I/O Performance of the SX-Aurora TSUBASA

Mitsuo Yokokawa
Graduate School of System Informatics
Kobe University
Kobe, Japan
yokokawa@port.kobe-u.ac.jp

Ayano Nakai
Graduate School of System Informatics
Kobe University
Kobe, Japan
1695093t@stu.kobe-u.ac.jp

Kazuhiko Komatsu
Cyberscience Center
Tohoku University
Sendai, Japan
komatsu@tohoku.ac.jp

Yuta Watanabe, Yasuhisa Masaoka, Yoko Isobe
AI platform Division
NEC Corporation
Minato-ku, Tokyo, Japan
{*yuta-watanabe-jt,ymsk-bx,y-isobe-pi*}*@nec.com*

Hiroaki Kobayashi
Graduate School of Information Sciences
Tohoku University
Sendai, Japan
koba@tohoku.ac.jp

***Abstract*—File outputs or checkpoints for intermediate results frequently appear at appropriate time intervals in large-scale time-advancement numerical simulations where they are utilized for simulation post-processing and/or for restarting consecutive simulations. However, file input/output (I/O) for large-scale data often takes excessive time due to bandwidth limitations between processors and/or secondary storage systems like hard disk drives (HDDs) and solid state drives (SSDs). Accordingly, efforts are ongoing to reduce the time required for file I/O operations in order to speed up such simulations, which means it is necessary to acquire advanced I/O performance knowledge related to high-performance computing systems used.**

In this study, I/O performance with respect to the connection bandwidth between the vector host (VH) server and the vector engines (VEs) for three configurations of the SX-Aurora TSUBASA supercomputer system, specifically the A300-2, A300-4, and A300-8 configurations, were measured and evaluated. The accelerated I/O function, which is a distinctive feature of the SX-Aurora TSUBASA I/O system, was demonstrated to have excellent performance compared to its normal I/O function.

***Keywords*-I/O performance; Accelerated I/O function; SX-Aurora TSUBASA; Vector supercomputer; DMA engine**

I. Introduction

Computer simulations, which are powerful tools for solving various problems, are now commonly used in a wide variety of science and engineering research fields, including weather/climate predictions and artificial structure designs. Such simulations usually require extremely long elapsed real times (wall-clock times) before meaningful and useful results can be obtained, even if cutting-edge high-performance supercomputer systems are used and also parallel computation on them can be applied to simulations.

In such situations, intermediate calculated results obtained by the simulations are often written to permanent files on electronic storage devices such as hard disk drives (HDDs) and/or magnetic tape libraries. Those results can be used to confirm that the simulations are proceeding as expected, and to prepare for consecutive batch jobs that will be chained to the final simulated time point.

However, since recent computer simulations treat complicated and large-size problems, and resulting in a lot of program variables, i.e. large memory capacity to be assigned, the intermediate file volume, which is output by such simulations, are often huge. In addition, since parallel computations on supercomputers generate the same number of logical files as the degree of parallelism of the simulation, they produce large numbers of files and vast total file volumes. This is true even if a parallel file system is used, and it is expected that such conditions will only be exacerbated in future advancements.

Because of the above, input/output (I/O) performance from/to sequential unformatted files is a topic of significant concern, and efforts are ongoing to decrease the related time requirements so that total simulation wall-clock times are as short as possible. Several intensive studies on parallel I/O systems and its benchmarking tools have been carried out so far, for example E. C. Inacio et al. [1], [2]. However, I/O performance evaluation for new architecture systems should be done.

Recently, the SX-Aurora TSUBASA vector-type supercomputer was introduced to the public [3], [4]. Although the ratio of vector-type supercomputers to the overall number of supercomputers in the TOP500 raking list is low, vector-type supercomputers have exceptionally large memory bandwidth, which allows them to achieve higher sustained performance for a variety of applications such as fluid dynamics. Furthermore, although the performance of the SX-Aurora TSUBASA was previously evaluated by Komatsu et al. [5], to the best of our knowledge, no sufficient evaluations that focus on its I/O performance have been performed.

Accordingly, in this paper, we report on I/O performance measurements of sequential unformatted files for three configurations of the SX-Aurora TSUBASA. In addition to the

978-1-7281-7445-7/20/$31.00 ©2020 IEEE
DOI 10.1109/IPDPSW50202.2020.00014

normal I/O function, scalability issues on the **accelerated I/O function** that has been implemented in the new architecture of the SX-Aurora TSUBASA are discussed.

The rest of this paper is organized as follows. Section II provides an overview of the SX-Aurora TSUBASA and its two I/O function types. Experiment environments such as system configurations and the pseudo-codes used in the measurements are described in Section III. The measurement results, including I/O performance comparisons of the three configurations are discussed in Section IV. The concluding remarks are given in Section V.

II. I/O FUNCSIONS ON SX-AURORA TSUBASA

A. Overview of SX-Aurora TSUBASA

The SX-Aurora TSUBASA is a vector-type supercomputer that has a distinguished system architecture compared to previous SX-series supercomputers [6], [7]. The system consists primarily of a vector host (VH) and one or more vector engines (VEs). The VH is a standard server with one or two x86 architecture processors on which a standard Linux operating system (OS) is operated. A VE is implemented as a Peripheral Component Inteconnect Express (PCIe) card, on which a vector processor is mounted with six high-bandwidth memory (HBM2) modules. The vector processor consists of eight vector cores, a 16 MB last-level-cache (LLC), and a VE direct memory access (DMA) engine. The LLC is connected to each core through a two-dimensional (2D) intra-network with a total cache bandwidth of 3.0 TB/s.

