The Design of Advanced Communication to Reduce Memory Usage for Exa-scale Systems

Shinji Sumimoto[1], Yuichiro Ajima[1], Kazushige Saga[1],
Takafumi Nose[1], Naoyuki Shida[1] and Takeshi Nanri[2]

[1] Fujitsu Ltd. 4-1-1 Kamikodanaka 4-Chome, Nakahara-ku,
Kawasaki-city, Kanagawa, 211-8588, Japan.
E-mail: {sumimoto.shinji,aji,saga.kazushige, nose.takafumi, shidax}@jp.fujitsu.com
[2] Kyushu University, 6-10-1 Hakozaki Higashi-ku, Fukuoka 812-8581 Japan.
E-mail: nanri@cc.kyushu-u.ac.jp

Abstract. Current MPI (Message Passing Interface) communication libraries require larger memories in proportion of the number of processes, and can not be used for exa-scale systems. This paper proposes a global memory based communication design to reduce memory usage for exa-scale communication. To realize exa-scale communication, we propose true global memory based communication primitives called Advanced Communication Primitives (ACPs). ACPs provide global address, which is able to use remote atomic memory operations on the global memory, RDMA (Remote Direct Memory Access) based remote memory copy operation, global heap allocator and global data libraries. ACPs are different from the other communication libraries because ACPs are global memory based so that house keeping memories can be distributed to other processes and programmers explicitly consider memory usage by using ACPs. The preliminary result of memory usage by ACPs is 70MB on one million processes.

1 Motivation

Many countries have been planning to develop exa-scale systems, and the Japanese government has also announced to develop an exa-scale system by the end of 2020. Many core based systems will be used for the exa-scale system and the number of cores will be in the 10 million class. We have to consider not only the impacts of number of cores and nodes, but also that of the number of processes on the system software stacks in this situation, and we are researching high performance communication libraries that are able to be used for 10 million process class parallel systems.

However, current communication libraries, such as Open MPI[1] and MPICH[2], require larger memories in proportion to the number of processes, and, they can not be used for exa-scale systems because they eventually exhaust the memories. Therefore, memory usage of them must be dramatically reduced.

We propose Advanced Communication Primitives (ACPs) with global memory access and management functions to reduce memory usage by communication libraries. ACPs are aimed at achieving low-level communication primitives, and be used to implement PGAS based languages on top of ACPs.

This paper is organized as follows, section 2 discusses memory usage issues for Exa-scale systems. Section 3 proposes our approach of ACPs, and describes global memory based communication design to reduce memory. Section4 shows evaluation of ACPs, and section 5 discusses related work.

2 Memory Usage Issues for Exa-scale Systems

We measured and evaluated memory usage by Open MPI with an InfiniBand network using DMATP-MPI[3] tool. InfiniBand interconnect has three types of communication protocol, i.e., Reliable Connection (RC) with Receive Queue (default and RC-RQ), RC with Shared Receive Queue (RC-SRQ), and Unreliable Datagram (UD). Fig. 1 plots the results. The results show that memory usage of Open MPI highly depend on communication protocols. Especially, RC-RQ and RX-SRQ require higher amount of memory because they require arrays of descriptors and buffers for messages for each destination.

