# Chapter 4
# System Software for Many-Core and Multi-core Architecture

**Atsushi Hori, Yuichi Tsujita, Akio Shimada, Kazumi Yoshinaga, Namiki Mitaro, Go Fukazawa, Mikiko Sato, George Bosilca, Aurélien Bouteiller, and Thomas Herault**

**Abstract** In this project, the software technologies for the post-peta scale computing were explored. More specifically, OS technologies for heterogeneous architectures, lightweight thread, scalable I/O, and fault mitigation were investigated. As for the OS technologies, a new parallel execution model, *Partitioned Virtual Address Space (PVAS)*, for the many-core CPU was proposed. For the heterogeneous architectures, where multi-core CPU and many-core CPU are connected with an I/O bus, an extension of PVAS, *Multiple-PVAS*, to have a unified virtual address

AQ1

A. Hori (✉)
RIKEN, Minato-ku, Tokyo, Japan
e-mail: ahori@riken.jp

Y. Tsujita
RIKEN, Kobe, Hyogo, Japan
e-mail: yuichi.tsujita@riken.jp

A. Shimada
Research and Development Group, Hitachi, Ltd., Yokohama, Kanagawa, Japan
e-mail: akio.shimada.ht@hitachi.com

K. Yoshinaga
eF-4 Co., Ltd., Meguro-ku, Tokyo, Japan
e-mail: k.yoshinaga@ef-4.co.jp

N. Mitaro
Tokyo University of Agriculture and Technology, Koganei-shi, Tokyo, Japan
e-mail: namiki@cc.tuat.ac.jp

M. Sato
Department of Embedded Technology, School of Information and Telecommunication Engineering, Tokai University, Minato-ku, Tokyo, Japan
e-mail: mikiko.sato@tokai.ac.jp

G. Fukazawa
Yamaha Corp., Naka-ku, Hamamatsu, Japan
e-mail: go.fukazawa@fzawa.net

G. Bosilca · A. Bouteiller · T. Herault
Innovative Computing Laboratory, University of Tennessee, Knoxville, TN, USA
e-mail: bosilca@icl.utk.edu; bouteill@icl.utk.edu; herault@icl.utk.edu



Mitsuhisa Sato (eds.), *Advanced Software Technologies for Post-Peta Scale Computing*, https://doi.org/10.1007/978-981-13-1924-2_4

space of multi-core and many-core CPUs was proposed. The proposed PVAS was also enhanced to have multiple processes where process context switch can take place at the user level (named *User-Level Process: ULP*). As for the scalable I/O, *EARTH*, optimization techniques for MPI collective I/O, was proposed. Lastly, for the fault mitigation, *user-level fault mitigation, ULFM* was improved to have faster agreement process, and *sliding methods* to substitute failed nodes with spare nodes was proposed. The funding of this project was ended in 2016; however, many proposed technologies are still being propelled.

## 4.1 Overview and Background

The goal of this project was to develop OS technologies needed for post-peta scale machines in the future under the assumption of heterogeneous architecture would have dominated. The goal of this project was to develop key software technologies required for the post-peta scale computers. There were ~~4~~ four research topics in this project~~;~~ :

- OS technologies for heterogeneous architectures,
- Lightweight thread,
- Scalable I/O, ~~and~~
- Fault mitigation.

This 5-year project was started from April 2011 and ended March 2016. When this project was started, the first commercial many-core CPU, Intel Knights Corner (known as KNC) [11], became available, and many people believed it was the dawn of many-core era. KNC was designed as a ~~co-processor~~ coprocessor and it needed another many-core CPU to be attached with. From the view point of OS research, this heterogeneous CPU architecture was new because an OS ran on each CPU, unlike the other ~~co-processor~~ coprocessor architectures such as GP-GPU or vector processing unit (e.g., ClearSpeed [8]) where no (explicit) OS runs on those ~~co-processors~~coprocessors. Our focus was to provide a single system image for such heterogeneous architecture.

The following research topics in this project were higher-level than the above OS research. The performance improvement of applications having dynamic workload characteristics is a big concern for the coming post-scale era. The oversubscribed lightweight multithread is thought to be the answer for this kind of applications. We proposed a novel technique for this topic.

I/O performance has been a headache in the long history of HPC. We focused on the collective I/O in MPI and proposed several optimization techniques to improve I/O performance.

Fault mitigation has been a big concern as supercomputers are scaling up. In this project, we proposed two techniques; one is a parallel execution environment in which a process failure can be mitigated, and another is the technique how failed nodes must be replaced with spare nodes.

