M&C 2017 - International Conference on Mathematics & Computational Methods Applied to Nuclear Science & Engineering, Jeju, Korea, April 16-20, 2017, on USB (2017)

MPI/OpenMP Hybrid Parallelization of a Monte Carlo Neutron/Photon Transport Code MVP

Yasunobu Nagaya,[*] Masaaki Adachi[†]

[*]*Nuclear Science and Engineering Center, Japan Atomic Energy Agency, Tokai-mura, Naka-gun, Ibaraki 319-1195, Japan*
[†]*Research Organization for Information Science and Technology, 2-4, Shirakata, Tokai-mura, Ibaraki 319-1106, Japan*
nagaya.yasunobu@jaea.go.jp, adachi.masaaki@jaea.go.jp

Abstract - *MVP is a general-purpose Monte Carlo code for neutron and photon transport calculations based on the continuous-energy method. To speed up the MVP code, hybrid parallelization is applied with a message passing interface library MPI and a shared-memory multiprocessing library OpenMP. The performance tests have been done for an eigenvalue calculation of a fast reactor subassembly, a fixed-source calculation of a neutron/photon coupled problem and a PWR full core model. Performance comparisons have been made for MPI, OpenMP and hybrid parallelisms. It has been found that the hybrid parallelization of MVP can achieve a reasonable speedup and can save required memories in comparison with MPI parallel calculations.*

I. INTRODUCTION

MVP[1] is a general-purpose Monte Carlo code for neutron and photon transport calculations based on the continuous-energy method. The code has been widely used in Japan for nuclear reactor applications such as reactor core design/analysis, criticality safety and reactor shielding.

The MVP code has been designed for efficient and large-scale Monte Carlo calculations since the early stage of the code development. The vectorized algorithm is thus adopted in the code[2]; in addition the distributed memory parallelism is possible with the algorithm (so-called vector-parallel processing)[3, 4]. Such an advantage enabled one to use MVP for a full core analysis of a PWR[5] and for a part of innovative reactor core designs[6, 7, 8].

In recent years, the needs for the Monte Carlo method are increasing in the field of reactor core design. The realistic full core modeling for PWR was proposed[9] for a grand challenge against the Monte Carlo method[10]. In addition, there are many recent works for Monte Carlo neutronics calculations coupled with thermal-hydraulics codes and Monte Carlo burnup calculations. This background urges us to speed up Monte Carlo calculations.

The objective of the present work is to speed up the MVP code with a message passing interface library MPI[11] and a shared-memory multiprocessing library OpenMP[12]. This work is the first step of the MPI/OpenMP hybrid parallelism with MVP. The performance tests are thus carried out for three benchmark problems; comparisons are made for computing time and used memories.

II. PARALLELIZATION METHOD

The MVP code employs the stack-driven zone-selection algorithm which is a variant of the event-based algorithm; it is suitable for vector computers while the history-based algorithm is suitable for scalar computers. The code was developed on vector supercomputers such as Fujitsu VPP series, NEC SX series and Cray XMP series. Though vector platforms are rarely available nowadays, the code still retains the event-based algorithm and works on scalar computers without any problems. However the code has no longer been tuned for vector processing.

Considering the event-based algorithm employed in MVP, a hybrid parallelism style similar to the vector parallel one is adopted in the present study. Namely particle histories are divided by MPI processes (coarse-grained parallelism) and the events during random walk of particles are processed with OpenMP multithreads in each MPI process (fine-grained parallelism). The MVP code has been already parallelized with MPI; thus parallelization with OpenMP is newly done for hybrid parallelism.

Figure 1 shows the parallel calculation flow of MVP with MPI. The first step is to set the number of MPI processing elements (PEs) and then input data is read. The number of batches per PE is determined for fixed-source problems or the batch size (the number of histories per batch) per PE is determined for eigenvalue problems; the batch size and the number of batches are equally divided by the number of used PEs set by the user in the current implementation. After a seed of an initial random number is generated for each PE and the data is sent to the PE, random walk starts. When all histories are terminated, the main process receives the results and performs statistical analysis.