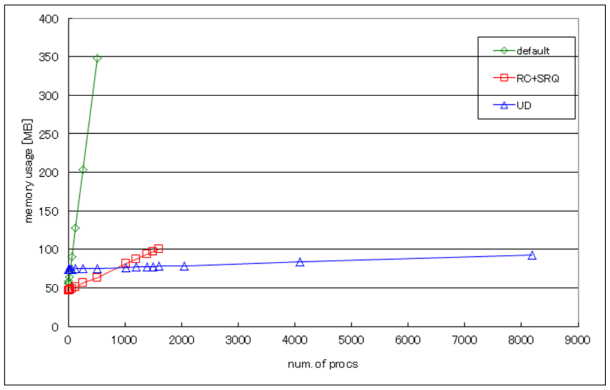

Fig. 1. Memory Usage by Open MPI 1.4.5

Table 1. Estimated Memory Usage by Open MPI

# of Procs	RC-RQ	RC-SRQ	UD
100,000	56.23	3.33	0.29 GB
1,000,000	561.87	32.86	2.24 GB
10,000,000	5,618.22	328.17	21.75 GB

By extrapolating the data from Fig. 1, Open MPI memory usage by the exa-scale system was estimated. Table 1 summarizes the estimated memory usage by up to 10 million processes, where memory usage by RC-RQ on one million processes is 561 GB and that by RC-SRQ is 32.9 GB. These results indicate that the connection oriented communication protocol is not scalable in essentials as described by Sumimoto et al.[4].

Table 1 also indicates that the MPI program on one million processes required about 2.2 GB of memory per process even if the UD protocol was used. This also indicates current MPI libraries usually allocate *o(Number of Processes)* amount of memory to the MPI buffer and control structure.

We analyzed the reason and found that MPI_Init function allocated memory in proportion to the number of processes because each process had redundant copies of information from the other processes'.

3 The Design of Advanced Communication for Exa-scale Systems

To realize high performance communication for Exa-scale system, not only reduction of memory usage of communication library but also memory reduction programing infrastructure is needed. This section discusses the design of advanced communication for Exa-scale systams.

3.1 Advanced Communication Primitves (ACPs) Design

As discussed in Section 2, MPI_Init function allocates memory in proportion to the number of processes bacause each process has redundant copies of information from other processes'. To eliminate the redundant copies, such process information should be located in the original process memory and accessed when needed for exa-scale communication.

To realize an easy access to such distributed process data, we decided to introduce global memory access scheme and global addresses which is able to use an address pointer as same as address pointer in data structures in local memory. By providing the global memory access scheme instead of message passing, communication library doesn't have to prepare message buffers and descriptors, and users can manipulate global memory pointers without considering the location of the pointer. This means that user program can make and access globally distributed data structures and doesn't need redundant copy data.

To provide the global memory access scheme, communication library should provide functions to handle global memory access, such as data copy, data hanle, global memory allocation and global memory data library functions.

To realize the global memory access functions, we developed the Advanced Communication Primitives(ACPs). ACPs provide global addresses, which are able to use remote atomic memory operations on the global memory, and RDMA based memory copy communication to effectively manipulate distributed structure. We chose RDMA based communication because it does not need an intermediate communication buffer such as message based communication and modern interconnects such as Tofu and InfiniBand to support it.

The ACPs consist of basic layer (ACPbl) and middle layers which consist of Communication Library (ACPcl) and Data Library (ACPdl), and all of the functions are able to handle global address pointers as they are. Each process on ACPs can individually register and unregister its local memory to the global memory without inter-process synchronization. The primal data transfer function of the layer is a 'copy' on the global memory. The initiator process for

Table 2. ACPbl Function Examples

Functions	Description	Functions	Description
acp_init()	Initialization	acp_register_memory()	Memory Registration
acp_finalize()	Finalization	acp_unregister_memory()	Memory Un-registration
acp_reset()	Reset	acp_copy()	Global memory copy
acp_sync()	Synchronization	acp_cas[48]()	Atomic compare and swap
acp_rank()	Getting rank number	acp_swap[48]()	Atomic swap operation
acp_procs()	Getting process Group	acp_complete()	Waiting completion
acp_query _address()	Query Local Address	acp_inquire()	Checking completion

the copy does not have to be the source nor the destination. The 'copy' function is directly implemented by using RDMA when network hardware, such as InfiniBand or Tofu Interconnect, has RDMA.

3.2 ACP Basic Layer: ACPbl

The basic ACP layer consists of an infrastructure, global memory management (GMM), and global memory access (GMA) functions to provide RDMA based memory copy communication, remote atomic operations, and initialize and finalize functions. It also provides fixed-size starter memory to exchange global addresses among processes after ACPs are initialized. The size of the starter memory can be used to change environment variables or argument options during execution.

