially ordered (i.e., *happens-before* partial ordering).

B. Floating home-based method of MENPS

To maximize the application productivity, MENPS is designed as a page-based DSM runtime system, which provides a transparent address space by interrupting the application memory accesses using the page protection mechanism. Page-based DSM sets the memory area that is not cached to inaccessible so that the special segmentation handler can pull the data from remote memory in the background and then expose it using the system call (e.g., mprotect).

Authorized licensed use limited to: FUDAN UNIVERSITY. Downloaded on March 12,2021 at 08:14:16 UTC from IEEE Xplore. Restrictions apply.

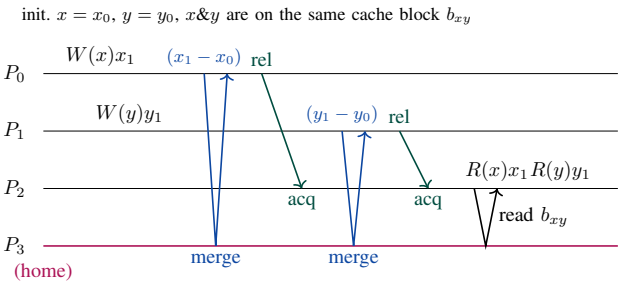


Fig. 2: A false sharing situation in home-based multiple-writer protocols. $\mathrm{W}(x)x_1$ is a write of x_1 to the variable x and $\mathrm{R}(x)x_1$ is a read from x resulting in x_1. Green edges represent happens-before relations between a release (rel) and an acquire (acq). Both the processes P_0 and P_1 are writing on two different variables (x, y) on the same block.

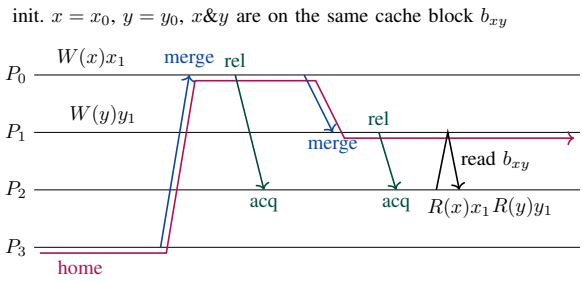


Fig. 3: Example of floating home-based DSM. When a release fence is issued, all of the memory blocks written in the process are merged. For the remotely owned blocks, they are migrated from the previous releasers first.

Most of the DSM systems handle the memory as a set of memory *blocks* to reduce metadata and utilize throughput. The page-based approach further restricts the block size to be a multiple of the page size. This cache granularity larger than that of multi-core processors increases the frequency of false sharing, meaning a situation that multiple cores write on distinct words sharing the same cache block.

To make the writes in false sharing coherent, hardware coherence employs a single-writer approach, which simply allows only a single core to write on the same block at a time. Because this approach is not suitable for page-based DSM systems due to their large cache granularity and high communication latency, DSM researchers have invented multiple-writer protocols instead, allowing multiple cores to write on the same block. Multiple-writer protocols use twinning [2] to resolve writes, which preserves a copy (= a *twin*) of the block before modifications and generates a *diff* by comparing the twin and the modified block. Multiple-writer protocols can reduce the latency of write operations because the system can delay communications for merging the diffs until the program issues a release fence next.

We employ a *home-based* multiple-writer protocol [4] because it needs only once to merge diffs to the *home* process for the block. Fig. 2 shows how home-based DSM protocols work. The variables x and y placed on the same block can be concurrently modified since their diffs are eventually merged into the block of the home process P_3. Although homeless protocols [3] can further delay diff merging until succeeding reads, we did not use them due to its computation and storage cost as mentioned in [4].

It is important for the performance of home-based DSM to properly decide the home processes. The simplest method is to statically assign the home processes, which may not match the actual writers and hence may degrade the write performance due to the increased diff transfers. To solve this problem, home migration techniques (e.g., [17]) have been proposed, which can dynamically decide the home processes based on the access patterns of the application.

