DOI: 10.14529/jsfi200401

Effects of Using a Memory Stalled Core for Handling MPI Communication Overlapping in the SOR Solver on SX-ACE and SX-Aurora TSUBASA

Takashi Soga[1], *Kenta Yamaguchi*[1], *Raghunandan Mathur*[2], *Osamu Watanabe*[3], *Akihiro Musa*[3,4], *Ryusuke Egawa*[4], *Hiroaki Kobayashi*[4]

Modern high-performance computing (HPC) systems consist of a large number of nodes featuring multi-core processors. Many computational fluid dynamics (CFD) codes utilize a Message Passing Interface (MPI) to exploit the potential of such systems. In general, the MPI communication costs increase as the number of MPI processes increases. In this paper, we discuss performance of the code in which a core is used as a dedicated communication core when the core cannot contribute to the performance improvement due to memory-bandwidth limitations. By using the dedicated communication core, the communication operations are overlapped with computation operations, thus enabling highly efficient computation by exploiting the limited memory bandwidth and idle cores. The performance evaluation shows that this code can hide the MPI communication times of 90% on the supercomputer SX-ACE system and 80% on the supercomputer SX-Aurora TSUBASA system, and the performance of the successive over-relaxation (SOR) method is improved by 32% on SX-ACE and 20% on SX-Aurora TSUBASA.

Keywords: Thermal plasma flows, SOR method, MPI, OpenMP, Performance tuning, SX-ACE, SX-Aurora TSUBASA.

Introduction

Recently, high-performance computing (HPC) systems have been attaining higher arithmetic operation performance. According to the latest Top500 ranking [1], the highest performance of an HPC system is 513 petaflop/s (Pflop/s). HPC systems consist of a large number of nodes with multi-core processors. An application on these systems has to be divided into parallel computation operations and is executed on the multiple cores in parallel. In these cases, a core cannot directly access the data on other nodes; rather, it has to access other nodes by using the data communication provided by the MPI library, which enables point-to-point communication among cores (processes) and collective communication with all cores (processes). In general, parallelization of programs is expected to enable their faster executions by increasing the number of cores. However, with an increased amount of parallelization performed, the decrease in data communication operation time is often small compared with the decrease in computation operation time. Therefore, decreasing the communication time in large-scale simulations is required.

Research studies have explored the combining MPI and OpenMP models (MPI+OpenMP models) to enable overlapping between computation and communication operations, and new features on OpenMP and MPI have been utilized. Sergent et al. studied task scheduling using the OpenMP Tools interface in OpenMP 5.0 and leveraged idle periods of computational threads to progress MPI communications [13]. Castillo et al. presented a mechanism for exchanging event information between MPI and task-based runtime through the MPI tools interface (MPI_T) in MPI 3.0 and enhanced a task scheduler for improving the performance of overlapping of

[1]NEC Solution Innovators, Osaka, Japan
[2]NEC Technologies India, New Delhi, India
[3]NEC Corporation, Tokyo, Japan
[4]Tohoku University, Sendai, Japan

computation with communication [3]. The MPI+OpenMP models have also been evaluated on various HPC systems. Gorobets et al. developed a parallel CFD algorithm of turbulent flows with a multilevel MPI+OpenMP+OpenCL parallelization and evaluated the performance of the CFD code on Intel Xeon, Xeon Phi, and Xeon with NVIDIA K80 GPU systems [5]. Oyarzun et al. also evaluated the performance of a thermo-fluid code with the similar parallelization on ARM-based CPUs and GPUs fused in a System-on-Chip (SoC) architecture [12]. Idomura et al. evaluated plasma turbulence with MPI+OpenMP parallelization on K-computer [6]. However, the performance of the MPI+OpenMP models on vector supercomputers SX-ACE and SX-Aurora TSUBASA has rarely been evaluated. Therefore, in this paper we clarify the potential of the models on SX-ACE and SX-Aurora TSUBASA using only the basic function of OpenMP, which is the schedule clause of the work-sharing constructs.

In Section 1, we present our simulation code and the specifications of the evaluation systems. Section 2 describes evaluated overlapping models using a dedicated communication core. In Section 3, we evaluate the performance of the models on the systems. The last section concludes this paper.