Current global address handles of ACPs are described as 64 bit unsigned integer type data so that they can directly use hardware atomic operation. Programs with ACPs do not have to recognize whether global address data exist on local memories or not. They only recognize them when directly accessing data. ACPs provide a translation function from global addresses to local logical memory addresses, and when the function fails, the data are on other process memories and need to be copied from the global data to local memory to access them.

Table 2 lists examples of ACPbl functions, where there are several infrastructure functions, and copy, compare and swap, swap, checking, and waiting operation functions.

3.3 Communication and Data Libraries

ACPs are also comprised of two main categories of interfaces, i.e., communication (ACPcl) and data libraries (ACPdl[5]). The communication libraries consist of channel interface, collective and neighbor interface, and global data libraries. These interfaces are built on the basic layer that provides a global memory model among processes. Programmers create channels when needed in a channel interface, and destroy them when communication has finished. The channel interface reduces memory usage by creating and destroying channels only when needed.

ACPdl privides five types of data structures which are vector, list, deque, set and map which are similar to the collection of the C++ language standard template library. It also provides a global memory allocator function named the acp_malloc which allocates a segment of global memory from current process on a specified process rank. A global memory segment allocated by the acp_malloc function can be easily freed by the acp_free function.

4 Evaluation of ACPbl and ACPdl

We are now developing ACP libraries and have finished ACPbl for UDP/IP and Tofu and some of ACPdl to evaluate it. Fujitsu Supercomputer PRIMEHPC FX10 was used for the evaluation on Tofu interconnect, Fujitsu Supercomputer PRIMEHPC FX100 for the evaluation on Tofu2 interconnect, and Fujitsu PRIMERGY RX200 S7 for the evaluation of UDP/IP. Fig. 2 plots the preliminary bandwidth performance of ACPbl for a Tofu interconnect using an acp_copy function with local memory to remote memory in Table 2.

4.1 Evaluation of ACPbl Communication Performance

The performance of the communication bandwidth of MPI has also been shown for comparison, and it can be seen ACPbl outperformed MPI in bandwidth. We also evaluated preliminary memory usage by ACPbl.

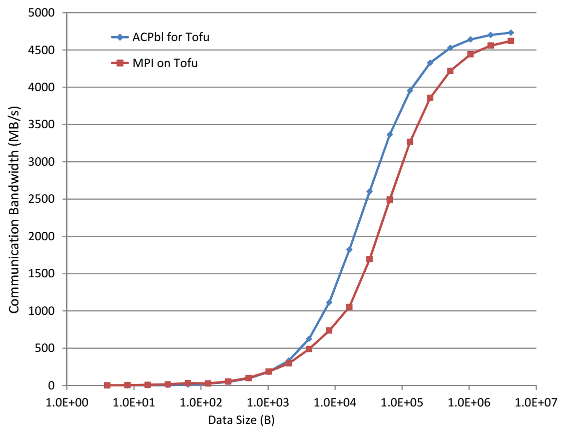

Fig. 2. Performance of ACPbl Communication on Tofu

4.2 Evaluation of ACPbl Memory Usage

Current estimated memory usage by ACPbl for Tofu is 70 MB on one million processes, and that for UDP/IP is 19 MB. These results are better than the results in Table. 1.

Table.3 shows the detail analysis of ACPbl memory usage on Tofu. In addition to the memory usage, 2 MBytes of memories are needed for Tofu hardware operation. Therefore, 72 MBytes of memories are needed for Tofu communication using ACPbl of Tofu. This means the combination of ACP and Tofu, which is not

Table 3. Memory Usage of ACPbl(Tofu) on 1 Million Processes

	ACPbl(Tofu)
Memories in proportion of the # of Processes –Command Receive Buffer –Tofu Address Table –Tofu Routing Table	69 MBytes@ 1M Per Process Information 64 Bytes / Process 4 Bytes / Process 1 Bytes / Process
Memories in proportion of the # of Memory Resistation	9 KBytes for 128 Entries
Misc. Buffers	262 KBytes

connection oriented, is able to realize exa-scale communication. However, MPI on Tofu requires about 2.2 GB of memory for one million processes as same as UD protocol on InfiniBand, because current MPI implementation requires MPI buffer and control structure for all ranks statically.

