Scalable RMA-based Communication Library Featuring Node-local NVMs

Ryo Matsumiya*†, Toshio Endo*†
*Tokyo Institute of Technology
†AIST-TokyoTech Real World Big-Data Computation Open Innovation Laboratory,
National Institute of Advanced Industrial Science and Technology
E-mail: matsumiya.r.aa@m.titech.ac.jp, endo@is.titech.ac.jp

***Abstract*—Remote Memory Access (RMA) is a useful communication interface to develop high-performance applications with complicated communication patterns. However, the data scales of such applications are still limited by the totally available main memory capacity.**

To accommodate extreme scale executions of those applications, we developed vGASNet, which is an RMA-based communication library that exploits the capacity of non-volatile memory (NVM) on each node. With vGASNet, NVM devices on nodes compose a large shared address space. Under this model, the key for good application performance is to reduce bandwidth bottlenecks. First, since NVM is much slower than DRAM, reducing the amounts of NVM accesses is important. For this purpose, vGASNet regards DRAM of each computation node as a cache of NVM. Next, one of bottleneck sources in RMA is caused by access contention. In order to mitigate its effects, vGASNet adopts cooperative cache mechanism, which make multiple caches of an object on several nodes. Our evaluation using vGASNet shows the above cache mechanism improves the scalability of RMA.

***Index Terms*—Non-volatile memory, Remote memory access, Cache, Cooperative caching, GASNet**

I. Introduction

Programming models based on Remote Memory Access (RMA) communication are known to ease to develop high-performance big-data applications including de novo genome assembly [1] and large-scale matrix manipulations [2]. Their advantages come from the model where distributed memories on multiple compute nodes are virtually integrated into a global memory pool, which are accessible from all nodes. However, the data scales of such applications are still limited by the totally available main memory capacity. To exceed this limitation, we focus on usage of non-volatile memory (NVM) including Flash SSD, which are widespreading toward the post-Moore era. Several supercomputers such as TSUBAME [3], [4] and Beacon [5] are equipped with node-local NVMs into computing nodes. To utilize their capacity, many researchers have developed memory management or communication libraries for NVMs [6]–[15]. However, few works have focused on RMA functionalities.

This paper describes design and implementation of a novel RMA-based communication library, named vGASNet. The basic idea of vGASNet is to compose a single memory pool from multiple node-local NVMs across all application processes, called ranks. vGASNet is designed to have similar interface to that of GASNet, which is a commonly used RMA communication library [16]. While this model achieves extremely large global memory pool, we need to mitigate performance overhead caused by heavier access costs of NVM than those of DRAM since NVMs tend to have less access bandwidth and larger latency. Towards this objective, vGASNet regards DRAM (main memory) on compute nodes as cache pools.

While vGASNet can reduce NVM access with DRAM cache, there is still performance bottleneck especially on large scale supercomputers. When many ranks are going to fetch the same data object, a naive way would be that the owner rank sends the data to all requester nodes. However, this is not scalable for two reasons; (1) the avaialble cache (main memory in our context) capacity is limited when only owner node caches the data. (2) communication congestion occurs on the owner node. With regard to (2), several PGAS systems such as XcalableMP [17] provides group communication interface to programmers, in order to utilize optimized communication algorithms. However, this approach tends to compromise on easiness of programming complicated communication patterns.

From the above discussion. the cache mechanism of vGASNet adopts *cooperative-caching* approach [18], which enables a rank to utilize the caches of other ranks. This is originally proposed in the context of file systems, and have shown to improve scalability of several distributed file systems. For better scalability, this approach may create multiple caches of a single object among multiple nodes. Since our focus is RMA based parallel programming, not file systems, we need to be more rigid in maintaining data coherency between multiple caches. We use a cache coherence protocol named MOESI-F protocol, a combination of MOESI and MESIF, inspired by protocol on multicore processors.

We conducted performance evaluation of vGASNet with our cache mechanism on 42-node cluster equipped with node local SSDs. As vGASNet has similar interface to GASNet, it can be easily integrated with high level partitioned global address space systems. Our evaluation includes evaluation of benchmark written in UPC++ [19]. Through the evaluation, we show that vGASNet achieves scalable performance.

Our main contributions are as follows:

- Designing and implementing vGASNet, a RMA-based communication library harnessing memory hierarchy towards extreme scale applications.

- Proposing a scalable cache mechanism.
- Integration of vGASNet and UPC++ PGAS runtime and performance evaluation.

II. GASNET

A. GASNet Overview

As we described above, vGASNet interface is based on GASNet, an existing RMA-based communication library. GASNet is used by various parallel programming systems such as UPC [20], UPC++ [19], and Legion [21]. GASNet provides high performance RMA communication by harnessing underlying APIs such as Verbs for InfiniBand, PSM for Omni-Path, or uGNI for Cray Aries/Gemini systems.