1. Modern Vector Supercomputers and Target Application

1.1. SX-ACE

The SX-ACE supercomputer system is composed of up to 512 nodes interconnected by a custom network switch. Figure 1 depicts an overview of the SX-ACE processor with four powerful vector-architecture cores. A node of SX-ACE consists of one processor and a local memory. The processor can provide a double-precision floating point operating rate of 256 Gflop/s with a memory bandwidth of 256 GByte/s, and its memory capacity is 64 GBytes. In order to achieve a high sustained performance, the ratio of memory bandwidth to floating-point operation (flop/s) rate (Bytes per Flop, B/F) is a key factor. The system B/F per processor of SX-ACE is 1.0, which represents a good balance between performance and memory bandwidth. Each core is composed of a scalar processing unit (SPU), a vector processing unit (VPU), and a vector on-chip cache called Assignable Data Buffer (ADB) implemented with Miss Status Handling Register (MSHR) [4, 10, 11]. VPU is a fundamental component of the SX-ACE vector architecture with its performance of 64 Gflop/s. SX-ACE can process up to 256 vector elements, eight bytes each, by a single vector instruction. The vector architecture works in a single instruction multiple

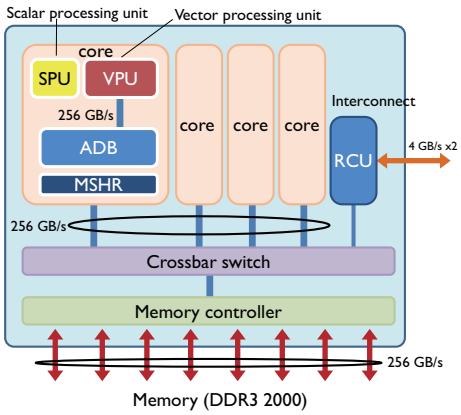


Figure 1. Overview of SX-ACE processor

data (SIMD) manner. The ADB with MSHR avoids redundant data transfers for vector load operations by keeping reusable data on a chip. SPU mainly works as a VPU controller and inherits the architecture of 64-bit RISC processors. The SX-ACE nodes are connected via a custom inter-connect network at a 4 GB/s bandwidth per direction.

The operating system (OS), SUPER-UX, is a production-proven environment based on UNIX System V with several extensions for performance and functionality.

The Fortran and C/C++ compilers support automatic vectorization and parallelization, and parallelization using OpenMP (OpenMP Version 2.5). NEC MPI is implemented in accordance with MPI-3.0 [2].

The Fortran compiler, FORTRAN90/SX, and the C/C++ compiler, C++/SX, support functions of automatic optimization, automatic vectorization, automatic parallelization, and OpenMP (OpenMP Version 2.5). NEC MPI is implemented in accordance with MPI-3.0 [2].

Here, Tab. 1 lists specifications of SX-ACE and SX-Aurora TSUBASA and options of each Fortran compiler.

Table 1. Specifications of SX-ACE and SX-Aurora TSUBASA

		SX-ACE	SX-Aurora TSUBASA
Core	Theoretical performance	64 Gflop/s	304 Gflop/s
	Memory bandwidth	256 GB/s	405.5 GB/s
	B/F	4.0	1.33
CPU	Number of cores	4	8
	Theoretical performance	256 Gflop/s	2.43 Tflop/s
	LLC capacity	1 MB (private)	16 MB (shared)
	LLC bandwidth	1000 GB/s	3244 GB/s
	Memory bandwidth	256 GB/s	1.35 TB/s
	B/F	1.0	0.55
Node	Number of CPUs	1	8
	Memory capacity	64 GB	384 GB
	Network bandwidth	8 GB/s	25 GB/s
Fortran compiler	Version	Rev.537	Rev.3.0.6
	Options	-Popenmp, -pi, -EP	-fpp, -finline-functions, -fopenmp

1.2. SX-Aurora TSUBASA

SX-Aurora TSUBASA is the newest vector supercomputer released in 2018 [8, 17]. It consists of one or more card-type vector engines (VEs) and a vector host (VH). The VE is the main part of SX-Aurora TSUBASA and contains a vector processor, a 16 MB shared last-level cache (LLC), and six High Bandwidth Memory 2 (HBM2) memory modules, as shown in Fig. 2. The vector processor has eight vector cores. As shown in Tab. 1, the peak performances of a vector core and a vector processor are 304 Gflop/s and 2.43 Tflop/s, respectively. The memory bandwidths are 405.5 GB/s per vector core and 1.35 TB/s per VE. The B/F rates are 1.33 per vector core and 0.55 per vector processor, respectively. The VH is a standard x86 Linux