4.3 Evaluation of ACPdl Execution Performance

This subsection presents evaluation results of ACPdl functions. In the evaluations, the acp_malloc, acp_free, acp_insert_map, and acp_find_map functions were evaluated. Every experiment used two nodes and all functions accessed memory of the other process on the remote node.

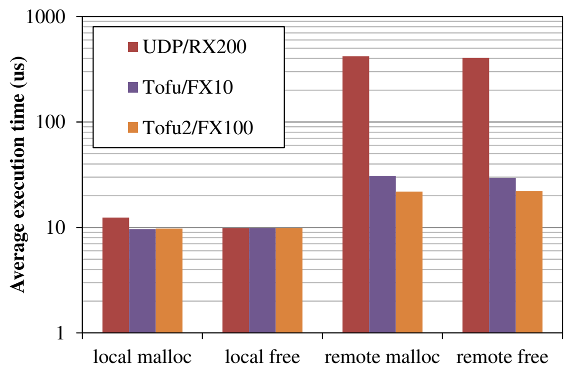

Fig. 3. Performance of ACPdl acp_malloc and acp_free

Fig. 3 shows the evaluation results of the acp_malloc and acp_free functions. The average execution times of the acp_malloc and acp_free functions with the initial algorithm were around 420 and 400 *u*secs using the UDP version of ACPbl, 31 and 29 *u*secs using Tofu, and 24 and 24 *u*secs using Tofu2.

Fig. 4 shows the results of the acp_insert_map and acp_find_map functions. The average execution times of the acp_insert_map and acp_find_map functions were around 1040 and 760 *u*secs using the UDP version of ACPbl, 86 and 64 *u*secs using Tofu, and 90 and 82 *u*secs using Tofu2.

These results show that ACPdl can be used effectively to handle distributed data structures with data allocation and manipulation on global memory space.

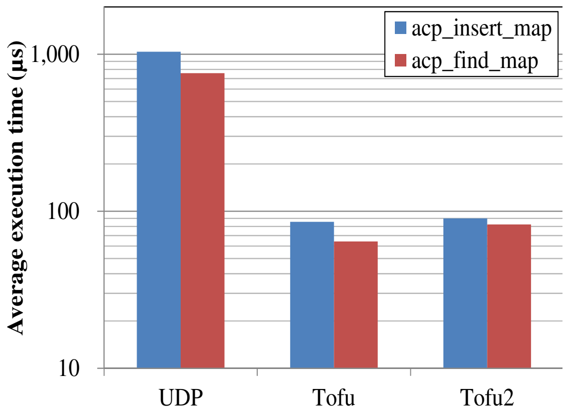

Fig. 4. Performance of ACPdl acp_insert_map and acp_find_map

5 Related Work

There have been several related work to reduce memory usage in related work.

MPICH, MVAPICH[6] and Open MPI use memory reduction techniques for InfiniBand. They use the UD and RC-SRQ communication protocol to reduce memory usage. Mellanox Dynamically Connected (DC) Transport Service, which also reduces the memory usage footprint drastically. However, DC is implemented on original InfiniBand RC protocol and some performance degradation exists when RC connections are disconnected and connected.

Open MPI only allocates a communication data buffer for communication. Open MPI for the K computer introduces two memory consumption models, i.e., high performance and memory saving modes[7]. It allocates the memory saving mode on first communication to a destination, and when the number of communications exceeds a predefined value, 16 at default, it switches the memory saving model into high performance mode.