Design concepts related to how operations are managed on the SX-Aurora TSUBASA are worth mentioning. For example, OS functions have been realized on the VH to the greatest extent possible so that the VEs are able to effectively utilize their computational power by concentrating program executions with vector instructions.

Therefore, in the VH, a minimum required function, which only manages the jobs running on the VEs, is newly introduced on top of the Linux OS in its software stack. A VEOS is a light software module that consists of a *ve_exec* command and *VEOS services*. The *ve_exec* command loads a VE program from memory in the VH, puts it into the memory of the VE, creates a VE process on the VE as well as a corresponding pseudo-process on the VH, and then kick-starts the program. The *VEOS services* provide tasks that bridge the system calls invoked in the programs running on the VE to the OS on the VH. It also handles interrupts and instruction exceptions that occur during program execution on the VE.

Since the GNU C library (*glibc*) is ported onto the VEs, applications can call the *glibc* functions such as I/O functions normally. Hence, applications do not need to change the way their programs operate; they must only be recompiled to generate execution binary codes using the VE instruction sets. Of course, program vectorization should also be applied to achieve higher computation performance.

To execute a VE program on the SX-Aurora TSUBASA, regardless of whether it is a multi-thread and/or a multi-process program, it is necessary to issue a execution command on the VH.

B. Implementation of I/O functions on SX-Aurora TSUBASA

The SX-Aurora TSUBASA has two I/O function types, normal and accelerated I/O functions as described in Figure 1. In the normal I/O, when a user process on the VE needs to use an I/O system call, the VEOS pins down a 4 KB temporary buffer on the memory of the corresponding pseudo-process running on the VH (hereafter referred to as *VH buffer*). At the same time, a buffer on the memory of the VE (hereafter referred to as *VE buffer*) is assigned by the *glibc* running on the VE. This task requires an address translation between virtual and physical addresses on the VH. Next, the VE DMA engine moves data from/to the VE buffer to/from the VH buffer. After the I/O process is finished, the VEOS releases the VH buffer.

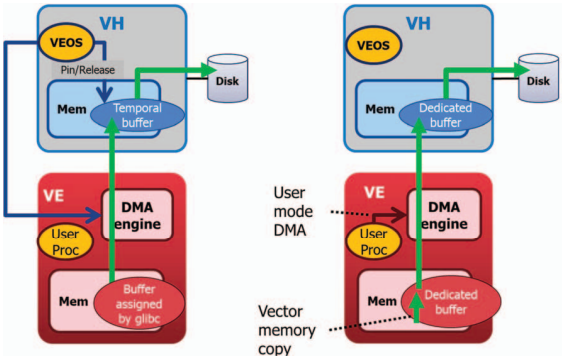

Figure 1. Normal I/O and Accelerated I/O

When a VE program invokes an I/O system call, memory allocation and de-allocation operations are required on the VH. These operations (memory pinning-down and releasing) are essential to prevent paging out a Linux memory page and to retain the page so that it can be accessed by the VE DMA engine while the system call is processed.

Since there is a latency to start/finish the system call in those operations, they take numerous clock cycles compared to those that operate only on the VH. In addition, since these operations for pinning down and releasing the VH buffer are executed at every I/O system call in the normal I/O function, they might lead to unacceptable overhead if system calls are processed multiple times in user application programs.

Therefore, the memory area accessed by the VE DMA engine should be fixed during program execution in order

to obtain better I/O performance. To realize a fixed memory area, an accelerated I/O function was developed to pin down the memory area used for data transfer. The VEOS is not involved in transferring the data to be read/written when the accelerated I/O function is used. Instead, the following process is used.

First, once the use of the accelerated I/O function is declared at the beginning of a job, a dedicated VE buffer of 8 MB per thread and a dedicated VH buffer of 64 MB per pseudo-process are assigned on the memory areas. When a user process needs a system call for I/O, the VE DMA engine moves data from/to the dedicated VE buffer to/from the dedicated VH buffer directly. In this case, it is not necessary to pin down and/or release the VH buffer for every I/O operation. Therefore, the accelerated I/O is expected to be faster than the normal I/O. The accelerated I/O function can be used simply by setting an environmental variable `VE_LD_PRELOAD` as `VE_LD_PRELOAD=libveaccio.so.1` prior to execution of the programs which contains I/O statements.

III. Measurement Environments

A. Configurations of A300-2, A300-4, and A300-8

In this study, three configurations of the SX-Aurora TSUBASA, A300-2, A300-4, and A300-8 (shown in Figure 2), were used for measurements.

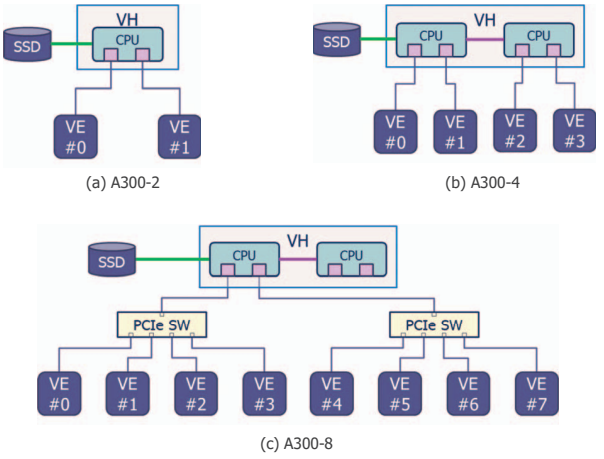

Figure 2. Three configurations of SX-Aurora TSUBASA A300 series

The A300-2 configuration, which is shown in Figure 2(a), consists of a VH with an Intel Xeon Gold 6126 (Skylake architecture) central processing unit (CPU) and two VEs (`VE #0` and `#1`). Each VE is connected to one of the PCIe (16 lanes) ports of the CPU. A secondary storage SSD is also connected to the CPU.