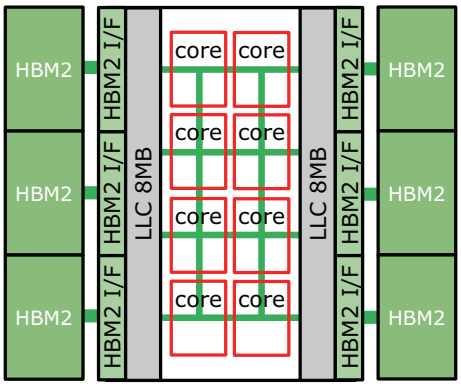


Figure 2. Block diagram of a vector engine

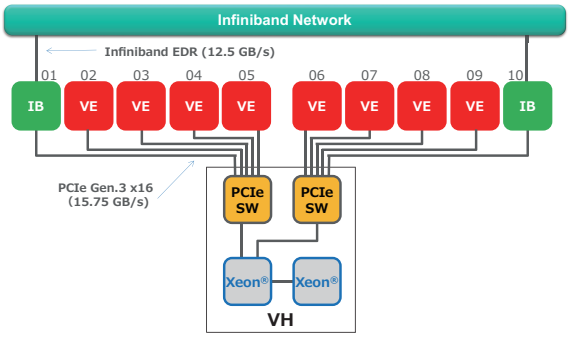


Figure 3. Block diagram of SX-Aurora TSUBASA system

server and executes the OS of the VE. While the conventional SX series runs the OS on a vector processor, SX-Aurora TSUBASA offloads OS-related processing to the VH. The VH consists of up to two Xeon processors and can handle up to eight VEs as shown in Fig. 3. Moreover, SX-Aurora TSUBASA can compose a large system by connecting the VHs via the InfiniBand switch.

The SX-Aurora TSUBASA system supports Red Hat Enterprise Linux and CentOS, and the VH provides OS functions such as process scheduling and handling of system calls invoked by the application running on the VEs. The programming environment includes NEC MPI and NEC Software Development Kit for Vector Engine (NEC SDK). NEC SDK contains an NEC Fortran compiler, NEC C/C++ compiler, NEC Numeric Library Collection, NEC Parallel Debugger, and NEC Ftrace Viewer. The Fortran and C/C++ compilers support functions of automatic optimization, automatic vectorization, automatic parallelization, and OpenMP (OpenMP Version 4.5). NEC MPI is implemented in accordance with MPI-3.1.

1.3. Thermal Plasma Flow Code

An evaluated code simulates the 3D turbulence on thermal plasma flow. This code solves the conservation that is simulated by solving the conservation equations of mass, momentum, and energy [14]. It uses the Hybrid Upwind K-K scheme and the Adams-Moulton method with third-order accuracy to discretize the convection terms and the time derivative terms, respectively. The discretized equations are numerically solved using the Successive Over-Relaxation (SOR) method. The simulation needs to be magnified in order to accurately reproduce the phenomenon in this code, and our target is the large scale simulation of 2.7 billion grids.

The code uses a Cartesian coordinate system and consists of triple loops. The two outer loops, Y-axis and Z-axis, are parallelized by the domain decomposition method, and the outermost loop, Z-axis, is also parallelized by OpenMP. The sub-routine with the SOR method has an unvectorizable loop structure, and the red-black parallelization method [7] is utilized to vectorize it on the SX-ACE supercomputer [16]. However, the computation cost of the sub-routine with the SOR method is the largest. The data sizes of the evaluated code are listed in Tab. 2.

The SOR method is an iterative method that executes a convergent calculation of the residual error. To perform this process, a collective communication operation for summing up residual values and a point-to-point communication operation for exchanging the boundary data are required. It takes longer to perform the point-to-point communication than the collective

Table 2. Data sizes of evaluated code

Model	Data size			Number of grids
	X	Y	Z	
Small data	516	204	204	21,473,856
Medium data	1280	512	512	1,342,177,280
Large data	2560	1024	1024	2,684,354,560

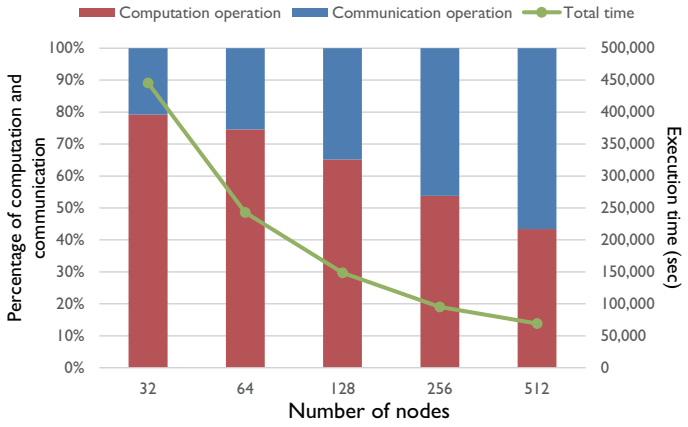


Figure 4. Breakdown of execution time

communication in this code. Therefore, the execution time for the point-to-point communication should be shortened.

Figure 4 shows the breakdown of execution time for the large data with 2.7 billion grids on the evaluation code when increasing the number of nodes. The blue and red parts of the bars indicate the execution times for the computation and communication on SX-ACE, respectively. When this code is executed on the 128-node SX-ACE, the computation operation takes up more time than the communication operation. However, when we increase the number of nodes to 512, the communication operation becomes dominant. As a result, the communication operation time becomes dominant in the entire operation time as the number of nodes increases.