## 4.2 OS Technologies for Heterogeneous Architecture

The ~~widely-used~~ widely used in-node parallel execution models are multi-process model (e.g., MPI) and multi-thread model (e.g., OpenMP). The multi-process model has the problem when to communicate with the others because each ~~procecc~~ process has its own virtual address space and shared memory is often used for inter-process communication. The widely used shared memory technique cannot avoid the 2-copy to exchange data between processes. On the other hand, the multi-thread model has the non-negligible overhead of ~~mutual-exclusion~~ mutual exclusion on shared variables. ~~Thise~~ These overheads become more striking when the number of cores increases.

To have the best of the two worlds~~;~~, multi-process and multi-thread, the third execution model had been proposed and implemented. SMARTMAP is to pack processes into one virtual address space, but SMARTMAP only runs on Kitten OS and relies on `x86` architecture [3]. MPC allows threads to have privatized variables, but MPC heavily depends on a language processing system, and it requires special language processing systems [10]. We decided to develop our own system to implement the third execution model. This is called *Partitioned Virtual Address Space (PVAS)* implemented by the patched Linux kernel. Its details ~~is~~ are explained in the next subsection.

PVAS provides the efficient parallel execution mode on many-core architectures. However, our final target architecture is heterogeneous architecture where many-core CPU and multi-core CPU are connected by an I/O bus. Each CPU has its own physical memory. An OS runs independently on each CPU. On KNC, Intel provides the SCIF to communicate between the KNC process and host process. Unfortunately, its communication performance was not so good (shown in Sect. 4.2.2). Our proposed solution to this is the extension of PVAS to pack processes running on KNC and processes running on the host processor into one virtual address space so that every process can access the data of the other processes regardless which process is running on which processor ~~. running~~ [14, 21]. This scheme was called *Multiple-PVAS (MPVAS)*, and more efficient communication than Intel provided SCIF could be achieved.

AQ2

### *4.2.1 Partitioned Virtual Address Space*

Basically, a process has an isolated virtual address space, and it cannot access the data in the other process. Shared memory technique is widely used to implement inter-process communication in HPC. Shared memory is a special memory region in which physical memory region is shared between processes and the data in the shared memory region can be accessed from the processes sharing the memory region. However, ~~when to communicate~~ during communication between two processes, sender process ~~write~~ writes a data into the shared memory segment

and receiver process ~~read~~ reads the data in the shared memory segment. Thus, two memory copies take place, and this memory copy overhead hinders the parallel execution.

AQ3

Each process has its own virtual address space so that the process is guaranteed not to be interfered by the other processes. Let's assume that there are two processes connected by a pipe. If a sender process dies for some reason, then the receiver process is killed by the `SIGPIPE` signal. Even if the `SIGPIPE` signal is ignored, then the receiver process might be blocked forever for receiving data, since there is no process to send data to the receiver process. Thus, communicating processes can be said to share the same fate. The protection mechanism for processes in modern OS incurs the overhead of the inter-process communication. In the coming many-core era, the intra-node communication will be more frequent. Therefore the development of efficient intra-node communication is important.

If all communication processes share the same virtual address space, then the data owned by a process can be accessed by the other processes without incurring the ~~overhad~~ overhead of 2-copy. This is the idea of PVAS [16–20]. There is no protection between the processes sharing the same virtual address space~~,~~. However, as mentioned above, the communicating processes share the same fate, and the protection mechanism has no meaning (Fig. 4.1).

AQ4

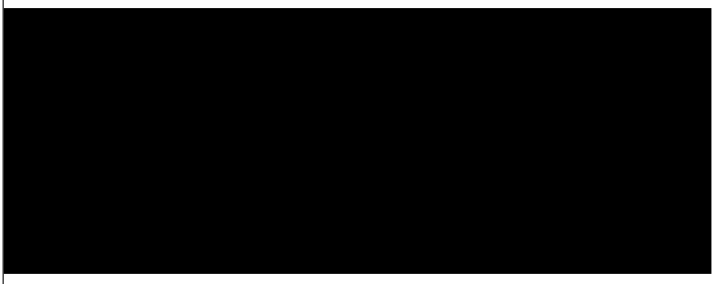

**Fig. 4.1** Address maps of multi-process, multi-thread and PVAS

In PVAS, the virtual address space is firstly partitioned, and process occupies one of the partitions. To implement PVAS, the Linux OS kernel is patched. Figure 4.2 shows the Open MPI intra-node communication performance comparisons among shared memory, denoted as "SM BTL," KNEM (a kernel assisted 1-copy messaging), denoted as "SM-KNEM BTL," XPMEM (allows process to expose a memory region to be accessed by the other process), denoted as "Vader BTL," and PVAS, denoted as "PVAS BTL." This evaluation was carried out on KNC (Intel Xeon Phi, 5110P 1.053 GHz). The left graph shows the PingPong latency measured by using IMB benchmark, and the right graph shows the system-wide memory usage of IMB ~~Alltoall~~ AlltoAll communication. As shown in those graphs, the intra-node communication using PVAS exhibits the lowest latency and the least memory usage.