The parallelization method differs for eigenvalue and fixed-source calculations. For fixed-source calculations, each PE can process each batch independently since all batches are independent. Thus each PE processes N_b/n batches (N_b is the number of batches, n is the number of PEs). For eigenvalue calculations, it is impossible to process each batch independently because of the source iteration. MVP thus divides the batch size by the number of PEs, solves the divided eigenvalue problems independently and finally collects tally results from PEs to perform the statistical processing as shown in Fig. 2.

The OpenMP multithread parallelization is applied to the random walk events of particles. The events are classified into 7 tasks in the MVP code: source particle generation (Source), free flight (Flight), collision analysis (Collision), next zone search (Search), boundary reflection (Reflection), repeated geometry (Lattice), particle destruction (Leakage/Kill) as shown in Fig. 3. The double-boxed tasks in Fig. 3 have their own stack for queuing particle pointers. The task processing order depends on the number of queued particles in the stacks. Particle descriptors addressed with the pointers are stored in

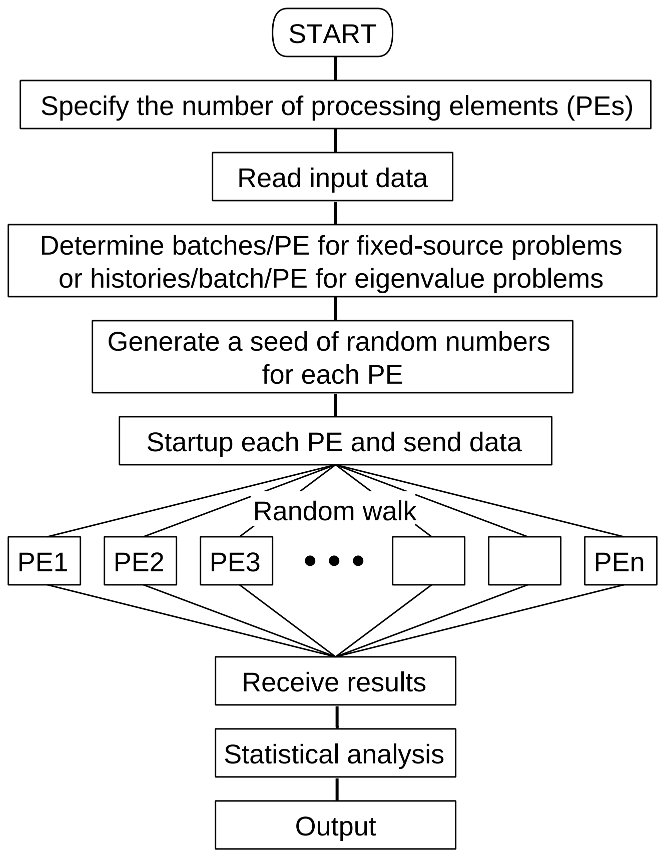
START
Specify the number of processing elements (PEs)
Read input data
Determine batches/PE for fixed-source problems or histories/batch/PE for eigenvalue problems
Generate a seed of random numbers for each PE
Startup each PE and send data
Random walk
PE1
PE2
PE3
PEn
Receive results
Statistical analysis
Output

Fig. 1. Parallel calculation flow of MVP.

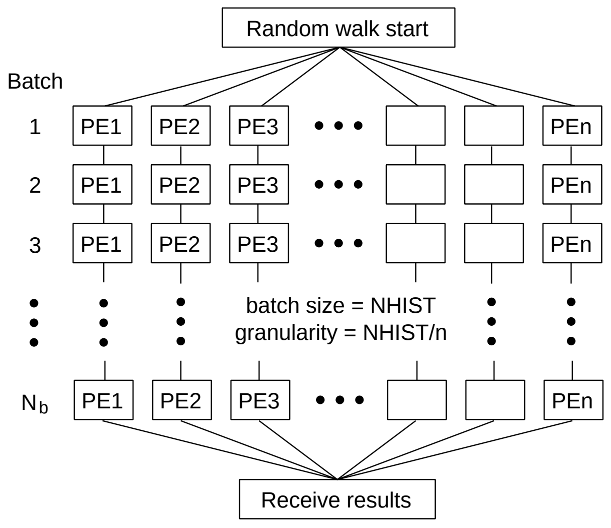

Fig. 2. Parallel processing for eigenvalue problems.