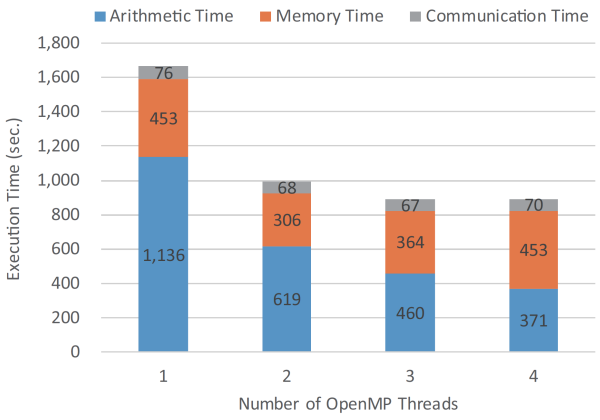


Figure 5. Relationship between execution time and number of cores on SX-ACE

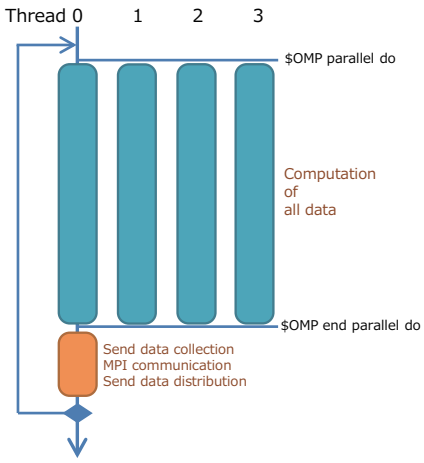


Figure 6. Diagram of the computation and communication operations of the SOR method

We evaluate the execution time (arithmetic time, memory time, and communication time) with changing the number of OpenMP threads per processor from one-thread to four-thread on SX-ACE. Figure 5 shows the relationship between the execution time and the number of OpenMP threads using four SX-ACE nodes with the small data. The memory time indicates the stall time of the core due to data load from the memory. The actual B/F [8] is 5.2 from the hardware counter of instructions on SX-ACE and indicates that the code is memory intensive for SX-ACE. The arithmetic time decreases with increasing the number of threads. However, the memory time increases from the two-thread case to the four-thread case. Therefore, the data transfer between the processor and the main memory becomes a bottleneck.

As mentioned above, even if the memory-intensive program uses all cores for computation operation, it cannot achieve a highly parallelized performance. Therefore, in our overlapping strategy, one core is assigned for the communication operation and the other cores for the computation operation. We examine the effect of the overlapping model from the viewpoint of hiding the communication operation time in memory-intensive applications such as those using the SOR method.

2. Evaluated MPI+OpenMP Models

Figure 6 shows the diagram of the parallel execution of the SOR method by MPI using OpenMP. This parallelization cannot overlap the computation operation with the communication one. Specifically, a thread on each core executes the calculation of the computation area that is divided by OpenMP, and then thread 0 starts processing of the communication operations after all the threads for computation have been completed. The communication operation gathers/scatters the boundary data for MPI communications.

The overlap of the computation operation and the communication operation is necessary to decrease the execution time. Figure 7 shows the basic concept of overlapping in the SOR method. First, the computation of the boundary data, which is sent by MPI communications, is executed for the communication operation. In the next step, both the computation of the non-boundary data and the communication of the boundary data are executed in parallel. Eventually, the time for the communication operation can be hidden.

Figure 8 shows the diagram of the overlapping model using the “schedule” clauses on OpenMP. First, all cores execute the computation of the boundary data. It then executes the

communication operation. After that, thread 0 begins to execute the non-boundary data, which is parallelized on four threads. Meanwhile, the other threads calculate the non-boundary data after the calculation of the boundary data. Therefore, the communication operation is overlapped with the calculation of the non-boundary data. When the communication operations are completed, thread 0 begins the execution of the computation operations for the non-boundary data. These operations are controlled automatically by OpenMP. The "schedule (static)" clause is used in this implementation because it has a smaller overhead for OpenMP than that of the "schedule (dynamic)" clause [9]. Although the overlapping can be achieved simply by using the directive statement, there is still an overhead of OpenMP.

In this study, we focus on the memory-intensive code that cannot effectively use all cores in a processor. Figure 9 shows the diagram of our execution model [15]. In the first step, all threads execute the computation operation of the boundary data. Then, thread 0 is used as a dedicated communication thread and executes the communication operation. At the same time, other threads simultaneously execute the computation operation of the non-boundary data. Each thread executes the prearranged DO loop as shown in Fig. 9, where the program calculates the starting and ending point of the DO loop on each thread.