In the MPI point-to-point communication, MPI datatypes which is an expression of non-contiguous data layout can be specified in the send function and the

**Fig. 4.2** IMB PingPong ~~Latency~~ latency (~~Left~~left) and ~~Memory Consumption~~ memory consumption (~~Right~~right)

receive function. Since datatype information is local~~and ,~~ the sender process has no knowledge on the datatype on receiver process. In the current MPI implementations, if a sender wants to send a non-contiguous data to a receiver, firstly the sender packs the non-contiguous data and then send the packed data to the receiver. And on the receiver process, the packed data is unpacked. Thus, costly message 2-copy for packing and unpacking must take place.

AQ5

With PVAS, however, the sender process can take a look at the datatype information on the receiver process. This is because the sender process and the receiver process share the same virtual address space in PVAS. And non-contiguous messages can be sent with 1-copy according to the datatypes of sender and receiver. Shimada [16] reported 20% improvement with the `ft2d_datatype` benchmark [15].

### *4.2.2 Heterogeneous Extension of PVAS*

On KNC, there are two sets of CPU and memory, and they are connected by the PCIe bus. Independent OS runs on each CPU. The multi-core CPU has a fewer number of fast cores and the many-core CPU has a large number of slow cores. Applications having large parallelism can exploit the power of many-core. When an application running on many-core CPU tries to send a message using MPI, for example, then MPI's complex protocol which is very hard to parallelize must be handled by the slow many-core CPU. This can add extra latency when compared with the case of using multi-core CPU.

The idea here is to delegate the heavy protocol handling from many-core CPU to multi-core CPU so that the complex MPI protocol can be handled by the faster multi-core CPU. The key technology here is the low-latency delegation mechanism from many-core CPU to multi-core CPU ~~,~~ and vice versa. Generally speaking, this idea can be applied not only MPI but many situations.

*Multiple-PVAS (MPVAS)* was designed for such purpose. The processes running on many-core CPU and the processes running on multi-core CPU are mapped

**Fig. 4.3** Multiple PVAS

into one virtual address space (Fig. 4.3) [4]. In the MPVAS-aware MPI, based on NMVAPICH, to send a message from a process on many-core CPU, the send request is passed to the dedicated process running on the multi-core CPU, and then the request is processed on the multi-core CPU. Since the data to be sent can be accessed by the multi-core process, there is no need of copying it.

**Fig. 4.4** Inter-CPU ~~Offloading Latency~~ offloading latency (KNC)

Figure 4.4 shows the latency of RPC between many-core (E5-2650v2, 2.60GHz) and multi-core (Xeon Phi 7120P). X-axis shows the payload length and Y-axis shows the latency. As shown in this figure, the latency of using the offloading mechanism provided by Intel is several order of magnitude slower. Fukazawa reported that the MPI latency with this delegation mechanism is almost halved compared with the one using only many-core CPU [4, 13].

## 4.3 Lightweight Thread

So far, each PVAS process has an associated OS kernel thread to run in parallel. There can be PVAS processes having OS kernel thread and there can be PVAS processes having no OS kernel thread. Assume PVAS process $A$ has no OS kernel thread and PVAS process $B$ has OS kernel thread. The OS kernel thread of $B$ can

switch to *A* at user level by calling the `swapcontext()` or `setcontext()` Glibc function. This mechanism is very similar to the one of user-level thread. However, each PAVS process may have different program and do have privatized variables, unlike (user-level) thread. We named this *User-Level Process (ULP)* because PVAS [20].

**Fig. 4.5** Comparison of context ~~swtching~~ switching time

Figure 4.5 shows the context switching times (Y-axis) comparing Linux processes, Pthreads, MassiveThreads [9] (a user-level thread implementation), and ULP over the number of processes or threads (X-axis). The context switching times of ULP is very close to the ones of MassiveThreads.