A program on top of GASNet consists of distributed several application processes, called ranks hereafter [1]. Each rank has a special region, called remote access segment hereafter. A remote access segment is accessible from other ranks via GASNet's APIs.

GASNet provides two kinds of APIs, Core API and Extended API. Core API includes basic functionalities such as initialization of GASNet library itself, allocation of remote access segments and functions related to active messages [22], which is the main communication protocol of Core API. The Extended API functions provide RMA and have a reference implementation using active messages. With high-performance network such as InfiniBand, Extended API functions utilize RMA functionalities on hardware primitives.

B. Core API

Each rank calls `gasnet_init()` to initialize GASNet, and then `gasnet_attach()`. The latter function has two meanings, (1) to register handlers for active messages (AM), and (2) to initialize its remote access segment. The size of remote access segment should be smaller than the available main memory capacity.

After that each rank can communicate with each other by using AM. When a rank sends an AM with a handler kind to another rank, the destination rank executes the corresponding handler function, which have been registered by `gasnet_attach()`. An AM is sent by Core API functions such as `gasnet_AMRequestMedium0()`.

A rank can obtain base pointers and sizes of remote access segments of all ranks by calling `gasnet_getSegmentInfo()`.

C. Extended API

Extended API provides high-level operations including put/get RMA functions, which enables access to remote access segments owned by other ranks. `gasnet_put()` is used to copy local data into remote access segment of another rank. Contrarily, `gasnet_get()` copies data on remote access segment of another rank to local memory. Each function is given a pointer on the local memory, the rank accessed, a pointer on the remote memory, and the size of the data.

[1]In the GASNet manual, the term "node" is used. However, this paper uses "rank" to distinguish processes and physical compute nodes.

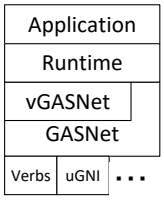

Fig. 1. Software stack of vGASNet

The followings are properties about memory access on GASNet.

- Data outside a remote access segment is not accessible from other ranks.
- Each node can access freely both to its own data inside the remote access segment and outside the remote access segment locally in the default configuration.

III. VGASNET

A. Overview

vGASNet is a RMA-based library based on GASNet and the software stack is shown as fig. 1. vGASNet provides GASNet-like interface functions such as `vgasnet_put()` and `vgasnet_get()` to the upper layer, which is typically a PGAS runtime such as UPC++. vGASNet utilizes the underlying original GASNet library in order to harness high-performance RMA facilities, while some GASNet functions are directly used by the upper layer.

On vGASNet, remote access segment of each node is allocated on the NVM of the node, which may be larger than DRAM capacity. Currently, the segment is implemented as a file on the NVM. The segment can be accessed by the other nodes using vGASNet functions.

vGASNet partitions DRAM on each node into three parts, local memory region, page cache pool and communication buffer as in figures 2 to 5. The page cache pool ("cache" in figures), which consists of pages with a fixed size, contains partial copy of NVMs. The cache pool is maintained by vGASNet internally and not accessed by the application explicitly.

Each node has a table in order to maintain the rank of the *owner node* of each cache page. The owner node of a cache page is a node that owns the original page on its NVM. Each page has its own reader-writer lock, which allows concurrent access for get operations, while put operations require exclusive access. The cache consistent policy of vGASNet is relaxed consistency; all caches are synchronized when memory barrier functions such as `vgasnet_barrier_wait()` are called.

The current implementation does not guarantee thread-safety, while the original GASNet does. We plan to solve it in the future.

B. Get Operation

The overview of get operations such as `vgasnet_get()` is drawn as fig. 2 and fig. 3. Here *Rank A* is being to copy from the data D (gray box) to the local (i.e. not shared among the ranks) memory region L. We assume D is included by a