3. Performance Evaluation

We evaluate three kinds of execution models: the non-overlapping model (Fig. 6), called "Original", the overlapping model (Fig. 8), called "Schedule", and our model (Fig. 9), called "Manual". This evaluation measures the calculation time of the SOR method in 20 time steps using three data sets in the code. First, the performance on SX-ACE is evaluated using the small and large data sets. The next evaluation uses the small and medium data sets on SX-Aurora TSUBASA. We will evaluate the performance on SX-Aurora TSUBASA using a large data set when a large system for it can be constructed.

3.1. Evaluation Results on SX-ACE

Figure 10 shows the execution times with the small data set using four and 16 nodes of SX-ACE. Here, computation time contains arithmetic time and memory time. Each node executes one MPI process and four OpenMP threads. Figure 10 (a) shows the case of using four nodes. The communication time includes the operation times for gathering and scattering of boundary data. For "Original", the computation time is 742 seconds and the communication time is 145 seconds, resulting in the total time of 887 seconds. "Manual" hides 121 seconds in the communication

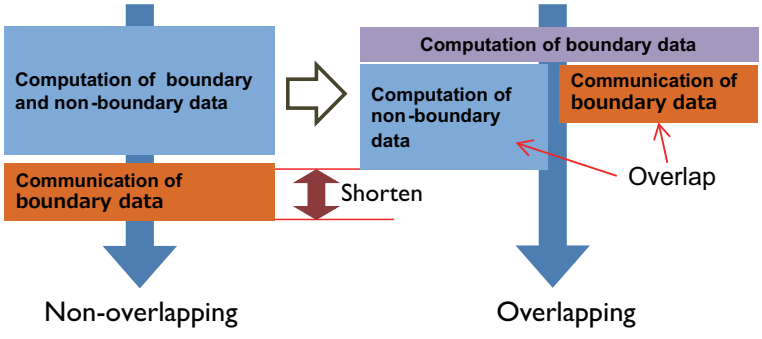


Figure 7. Basic concept of overlapping

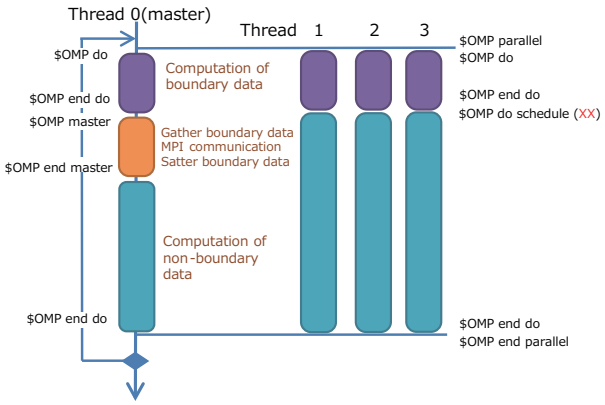


Figure 8. Diagram of implementation using "schedule" clauses on OpenMP

time, and it coresponds to 83% of the communication time. "Schedule" hides 40 seconds. The overlapping effect of "Schedule" is smaller than that of "Manual" because the OpenMP overhed of "Schedule" is larger. Figure 10 (b) shows the result for the 16-node case. The computation time for "Original" decreases from 742 seconds to 208 seconds compared with the four-node case. On the other hand, the communication time for "Original" decreases by only 13 seconds, and the percentage of communication time to the total operation time becomes about 40%. "Manual"