Both of the VHs of the A300-4 and A300-8 configurations, which are shown in Figures 2(b) and (c), have two Xeon CPUs that are connected by three ultra-path connect links to each other. The A300-4 configuration consists of a VH and four VEs (`VE #0` to `#3`), and each VE is connected to one port of the four PCIe ports of the two CPUs as shown in Figure 2(b). The A300-8 configuration consists of a VH, eight VEs (`VE #0` to `#7`), and two PCIe switches (PCIe SWs) as depicted in Figure 2(c). Each PCIe SW is connected to one of the two PCIe ports of a CPU. The secondary storage consisting of solid state drives (SSDs)

Pseudo-code 1 Measurement code written in Fortran

```
program
use mpi
real(kind=8) :: var(N)          ← N is determined
according to file size.
    — Initialization —
call file_read(myrank, var)
call file_write(myrank, var)
    — Finalization —
end program
```

```
subroutine file_read(myrank, var)
open(ur, file="foo.myrank",format="unformatted")

                                ← MPI_Barrier() &
Timer starts
read(ur) var
                                ← MPI_Barrier() &
Timer ends
end subroutine
```

```
subroutine file_write(myrank, var)
open(uw, file="boo.myrank",format="unformatted")

                                ← MPI_Barrier() &
Timer starts
write(uw) var
flush(uw)
                                ← MPI_Barrier() &
Timer ends
end subroutine
```

Pseudo-code 2 Measurement code written in C

```
void file_read_(int *myrank, double *var){
FILE *fd;
fd=fopen( "foo.myrank", "rb");

                                ← MPI_Barrier() &
Timer starts
fread(var, sizeof(double), N, fd);
                                ← MPI_Barrier() &
Timer ends
}
```

```
void file_write_(int *myrank, double *var){
FILE *fd;
fd=fopen( "foo.myrank", "wb");

                                ← MPI_Barrier() &
Timer starts
fwrite(var, sizeof(double), N, fd);
fflush( fd );
                                ← MPI_Barrier() &
Timer ends
}
```

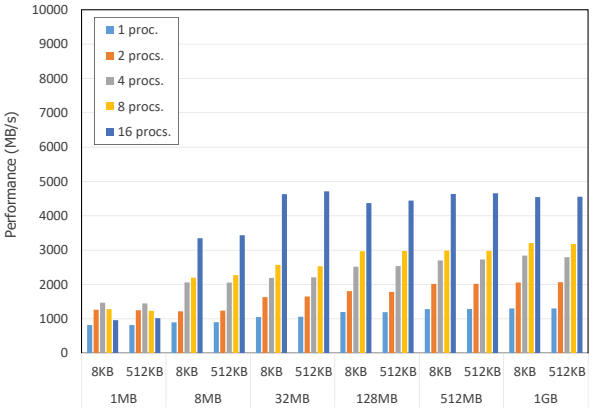

Figure 3. Read performance of A300-2 configuration in Fortran code

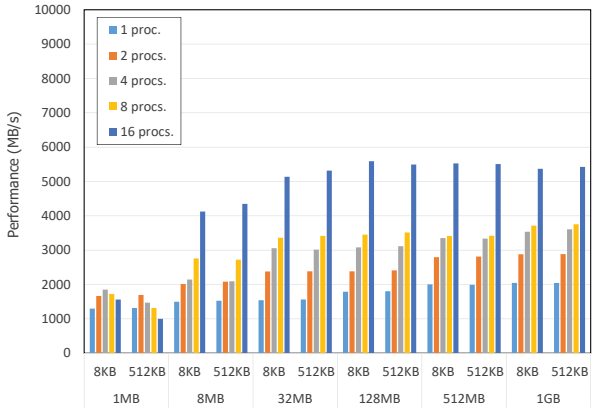

Figure 4. Read performance of A300-2 configuration in C code

connected to the VH is used for a file system.

Every CPU on the VH has a maximum of 48 PCIe lanes, of which 16 lanes are bundled for a VE connection whose theoretical and sustained bandwidth levels are approximately 15.75 GB/s and 10.0 to 12.0 GB/s, respectively. The A300-4 configuration has four PCIe connections between the VH and the VEs, which is the highest of all three configurations. The sustained bandwidth between the VH and VEs of the A300-4 is expected to achieve approximately 40.0 GB/s.

B. Measurement programs

Two Message Passing Interface (MPI) programs were implemented for the measurement as described in **Pseudo-codes 1** and **2**. The main program (written in Fortran) creates a number of MPI processes according to the command line parameter that indicates the number of MPI processes to be invoked at a job start. The MPI ranks were bound to the cores and VEs as the ascending order of the core and VE number. Each MPI process calls the same sub-program, which accesses a sequential unformatted file (i.e. a binary file) when it is necessary to read and write. **Pseudo-codes 1** and **2** for the sub-programs are written in Fortran and C, respectively. These codes were compiled by using the compilers `nfort v2.4.1` and `ncc v2.4.1` with an optimization flag `-O3`.

Several file sizes from 8 KB to 1GB and three I/O buffer sizes of 8 KB, 64 KB, and 128 KB assigned by *glibc* were used for the measurements. Though the performance values for the files whose sizes were less than 1MB were low, those sizes are not necessary for discussions on performance of the bandwidth between the VH and VEs. And differences with respect to the buffer sizes were not observed. Thus, the file sizes of 1 MB, 8 MB, 32 MB, 128 MB, 256 MB, and 1 GB and two I/0 buffer sizes of 8 and 512 KB were taken in our performance analyses, Finally, the number of measurements was set at 12 cases for each code.