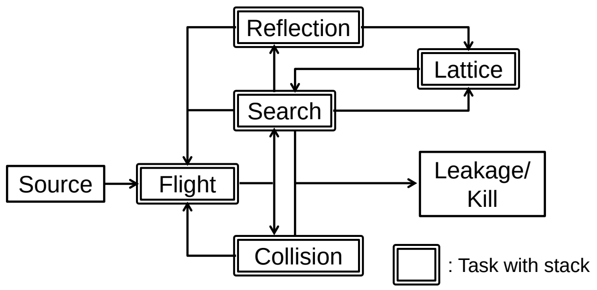

Fig. 3. Tasks of Monte Carlo random walk in the MVP code.

a large bank; they are retrieved out of and stored back to the bank for vector processing. References [2, 13, 14] should be consulted for the details of the algorithm.

Since the MVP code adopts the event-based algorithm as mentioned above, many Fortran DO-loops are implicitly and explicitly vectorizable with Fortran compilers and vectorization directives, respectively. It should be noted that current vectorization directives are useless because they are directives such as "`*VOCL LOOP,NOVREC`" for Fujitsu VP Fortran and are treated as comment lines. In the present work, the basic strategy is to implement the OpenMP directives for the DO-loops. DO-loops to be multithreaded are determined with a performance analysis tool PerfSuite[15]; DO-loops that spend more than 1% of total execution time are multithreaded. DO-loops with recursive dependencies that cannot be multithreaded without modifying the algorithm are not multithreaded.

III. PERFORMANCE TESTS

The performance tests have been carried out for following 3 benchmark problems to examine the speedup of MVP by the hybrid parallelization. All test calculations have been performed on a node of SGI ICE X: 2 CPUs of Intel Xeon E5-2680 v3 (2.5 GHz, 30 Mbyte cache) with 12 cores per CPU and memories of 64 Gbytes. The MVP source codes have been compiled by using Intel Fortran compiler with the optimization option of "-O2"; the SGI message passing toolkit (MPT) and the Intel OpenMP library have been used for MPI and OpenMP, respectively. All the calculations have occupied a single node even if all 24 cores have not been used because sharing resource in the node with other jobs deteriorates computation performance; namely unused cores have been left idle.

1. Fast Reactor Subassembly

The first performance test is for an eigenvalue calculation of a fast reactor subassembly; the infinite multiplication factor (k_∞) is calculated for the two-dimensional geometry with reflective boundary conditions. MVP calculations have been done for 8 million histories; the number of histories per batch (batch size) is 32,000, the number of active and inactive batches are 200 and 50, respectively.

Table I shows the comparison of elapsed time and used memories for the fast reactor subassembly problem. The speedup factor is defined as the ratio of the elapsed time in a single-process calculation to that in a parallel calculation. The parallelization efficiency is defined as the ratio of the speedup factor to the number of used cores. The MPI results show almost ideal performance: speedup factors of 4.5 and 12.4 for 4- and 12-process calculations, respectively. The hybrid parallelism with 4 processes × 3 threads yields the speedup factor of 6.1; the parallelization efficiency is however almost

50%. This is caused by the low parallelization efficiency with OpenMP. Figure 4 shows the speedup by individual parallelization with MPI and OpenMP. MPI parallelism shows an ideal scalability with the number of processes; OpenMP parallelism yields a speedup factor of 2 at most. The MPI results are slightly better than the ideal scalability. The reason is unclear; one of the possible reasons is the effect of cache memories.

From the viewpoint of used memories, the hybrid parallelism shows a good performance. The 12-process MPI calculation requires 5.6 Gbytes but the hybrid one with 4 processes × 3 threads does 1.7 Gbytes. The required memory size is almost the same as for the 4-process MPI calculation. The memory saving will be useful for actual large-scale full-reactor-core calculations because huge memories are required for cross section data, tally data, etc.