Balaji et al.[8] discusses MPI on a million processors on MPICH and current implementation requires 80% (1.6GB) of memory on 128K BlueGene/P process system. It points out memory usage by communicator creation.

There are several low level communication libraries that support RDMA access and multiple networks such as UCCS[9], Portals[10], PAMI[11], and so on. These communication libraries provide RDMA based communication and message communication and memory usages by them depend on how to use the message communication. They do not focus memory usage reduction.

However, ACPs are true global memory based so that house keeping memories such as the other process'es information can be distributed to other processes and programmers explicitly consider memory usage by using ACPs to reduce memory usage even if for message communication.

6 Summary and Future Work

This paper proposed a global memory based communication design to reduce memory usage in exa-scale communication. We analyzed memory usage by current communication libraries and clarified issues with reducing memory usage by house-keeping memory such as the other process'es information in communication libraries.

We proposed global memory based communication primitives called ACPs to solve these issues. ACPs provide global addresses, which are able to use remote atomic memory operations on the global memory, and RDMA based memory copy communication, global heap allocator and global data libraries. ACPs are different from the other communication libraries because ACPs are global memory based so that house keeping memories can be distributed to other processes and programmers explicitly consider memory usage by using ACPs.

We have finished implementing ACPbl for UDP/IP, Tofu and InfiniBand, and ACPdl including global heap memory allocation and manipulation of five types of data structures. The preliminary evaluation results show that performance and memory usage of ACPbl outperform current MPI libraries and the preliminary result of memory usage by ACPs is 70MB on one million processes. We intend to evaluate and optimize ACPbl and apply to several libraries such as global array, co-array and scripting languages such as python.

References

1. Open MPI:http://www.open-mpi.org/.
2. MPICH-A Portable Implementation of MPI: http://www-unix.mcs.anl.gov/mpi/mpich/.
3. Shinji Sumimoto, Takayuki Okamoto, Hideyuki Akimoto, Tomoya Adachi, Yuichiro Ajima, and Kenichi Miura. Dynamic Memory Usage Analysis of MPI Libraries Using DMATP-MPI. In *Proceedings of the 20th European MPI Users' Group Meeting*, EuroMPI '13, pp. 149–150. ACM, 2013.
4. S. Sumimoto, A. Naruse, K. Kumon, K. Hosoe, and T. Shimizu. PM/InfiniBand-FJ: a high performance communication facility using InfiniBand for large scale PC clusters. In *High Performance Computing and Grid in Asia Pacific Region, 2004. Proceedings. Seventh International Conference on*, pp. 104–113, July 2004.
5. Yuichiro Ajima, Takafumi Nose, Kazushige Saga, Naoyuki Shida, and Shinji Sumimoto. ACPdl: Data-Structure and Global Memory Allocator Library over a Thin PGAS-Layer. In *First International Workshop on Extreme Scale Programming Models and Middleware ESPM2*, 2015.
6. MVAPICH: http://mvapich.cse.ohio-state.edu/.
7. Shinji Sumimoto. The mpi communication library for the k computer: Its design and implementation. In *EuroMPI*, p. 11, 2012.
8. Pavan Balaji, Darius Buntinas, David Goodell, William Gropp, Sameer Kumar, Ewing Lusk, Rajeev Thakur, and Jesper Larsson Träff. MPI on a Million Processors. In *Proceedings of the 16th Euro PVM/MPI*, pp. 20–30, 2009.
9. UCCS-Universal Common Communication Substrate: http://uccs.github.io/uccs/.
10. Portals4: http://www.cs.sandia.gov/Portals/portals4.html.
11. S. Kumar, A.R. Mamidala, D.A. Faraj, B. Smith, M. Blocksome, B. Cernohous, D. Miller, J. Parker, J. Ratterman, P. Heidelberger, Dong Chen, and B. Steinmacher-Burrow. Pami: A parallel active message interface for the blue gene/q supercomputer. In *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pp. 763–773, May 2012.