Here, it should be noted that since each VE socket has eight cores and each core can be assigned an MPI process, up to 16, 32, and 64 MPI processes can be created on the A300-2, A300-4, and A300-8 configurations, respectively.

The number of measured I/O time values was the same number of MPI processes, and a total 10 measurements were carried out. Therefore, we obtained the measured values of 10 multiplied by the number of MPI processes for each case. For example, 16 measured I/O time values were obtained via a measurement of 16-MPI-process execution, and 10 measurements were carried out. Consequently, a total of 160 measured values were obtained for the 16-MPI-process execution. All the measured values for each case were then averaged to obtain the final value as an I/O performance value for each case.

It is noted that, in the experiments, we focus on the time when data move between the VH buffer and VE memory, not the time for reading from and writing to the disk drives.

IV. Measurement Results

A. Comparison of normal I/O performance between Fortran and C codes on A300-2 configuration

Figures 3 and 4 show the read performance values using the normal I/O function for **Pseudo-codes 1** and **2**, respectively. For the Fortran code, approximately 4.6 GB/s read performance was achieved for cases with 16 or more MPI processes and 32 KB data. No significant differences were observed between cases with 8 and 256 KB buffer sizes. For the C code, more than 5 GB/s performance was obtained for cases with 16 or more MPI processes and 32 KB data. The values for the VH buffer size of 8 KB are slightly smaller than those of the 256 KB buffer size for all C code cases.

In case of 1 MB data, lower read performance of around 2 MB/s was observed. In addition, we found that the performance decreased as the number of MPI processes increased. It appears that the latency for processing a system

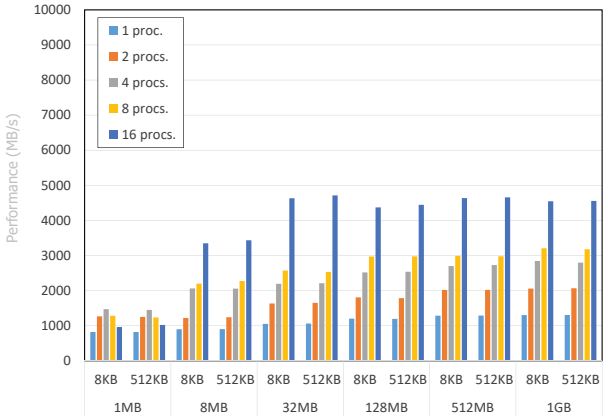

Figure 5. Write performance of A300-2 configuration in Fortran code

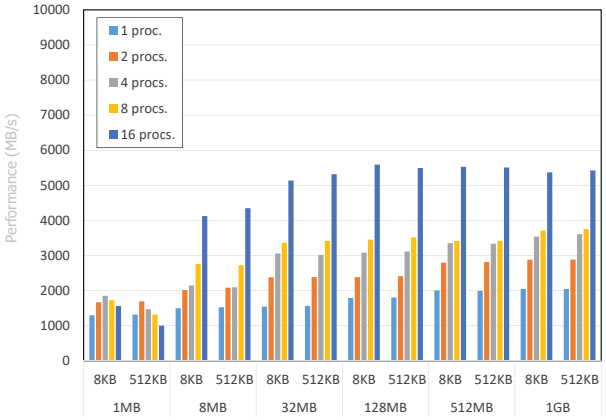

Figure 6. Write performance of A300-2 configuration in C code

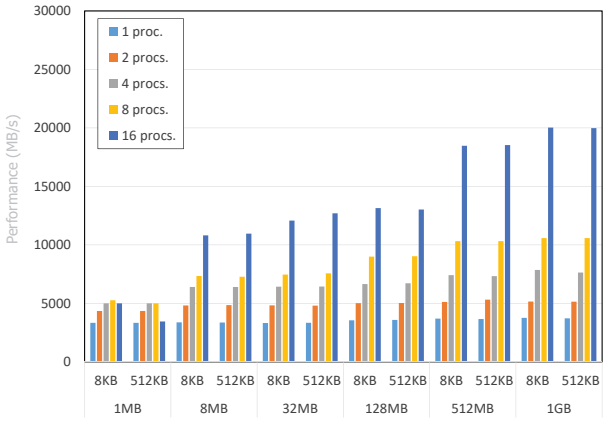

Figure 7. Read performance of A300-2 configuration in C code with accelerated I/O

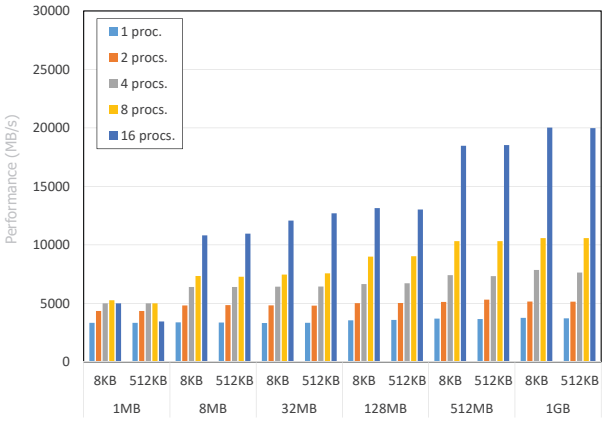

Figure 8. Write performance of A300-2 configuration in C code with accelerated I/O

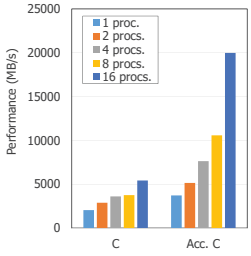

Figure 9. Read performance of A300-2 configuration for 1 GB file and 512 KB buffer by using the normal and accelerated I/O

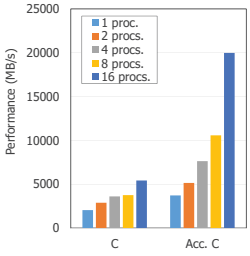

Figure 10. Write performance of A300-2 configuration for 1 GB file and 512 KB buffer by using the normal and accelerated I/O

call to move data between the VH and VEs was observed and resulted in the lower performance.