If some MPI processes in a node are implemented by using ULP and they are oversubscribed by having more number of MPI processes than the number of CPU cores, then those MPI processes can be switched from one to the other in a fast way. This would result in leveling the load imbalance between CPU cores. Thus, the load imbalance of dynamic workload can run more efficiently.

## 4.4 Scalable I/O

The current optimization technique of collective MPI-IO [22] is based on ~~Two-Phase~~ two-phase I/O [12], where the scattered I/O requests on compute nodes are sorted according to the parallel file offset, and then actual I/O requests are sent to I/O servers. This technique is very effective because the smaller I/O requests on compute nodes are gathered on aggregator processes so that larger I/O requests can be sent to I/O servers.

In general, not limited to HPC, it is the file servers' nature that a number of simultaneous I/O requests can degrade its performance. To prevent this, I/O throttling technique was proposed. In the ~~Two-Phase~~ two-phase I/O, a number of I/O requests are also sent to I/O servers. Our first idea is to throttle the number of request to the IO servers to get higher performance. There is another bottleneck in ~~Two-Phase~~ two-phase I/O. Heavy ~~all-to-all like~~ all-to-all-like communication between user processes and aggregator processes to sort and merge the user's

I/O requests takes place. If the I/O requests to the server are throttled, then this ~~all-to-all like~~ all-to-all-like communication can also be throttled. Consequently, the communication bottleneck can be relaxed. Thus, the flow of the I/O requests from user process to I/O servers can be done in a way of step-wise pipeline to level the loads of the communication network and I/O servers.

This optimization technique was named *EARTH* which includes the above step-wise communication, throttling to the I/O servers, striping-aware data blocking and distribution, and optimized allocation of the aggregator processes to decrease the message congestion in communication network [23, 24]. Figure 4.6 shows the write bandwidth on the K computer [25] comparing the original MPI-IO and EARTH optimized MPI-IO. In the EARTH cases, the number of requests at each step of the step-wise pipeline is also varied (~~denited~~ denoted as "req = *N*"). As shown in this figure, EARTH performs more better when the number of processes increases.

**Fig. 4.6** HP-IO ~~Write Performance~~ write performance (the K computer)

## 4.5 Fault Mitigation

Checkpointing is a well-known technique to recover from a failure. However, its cost can be very high, and the fault mitigation becomes the hot topic in the ~~fault tolerant research. User-Level Fault Mitigation~~ fault-tolerant research. User-level fault mitigation (ULFM) which is a user-level library for a parallel program to survive from a process failure is gathering the attentions as a ~~parctical~~ practical technique of fault mitigation. In this project, a technique to improve the performance of ULFM was developed. The implementation of ULFM is not easy. The signal of a process failure must be propagated to the other processes to reach a consensus on the failure among the healthy processes. Another process failure may happen during the propagation and consensus ~~protools~~protocols. ULFM must survive from any process failure that can happen at any time. In the next subsection, an optimization technique for ULFM to be more efficient is explained.

Since ULFM provides the way where user program can handle process failure events. It is up to the user program how to handle such situation. Having spare

nodes to replace failed node is considered to be effective~~,~~; however, study on the spare nodes is not active so far. We also focused on this issue and propose how to allocate spare nodes and how failed nodes should be replace by the spare nodes.

### *4.5.1 User-Level Fault Mitigation (ULFM)*

#### 4.5.1.1 Conceptual Design

The main goal of the ~~*User-Level Fault Mitigation*~~ *user-level fault mitigation ULFM* [1, 2, 5] approach remained unchanged over the period of this project: define a minimal set of features to notify application processes about failures, and allow the user to interrupt the normal behavior of his application and to restore communication capability after process failures. Exchanges with developers of MPI applications who envision to introduce resilience in their codes, or who have tried to do so using the ongoing ULFM specification, as well as fruitful discussion in the MPI Forum, mainly inside the Fault Tolerance Working Group and in plenary sessions, have highlighted few opportunities for improvements in the ULFM specification. Ambiguities in the proposed text have been removed ~~,~~ and replaced by a clearer description of the required behavior. The RMA chapter has been entirely rewritten, to account for the changes in the RMA chapter introduced in the version 3 of the MPI standard. The `MPI_Comm_shrink()` and `MPI_Comm_agree()` routines, which are part of the Fault Tolerance extensions, have been simplified, allowing for a more diverse usage. Non-blocking versions of some of these constructs have been also added, to allow for more flexible recovery strategies.

#### 4.5.1.2 Implementation

In same time as improving the concepts design, we wanted to provide a decent implementation that can be used by interested parties for their own research and developments. The first part of the award period was dedicated to providing correct, but non necessary efficient and scalable, support for all concepts. In parallel we advanced on the theoretical side by developing all the necessary tools and algorithms to design and implement more scalable versions of the needed concepts.