When migrating the home process, it must be possible for every process to know the correct home for each block. To keep this metadata coherent, there is a method called *probable owner* [1], in which each process maintains a link to a possible home process at a certain time. Each process has a link for each block pointing to another possible owner and updates it only when the process notices. If the link is stale and not pointing to the latest home, it is still possible to find the latest home by traversing the distributed links. Because this method does not require any data structures fixed to a certain process, it can accomplish decentralized home migration.

We found that migration with probable owners can be efficiently implemented using RDMA ATOMIC operations. Besides, as shown in Fig. 1, we knew that it is costly to merge diffs on remote memory, which is required by the existing home-based protocols. From these two observations, our floating home-based approach is designed to aggressively migrate blocks at every release fence using RDMA ATOMIC and READ operations so that diff merging in release fences can be always processed locally. Fig. 3 shows the behavior of our approach. Instead of sending diffs to the home process as in typical home-based methods, our method migrates the home to the last releaser regarding the block and applies the diffs locally. If the application has proper temporal locality of writes, this method works best because the last releaser is supposed to write the block again minimizing communications.

The disadvantage of this proposed method is the necessity of serializing diff merging because it is almost impossible to simultaneously merge diffs from multiple processes while migrating. Migration can be implemented using a global mutex associated with each block. In false sharing cases, our approach may not perform better than other home-based approaches because they can merge diffs in parallel, but ours is still supposed to work efficiently in false sharing than single-writer approaches because it can delay actual communications for diff merging as in other multiple-writer approaches.

It is known that the home nodes in home-based DSM can skip twinning on fences because they can directly write on the blocks owned locally. We realized this acceleration as the *fast release* mode, which needs special care for migrating via

Authorized licensed use limited to: FUDAN UNIVERSITY. Downloaded on March 12,2021 at 08:14:16 UTC from IEEE Xplore. Restrictions apply.

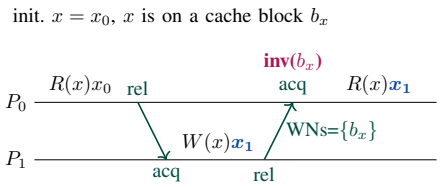


Fig. 4: Example of write notice invalidation. The cache block holding x is written by P_1 and then P_1 sends its invalidation to other processes as a set of cache blocks (in this case, $\{b_x\}$) during the synchronization.

RDMA from the previous owners that may be locally writing concurrently.

C. Cache invalidation method of MENPS

Once writes are properly handled in a shared-memory system, another problem is how to manage reads according to the consistency model. Invalidation-based coherence provides correct read results by invalidating old blocks on writes. The standard method for cache invalidation is directory-based coherence, in which the sharers of a block are tracked in a directory, but it has well-known problems including the increase of invalidation messages proportional to the sharer counts. To avoid this problem and exploit RDMA, we developed a hybrid approach for MENPS which unifies two orthogonal ideas of directory-less coherence: write notices and logical leases.

Write notice (WN) invalidation was introduced in a DSM system TreadMarks [3] based on RC. Fig. 4 shows the behavior of write notice invalidation. Because RC only guarantees to propagate the writes when they happen-before reads, it allows to aggregate invalidation messages as write notices, representing the writes (e.g., on the variable x in Fig. 4) of the preceding releaser processes in the synchronization order, and reduce the count of messages. This method, however, requires to discard old write notices while preserving coherence because it keeps consuming the storage during execution. TreadMarks implemented a global garbage collection method for write notices, which complicated its overall design.

To simplify the garbage collection of write notices, we focused on lease-based coherence called Tardis [12], which utilizes logical timestamps for coherence. The basic idea of this method is that readers register a logical timestamp representing when all of the cache copies expire for each block. Fig. 5 shows the behavior of Tardis. In Tardis, every replica of a block has a read timestamp (rts) and a write timestamp (wts). A reader adds the lease value to the read timestamp on the home process[1] to get the lease. When a new write happens, a writer increases the write timestamp to a greater value than the read timestamp. The writer can immediately proceed after the write because the writer can independently increment its logical timestamp. The lease-based method enables to decentralize coherence actions as

[1]Since Tardis was originally defined on multi-core coherence, the shared cache was instead used to hold the timestamps.