Figures 5 and 6 show the write performance values using the normal I/O set up for **Pseudo-codes 1** and **2**, respectively. Here, it can be seen that almost 5 GB/s write performance was achieved for cases with 16 or more MPI processes and 32 KB data using the Fortran code, and that a greater than 5 GB/s write performance was obtained using the C code for cases with 16 or more MPI processes and 32 KB data.

For all cases, performance levels by eight MPI processes were not scalable when compared to the levels achieved by four MPI processes. This is because 8 MPI read/write operations are the most that can be applied to the eight cores of a VE socket, and it appears that processing of system calls from eight cores creates contentions on some parts in a socket.

We observed consistently that the read/write performance values measured by the C codes were superior to those measured by the Fortran code on the A300-2, A300-4, and A300-8 configurations, because the C code can call *glibc* directly. On the other hand, an interface code bridging from the Fortran language to *glibc* is called when executing I/O Fortran statements such as `READ` and `WRITE`, and an additional time is required for the execution. Therefore, we

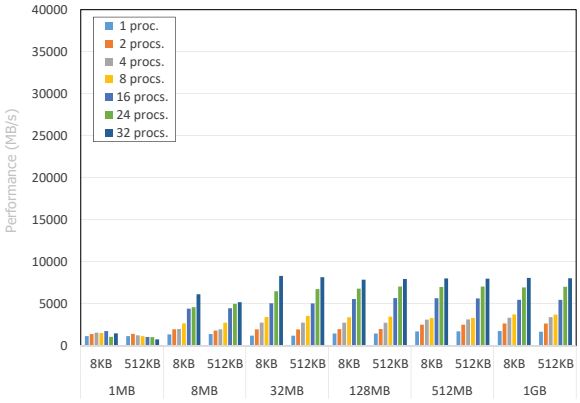

Figure 11. Read performance of A300-4 configuration in C code with normal I/O

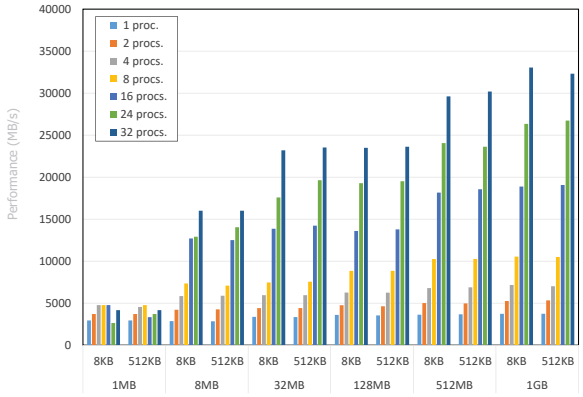

Figure 12. Read performance of A300-4 configuration in C code with accelerated I/O

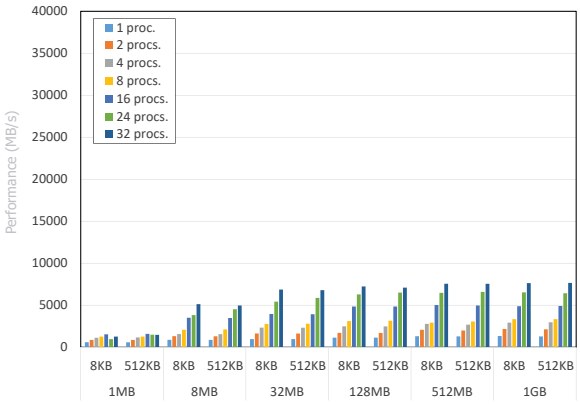

Figure 13. Write performance of A300-4 configuration in C code with normal I/O

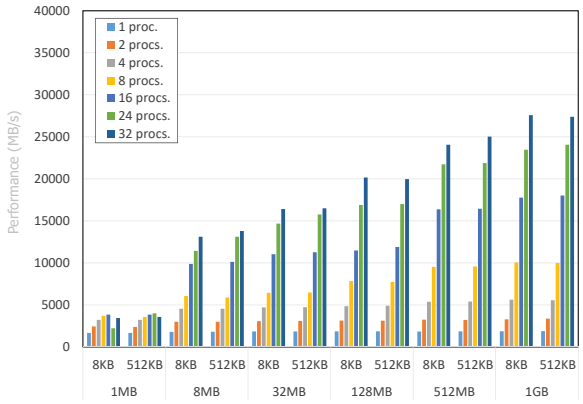

Figure 14. Write performance of A300-4 configuration in C code with accelerated I/O

will discuss results for the C code in the rest of the paper.

B. Comparison of normal and accelerated I/O performance levels

1) Results for A300-2 configuration: Figures 7 and 8 show the read and write performance values when using the accelerated I/O for the C code on A300-2 configuration, respectively. Here, it can be seen that the performance level increased as the file size increased. Approximately 20 GB/s performance was obtained for the case of a 1 GB file size and 512 KB buffer size with 16 MPI processes. Since two PCIe ports are used for data transfer, this performance measurement is close to the maximum sustained value of the two PCIe connections among VH and two VEs. There were no differences with respect to the buffer size differences.