Case	Elapsed time (sec)	Speedup factor	Memories (Gbytes)
Single (1 process)	7,349	1	0.3
MPI (4 processes)	1,625	4.5(1.13)†	1.8
MPI (12 processes)	592	12.4(1.03)	5.6
Hybrid (4×3)*	1,213	6.1(0.50)	1.8
OpenMP (12 threads)	3,197	2.3(0.19)	0.3

* processes × threads, † The value in parentheses indicates the parallelization efficiency.

TABLE I. Comparison of elapsed time and used memories for the fast reactor subassembly problem.

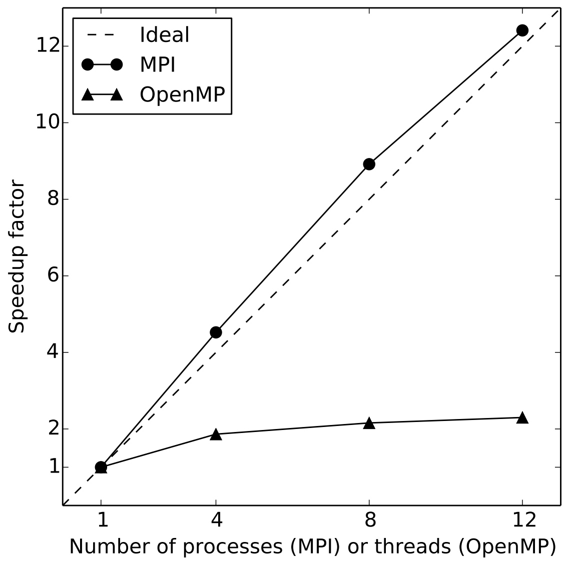

Fig. 4. Speedup by individual parallelization with MPI and OpenMP for the fast reactor subassembly problem.

2. Experimental Analysis for Fusion Neutronics Source

The second performance test is for a fixed-source and a neutron/photon coupled problem with an experimental analysis for Fusion Neutronics Source (FNS)[16] installed at JAEA. Both neutron and photon fluxes are calculated at detector regions in this benchmark problem. MVP calculations have been done for 12,000,000 histories; the number of histories per batch (batch size) is 10,000 and the number of batches is 1,200. The computing platform and the calculation condition are the same as the previous problem.

Table II shows the comparison of elapsed time and used memories for the FNS problem. Figure 5 shows the the speedup by individual parallelization with MPI and OpenMP for the problem. MPI parallelism yield the parallel efficiency of about 50% at most. This is because the problem scale is relatively small; namely 1,200 batches are processed in parallel with the number of MPI processes. In this case, 4×3 hybrid parallel calculation yields almost the same speed up as the 4-process MPI parallel calculation. This is because OpenMP parallelism is not effective as shown in Fig. 5.

Case	Elapsed time (sec)	Speedup factor	Memories (Gbytes)
Single (1 process)	3,420	1	0.2
MPI (4 processes)	1,372	2.5(0.62)†	1.1
MPI (12 processes)	620	5.5(0.46)	3.6
Hybrid (4×3)*	1,174	2.9(0.24)	1.1
OpenMP (12 threads)	3,891	0.9(0.07)	0.2

* processes × threads, † The value in parentheses indicates the parallelization efficiency.

TABLE II. Comparison of elapsed time and used memories for the FNS problem (batch size = 10,000).

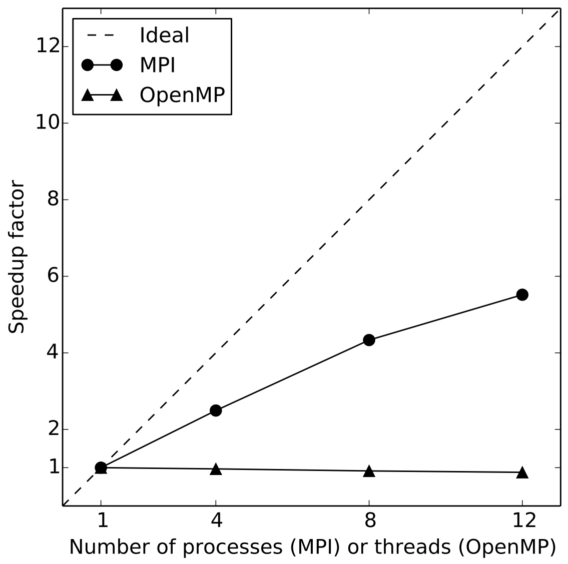

Fig. 5. Speedup by individual parallelization with MPI and OpenMP for the FNS problem (batch size = 10,000).