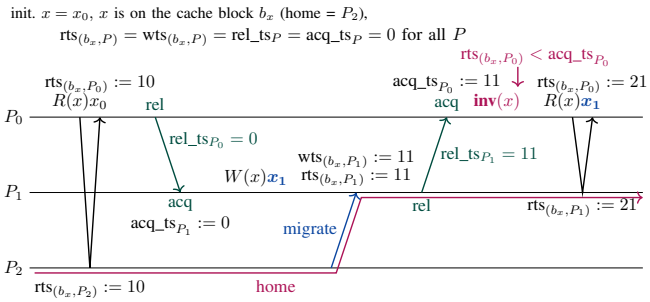


Fig. 5: Example of logical lease-based invalidation. The lease value is set to 10. P_1 only sends the logical timestamp value (rel_ts = 11) in the synchronization with P_0, and then P_0 invalidates x because the read timestamp (= 10) is smaller than acq_ts.

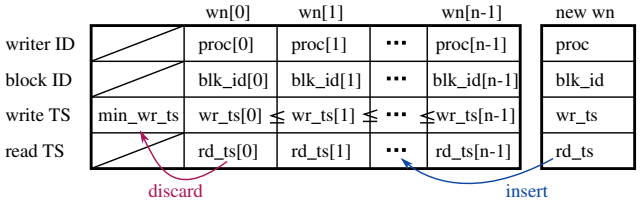


Fig. 6: Signature structure of MENPS. When a new write notice (new_wn) is inserted, the oldest wn (wn[0]) is removed while taking the maximum of the timestamps.

the writers can solely invalidate their blocks. However, it has a drawback of incurring over-invalidation and unnecessary cache misses because it cannot accomplish the block-wise invalidation based on the happens-before ordering, which cannot be fully captured by logical timestamps.

In our invalidation method, when a synchronization event occurs between two processes, the releaser process sends the metadata named a *signature* [18] to the acquirer process. The data structure for a signature is shown in Fig. 6. A signature is constructed of a minimum write timestamp and the list of a limited number of write notices with read and write timestamps. When a process merges a cache block in the release fence, a new write notice is inserted into the signature. The write notices with old write timestamps are discarded if necessary to limit the length of the write notice list. To maintain the coherence, the largest write timestamp of all of the discarded write notices is extracted and recorded as a minimum write timestamp. When the acquirer process receives a signature from the releaser, it invalidates blocks based on both the minimum write timestamp and the write notices. Because the minimum write timestamp represents all of the discarded write notices, our method can remove the write notices without breaking the consistency. It is reasonable to discard the oldest write notices first because of temporal locality.

Write notices accompanying logical leases enabled us to

Authorized licensed use limited to: FUDAN UNIVERSITY. Downloaded on March 12,2021 at 08:14:16 UTC from IEEE Xplore. Restrictions apply.

implement the fast read method, which can finish a read operation with a single RDMA READ in the best case. We noticed that if a cache block is invalidated with a write notice on a certain process and is read in the same process again, the updated data can be transferred from the releaser which produced the write notice rather than the latest releaser in physical time. This is because SC-for-DRF only guarantees that the preceding writes in the happens-before ordering are visible to the reads. Reading the data from the write notice sender can be efficiently implemented with an RDMA READ because it does not need to traverse the probable owner links to find the latest home process.

When the timestamp-based invalidation causes a read miss later, the reader process cannot determine which process wrote to the block at last in the happens-before order. In such a case, it enters a critical section as in the release fence in order to update the read timestamp value of the latest owner process.

III. Implementation of MENPS

MENPS is implemented in C++ and supports C/C++ applications with OpenMP. It also has experimental support for Fortran applications because our runtime design is independent of application languages. The launcher of MENPS starts as a normal MPI program and executes a single shared-memory application using multiple processes. We have developed a portable communication layer for MENPS [19], which can use UCT (from UCX [20]), a library for exploiting RDMA, in addition to MPI for inter-node communication.

MENPS implements and replaces the same OpenMP ABI functions as the compiler defines. We have implemented only important directives for our purposes. MENPS can share not only heap memory but also call stacks and global variables. To place on DSM, global variables need to be annotated for sharing and heap allocation should call the special functions.