More specifically, the last year of the project has been almost entirely focused on advancing on the performance side of the two major operations proposed for the handling of the faults (revocation and agreement) ~~,~~ and on promulgating the extension to the MPI standard to the ~~users~~ user communities. In same time, a significant effort has been done to improve the code quality of the available implementation, in order to provide a stable, efficient, and portable implementation to the user communities.

Similarly to past years, exchanges with developers of MPI applications who envision to introduce resilience in their codes, or who have tried to do so using the ongoing ULFM specification, as well as fruitful discussion in the MPI Forum, inside the Fault Tolerance Working Group and in plenary sessions, have highlighted a few places for improvements in the ULFM specification. The main goal of the ULFM approach remains unchanged: define a minimal set of features to notify application processes about failures, and allow the user to interrupt the normal behavior of his application and to restore communication capability after process failures. Ambiguities in the proposed text have been removed ~~,~~ and replaced by a clearer description of the required behavior. The RMA chapter has been entirely rewritten, to account for the changes in the RMA chapter introduced in the version 3 of the MPI standard. The `MPI_Comm_shrink()` and `MPI_Comm_agree()` routines, which are part of the Fault Tolerance extensions, have been simplified, allowing for a more diverse usage.

**Fig. 4.7** Binomial graph topology

On the implementation front, the `MPI_Comm_revoke()` and `MPI_Comm_agree()` operations have been the focus of our efforts for this year. These two routines are critical to the efficiency of a resilient code ~~,~~ and are by nature costly because they need to be fault-tolerant themselves. `MPI_Comm_revoke()` is provided to the user to notify (asynchronously) all processes belonging to a communicator that an exception happened ~~,~~ and to stop the normal execution flow (e.g., to trigger a collective repair of the communicator). At the conceptual level, it implements a form of reliable broadcast: if one process receives such revoke notification, all processes of the communicator must receive one. Because failures happening during the revoke operation can introduce messages loss, the protocol that implements the revoke operation must be fault-tolerant ~~,~~ and ensure that if one notification is delivered, all notifications are delivered. The first implementation of the revocation was done at the ~~Runtime Environment~~ runtime environment (RTE) level, over the ~~Out-Of-Band~~ out-of-band (OOB) network. This implementation was not fault tolerant and required a large number of messages ($O(N2)$). We redesigned the algorithms integrating research in resilient graph topologies and successfully improved not only the resilience and performance of the revocation algorithm but also the impact on the network infrastructure and the number of messages. The

final algorithm that we will be delivered with ULFM is based on the ~~Binomial Graph~~ binomial graph topology (Fig. 4.7). This graph ~~minimize~~ minimizes the number of messages and the degree of the graph ~~,~~ while providing a strongly resilient topology ~~,~~ that can only be bipartite by the sudden disappearance of more than half of the processes. Moreover, the new algorithm is implemented at the BTL level ~~,~~ and takes advantage of the fast network interconnect available on most of the HPC clusters.

**Fig. 4.8** Example of dynamic topology

The `MPI_Comm_agree()` routine was also relying, for parts of its services (namely, the all-reduce tree maintenance), on messages at the OOB level. The overhead imposed by the OOB mechanism has been hindering the performance and scalability of the agreement. We completely reworked the agreement to use directly the fast network communication infrastructure available in Open MPI (namely, the BTL). This step provided a significant boost in terms of performance of the agreement algorithms. However, this highlighted few implementation issues with quadratic search involved in the original algorithms (due mainly to the MPI semantics of translating ranks between communicators). As a result, we decided to redesign the agreement implementation from the ground up. Dynamic topologies have been introduced, topologies that ~~remains~~ remain optimal as long as no new processes disappear ~~,~~ and that are updated upon a successful completion of an agreement (due to the semantic of the agreement itself, a consensus on the dead processes is established during the agreement, allowing for a consistent global view of the alive processes). This topology is described in the Fig. 4.8. The important change is the coarsest dashed lines, which represent the new connections that are established to cope with the discovery of dead processes (signaled by the next coarsest dashed line). These lines connect a process to the leftmost ancestor still alive ~~,~~ and count for one of the most costly, but critical, ~~operation~~ operations during the agreement. Without going in too much details, we have designed a new consensus algorithm named Early Returning Agreement, with a worst case cost of $\sum f = 1 \ldots F(\log(P - f))$ where $F$ is the number of fault discovered during a single agreement and $P$ is the total number of processes.