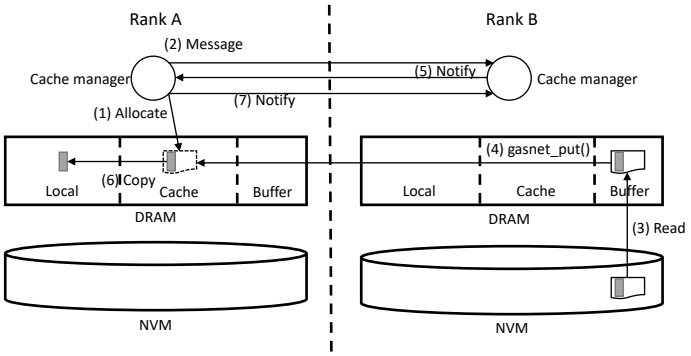

Fig. 2. Basic get operation of vGASNet

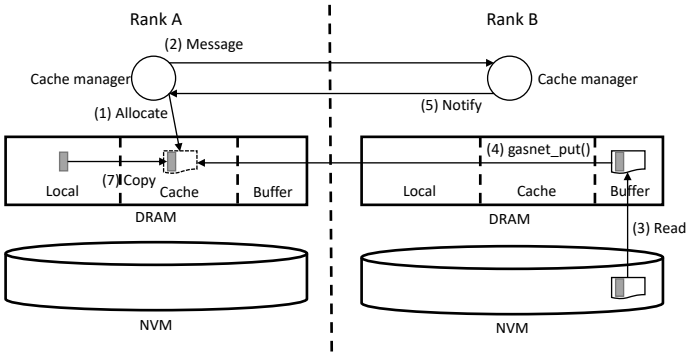

Fig. 4. Basic put operation of vGASNet

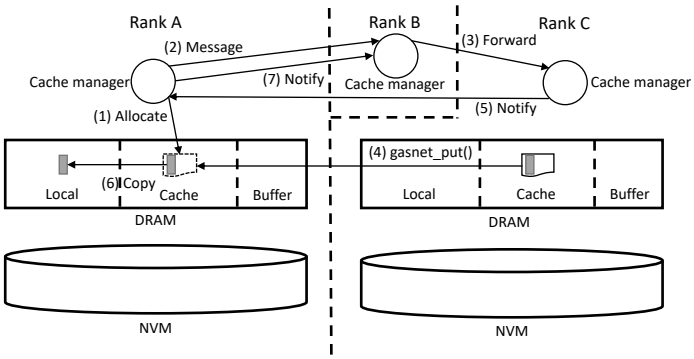

Fig. 3. Get operation of vGASNet with cooperative-caching

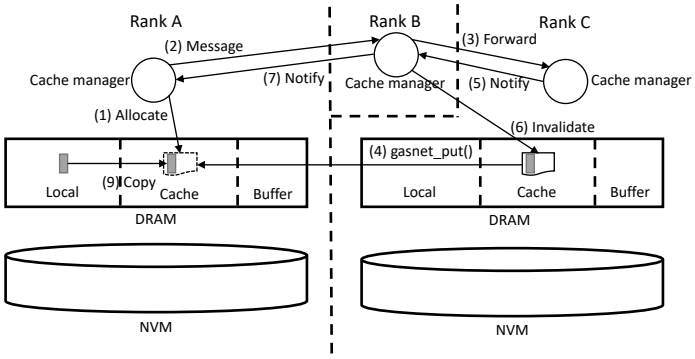

Fig. 5. Put operation of vGASNet with cooperative-caching

page P (white boxes contain D), which is stored in the NVM of its owner, *Rank B*. The figures also show a *cache manager* of each rank, which is a module that communicates with other ranks. The all messages exchanged among cache managers are passed using `gasnet_AMRequestMedium0()`.

When rank A invokes a get operation, it firstly checks whether P is cached in its local cache pool. If it is, rank A simply copies it to L locally. Otherwise, the get operation works as follows: (1) Rank A allocates a page for P in its cache pool. This may cause eviction of an old cache page. (2) Rank A sends Rank B a request message to get P.

Next, Rank B checks any other ranks has the cache of P. To achieve this, each rank maintains a list of nodes that have cache for each page on NVM. If Rank B finds it is not cached by others, the sequence continues as in fig. 2: (3) Rank B reads P from its NVM to the communication buffer. (4) Rank B puts the buffer to the cache pool of rank A with `gasnet_put()`. (5) Rank B notifies rank A that the request has been completed. (6) Rank A copies from D as a part of the cache of P to L. (7) Rank A notifies Rank B that Rank A now has a cache of P.

If P is cached by *Rank C*, we can harness cooperative-caching. The following operations occur as in fig. 3. (3) Rank B forwards the request to Rank C. (4) Rank C puts the content of P to the cache pool of Rank A with `gasnet_put()`. (5) Rank C notifies Rank A that the request has been completed. Steps (6) and (7) are same as above.

C. Put Operation

The overview of put operations such as `vgasnet_put()` is drawn as figures 4 and 5. *Rank A* is being to copy the data L in its local access segment into D (gray box). D is included by a page P, whose owner is *Rank B*.

Unlike a get operation, each put operation invokes communication to the owner regardless the status of local cache for cache coherency. A put operation starts with: (1) Rank A allocates a page for P in its cache pool if the cache does not exist. (2) Rank A sends Rank B a message to get P.

Here Rank B checks if P is cached by other ranks. If not, the following operations are done as in fig. 4: (3) Rank B reads P from its NVM to its communication buffer. (4) Rank B puts the buffer to the cache pool of rank A with `gasnet_put()`. (5) Rank B notifies rank A that the request has been completed. (6) Rank B records that rank A has the cache of P. (7) Rank A stores the content of L into D as the cache of P.