A. Implementing memory fences and barriers

This paper provides an implementation method of barrier synchronization as it is one of the most frequently used directives of OpenMP. To additionally implement fine-grained synchronizations, our implementation decomposes the barrier into a release fence and an acquire fence. Both fences can be individually issued in each process.

A release fence applies the diffs of all of the writable blocks. When the release fence is issued, merge operations on all of the writable blocks within the process can be done in parallel. An acquire fence takes a signature as its argument to correctly invalidate the caches. This signature is transferred from the releaser process which synchronizes with the acquirer process.

A barrier operation first issues a release fence and then the signature returned by the fence is exchanged with all of the running processes. In the current implementation, MPI_Allgather() is used for this all-to-all communication. Because this part is not critical to the performance, we leave implementing the exchange of signatures purely with RDMA as future work. Finally, an acquire fence is issued to invalidate the caches based on the signatures.

TABLE I: Evaluation environment of MENPS [21].

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

IV. Evaluation

TABLE I shows the evaluation environment, which can run programs using up to 32 nodes. We used a user-level threading (ULT) library [22] for intra-node threading. We ran MENPS using two processes per node to avoid NUMA issues and each process executed 17 worker threads of ULT[2]. Each process also ran 16 OpenMP worker threads on ULT and one core was reserved for MPI. Therefore, we ran 32 OpenMP worker threads per node with MENPS in this setting. We also measured the performance in ICC OpenMP or Intel MPI using all of the cores (36 cores per node).

A. Application benchmark: NAS Parallel Benchmarks

To evaluate MENPS, we selected an unofficial version [23] of NAS Parallel Benchmarks (NPB) ported to C and OpenMP because most of the original benchmark programs are written in Fortran (only IS is written in C). For the evaluation in MPI, we used the original Fortran codes of NPB 3.3.1. We did not evaluate some applications including NAS LU, SP, and MG due to the implementation issues in MENPS or the ported application programs.

Because the applications of NPB depend on the OpenMP features that are not currently supported by MENPS (e.g., reductions, threadprivate variables), we replaced them with alternatives for the evaluation. This change was disabled during the evaluation in ICC OpenMP.

We also compared the performance of Argo [9] with MENPS using NAS EP and CG. Because the authors of Argo did not publish their benchmark programs[3], we needed to port the benchmarks by ourselves. In Argo, we used 36 worker threads for each process. We needed to modify the programs because Argo does not support OpenMP directives directly. We implemented a thin wrapper library that imitates OpenMP directives using Pthreads and converted every directive to a library function call using C++ lambda functions.

TABLE II shows the execution time of NPB without parallelization by disabling the feature of OpenMP. All of the error values in this paper represent 95% confidence intervals.

TABLE III shows the DSM parameters which we used for NPB. We adjusted these values to maximize the performance in multiple nodes. The heap size may be larger than the actual demand of the application because we set conservative values.

[2]We left one core for the operating system and other services.

[3]We asked them to provide the benchmarks, but they did not respond.

Authorized licensed use limited to: FUDAN UNIVERSITY. Downloaded on March 12,2021 at 08:14:16 UTC from IEEE Xplore. Restrictions apply.

TABLE II: Sequential execution time of NPB (the unofficial C+OpenMP version, OpenMP is disabled).

Name	CLASS	Time [s]
EP	D	4778.36 ± 1.06
	C	301.03 ± 1.02
CG	D	11619.82 ± 782.61
	C	365.13 ± 16.22
FT	C	357.55 ± 2.04
IS	C	12.30 ± 0.07
BT	A	88.06 ± 2.97

TABLE III: Parameters of MENPS for NPB.