Moreover, users reports signaled recurring corner cases in the previous agreement implementations, where the agreement routine was unable to provide its service because of the presence of failures. We wrote a reference implementation

of the agreement, based on an existing Early Termination Agreement protocol, which work in asynchronous all-to-all phases, ensuring a correct result in any possible failure scenario. In addition to this reference implementation, we initiated works on more efficient implementations, which will take advantage of the high resilience property of the ~~Binomial Graph~~binomial graph, to reduce the number of messages while still providing the insurance of safe agreement despite failures in most situations.

### *4.5.2 Spare Node Substitution*

This subsection considers the questions of how spare nodes should be allocated, how to substitute them with faulty nodes, and how much the communication performance is affected by the substitution. The third question stems from the modification of the rank mapping by node substitutions, which can incur additional message collisions. In a stencil computation, rank mapping can be done in a straightforward way on a Cartesian network without incurring any message collisions. However, once a substitution has occurred, such collision-free node-rank mapping can be destroyed. Figure 4.9 shows an example of message collisions (right figure). In this case, the failed node substitution results in having at most ~~5~~ five message collisions. When two ~~message collides~~messages collide, then the latency is doubled. Thus, the number of message collisions should be as low as possible.

**Fig. 4.9** Message collisions on 5P 2D stencil on a Cartesian ~~Network~~ network (X-Y routing)

The number of message collisions depends on the spare node allocation and how the failed node is substituted. Although the substitution idea proposed here can be applied to Cartesian networks having any order, the explanations are mostly using 2D network topology, due to the simplicity.

Figure 4.10 shows how the failed node is substituted in this proposed way. In this example, spare nodes are allocated at the edges of the 2D node space. The *0D sliding* method is the most simple one~~,~~: just replace the failed node with a spare node which is located to the nearest Manhattan distance from the failed node. In the

**Fig. 4.10** 0D, 1D, and @d sliding methods on 2D Cartesian ~~Network~~network

*1D sliding* method, firstly the spare node having the same ~~X-akis~~ X-axis or Y-axis is chosen and the nodes between the failed node and the spare node are shifted. In *2D sliding* method, all nodes having the larger or equal to X-axis or Y-axis location are shifted. In general, the sliding method can be extended up to *ND sliding*, where $N$ is the dimension order of a Cartesian topology network. When a substitution of a failed node happens, the inter-node communication of a stencil computation takes place between the neighbor nodes of the spare node. And the message collisions can happen on the paths between the neighbor nodes and the spare node. This is the reason why the ~~nerest~~ nearest spare node is chosen so that the paths can be minimized.

The ~~highest order~~ highest-order sliding method can be applied to only the number of the order of the network, assuming spare nodes are allocated on every single edge of the network. And this ~~higest order~~ highest-order sliding method increases one hop count, but no collision happens.

AQ6

**Fig. 4.11** Hybrid sliding methods on 2D network

Those substitution methods can be combined, and this is named *hybrid sliding* method as shown in Fig. 4.11. In 2D sliding method, the nodes in the gap which is created by the 2D sliding method can be used as a new spare node set. In this hybrid sliding method, the method having the highest order must be applied. This is ~~becuse the higest order~~ because the highest-order sliding method does not ~~impacts~~ impact the network performance so much. And if a sliding method cannot be applied, then

the ~~lower order~~ lower-order sliding method is applied. This procedure is repeated until the 1D sliding method fails. Since the 0D sliding method has no limitation as long as spare nodes exits, it can be used as a last resort.

We developed a simulator of the proposed sliding methods to count the maximum message collisions and ~~compares~~ compare the performance (message ~~colliskions~~collisions) of the proposed sliding methods. We also evaluated the sliding methods on the K computer, BG/Q, and Tsubame2.5 [7].

## 4.6 Subsequent Development

Although this project was started with the prediction of heterogeneous CPU architecture would be a common many-core architecture, this does not come true. Intel announced ~~standalone~~ stand-alone many-core CPU, Knights Landing (KNL). Although MPVAS software developed in this project became useless for the ~~standalone~~ stand-alone many-core CPUs, the idea of packing multiple processes running on heterogeneous CPUs into one virtual address space is still effective.

The PVAS project is still active. One of the most ~~drawback~~ drawbacks of PVAS was the implementation by the patched Linux, and PVAS was hard to port to the other OSes and architectures. To overcome this, a new user-level implementation has been being developed, named *Process-in-Process (PiP)* [6]. ULP on PVAS is also ~~palnned~~ planned to implement on top of PiP. The other techniques developed in this project excepting MPVAS are applicable for the exa-scale computing. The research on scalable MPI-IO, EARTH, is also going on. The ULFM development team is trying to push ULFM into the MPI standard.