If Rank B sees P is cached by others (it may be cached by A), the following operations are done as in fig. 5: (3) The owner forwards the request to rank C. (4) Rank C puts the content of P to the cache pool of Rank A with `gasnet_put()`. (5) Rank C notifies Rank B that the request has been completed. (6) Rank B requests all ranks that have cache of P (except Rank A) to invalidate them. Steps (7), (8), (9) are the same as (5), (6), (7) in the former sequence.

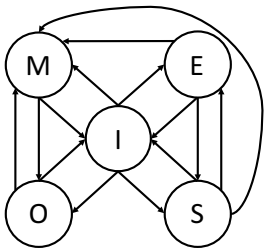

Fig. 6. State transition of the MOESI-F protocol

D. Local Memory Access

With the original GASNet, each rank can have access not only to its local access segment but to its own remote access segment. This is possible because the remote access segment is allocated on DRAM. On the other hand, this causes an issue on vGASNet, which allocates the remote access segment on the node-local NVMs. We noticed that several applications use this facility, thus we support it on vGASNet as follows.

Our current implementation is based on signal handling of `SIGSEGV`. In vGASNet initialization, each rank allocates a virtual memory address space for remote access segment using `mmap()` specifying `MAP_ANONYMOUS` and `PROT_NONE`.

When rank A tries to access the segment first, a `SIGSEGV` is invoked. In the `SIGSEGV` handler, `si_addr` (virtual memory address at fault) is checked. If the address is out of the remote access segment, the program is simply terminated. Otherwise, the handler makes the page including `si_addr` accessible as follows. (1) Using `mprotect()`, the handler makes the virtual memory space of the page accessible. (2) If no other rank has caches of the page, the handler loads the page from its own NVM. Otherwise, the handler requests one of the ranks that has the cache to send it. (3) For memory consistency, all remote caches of the page are invalidated, since the virtual memory space can be dirty and this page is regards as "Modified" in rank A. (4) The handler records that the target page is cached at the address `si_addr` itself. This mechanism allows the virtual memory space can be used as a cache.

Afterward, rank A may receive invalidate requests for the cache page. In these cases, it writes the content of the virtual memory space to the NVM. Then, the virtual memory is protected with `mprotect()` again.

E. MOESI-F Protocol

MOESI-F protocol is the cache coherence protocol used in vGASNet. The protocol is inspired by two practical protocols of multicore processors. One protocol is MOESI protocol, which is implemented in AMD multicore processors [23]. MOESI protocol allows the dirty cache not to be written-back to the main memory when false-sharing access is caused. The other protocol is MESI-F protocol, which is implemented in Intel multicore processors [24]. MESI-F protocol assigns each shared cache to a node which has the cache. Only assigned node can transfer the cache to another node.

Shown as fig. 6, under MOESI-F protocol, each page transits among five states.

Modified is a state which means the cache is dirty and that any other nodes do not have the cache pointing to the same cache line. When the cache is evicted, it is written back to the NVM stored the original data. If a modified cache is transferred to another node, the cache is changed to owned.

Owned is a state which means the cache is dirty and shared among multiple ranks. When the cache is evicted, it is not written back to NVM. Once a owned cache is written, the cache is changed to modified.

Exclusive is a state which means the cache is clear and that any other nodes does not have the cache pointing to the same cache line. When the cache is evicted, it is not written back to the NVM. If an exclusive cache is transferred to another node, the cache is changed to shared. Once a exclusive cache is written, the cache is changed to modified.

Shared is a state which means the cache is clear and shared among multiple nodes. When the cache is evicted, it is not written back to the NVM. Once a shared cache is written, the cache is changed to modified.

Invalidate is a state which means the cache is invalidated. This state is the initial state of the caches under the MOESI-F protocol.

Moreover, the owner assigns each cache line with a node which has the cache. The assigned cache is called **forward** in addition to the other five states. Modified caches and exclusive caches must be forward because these are only caches pointing to their cache lines. Owned caches and shared caches can be forward. The forward cache of a cache line is selected using a stack-based array. The stack-based array consists of three operations, push, access, remove. When an element is pushed, the element is put on the top of the array. An access operation returns the value of the top element. Then, the top element is re-pushed the bottom of the array. A remove operation erases the specified element from the array. Each cache line is assigned to one of the stack-based array.

F. Cache Replacement Policy

The cache replacement policy of vGASNet is similar to Least Recently Used (LRU). Although pure LRU uses only one LRU queue, our policy additionally uses one FIFO queue. Every cache is firstly enqueued into the LRU queue. When cache pool is filled, vGASNet dequeues a cache from the bottom of the LRU queue. In case that the cache is exclusive or modified, the cache is enqueued to the FIFO queue, and vGASNet tries to evict the next bottom cache of the LRU queue. When the FIFO-based queue is larger than half of the cache pool, the cache at the bottom of the FIFO-based queue is evicted. An element of the FIFO queue is erased and enqueued to the LRU queue when and only when the assigned cache line is accessed.