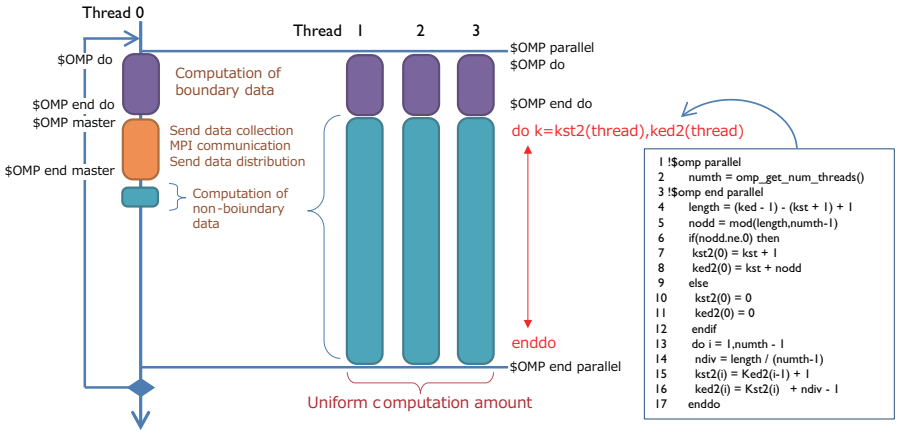
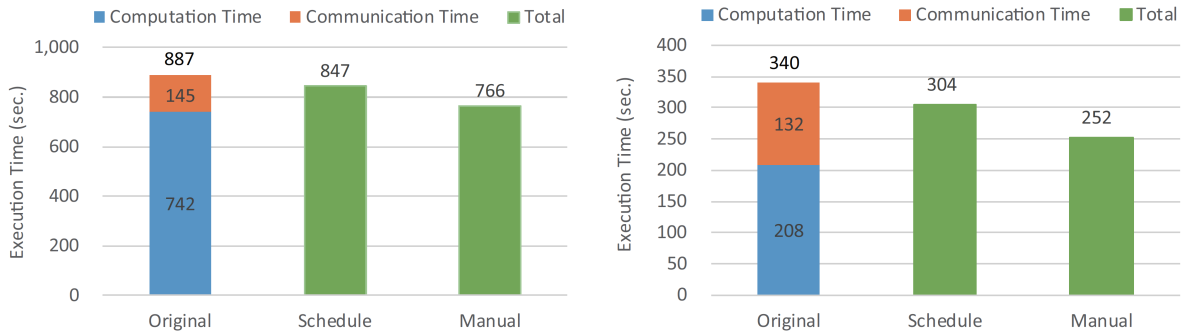
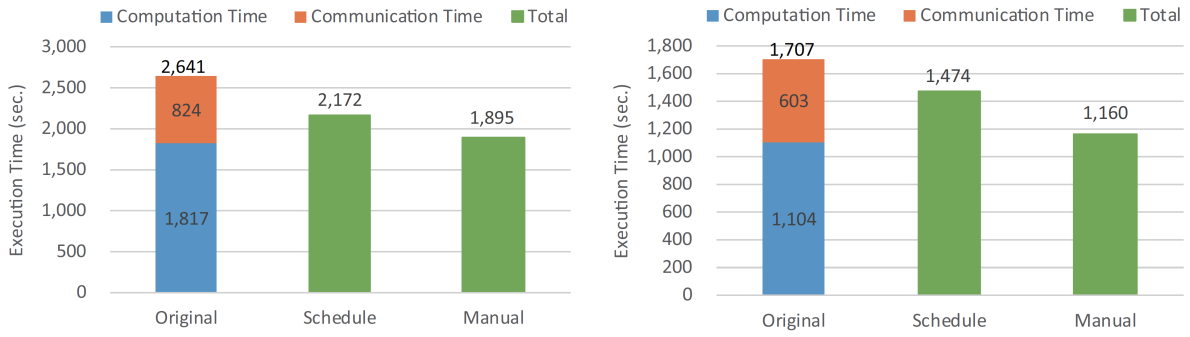


Figure 9. Diagram of our overlapping model



(a) 4-node case (b) 16-node case

Figure 10. Evaluation results with small data set on SX-ACE



(a) 256-node case (b) 512-node case

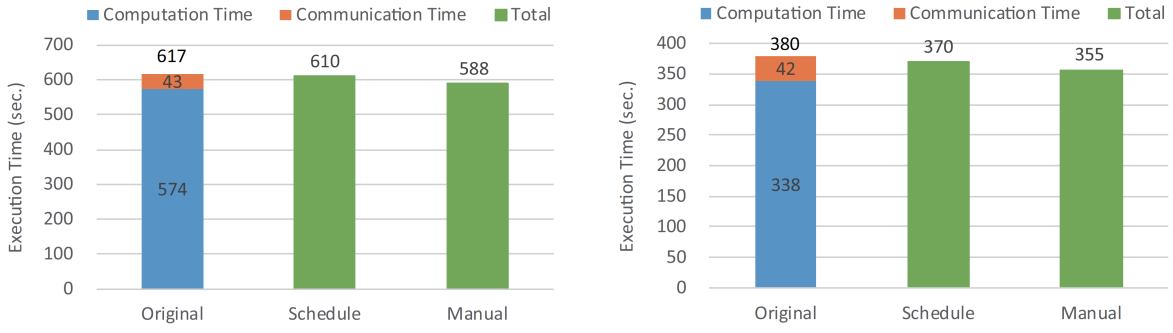
Figure 11. Evaluation results with the large data set on SX-ACE

hides the communication time of 88 seconds, which coresponds to 67% of the communication time of "Original". "Schedule" hides 36 seconds in the communication time of "Original".

Figures 11 (a) and (b) show the results of the 256-node and 512-node cases with the large data set, respectively. Each node of SX-ACE uses one MPI process and four OpenMP threads in this evaluation. In Fig. 11 (a), "Manual" hides the communication time of 746 seconds, which coresponds to 90% of the communication time of "Original", and "Schedule" hides 469 seconds in the communication time of "Original". When the number of nodes increases from 256 to 512, the percentage of communication time to the total time becomes large. However, "Manual" in Fig. 11 (b) is able to hide 547 seconds in the communication time of "Original", which represents a 90% decrease in the communication time. Meanwhile, the hidden time on "Schedule" decreases to only 233 seconds. Our overlapping model "Manual" can hide a great part of the communication time, and the execution times for the 512-node decreases by about 32%. On the basis of the above results, we show that SX-ACE has a potential to improve the performance of MPI+OpenMP models for the memory-intensive code.