		DSM heap		DSM global	Length of
Name	CLASS	Total size	Block size	Block size	a WN list
EP	D	1 MiB	32 KiB	32 KiB	1024
	C	16 MiB	32 KiB	32 KiB	1024
CG	D	32 GiB	256 KiB	4 KiB	4096
	C	8 GiB	32 KiB	32 KiB	128
FT	C	4 GiB	1 MiB	32 KiB	128
IS	C	4 GiB	1 MiB	32 KiB	1024
BT	A	1 MiB	32 KiB	128 KiB	1024

V. Evaluation Results

A. Scalability of NPB

The scalability results of NPB are shown in Fig. 7 and TABLE IV. All of the speedups were compared to the sequential results in TABLE II. First, it is visible that MENPS could scale NAS EP. Because EP initializes thread-private arrays before its main computation and includes a reduction phase after the computation, running on MENPS slightly degraded the performance with 32 processes compared with MPI. NAS CG is another benchmark program that MENPS could perform better in multiple nodes than the single-node OpenMP. The speedup benefit was saturated at around 63 times with 128 cores on MENPS. On the other hand, the performance improvement in MPI became flat with two to four nodes and then scaled better than other systems. There are many differences between ours and MPI's version including not only the memory interfaces but also the algorithms [24].

In NAS CG, we also compared the two conditions using MPI+UCT vs. MPI-only in MENPS. When UCT is enabled, RDMA operations are executed in UCT, and when disabled, they call MPI-3 RMA functions instead. Because UCT provides a low-overhead interface to the interconnects, the performance of MPI+UCT was 21% better than MPI-only with 128 cores (four nodes).

TABLE IV: Speedup comparisons between MENPS and ICC OpenMP in the benchmarks in which MENPS has lost. Only the settings of the best speedups 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

Although MENPS could successfully execute the other three benchmarks, there was no performance gain compared to single-node configurations. Several possible reasons prohibited our system from scaling these benchmarks. Because the problem sizes of these benchmarks especially in NAS IS are smaller than EP and CG, it is hard to get the performance benefit of multiple nodes. Our implementation could not run larger problem sizes in these benchmarks due to the size of the physical memory per node.

B. Comparing floating and timestamp-based methods with the baseline methods

Fig. 8 compares our proposed methods with the baseline methods. The floating home-based method is compared with the *fixed-home* mode, which statically assigns a home process cyclically based on a block ID. We note that our implementation of the fixed-home method performs differently from Argo's method because ours serializes the diff merge but Argo merges diffs with distinct RDMA WRITEs. We observed that the floating home-based method accelerated NAS BT by a factor of 2.47 times compared to the fixed-home one. We think that BT continuously repeats the writes to the same blocks on the same process across barriers and the home migration is indeed important in this case. It is also observed that the fast release mode further accelerated BT.

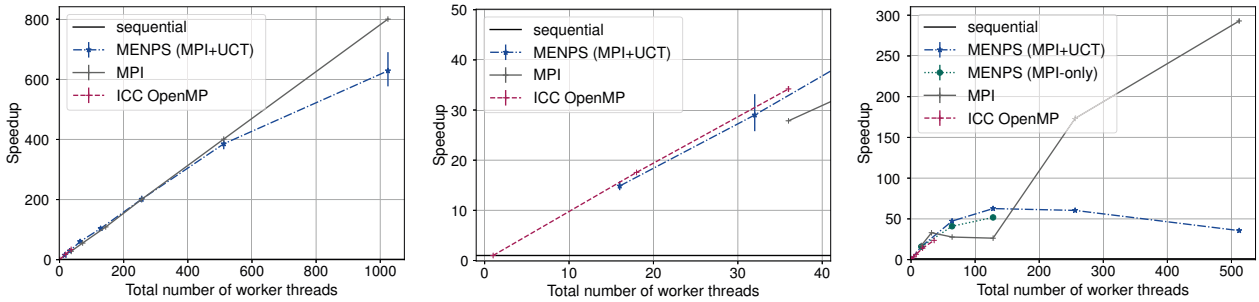
We compared our timestamp-based invalidation scheme with directory-based coherence on MENPS. The performance result of CG using timestamp-based invalidation was 2.56 times better than directory-based coherence. Our invalidation method works efficiently if there is enough space for holding the write notices in a signature and logical leases can roughly track the ordering of cache blocks. Otherwise, precise sharer tracking using directories may provide better performance, but none of these applications exhibited such a result.

Overall, the combination of the floating home-based method and the hybrid invalidation method was the most stable approach to run these benchmarks. It also achieves the best performance in CG, in which MENPS can surpass the single-node OpenMP runtime in the absolute performance.