Read/write performance values using the normal and accelerated I/O types for 1 GB file size and 512 KB buffer size are compared as shown in Figures 9 and 10. Here, it can be seen that, in the case of 16 MPI processes, the accelerated I/O had performance levels approximately four times those of the normal I/O.

2) Results for A300-4 configuration: Figures 11 and 12 show the read performances achieved by the C code for the normal and accelerated I/O types, respectively, on the A300-4 configuration. It is noted that the same vertical scale is taken in the both figures.

Here, it can be seen that a read performance of approximately 8 GB/s was obtained for the normal I/O in cases with more than 32 MB files and 32 MPI processes. In contrast, read performance in using the accelerated I/O increased gradually as the file size increased, and 32.3 GB/s read performance was obtained with 32 MPI processes for the case of 1 GB file and 512 KB buffer. However, a scalable performance increase for 32 MPI processes compared to 16 MPI processes was not observed.

Figures 13 and 14 show the write performance by the C code for both the normal and accelerated I/O types, respectively. These figures show almost 8 GB/s write performance was obtained for files larger than 512 MB by the normal I/O. They also show that performance using the accelerated

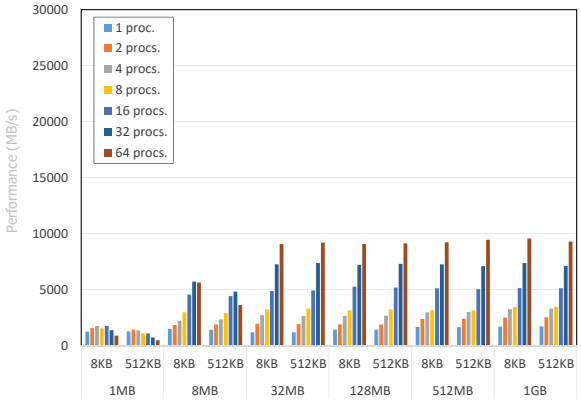

Figure 15. Read performance of A300-8 configuration in C code with normal I/O

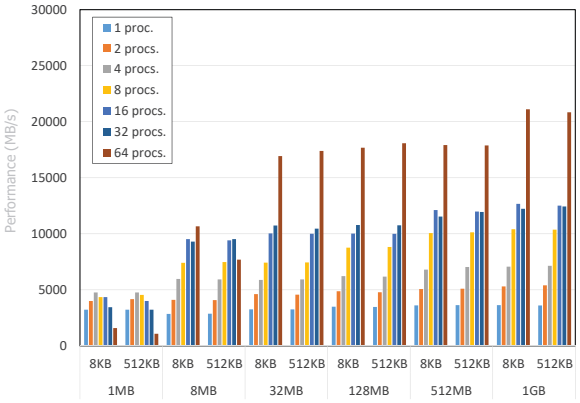

Figure 16. Read performance of A300-8 configuration in C code with accelerated I/O

I/O was gradually increased as the file size increased, and that 27.4 GB/s write performance was obtained with 32 MPI processes for case of a 1 GB file and 512 KB buffer. However, no significant differences were found between buffer sizes for each file size.

Since, as stated in Section III-A, there are four PCIe connections between the VH and four VEs, we expected that approximately 40.0 GB/s read/write performance should be achieved, which is the total sustained performance for four PCIe connections. However, the actual measured performance level was less than the expected value, apparently because the bandwidth produced by the three ultra-path connection links between two CPUs or the I/O performance of the SSDs is affected somehow. Further studies will be needed to clarify this point.

3) Results for A300-8 configuration: Figures 15 and 16 show the read performance values by the C code for the normal and accelerated I/O types, respectively, on the A300-8 configuration. Here, we can see that approximately 9 GB/s read performance was obtained for cases involving over 32 MB files with 64 MPI processes when the normal I/O was used. We can also see that the maximum performance of approximately 20 GB/s was achieved for the case of 1 GB file and 512 KB buffer with 64 MPI processes, because all the VEs are connected to the VH via two PCIe SWs, and the sustained bandwidth between the VH and PCIe SWs is almost 20 GB/s, that is close to the upper bandwidth limit for the A300-8 configuration.

It is found that the performance values for the cases of the 16 and 32 MPI processes were almost the same. Additionally, approximately a 12.5 GB/s bandwidth was obtained for the cases of 1 GB file and 512 KB buffer. Since MPI processes mapping to the VEs conducted in the order of the smallest to largest VE ID (`VE #0` to `#3`) in these experiments, one PCIe SW provides the only measurement path to the VH. This resulted in a maximum obtained

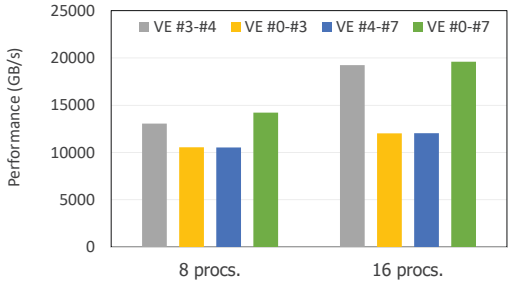

Figure 17. Comparison of read performance levels using different MPI mapping to VEs

bandwidth value for a 10 GB/s sustained PCIe connection. To clarify whether the PCI SW creates a bottleneck that prevents us from obtaining scalable performance from 16 to 32 MPI processes, we carried out an additional experiment in which the MPI processes were divided into two equal groups and each group was assigned to VEs in a way that ensured the route from the VEs to the VH can pass both the two PCIe SWs.