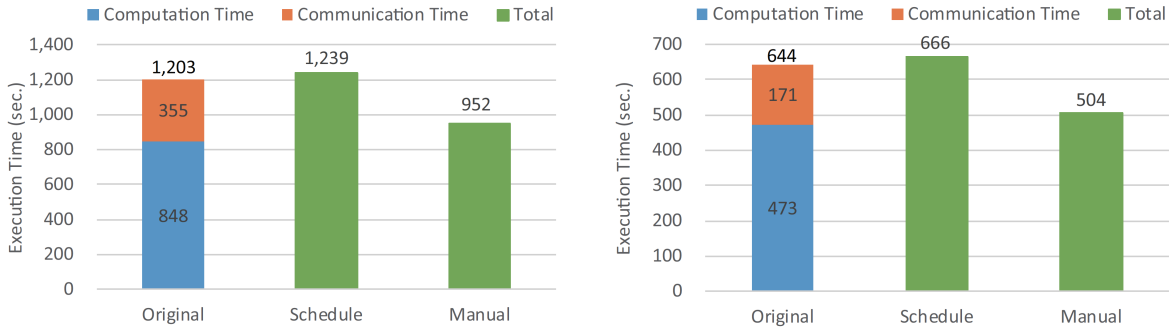
3.2. Evaluation Results on SX-Aurora TSUBASA

Figure 12 shows the execution times with the small data set using two and four VEs of SX-Aurora TSUBASA. Each VE is combined via a PCIe switch and executes one MPI process and eight OpenMP threads. Figure 12 (a) shows the results using two VEs. The communication time of "Original" in Fig. 12 (a) is 43 seconds. "Manual" hides 29 seconds, which represents a



(a) 2-VE case (b) 4-VE case

Figure 12. Evaluation results with small data set on SX-Aurora TSUBASA



(a) 16-VE case

(b) 32-VE case

Figure 13. Evaluation results with medium data set on SX-Aurora TSUBASA

roughly 67% decrease in the communication time. Figure 12 (b) shows the results using four VEs. The communication time of "Original" is nearly the same as that in Fig. 12 (a), although increasing the number of MPI processes. "Manual" also hides 25 seconds, which coresponds to roughly 60% of the communication time. However, "Schedule" in the case of two and four VEs has little overlapping effort.

We utilize 16 and 32 VEs of SX-Aurora TSUBASA for evaluation in the medium data set. The 16-VE and 32-VE systems contain two VHs and four VHs, respectively. Each VH is connected via an InfiniBand switch, and executes one MPI process and eight OpenMP threads. Figures 13 (a) and (b) show the evaluation results for 16 VEs (2 VHs) and 32 VEs (4 VHs). The computation time in "Original" on 32 VEs decreases from 848 seconds to 473 seconds compared with the case of 16 VEs, and the ratio of the communication times to the total time on the 16 and 32 VEs are about 30% and 27%, respectively. In both cases, "Manual" can hide a part of the communication time: on 16 VEs, it hides about 251 seconds among the communication time of 355 seconds, which coresoponds to roughly 70% of the communication time, and 32 VEs, it hides about 140 seconds among the communication time of 171 seconds, which represents a roughly 80% in the communication time. Then, the overall execution times on each case can decrease by about 20%. Meanwhile, "Schedule" was unable to decrease the execution times. SX-Aurora TSUBASA was released in 2018, and the fortran compiler was newly developed. Therefore, the overhead of OpenMP on SX-Aurora TSUBASA is high. In this evaluation, our overlapping model, "Manual", can hide a great part of the communication time on SX-Aurora TSUBASA. We expect our overlapping model to produce increasing performance on a larger data set with a larger system.

Conclusions

In this work, we focused on the overlapping between the computation and MPI communication operations of the SOR method in a thermal plasma flow simulation code, and examined an implementation of MPI+OpenMP models where OpenMP thread 0 assigns the communication operation and a part of the computation operation. Evaluation results demonstrated that SX-ACE and SX-Aurora TSUBASA show a good potential for overlapping computation and MPI communication on memory intensive codes, and our overlapping model, "Manual", can hide the MPI communication times of 90% on SX-ACE and 80% on SX-Aurora TSUBASA. Our overlapping model with SX-ACE and SX-Aurora TSUBASA is expected to increase perforamnce of memory intensive codes.