## 4.7 Summary

We proposed PVAS as a new parallel execution model especially for many-core CPU, MPVAS for heterogeneous CPU architectures, ULP to have fast context switching between user-level ~~proecsses~~processes, EARTH for efficient MPI ~~Collective~~ collective I/O, ULFM enhancement, and sliding substitution methods to replace failed nodes with spare nodes. Successor projects are contributing to the exa-scale computing.

## References

1. Bland, W., Bouteiller, A., Herault, T., Bosilca, G., Dongarra, J.: Post-failure recovery of mpi communication capability: Design and rationale. The International Journal of

High Performance Computing Applications **27**(3), 244–254 (2013). https://doi.org/10.1177/1094342013488238. https://doi.org/10.1177/1094342013488238
2. Bouteiller, A., Bosilca, G., Dongarra, J.J.: Plan b: Interruption of ongoing mpi operations to support failure recovery. In: Proceedings of the 22Nd European MPI Users' Group Meeting, EuroMPI '15, pp. 11:1–11:9. ACM, New York, NY, USA (2015). https://doi.org/10.1145/2802658.2802668. http://doi.acm.org/10.1145/2802658.2802668
3. Brightwell, R., Pedretti, K., Hudson, T.: SMARTMAP: Operating System Support for Efficient Data Sharing among Processes on a Multi-Core Processor. In: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, SC '08, pp. 25:1–25:12. IEEE Press, Piscataway, NJ, USA (2008). http://dl.acm.org/citation.cfm?id=1413370.1413396
4. Fukazawa, G.: Multiple PVAS: a systems software for HPC application programs on multi-core and many-core (2014). (Master Thesis, in Japanese)
5. Herault, T., Bouteiller, A., Bosilca, G., Gamell, M., Teranishi, K., Parashar, M., Dongarra, J.: Practical scalable consensus for pseudo-synchronous distributed systems. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '15, pp. 31:1–31:12. ACM, New York, NY, USA (2015). https://doi.org/10.1145/2807591.2807665. http://doi.acm.org/10.1145/2807591.2807665
6. Hori, A., Si, M., Gerofi, B., Takagi, M., Dayal, J., Balaji, P., Ishikawa, Y.: Process-in-Process: Techniques for Practical Address-Space Sharing. In: The 27th International Symposium on High-Performance Parallel and Distributed Computing (HPDC'18). ACM (2018). (to appear)
7. Hori, A., Yoshinaga, K., Herault, T., Bouteiller, A., Bosilca, G., Ishikawa, Y.: Sliding Substitution of Failed Nodes. In: Proceedings of the 22Nd European MPI Users' Group Meeting, EuroMPI '15, pp. 14:1–14:10. ACM, New York, NY, USA (2015). https://doi.org/10.1145/2802658.2802670. http://doi.acm.org/10.1145/2802658.2802670
8. Matsuoka, S.: The road to tsubame and beyond. In: M. Resch, S. Roller, P. Lammers, T. Furui, M. Galle, W. Bez (eds.) High Performance Computing on Vector Systems 2007, pp. 265–267. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
9. Nakashima, J., Taura, K.: MassiveThreads: A Thread Library for High Productivity Languages, pp. 222–238. Springer Berlin Heidelberg, Berlin, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44471-9_10. https://doi.org/10.1007/978-3-662-44471-9_10
10. Pérache, M., Jourdren, H., Namyst, R.: MPC: A Unified Parallel Runtime for Clusters of NUMA Machines. In: Proceedings of the 14th International Euro-Par Conference on Parallel Processing, Euro-Par'08, pp. 78–88. Springer-Verlag, Berlin, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85451-7_9. http://dx.doi.org/10.1007/978-3-540-85451-7_9
11. Reinders, J.: An Overview of Programming for Intel Xeon processors and Intel Xeon Phi coprocessors (2012)
12. del Rosario, J.M., Bordawekar, R., Choudhary, A.: Improved parallel i/o via a two-phase run-time access strategy. SIGARCH Comput. Archit. News **21**(5), 31–38 (1993). https://doi.org/http://doi.acm.org/10.1145/165660.165667
13. Sato, M., Fukazawa, G., Shimada, A., Hori, A., Ishikawa, Y., Namiki, M.: Design of multiple pvas on infiniband cluster system consisting of many-core and multi-core. In: Proceedings of the 21st European MPI Users' Group Meeting, EuroMPI/ASIA '14, pp. 133:133–133:138. ACM, New York, NY, USA (2014). https://doi.org/10.1145/2642769.2642795. http://doi.acm.org/10.1145/2642769.2642795
14. Sato, M., Fukazawa, G., Yoshinaga, K., Tsujita, Y., Hori, A., Namiki, M.: A hybrid operating system for a computing node with multi-core and many-core processors. In: International Journal Advanced mputer Science (IJACSci), vol. 3, pp. 368–377 (2013)
15. Schneider, T., Gerstenberger, R., Hoefler, T.: Micro-Applications for Communication Data Access Patterns and MPI Datatypes. In: Recent Advances in the Message Passing Interface – 19th European MPI Users' Group Meeting, EuroMPI 2012, Vienna, Austria, September 23–26, 2012. Proceedings, vol. 7490, pp. 121–131. Springer (2012)
16. Shimada, A.: A Atsudy on Task Models for High-performance and Efficient Intra-node Communication in Many-core Environments. Ph.D. thesis, Keio University (2017). (in Japanese)