The results are shown in Figure 17. Here, we can see that (as depicted in the first gray bar from the left), approximately 20 GB/s was obtained for 16 MPI processes. In this experiment, a group of eight MPI processes was mapped to `VE #3`, which is connected to the left PCIe SW in Figure 2(c), whereas the other group was mapped to `VE #4`, which is connected to the right PCIe SW. Therefore, both PCIe SW paths were used.

On the other hand, approximately a 10 GB/s read performance value was obtained for the cases in which all 16 MPI processes are mapped to the VEs that connect either the right or the left PCIe SW (as depicted by the second yellow and the third blue bars from the left), since only one of PCIe connections between the PCIe SWs and the VH was

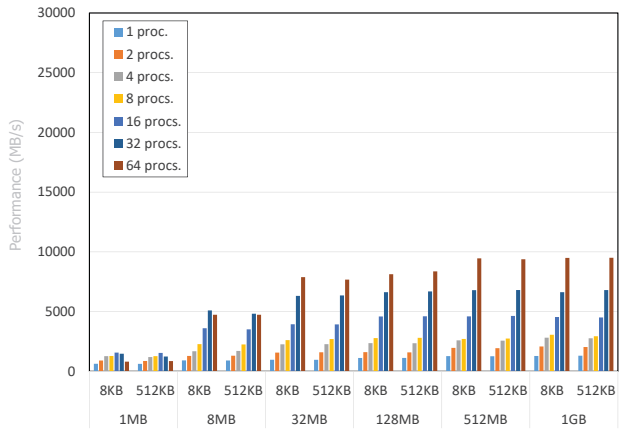

Figure 18. Write performance of A300-8 configuration in C code with normal I/O

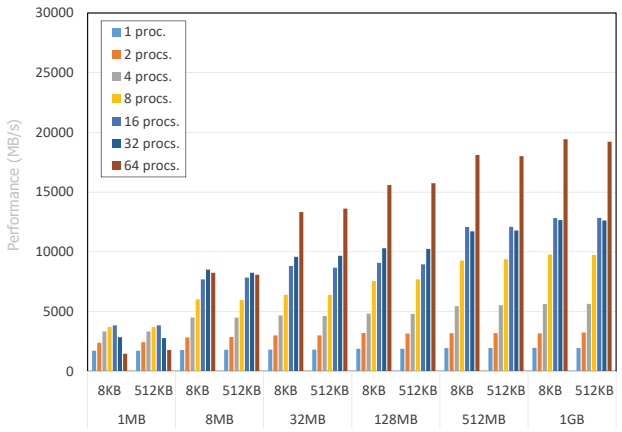

Figure 19. Write performance of A300-8 configuration in C code with accelerated I/O

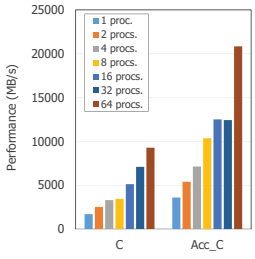

Figure 20. Read performance of A300-8 configuration for 1GB file and 512KB buffer by using the normal and accelerated I/O

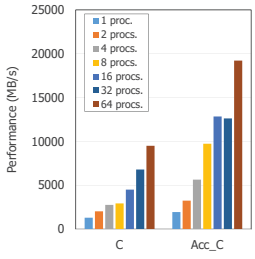

Figure 21. Write performance of A300-8 configuration for 1GB file and 512KB buffer by using the normal and accelerated I/O

used for all cases.

Figures 18 and 19 show the write performance achieved by the C code for the normal and accelerated I/O, respectively, on the A300-8 configuration. Here, we can see that the write performance was almost the same as the read performance.

Figures 20 and 21 show comparisons between the normal I/O and accelerated I/O types in the case of 1 GB file size and 512 KB buffer. Here, we can see that the accelerated I/O achieved significantly higher read and write performance levels than the normal I/O. In fact, approximately 20 GB/s performance was obtained for the accelerated I/O in the case of 64 MPI processes. This sustained value is close to the limit of two PCI-e interfaces of A300-8 configuration and shows that good scalability was obtained as the number of MPI processes increased.

C. Comparison of three configurations

Read and write performance levels for the three configurations using the normal I/O and accelerated I/O types for 1 GB file size and a 512 KB buffer size in the C code are shown all together in Figures 22 and 23, respectively. Here, we can see that the performance obtained using the accelerated I/O were better than those obtained using the normal I/O.

It is also found that almost the sustained PCIe bandwidths are obtained for any of configurations by using the accelerated I/O function, when the data size is large and the number of MPI processes is large. As expected, the A300-4 configuration, which has four PCIe connections between the VH and four VEs, was found to achieve the highest performance level using the accelerated I/O. More specifically, a 32.5 GB/s read performance was obtained by 32 MPI processes for a 1 GB file size and a 512 KB buffer.

V. Conclusion

I/O performance from/to sequential unformatted files is a matter of significant concern, and efforts are ongoing to reduce simulation wall-clock time, including I/O time, as much as possible. However, since performance levels depend on the configuration of the high-performance computing systems to be used, careful I/O performance evaluations are necessary to recognize system characteristics and optimal code implementations should be chosen.