In future work, we will evaluate our three models using memory intensive codes from various application fields on SX-Aurora TSUBASA and other systems: such as Intel Xeon and AMD EPYC.

Acknowledgments

The authors would like to thank Associate Professor Masaya Shigeta of Osaka University for a lot of useful advice. This research uses the SX-ACE systems of Cyberscience Center at Tohoku University.

References

1. Top 500 the list. `https://www.top500.org/`

2. Vector supercomputer SX series SX-ACE. `https://de.nec.com/de_DE/en/documents/SX-ACE-brochure.pdf`

3. Castillo, E., Jain, N., Casas, M., et al.: Optimizing computation-communication overlap in asynchronous task-based programs. In: Proceedings of the ACM International Conference on Supercomputing, ICS '19, June 2019, Phoenix, Arizona, USA. pp. 380–391. ACM (2019), DOI: 10.1145/3330345.3330379

4. Egawa, R., Komatsu, K., Takizawa, H., et al.: Early evaluation of the SX-ACE processor. In: Proceedings of the 27th International Conference for High Performance Computing, Networking, Storage and Analysis (2014)

5. Gorobets, A., Soukov, S., Bogdanov, P.: Multilevel parallelization for simulating compressible turbulent flows on most kinds of hybrid supercomputers. Computers & Fluids 173, 171–177 (2018), DOI: https://doi.org/10.1016/j.compfluid.2018.03.011

6. Idomura, Y., Nakata, M., Yamada, S., et al.: Communication-overlap techniques for improved strong scaling of gyrokinetic Eulerian code beyond 100k cores on the K-computer. The International Journal of High Performance Computing Applications 28(1), 73–86 (2014), DOI: 10.1177/1094342013490973

7. Iwashita, T., Shimasaki, M.: Algebraic block red-black ordering method for parallelized ICCG solver with fast convergence and low communication costs. IEEE Transactions on Magnetics 39(3), 1713–1716 (2003), DOI: 10.1109/TMAG.2003.810531

8. Komatsu, K., Momose, S., Isobe, Y., et al.: Performance evaluation of a vector supercomputer SX-Aurora TSUBASA. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 2018, 11-16 November 2018, Dallas, Texas, USA. IEEE Press (2018), DOI: 10.5555/3291656.3291728

9. Mattson, T.G., He, Y., Koniges, A.E.: The OpenMP Common Core. The MIT Press (2019)

10. Momose, S., Hagiwara, T., Isobe, Y., et al.: The brand-new vector supercomputer, SX-ACE. In: Kunkel, J.M., Ludwig, T., Meuer, H.W. (eds.) Supercomputing. Lecture Notes in Computer Science, vol. 8488, pp. 199–214. Springer, Cham (2014), DOI: 10.1007/978-3-319-07518-1_13

11. Musa, A., Sato, Y., Soga, T., et al.: Effects of MSHR and prefetch mechanisms on an on-chip cache of the vector architecture. In: 2008 IEEE International Symposium on Parallel and Distributed Processing with Applications, 10-12 Dec. 2008, Sydney, NSW, Australia. pp. 335–342. IEEE (2008), DOI: 10.1109/ISPA.2008.100

12. Oyarzun, G., Borrell, R., Gorobets, A., et al.: Efficient CFD code implementation for the ARM-based Mont-Blanc architecture. Future Generation Computer Systems 79, 786–796 (2018), DOI: 10.1016/j.future.2017.09.029

13. Sergent, M., Dagrada, M., Carribault, P., et al.: Efficient communication/computation overlap with MPI+OpenMP runtimes collaboration. In: Aldinucci, M., Padovani, L., Torquati, M. (eds.) Euro-Par 2018: Parallel Processing. Lecture Notes in Computer Science, vol. 11014, pp. 560–572. Springer, Cham (2018), DOI: 10.1007/978-3-319-96983-1_40

14. Shigeta, M.: Turbulence modelling of thermal plasma flows. Journal of Physics D: Applied Physics 49(49), 493001 (2016), DOI: 10.1088/0022-3727/49/49/493001

15. Soga, T., Yamaguchi, K., Mathur, R., et al.: Effects of using a memory-stalled core for handling MPI communication overlapping in the SOR solver. In: The 29th International Conference on Parallel Computational Fluid Dynamics, 15-17 May 2017, Glasgow, UK (2017)

16. Soga, T., Musa, A., Okabe, K., et al.: Performance of SOR methods on modern vector and scalar processors. Computers & Fluids 45(1), 215–221 (2011), DOI: 10.1016/j.compfluid.2010.12.024

17. Yamada, Y., Momose, S.: Vector Engine Processor of NEC Brand-New supercomputer SX-Aurora TSUBASA. In: International symposium on High Performance Chips, Hot Chips 2018, August 2018, Cupertino, USA (2018)