IV. Performance Evaluation

A. Evaluation Setup

We evaluated vGASNet performance on TSUBAME-KFC/DL, a 42-node cluster [4]. Each node has two Intel Xeon E5-2620 v2 CPUs and two 480 GB node-local SATA SSDs as NVM. All nodes connected with InfiniBand 4X FDR. The GASNet version is 1.30.0.

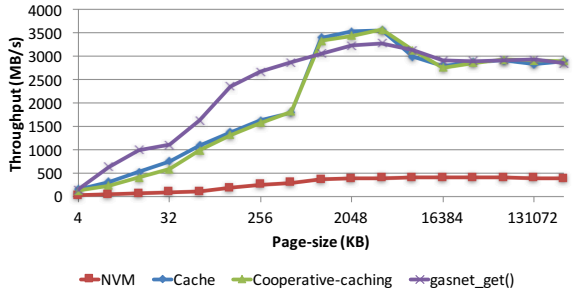

Fig. 7. Throughputs of vGASNet protocol

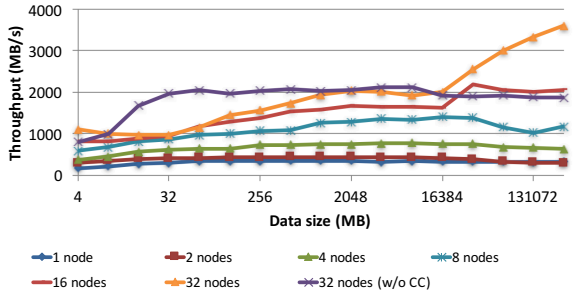

Fig. 8. Sequential access throughput

B. Preliminary Evaluation

To optimize the page size of vGASNet, we conducted a preliminary evaluation. For this purpose, we implemented a simple program that emulates our vGASNet protocol including cooperative caching on top of GASNet.

We used three nodes in TSUBAME-KFC, node A, B, and C. We observed the throughput when node A requests a page whose owner is node B.

Fig. 7 shows the throughput in three cases. "NVM" denotes a case where no node has caches of the page. Here the page is read from node B's NVM. "Cache" means that the page is already cached in DRAM of node B. The page cache is sent to node A without NVM access. "Cooperative-caching" denotes a case where the page is cached in DRAM of node C. Here the request from node A is forwarded from node B to node C. For comparison reason, we also evaluated the throughput of `gasnet_get()`. Here the page is placed on DRAM of node B.

In fig. 7, we observe that throughputs of "Cache" and "Cooperative-caching" are much better than that of "NVM", which suffers from NVM bandwidth directly. When a page size is 1 MB or larger, throughputs of the two cases are comparable to or better than `gasnet_get()`. Also, the gap between "Cache" and "Cooperative-caching" are very small with 1 MB page or larger, which shows that costs of request forwarding is well hidden in this condition. Hereafter, we adopt 4 MB, which achieves the maximum throughput in this experiment, as a page size.

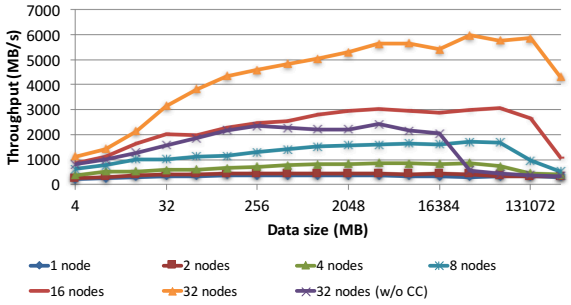

Fig. 9. Random access throughput

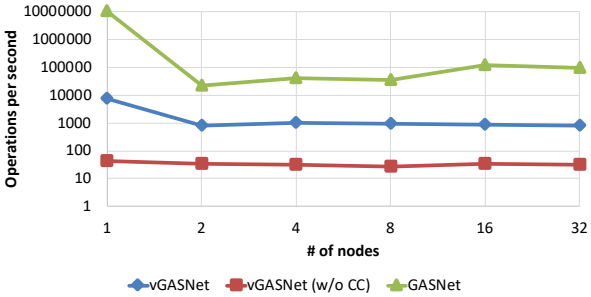

Fig. 10. False sharing performance

C. Basic Operations

1) Overview: To investigate vGASNet scalability, we performed our benchmark programs developed with vGASNet and GASNet. In each experiment, the all nodes get or put a remote access segment in one of the nodes. To observe scalability improvement by cooperative-caching, we developed vGASNet without cooperative-caching (denotes "w/o CC" in the graphs). The cache pool size was 16 GB. A node is assigned to only one rank.

2) Sequential access: In this experiment, each node get a datum on a continuous memory region in a rank sequentially. In fact, the program gets the content of the remote access segment with `vgasnet_get()` page by page, then measure the time of the communication.