To investigate the impact of the batch size for OpenMP parallelism, MVP calculations have been done for the same FNS problem with a larger batch size of 40,000. As a result, the total number of histories are 48,000,000 histories and the number of batches is 1,200. Table III and Fig. 6 show the comparison of elapsed time/used memories and individual parallelization with MVP/OpenMP, respectively, for the FNS problem. The parallelization efficiency for MPI becomes better: 91%, 81% and 68% for 4, 8 and 12 MPI processes, respectively. It is considered that relatively small overheads lead to these slightly better efficiencies because the number of batches is unchanged. The speedups by OpenMP are observed in Fig. 6: 2.0, 2.4 and 2.5 for 4, 8 and 12 threads, respectively. The OpenMP speedup results in the better speedup of 6.1 for the 4×3 hybrid parallel calculation in comparison with the speedup of 3.6 for the 4-process MPI calculation. The hybrid result is not as good as the 12-process MPI result in terms of computation time but the memory saving of 68% is achieved.

Case	Elapsed time (sec)	Speedup factor	Memories (Gbytes)
Single (1 process)	24,968	1	0.2
MPI (4 processes)	6,863	3.6(0.91)†	1.3
MPI (12 processes)	3,057	8.2(0.68)	4.1
Hybrid (4×3)*	4,076	6.1(0.51)	1.3
OpenMP (12 threads)	10,010	2.5(0.21)	0.2

* processes × threads, † The value in parentheses indicates the parallelization efficiency.

TABLE III. Comparison of elapsed time and used memories for the FNS problem (batch size = 40,000).

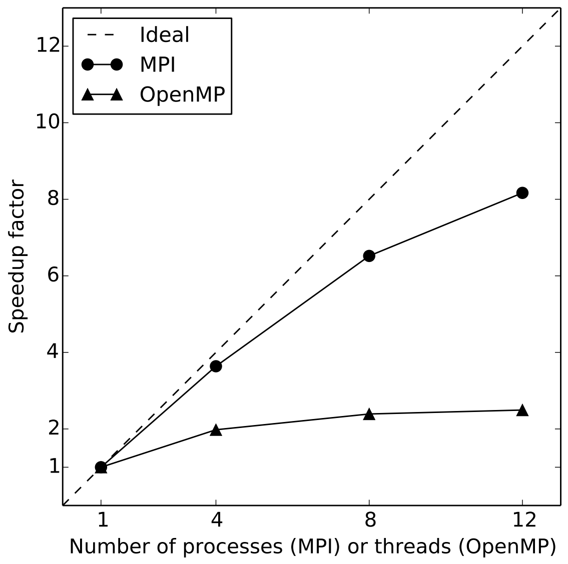

Fig. 6. Speedup by individual parallelization with MPI and OpenMP for the FNS problem (batch size = 10,000).

3. PWR Full Core

The last performance test is for a large-scale eigenvalue problem with a PWR full core model. The details of the model should be consulted for Reference [5]. This model includes 193 fuel assemblies and 55,777 fuel pins. In the present work, the pin power distribution has been calculated in 5 assemblies; each assembly includes 289 fuel pins (tally regions). MVP calculations have been done for 69,120,000 histories; the number of histories per batch (batch size) is 115,200, the number of active and inactive batches are 500 and 100, respectively. In this test all 24 cores in a node have been used for various combinations of the number of processes and threads.

