Flux Region Method Assignment using Ray Trace Information for the Method of Characteristics to Improve Cache Efficiency

Akio YAMAMOYO*), Akinori GIHO, Tomohiro ENDO*)

*) Graduate School of Engineering, Nagoya University

Corresponding author: Akio YAMAMOTO

E-mail: a-yamamoto@energy.nagoya-u.ac.jp

Address: Nagoya University, Furo-cho, Chikusa-ku, Nagoya, Japan, 464-8603

Tel, Fax: +81-52-789-5121, +81-52-789-3606

Number of Pages: 15

Number of Tables: 6

Number of Figures: 13

ABSTRACT

A flux region assignment algorithm to increase cache efficiency for the method of characteristics is proposed. In order to minimize the stride of memory access, flux region identifications (IDs) are assigned based on the ray trace sequence during the method of characteristics calculation. The present method is implemented in the three-dimensional transport code GENESIS and its performance is confirmed through verification calculations ranging from single PWR fuel assembly to PWR full core benchmark problems. Quantitative comparison of cache efficiency is carried out and the present method shows improved cache efficiency, which results in a reduction in computation time. The present method can reduce computational time by improving cache efficiency while suppressing memory requirement.

Keywords: Method of characteristics, Cache, GENESIS, Flux region ID, Ray trace

I. INTRODUCTION

The ray trace-based transport calculation, *i.e.*, the method of characteristics (MOC), has been widely used in today's lattice physics and core calculations [1]. In MOC, neutron propagation along a characteristics line is simulated considering production and annihilation of neutrons. By applying the multi-group and the flat flux approximations, the Boltzmann equation on a characteristics line can be reduced to a very simple form for which the analytic solution can be easily obtained. This efficiency makes MOC attractive as a numerical solution method for the Boltzmann transport equation. A very complicated geometry can be explicitly handled as long as the intersections between a characteristics line and spatial regions can be obtained.

In MOC, the spatial distribution of the neutron source (thus neutron flux) can be treated through the flat flux or the low-order polynomial (*e.g.*, linear) approximation. In order to reduce the spatial discretization error, a spatial region should be sub-divided into finer regions even if the macroscopic cross section is constant within the spatial region. Consequently, MOC should manage a numerous number of flux regions. Typical numbers for flux regions for a fuel assembly and a two-dimensional PWR full-core are 10^4-10^5 and 10^7-10^8, respectively.

As described above, MOC usually handles a complicated heterogeneous geometry such as fuel assembly as shown in Fig.1. Each flux region should be independently managed thus it has its own unique flux region identification (ID) usually expressed by an integer. The assignment of the flux region IDs is one of the cumbersome parts in the actual implementation of MOC. If a geometry is very simple, *e.g.*, a square cell with centered cylindrical fuel pin, an empirical approach, for example, numbering the flux region IDs from the inner to the outer regions, can be used. This approach will also be valid even if the cell is subdivided azimuthally. In this case, numbering can be performed from the inner to the outer regions, with the anti-clockwise order. However, such an approach is clearly impractical for a generalized geometry thus an alternate approach should be considered. The factorial geometry method [2] is a good alternative for region numbering. It does not adopt any empirical approach. Instead, a complicated geometry is divided into elements, *e.g.*, line or circle, and inner and outer spaces are defined for each element. By combining the inner and outer spaces of each element, all regions in the geometry can be identified. In order to distinguish inner and outer spaces, ray trace information is used as shown in Fig.2. This discussion suggests a difficulty of flux region ID treatment in MOC.

Now we consider the calculation scheme of MOC. In MOC, average angular flux on a characteristics line is numerically obtained during a neutron propagation calculation on a characteristics line. The average angular flux on a characteristics line within a flux region (segment) is summed up with the weight of segment volume (segment length multiplied by ray trace width) and solid angle for neutron flight direction. In actual implementations, an array is assigned to preserve flux in each flux region. Neutron propagation is performed on a characteristics line, which usually penetrates a whole calculation geometry. Since flux region ID is usually defined by a dedicated algorithm such as the factorial geometry, access to the array preserving the tentative value of flux becomes scattered as shown in Fig.3. In the case of Fig.3, flux region IDs encountered during a ray trace are 37, 29, 21, 13, 5, 6, 7, 8, 1, 9, 17, 25, 33, 76, 68, 60, 52, 51, 50, 58, 66, and 74, which are far from the sequential access.