17. Shimada, A., Gerofi, B., Hori, A., Ishikawa, Y.: Pgas intra-node communication towards many-core architecture. In: In PGAS 2012: 6th Conference on Partitioned Global Address Space Programing Model, PGAS'12 (2012)
18. Shimada, A., Gerofi, B., Hori, A., Ishikawa, Y.: Proposing a new task model towards many-core architecture. In: Proceedings of the First International Workshop on Many-core Embedded Systems, MES '13, pp. 45–48. ACM, New York, NY, USA (2013). https://doi.org/10.1145/2489068.2489075. http://doi.acm.org/10.1145/2489068.2489075
19. Shimada, A., Hori, A., Ishikawa, Y.: Eliminating costs for crossing process boundary from mpi intra-node communication. In: Proceedings of the 21st European MPI Users' Group Meeting, EuroMPI/ASIA '14, pp. 119:119–119:120. ACM, New York, NY, USA (2014). https://doi.org/10.1145/2642769.2642790. http://doi.acm.org/10.1145/2642769.2642790
20. Shimada, A., Hori, A., Ishikawa, Y., Balaji, P.: User-level process towards exascale systems **2014**(22), 1–7 (2014). http://ci.nii.ac.jp/naid/110009850784/
21. Shimosawa, T., Gerofi, B., Takagi, M., Shirasawa, T., Shimizu, M., Hori, A., Ishikawa, Y.: Interface for Heterogeneous Kernels: A Framework to Enable Hybrid OS Designs Tergeting High Performance Computing. In: IEEE International Conference on High Performance Computing (HiPC). IEEE (2014)
22. Thakur, R., Lusk, E., Gropp, W.: Users Guide for ROMIO: A High-Performance, Portable MPI-IO Implementation. Technical Memorandum ANL/MCS-TM-234, Argonne National Laboratory (2004)
23. Tsujita, Y., Hori, A., Ishikawa, Y.: Locality-aware process mapping for high performance collective mpi-io on fefs with tofu interconnect. In: Proceedings of the 21th European MPI Users' Group Meeting, pp. 157–162. ACM (2014). https://doi.org/10.1145/2642769.2642799. Challenges in Data-Centric Computing
24. Tsujita, Y., Hori, A., Kameyama, T., Uno, A., Shoji, F., Ishikawa, Y.: Improving collective MPI-IO using topology-aware stepwise data aggregation with I/O throttling. In: Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region, pp. 12–23. ACM (2018). https://doi.org/10.1145/3149457.3149464
25. Yokokawa, M., Shoji, F., Uno, A., Kurokawa, M., Watanabe, T.: The K Computer: Japanese Next-Generation Supercomputer Development Project. In: Proceedings of the 17th IEEE/ACM international symposium on Low-power electronics and design, ISLPED '11, pp. 371–372. IEEE Press, Piscataway, NJ, USA (2011). http://dl.acm.org/citation.cfm?id=2016802.2016889

AUTHOR QUERIES

AQ1. Please check if author affiliation details are okay.
AQ2. Please check if edit to latter part of sentence starting “Our proposed solution…” is okay.
AQ3. Please check if edit to sentence starting “However, during communication…” is okay.
AQ4. Please check if inserted citation for “Fig. 4.1” is okay.
AQ5. Please check if edit to sentence starting “Since datatype information…” is okay.
AQ6. Please check if edit to sentence starting “The highest-order sliding…” is okay.