C. Effect of the length of a write notice list

To evaluate our invalidation method, we examined the effect of various lengths of a write notice list in CG (CLASS=D) with two nodes. The benchmark was first run with 4096 WNs and took 246 seconds. When the list length was reduced to 128, the performance kept almost the same, however, when the length was set to 64, the benchmark took 563 seconds to complete. When the WN invalidation was disabled (i.e., where the length is zero), the execution time was further increased to 608 seconds. We have found that write notice invalidation could effectively improve the performance of lease-based invalidation because it can precisely invalidate blocks and the invalidated blocks can be later transferred via RDMA READ. We statically decided the length of WN lists and left its dynamic adjustment for future work.

Authorized licensed use limited to: FUDAN UNIVERSITY. Downloaded on March 12,2021 at 08:14:16 UTC from IEEE Xplore. Restrictions apply.



(a) NAS EP (CLASS=D) (b) NAS EP (CLASS=D) (zoomed) (c) NAS CG (CLASS=D)

Fig. 7: Speedup comparisons between MENPS, ICC OpenMP and MPI using NPB.

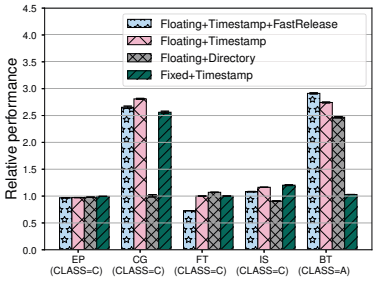


Fig. 8: Relative performance of different home-based methods and invalidation methods with 64 cores (two nodes) compared to Fixed+Directory.

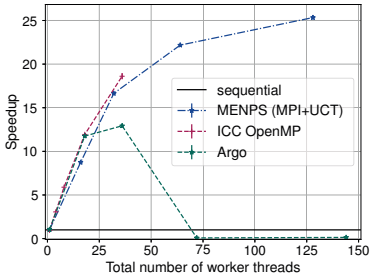


Fig. 9: Scalability comparison of NAS CG (CLASS=C) between MENPS, Argo, and ICC OpenMP.

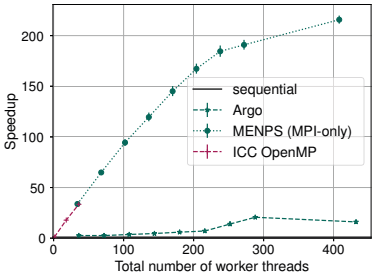


Fig. 10: Scalability comparison of NAS EP (CLASS=C) between MENPS, Argo, and ICC OpenMP.

D. MENPS vs. Argo

Fig. 9 compares the scalability of CG (CLASS=C) with Argo. Although Argo's paper reported a good scalability result with the same application and problem size, we observed that Argo was not scaling with multiple processes in our environment. On the other hand, MENPS performed better than ICC OpenMP with multiple processes. The performance benefit of MENPS was smaller than that of CG (CLASS=D) (Fig. 7) due to the insufficient problem size.

We also compared the performance of EP with Argo. Fig. 10 shows the scalability comparison of EP (CLASS=C) with Argo. MENPS scaled EP with less than 238 cores (= 7 nodes). However, with more nodes, it stopped scaling due to critical sections for reducing the results. Compared with the CLASS=D result (Fig. 7), this smaller problem finished within two seconds with 238 cores, which seems hard to scale. Although the performance of EP on Argo was improved with more nodes, the absolute speedup on Argo did not reach the best speedup using single-node OpenMP.

There are several differences between MENPS and Argo. MENPS utilizes user-level threading, but Argo heavily depends on the usage of Pthreads and incurs overhead in multi-threading [25]. Argo partially depends on directory-based coherence, which increases the number of coherence operations for each memory operation.

TABLE V: Changes in SLOC to NPB for two DSM systems.

Name	Original	Changes for MENPS	Changes for Argo
EP	269	+ 80, − 16	+497, − 90
CG	921	+141, − 33	+945, − 444
FT	1263	+166, − 24	
IS	710	+115, − 34	
BT	3713	+ 66, − 33	

E. Measuring Productivity

To examine the productivity of two DSM systems, we measured the changes made to the original programs in SLOC. TABLE V shows the changes to NPB to run on two DSM systems. Because MENPS supports the transparent execution of OpenMP programs, it required fewer lines of modifications on source code than Argo. Argo did not allow us to run OpenMP programs directly and required the translation with Argo's API. This translation came to be the main difference in the number of modification lines.