The result is shown as fig. 8. Compared with 1 node, 32 nodes under vGASNet obtained 3.23 – 11.1 times faster throughput. This result shows that cooperative-caching reduces the throughput of vGASNet when the data size is small due to MOESI-F protocol overhead. However, these cases are that the data is fitted in cache pool. Indeed, cooperative-caching gains 1.34 – 1.92 times throughput when the size is larger than cache pool size.

3) Random access: In this experiment, each node gets the contents of a continuous remote access segment in a rank randomly. Like sequential access experiment described above, the program also gets the content of the remote access segment with `vgasnet_get()` page by page, then measure the time of the communication. However, the order of gotten pages is shuffled.

The result is shown as fig. 9. Surprisingly, the throughputs of multiple nodes are faster than sequential cases. This is because

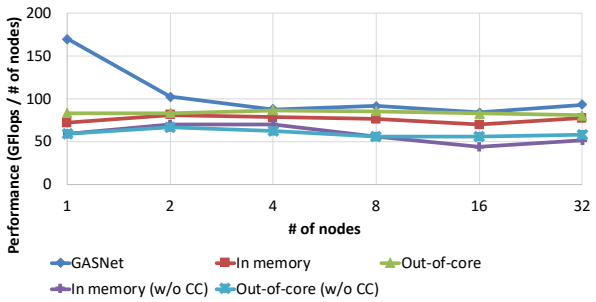

Fig. 11. Weak scale performance per node of the DGEMM program

the orders of gotten pages are different among nodes. Since each node caches different pages, the caches of the remote access segment are distributed. Similarly, even if the data size is smaller than cache pool, cooperative-caching provides higher scalability.

4) False sharing: To estimate the performance when false sharing is occurred, we developed a program which causes false sharing. In this program, all nodes puts one byte data in the same page simutaneously.

The result is shown as fig. 10. Except one node, vGASNet with cooperative-caching is 23.5 – 33.7 times faster than vGASNet without cooperative-caching. This is because the node which has the cache should write back to the owner. With cooperative-caching, the cache can be transferred to the requesting node directly. On the other hand, vGASNet with cooperative-caching is 138 times slower than GASNet. This result tells that false sharing should be avoided as possible even vGASNet supports it.

D. Integrating with UPC++

We integrated vGASNet with UPC++ (version 0.1), a practical PGAS library using GASNet. We used a DGEMM sample program implemented in the UPC++ official repository.

The program calculates $C = A \times B$ using SUMMA [25]. A, B, and C are square matrices of the same sizes. Each matrix is divided into submatrices. Each block of C is calculated by the rank whose NVM hold the block. The program is single-threaded. Instead, our evaluation was conducted on 12 ranks per node. The cache pool size was 2 GB. The size of each submatrix was 512.

The weak scale peformance of the DGEMM program is drawn as fig. 11. "GASNet" is the program using original UPC++. In other words, "GASNet" does not use vGASNet. "In memory" and "Out-of-core" are the programs using our UPC++ with vGASNet. In "GASNet" and "In memory", each matrix size is $12288\sqrt{N} \times 12288\sqrt{N}$ where N is the number of nodes. On the other hand, the size is $24576\sqrt{N} \times 24576\sqrt{N}$ in "Out-of-core". In this experiment, the size of the remote access segment of "GASNet" in each rank was 2 GB. Therefore, executing "GASNet" with $24576\sqrt{N} \times 24576\sqrt{N}$ matrices was failed due to an out of memory error.

Although "Out-of-core" caused NVM accesses, we observed that "Out-of-core" is 1.03 – 1.16 times faster than "In memory". The reason is that the computation time of each submatrix is $O(M^3)$ although the communication time is $O(M^2)$ where M is size of submatrix. Contrarily, on one node, "Out-of-core" is 51 % slower than "GASNet". In our evaluation, a node can have duplicated caches, which point to the same cache line. This is because each node has its own cache pool and cache table. Both the pool and the table cannot be accessed by another rank even if the rank is assigned to the same node. This leads to not only wasting the cache pools but also extra memory copies inside a node.

Like the throughputs of basic operations, adapting cooperative-caching into vGASNet can improve its scalability. Indeed, "In memory" is 1.12 – 1.59 times faster than "In memory (w/o CC)". This is because some ranks access the same sub matrices in this matrix multiplication program. Cooperative-caching can avoid the congestion of communication caused by simultaneously accessing to the same node. For the similar reason, "Out-of-core" is 1.25 – 1.53 times faster than "Out-of-core (w/o CC)".