Table IV and Fig. 7 show the comparison of elapsed time, the speedup factor and used memories for the PWR full core problem. For the comparison of the results with the same number of MPI processes, hybrid parallelism gives the better performance than MPI parallelism by the parallelization with OpenMP; the speedup factors are 7.4, 10.9 and 12.9 for hybrid 4 (× 6), 8 (× 3) and 12 processes (× 2 threads), respectively, in Table IV while they are 3.8, 7.7 and 9.0 for MPI 4, 8 and 12 processes, respectively, in Fig. 7. It can also reduce a large amount of required memories for the same number of used cores as observed from the comparison with the 24-process MPI result. It is, however, apparent that speedup by hybrid parallelism is dominantly achieved by MPI parallelism; the parallelization efficiency is about 50% at most and it decreases as the number of threads increases. The larger-scale Monte Carlo calculations have been done for the PWR problem in comparison with the fast reactor subassembly problem regarding the total number of histories and the batch size. However, OpenMP parallelism achieves the speedup of 2.5 at a maximum for 12 threads as shown in Fig. 7. Optimized implementation for OpenMP parallelism is required for further speedup.

Case	Elapsed time (sec)	Speedup factor	Memories (Gbytes)
Single (1 process)	37,941	1	1.2
MPI (24 processes)	1,846	20.6(0.86)†	34.1
Hybrid (12×2)*	2,950	12.9(0.54)	17.0
Hybrid (8×3)	3,492	10.9(0.45)	11.2
Hybrid (6×4)	4,119	9.2(0.38)	8.3
Hybrid (4×6)	5,122	7.4(0.31)	5.5
Hybrid (3×8)	8,336	4.6(0.19)	4.0
Hybrid (2×12)	7,985	4.8(0.20)	2.6
OpenMP (24 threads)	18,202	2.1(0.09)	1.2

* processes × threads, † The value in parentheses indicates the parallelization efficiency.

TABLE IV. Comparison of elapsed time and used memories for the PWR full core problem.

IV. CONCLUSIONS

The MVP code has been parallelized with MPI and OpenMP. Multithread and hybrid distributed/shared memory parallel calculations with MVP have become possible through the present work. The performance test has been also carried out. The results have unveiled that the speed-up of a Monte

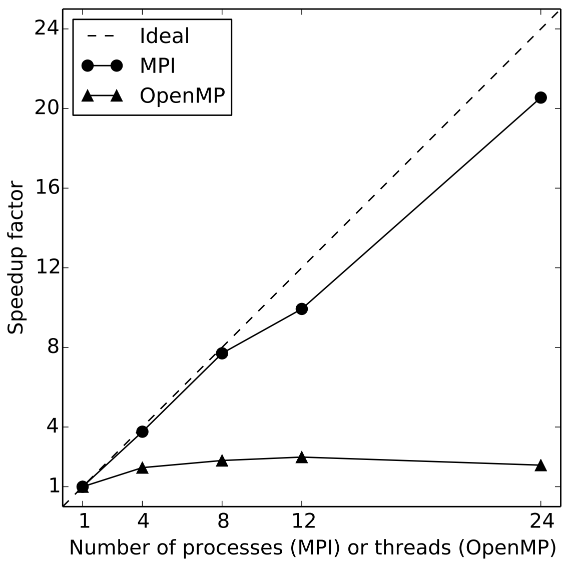

Fig. 7. Speedup by individual parallelization with MPI and OpenMP for the PWR full core problem.

Carlo code based on the event-based algorithm is possible with hybrid parallel processing.

The current OpenMP parallelization of MVP can archive a speedup of 2.5 at most. The parallelization efficiency with OpenMP is not as good as that with MPI; it deteriorates speedup performance for hybrid parallelism in comparison with ideal scalability expected for the same number of available cores. Hybrid parallelism can, however, save a great amount of memories in comparison with MPI parallelism. Recent hybrid distributed- and shared-memory computers implement a lot of cores but the available memories per core are usually limited. The required memory size for realistic full reactor core calculations is often larger than the memory limitation per core. The current hybrid parallelization would be one of the solution in such cases.