VI. Related Work

There have been numerous studies of DSM systems until the 1990s. They explored relaxing consistency (e.g., RC [16]) to achieve better performance in DSM systems. Most of DSM systems only achieved the limited scalability (roughly less than 100 cores). Traditional DSM systems could ignore

Authorized licensed use limited to: FUDAN UNIVERSITY. Downloaded on March 12,2021 at 08:14:16 UTC from IEEE Xplore. Restrictions apply.

the complexities of today's computing environments such as multi-core architectures and efficient interconnects.

The concept of Moving Home-based Lazy Release Consistency (MHLRC) [17] is close to that of our floating home-based approach because MHLRC proactively migrates the home nodes to reduce the diff transfer overhead. The main difference is that ours is carefully designed to match the RDMA's semantics.

Argo [9] is one of the RDMA-based DSM systems, which has inspired the design of MENPS. Argo's invalidation method still depends on centralized directory structures and their runtime is implemented on MPI (particularly on MPI-3 RMA), which does not always map to the actual RDMA operations. MENPS is based on the decentralized schemes and its communication is surely mapped to a single RDMA operation.

Early PGAS systems (e.g, [5], [6]) appeared in the 2000s, and after that, the research on global address space systems started to be dominated by PGAS instead of DSM. PGAS is not as productive as DSM because it does not support transparent caching and require the applications to invoke explicit communication calls.

To solve the problems of directory-based coherence, there are many proposals to accomplish directory-less coherence. One of the examples is *Self-invalidation* [26], in which readers spontaneously invalidate caches rather than waiting for the writer-initiated invalidations. The idea of our cache invalidation method incorporates Tardis [12], which invalidates caches with logical timestamps. The difference from Tardis is that our method effectively combines write notices.

VII. Conclusions

To improve the productivity of distributed-memory programming, we developed a DSM runtime system MENPS that can exploit RDMA. Two proposed methods for MENPS, the floating home-based approach and the hybrid invalidation method, solve the performance issues of traditional coherence protocols when combined with RDMA. MENPS can transparently execute shared-memory applications with simple OpenMP directives and achieve good application productivity. In the evaluation using NAS Parallel Benchmarks, MENPS does not always perform better than a normal OpenMP runtime system even with multiple nodes but can scale NAS EP and CG with almost the same program. The experimental result also showed that MENPS was more efficient than the existing RDMA-based DSM library Argo.

Acknowledgment

We would like to thank the Information Technology Center at the University of Tokyo for providing us with computing resources. This work was supported by JSPS KAKENHI Grant Number 19J14231.

References

[1] K. Li and P. Hudak, "Memory Coherence in Shared Virtual Memory Systems," *ACM Transactions on Computer Systems*, vol. 7, no. 4, pp. 321–359, 1989.

[2] J. B. Carter, J. K. Bennett, and W. Zwaenepoel, "Implementation and Performance of Munin," in *SOSP '91*, vol. 25, 1991, pp. 152–164.

[3] P. Keleher, A. Cox, and W. Zwaenepoel, "Lazy Release Consistency for Software Distributed Shared Memory," in *ISCA '92*, 1992, pp. 13–21.

[4] Y. Zhou, L. Iftode, and K. Li, "Performance Evaluation of Two Home-Based Lazy Release Consistency Protocols for Shared Virtual Memory Systems," in *OSDI '96*, 1996, pp. 75–88.

[5] T. El-Ghazawi and F. Cantonnet, "UPC Performance and Potential: A NPB Experimental Study," in *SC '02*, 2002, p. 26.

[6] J. Nieplocha, B. Palmer, V. Tipparaju, M. Krishnan, H. Trease, and E. Aprà, "Advances, Applications and Performance of the Global Arrays Shared Memory Programming Toolkit," *The International Journal of High Performance Computing Applications*, vol. 20, no. 2, pp. 203–231, 2006.

[7] A. Kalia, M. Kaminsky, and D. G. Andersen, "Design Guidelines for High Performance RDMA Systems," *USENIX ATC '16*, pp. 437–450, 2016.