V. Related Work

Many researchers have developed memory managed systems or communication libraries for NVMs [6]–[15]. Especially, Papyrus [12], [13] and HHRT [11] are available for practical supercomputers such as TSUBAME and Beacon. However, except ComEx-PM, they do not focus on RMA functionalities. vGASNet accomodates RMA functionalities such as `vgasnet_get()` and `vgasnet_put()`. ComEx-PM [10] supports RMA functionalities featuring node-local NVMs. However, cache mechanism of ComEx-PM depends on Linux kernel VFS cache. In this study, we propose an efficient cache mechanism for RMA-based communication library optimized for node-local NVMs.

Cooperative-caching was firstly proposed by Dahlin et al [18]. After that, many distributed filesystems have been equipped with cooperative caching [26]–[31]. Like vGASNet, Hwang et al. [26] proposed write-enabled cooperative caching mechanism for NFS.

Ferguson et al. [32] proposed a cache mechanism for a PGAS language. They adapted caching to RMA-based Put/Get operations. Their cache mechanisms are not for NVMs.

VI. Conclusion

In this paper, we introduce the mechanism of vGASNet, an RMA-based communication library. vGASNet considers NVM as main memory. For performance improvement, vGASNet also uses DRAM as a cache pool. Under vGASNet, each rank uses not only local caches but also remote caches. This methodology is called cooperative-caching. For introduction of cooperative-caching to vGASNet, we propose a cache coherence protocol, namely MOESI-F protocol. MOESI-F protocol is a combination of two existing cache coherence protocols, MOESI protocol and MOSI-F protocol. Our evaluation shows our cooperative-caching mechanism can gain 1.12 – 1.53 times performance measured by a DGEMM program.

ACKNOWLEDGMENT

We would like to thank Dan Bonachea of Lawrence Berkeley National Laboratory and some anonymous reviewers for useful comments. This work is partly supported by JST-CREST.

REFERENCES

[1] E. Georganas, A. Buluc, J. Chapman, S. Hofmeyr, C. Aluru, R. Egan, L. Oliker, D. Rokhsar, and K. Yelick, "HipMer: An Extreme-Scale De Novo Genome Assembler," in *Proceedings of the ACM/IEEE 28th International Conference for High Performance Computing, Networking, Storage and Analysis (SC '15)*, 2015.

[2] S. French, Y. Zheng, B. Romanowicz, and K. Yelick, "Parallel Hessian Assembly for Seismic Waveform Inversion Using Global Updates," in *Proceedings of the IEEE 29th International Parallel and Distributed Processing Symposium (IPDPS '15)*, 2015.

[3] S. Matsuoka, T. Endo, A. Nukada, S. Miura, A. Nomura, H. Sato, H. Jitsumoto, and A. Drozd, "Overview of TSUBAME3.0, Green Cloud Supercomputer for Convergence of HPC, AI and Big-Data," *e-Science Journal*, vol. 16, pp. 2–9, 2017.

[4] T. Endo, A. Nukada, and S. Matsuoka, "TSUBAME-KFC: a Modern Liquid Submersion Cooling Prototype towards Exascale Becoming the Greenest Supercomputer in the World," in *Proceedings of the 2014 IEEE International Conference on Parallel and Distributed Systems (ICPADS '14)*, 2014.

[5] R. G. Brook, A. Heinecke, A. B. Costa, P. Peltz, V. C. Betro, T. Baer, M. Bader, and P. Dubey, "Beacon: Deployment and Application of Intel Xeon Phi Coprocessors for Scientific Computing," *Computing in Science & Engineering*, vol. 29, no. 2, pp. 65–72, 2015.

[6] H. Volos, A. J. Tack, and M. M. Swif, "Mnemosyne: Lightweight Persistent Memory," in *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '11)*, 2011.

[7] J. Coburn, A. M. Caulfield, A. Akel, L. M. Grupp, R. K. Gupta, R. Jhala, and S. Swanson, "NV-Heaps: Making Persistent Objects Fast and Safe with Next-Generation, Non-Volatile Memories," in *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '11)*, 2011.

[8] C. Wang, S. S. Vazhkudai, X. Ma, F. Meng, Y. Kim, and C. Engelmann, "NVMalloc: Exposing an Aggregate SSD Store as a Memory Partition in Extreme-Scale Machines," in *Proceedings of the IEEE 26th International Parallel and Distributed Processing Symposium (IPDPS '12)*, 2012.

[9] D. Schwalb, T. Berning, M. Faust, M. Dreseler, and H. Plattner, "nvm_malloc: Memory Allocation for NVRAM," in *Proceedings of the 6th International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures (ADMS '15)*, 2015.

[10] R. Matsumiya and T. Endo, "PGAS Communication Runtime for Extreme Large Data Computation," in *Proceedings of the 2nd International Workshop on Extreme Scale Programming Models and Middleware (ESPM2 '16)*, 2016.

[11] T. Endo, "Realizing Out-of-Core Stencil Computations using Multi-Tier Memory Hierarchy on GPGPU Clusters," in *Proceedings of the IEEE International Conference on Cluster Computing 2016 (CLUSTER '16)*, 2016.