V. ACKNOWLEDGMENTS

The authors wish to express their gratitude to Kensaku Sakamoto of Center for Computational Science & e-Systems (JAEA) for his administrative support. They would like to thank Tsuruoka Takuya of Research Organization for Information Science and Technology, and Futoshi Shimizu of Center for Computational Science & e-Systems (JAEA) for fruitful discussion and helpful comments. The authors also would like to thank Takamasa Mori of Nuclear Science and Engineering Center (JAEA) for technical comments about the vector algorithm of the MVP code.

REFERENCES

1. Y. NAGAYA, K. OKUMURA, T. MORI, and M. NAKAGAWA, "MVP/GMVP II: General Purpose Monte Carlo Codes for Neutron and Photon Transport Calculations Based on Continuous Energy and Multigroup Methods," Tech. Rep. JAERI 1348, Japan Atomic Energy Research Institute (2005).
2. T. MORI, M. NAKAGAWA, and M. SASAKI, "Vectorization of Continuous Energy Monte Carlo Method for Neutron Transport Calculation," *Journal of Nuclear Science and Technology*, **29**, *4*, 325–336 (1992).
3. M. SASAKI, M. NAKAGAWA, and T. MORI, "An application study of parallel processing to the particle transport simulation," *Assisted Mechanics and Engineering Sciences*, **1**, 177–189 (1994).
4. M. NAKAGAWA, *Quality of Numerical Software*, Springer US, chap. Development of efficient general purpose Monte Carlo codes used in nuclear engineering, pp. 349–360 (1997).
5. M. NAKAGAWA and T. MORI, "Whole Core Calculations of Power Reactors by Use of Monte Carlo Method," *Journal of Nuclear Science and Technology*, **30**, *7*, 692–701 (1993).
6. T. KUGO, "Conceptual Designing of a Reduced Moderation Pressurized Water Reactor by Use of MVP and MVP-BURN," in "Advanced Monte Carlo for Radiation Physics Particle Transport Simulation and Application, Proceedings of the Monte Carlo 2000 Conference," Springer, Lisbon (23-26 October 2000), pp. 821–826.
7. Y. FUKAYA, M. GOTO, and T. NISHIHARA, "Study on erbium loading method to improve reactivity coefficients for low radiotoxic spent fuel HTGR," *Nuclear Engineering and Design*, **293**, 30–37 (2015).
8. Y. FUKAYA, M. GOTO, and T. NISHIHARA, "Reduction on high level radioactive waste volume and geological repository footprint with high burn-up and high thermal efficiency of HTGR," *Nuclear Engineering and Design*, **307**, 188–196 (2016).
9. N. HORELIK, B. HERMAN, B. FORGET, and K. SMITH, "Benchmark for evaluation and validation of reactor simulations (BEAVRS)," in "Proceedings of International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering (M&C 2013)," Sun Valley, Idaho (May 5-9 2013), pp. 2986–2999.
10. W. R. MARTIN, "Challenges and prospects for whole-core Monte Carlo analysis," *Nuclear Engineering and Technology*, **44**, *2*, 151–160 (2012).
11. MPI FORUM, "Message Passing Interface (MPI) Forum Home Page," `http://www.mpi-forum.org/`.
12. "OpenMP API for parallel programming," `http://openmp.org/wp/`.
13. M. NAKAGAWA, T. MORI, and M. SASAKI, "Monte Carlo calculations on vector supercomputers using GMVP," *Progress in Nuclear Energy*, **24**, 183–193 (1990).
14. M. NAKAGAWA, T. MORI, and M. SASAKI, "Comparison of Vectorization Methods Used in a Monte Carlo Code," *Nuclear Science and Engineering*, **107**, 58–66 (1991).
15. R. KUFRIN, "PerfSuite: An Accessible, Open Source Performance Analysis Environment for Linux," in "Proceedings of the 6th International Conference on Linux Clusters: The HPC Revolution 2005 (LCI-05)," Chapel Hill, NC (April 2005).

16. F. MAEKAWA, C. KONNO, Y. OYAMA, M. WADA, Y. UNO, Y. M. VERZILOV, and H. MAEKAWA, “Validity assessment of shielding design tools for ITER through analysis of benchmark experiment on SS316/water shield conducted at FNS/JAERI,” *Fusion Technology*, **30**, *3*, 1081–1087 (1996).