This paper clarified the I/O performance of the SX-Aurora TSUBASA, with particular attention to the accelerated I/O function that is dedicatedly implemented by using the VE DMA engine installed in the VE. Three SX-Aurora TSUBASA A300 configurations have been used in our experiments, and the results showed that the I/O performance levels increased as the number of MPI processes and file sizes increased, and that the performance provided by the accelerated I/O is significantly better than that provided by the normal I/O. In fact, A 32.5 GB/s read performance level is achieved by using the accelerated I/O function with 64 MPI processes for 1 GB file size and 512 KB buffer on the A300-4. In order to use the accelerated I/O function, users simply set an environmental variable as specified.

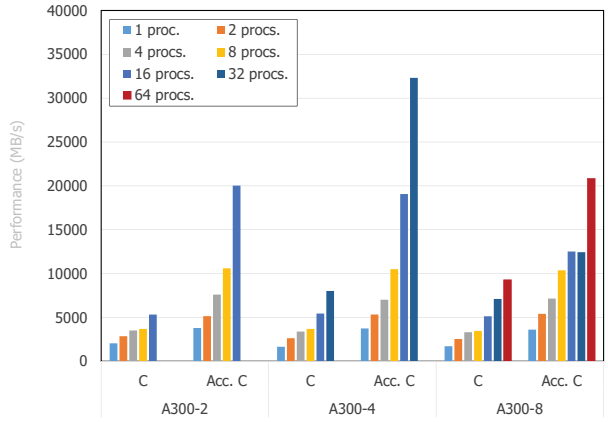

Figure 22. Comparison of read performance of A300 configuration in C code

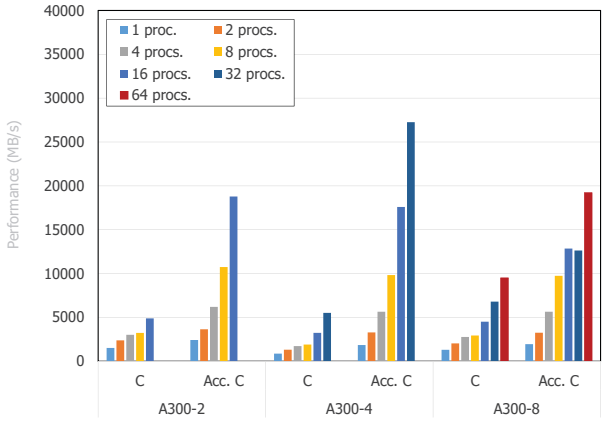

Figure 23. Comparison of write performance of A300 configuration in C code

As expected, the performance level of the A300-4 configuration, which has the largest number of the PCIe connections between the VH and VEs, was the highest of the three. It was also found that the mapping of MPI processes to VEs is important for improving the performance on the A300-8 configuration because two PCIe connections between the VH and the PCIe SWs, each of which has four connections with four VEs, are configured.

It is observed that the larger the data size to be read/written are, the greater the performance of the accelerated I/O obtained on any configuration is. Therefore, data size should be carefully chosen to achieve higher performance. Program modification in I/O parts of applications by using C functions such as `fread` and `fwrite` is preferable even if the original code was written in Fortran.

In general, however, the entire file size of a realistic simulation program to be output as checkpoints is fixed according to the problem size to be solved, and the output data capacity from simulation codes is usually partitioned into multiple files based on the degree of parallelism, even though the partitioned data are kept together as a parallel file. This means the file size to be read and written in each MPI process becomes smaller.

In these experiments, we set all file sizes the same so that system performance could be revealed. From the results obtained, we conclude that an appropriate file size should be determine carefully according to the system configuration and the degree of parallelism for realistic simulation problems.

ACKNOWLEDGMENT

This work is supported partially by MEXT Next Generation High-Performance Computing Infrastructure and Applications R&D Program, entitled R&D of a Quantum-Annealing-Assisted Next Generation HPC Infrastructure and Its Applications.

REFERENCES

[1] E. C. Inacio, P. A. Barbetta, and M. A. Dantas, "A statistical analysis of the performance variability of read/write operations on parallel file systems," *Procedia Computer Science*, vol. 108, pp. 2393 – 2397, 2017, international Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland.

[2] E. C. Inacio and M. A. Dantas, "An i/o performance evaluation tool for distributed data-intensive scientific applications," in *Proceedings of the Latin America Data Science Workshop*, 08 2018, pp. 1–8.

[3] NEC Corporation, "NEC releases new high-end HPC product line, SX-Aurora TSUBASA," 2017. [Online]. Available: https://www.nec.com/en/press/201710/global_20171025_01.html

[4] ——, "NEC SX-Aurora TSUBASA - Vector Engine," 2018. [Online]. Available: https://www.nec.com/en/global/solutions/hpc/sx/vector-engine.html

[5] K. Komatsu, S. Momose, Y. Isobe, O. Watanabe, A. Musa, M. Yokokawa, T. Aoyama, M. Sato, and H. Kobayashi, "Performance Evaluation of a Vector Supercomputer SX-Aurora TSUBASA," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, ser. SC '18. Piscataway, NJ, USA: IEEE Press, 2018, pp. 54:1–54:12.

[6] R. Egawa, K. Komatsu, S. Momose, Y. Isobe, A. Musa, H. Takizawa, and H. Kobayashi, "Potential of a modern vector supercomputer for practical applications: performance evaluation of SX-ACE," *The Journal of Supercomputing*, vol. 73, no. 9, pp. 3948–3976, Sep 2017. [Online]. Available: https://doi.org/10.1007/s11227-017-1993-y

[7] T. Soga, A. Musa, Y. Shimomura, R. Egawa, K. Itakura, H. Takizawa, K. Okabe, and H. Kobayashi, "Performance evaluation of NEC SX-9 using real science and engineering applications," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, ser. SC '09, 2009, pp. 28:1–28:12.