[12] J. Kim, K. Sajjapongse, S. Lee, and J. S. Vetter, "Design and Implementation of Papyrus: Parallel Aggregate Persistent Storage," in *Proceedings of the 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS '17)*, 2017.

[13] J. Kim, S. Lee, and J. S. Vetter, "PapyrusKV: A High-Performance Parallel Key-Value Store for Distributed NVM Architectures," in *Proceedings of the ACM/IEEE 30th International Conference for High Performance Computing, Networking, Storage and Analysis (SC '17)*, 2017.

[14] Y. Shan, S.-Y. Tsai, and Y. Zhang, "Distributed Shared Persistent Memory," in *Proceedings of the 2017 ACM Symposium on Cloud Computing (SoCC '17)*, 2017.

[15] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin, "Efficient Memory Disaggregation with INFINISWAP," in *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI '17)*, 2017.

[16] D. Bonachea, "GASNet Specification, v1.1," EECS Department, University of California, Berkeley, Tech. Rep. UCB/CSD-02-1207, Oct 2002. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2002/5764.html

[17] J. Lee and M. Sato, "Implementation and Performance Evaluation of XcalableMP: A Parallel Programming Language for Distributed Memory Systems," in *Proceedings of the International Conference on Parallel Processing Workshops (ICPPW '10)*, 2010.

[18] M. D. Dahlin, R. Y. Wang, T. E. Anderson, and D. A. Patterson, "Cooperative Caching: Using Remote Client Memory to Improve File System Performance," in *Proceedings of the 1st USENIX Conference on Operating System Design and Implementation (OSDI '94)*, 1994.

[19] Y. Zheng, A. Kamil, M. B. Driscoll, H. Shan, and K. Yelick, "UPC++: A PGAS extension for C++," in *Proceedings of the IEEE 28th International Parallel and Distributed Processing Symposium (IPDPS '14)*, 2014.

[20] UPC Consortium, "UPC Language and Library Specifications, v1.3," Lawrence Berkeley National Laboratory, Tech. Rep. LBNL-6623E, Nov 2013. [Online]. Available: https://escholarship.org/uc/item/0n7734mg

[21] M. Bauer, S. Treichler, E. Slaugher, and A. Aiken, "Legion: Expressing Locality and Independence with Local Regions," in *Proceedings of the ACM/IEEE 25th International Conference for High Performance Computing, Networking, Storage and Analysis (SC '12)*, 2012.

[22] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser, "Active Messages: a Mechanism for Integrated Communication and Computation," in *Proceedings of the 19th Annual International Symposium on Computer Architecture (ISCA '92)*, 1992.

[23] Advanced Micro Devices, "AMD64 Technology AMD64 Architecture Programmer ' s Manual Volume 2: System Programming," http://developer.amd.com/wordpress/media/2012/10/24593_APM_v21.pdf.

[24] D. Kanter, "The Common System Interface: Intel' s Future Interconnect," *Real World Tech*, no. 5, 2012.

[25] R. A. Van De Geijn and J. Watts, "Summa: scalable universal matrix multiplication algorithm," *Concurrency: Practice and Experience*, vol. 9, no. 4, pp. 255–274, 1997.

[26] I.-C. Hwang, H. Jung, S.-R. Maeng, and J.-W. Cho, "Design and Implementation of the Home-Based Cooperative Cache for PVFS," in *Proceedings of the 5th International Conference on Computational Science (ICCS '05)*, 2005.

[27] P. Sarkar and J. Hartman, "Hint-based Cooperative Caching," *ACM Transactions on Computer Systems*, vol. 18, no. 4, pp. 387–419, 2000.

[28] S. Annapureddy, M. J. Freedman, and D. Mazières, "Shark: Scaling File Servers via Cooperative Caching," in *Proceedings of the 2nd Symposium on Networked Systems Design & Implementation (NSDI '05)*, 2005.

[29] S. Sasaki, R. Matsumiya, K. Takahashi, Y. Oyama, and O. Tatebe, "RDMA-based Cooperative Caching for a Distributed File System," in *Proceedings of the 21st IEEE International Conference on Parallel and Distributed Systems (ICPADS '15)*, 2015.

[30] Y. Xu and B. D. Fleisch, "NFS-cc: Tuning NFS for Concurrent Read Sharing," *International Journal of High Performance Computing and Networking*, vol. 1, no. 4, pp. 203–213, 2004.

[31] A. Batsakis and R. Burns, "NFS-CD: Write-Enabled Cooperative Caching in NFS," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 3, pp. 323–333, 2007.

[32] M. P. Ferguson and D. Buettner, "Caching Puts and Gets in a PGAS Language Runtime," in *Proceedings of the 9th International Conference on Partitioned Global Address Space Programming Models (PGAS '15)*, 2015.