[8] S. V. Adve and M. D. Hill, "Weak ordering—a new definition," in *ISCA '90*, vol. 18, 1990, pp. 2–14.

[9] S. Kaxiras, D. Klaftenegger, M. Norgren, A. Ros, and K. Sagonas, "Turning Centralized Coherence and Distributed Critical-Section Execution on their Head: A New Approach for Scalable Distributed Shared Memory," in *HPDC '15*, 2015, pp. 3–14.

[10] R. Noronha and D. Panda, "Designing High Performance DSM Systems using InfiniBand Features," in *CCGrid 2004*, 2004, pp. 467–474.

[11] Y. Hong, Y. Zheng, F. Yang, B. Y. Zang, H. B. Guan, and H. B. Chen, "Scaling out NUMA-Aware Applications with RDMA-Based Distributed Shared Memory," *Journal of Computer Science and Technology*, vol. 34, no. 1, pp. 94–112, 2019.

[12] X. Yu and S. Devadas, "Tardis: Time Traveling Coherence Algorithm for Distributed Shared Memory," in *PACT '15*, 2015, pp. 227–240.

[13] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga, "The NAS Parallel Benchmarks," *The International Journal of High Performance Computing Applications*, vol. 5, no. 3, pp. 63–73, 1991.

[14] W. Endo, "A Decentralized Implementation of Software Distributed Shared Memory," Ph.D. dissertation, The University of Tokyo, 2020.

[15] https://github.com/endowataru/menps.

[16] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy, "Memory consistency and event ordering in scalable shared-memory multiprocessors," in *ISCA '90*, 1990, pp. 15–26.

[17] J. W. Chung, B. H. Seong, K. H. Park, and D. Park, "Moving Home-Based Lazy Release Consistency for Shared Virtual Memory Systems," in *ICPP '99*, 1999, pp. 282–290.

[18] Y. Yao, W. Chen, T. Mitra, and Y. Xiang, "TC-Release++: An Efficient Timestamp-Based Coherence Protocol for Many-Core Architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 11, pp. 3313–3327, 2017.

[19] W. Endo and K. Taura, "Parallelized Software Offloading of Low-Level Communication with User-Level Threads," in *HPC Asia 2018*, 2018, pp. 289–298.

[20] P. Shamis, M. G. Venkata, M. G. Lopez, M. B. Baker, O. Hernandez, Y. Itigin, M. Dubman, G. Shainer, R. L. Graham, L. Liss, Y. Shahar, S. Potluri, D. Rossetti, D. Becker, D. Poole, C. Lamb, S. Kumar, C. Stunkel, G. Bosilca, and A. Bouteiller, "UCX: An Open Source Framework for HPC Network APIs and Beyond," in *HOTI '15*, 2015, pp. 40–43.

[21] Supercomputing Division, Information Technology Center, The University of Tokyo, "Introduction to the Reedbush Supercomputer System," https://www.cc.u-tokyo.ac.jp/en/supercomputer/reedbush/system.php.

[22] J. Nakashima and K. Taura, "MassiveThreads: A Thread Library for High Productivity Languages," in *Concurrent Objects and Beyond*, 2014, vol. 8665, pp. 222–238.

[23] University of Versailles Saint Quentin en Yvlines, "NAS-C-OpenMP3.0," http://benchmark-subsetting.github.io/cNPB.

[24] O. Kwon, F. Jubair, R. Eigenmann, and S. Midkiff, "A Hybrid Approach of OpenMP for Clusters," in *PPoPP '12*, 2012, pp. 75–84.

[25] J. Gracia, "D5.3 - Second-year Report on Proof-of-Concept Activities Version 1.0," POP Consortium Partners, Tech. Rep., 2017.

[26] A. R. Lebeck and D. A. Wood, "Dynamic self-invalidation: reducing coherence overhead in shared-memory multiprocessors," in *ISCA '95*, 1995, pp. 48–59.

Authorized licensed use limited to: FUDAN UNIVERSITY. Downloaded on March 12,2021 at 08:14:16 UTC from IEEE Xplore. Restrictions apply.