In today's computers, the clock frequency of CPU reaches GHz, implying memory access speed is much slower than processing speed of CPU and becomes a crucial bottleneck for the overall computational performance. In order to mitigate the latency of memory access, the cache memory is utilized. The cache memory is physically placed near CPU and its access speed is much faster than that of main memory, though its size is limited. In many CPU architectures, the multi-level cache is implemented. For example, cache sizes are 32 KB+32 KB, 256 KB, 8 MB for L1, L2, and L3 caches, respectively, which are significantly smaller than that of main memory.

Let us consider a large array in which flux is stored during a MOC calculation. Now an element of the array is accessed in order to sum up the average angular flux on a segment of a flux region. At first, the CPU "searches" for the data of this element in the caches. If it does not exist in the caches, then the CPU accesses to the main memory. If the main memory is accessed by the CPU, not only the element accessed from main memory but also contents of main memory near the accessed position are simultaneously copied and stored in the cache memory. If the next access to the position in a flux array is not far from the current access position, the required data would exist on the cache memory thus its access speed would be very fast. Contents of an array are contiguously mapped on the main memory. Therefore, the "stride" of the memory access can dominate computational speed when access to main memory is intensive. Ideally, sequential access to the array maximizes memory access speed by the efficient utilization of cache. This is not a particular issue in a MOC calculation but is a generic (and a very important) issue in the field of large-scale scientific computations. For example, in the basic linear algebra subprograms (BLAS), memory access and cache utilization are one of the important factors in its implementation [3].

In general, partly due to its complexity, flux region IDs are assigned independently from the access sequence to the flux array during transport sweeps on characteristics lines. This can be one of the causes of significant degradation in computational performance since access to the flux array is far from contiguous.

Importance of this issue has been pointed out in the previous studies. In the OpenMOC code developed by the reactor physics group of MIT, iterations on energy and polar angles are implemented as the most inner loops [4]. Since energy and polar angle indices can be accessed in a sequential manner during iterations, its iteration strategy increases the cache efficiency. However, by moving the energy iteration to the innermost loop, energy iteration should be carried out by the Jacobi method rather than the Gauss-Seidel method adopted in other MOC codes. Though convergence of the Jacobi method is slower than that of the Gauss-Seidel method, its shortcoming can be compensated by an increase of computational speed up by cache utilization and efficient acceleration by the CMFD method. The same observation is obtained in ref. [5].

Kochunas et al. developed a theoretical model for the computational efficiency of the 3-D MOC parallel calculation including the cache and memory latencies and verified the theoretical model by numerical calculations. Their study indicated that a typical 3-D MOC utilizes approximately 50% of the peak performance of a computer [6]. The results imply the importance of efficient cache utilization.

In the present paper, we propose an alternate approach for the flux region assignment, which increases cache utilization. Usually, assignment of flux region ID is completely independent of transport sweeps along characteristics lines. Instead, we propose a new flux region ID assignment algorithm considering the access sequence to the flux array during a transport sweep. The present method is implemented in the GENESIS code, which is a neutron transport code in two- or three-dimensional geometry-based on MOC or Legendre polynomial Expansion of Angular Flux (LEAF) method. The present method can be used with the Jacobi method, which is also used in the OpenMOC code.

The remainder of this paper is organized as follows. In section II, overview and recent developments of the GENESIS code are briefly reviewed. A new flux region assignment method is proposed in section III. Numerical results using the proposed and the conventional flux region assignment methods are shown in section IV with discussions. Finally, concluding remarks are summarized in section V.

II. OVERVIEW AND RECENT DEVELOPMENTS OF THE GENESIS CODE

The GENESIS code is a three-dimensional multi-group neutron transport code based on the MOC (2D) or the LEAF method (3D) [7]. MOC used in two-dimensional geometry is the conventional one and its methodology is well described in the textbook [1]. In the LEAF method, three-dimensional geometry is covered by parallel "characteristics planes" that are a straightforward extension of 2D MOC ray traces to the axial direction [7]. A characteristics plane consists of multiple rectangular regions as shown in Fig.4. In the LEAF method, neutron transport within a 2D rectangular region is repeatedly carried out instead of neutron transport along 1D characteristics line in the conventional 2D or 3D direct MOC.

From the viewpoint of computational efficiency, neutron transport within a rectangle is carried out by the angular dependent transmission probability (ADTP) method considering spatial distribution of incoming and outgoing angular fluxes along the face and neutron source [7], [8]. Once the average angular flux within a rectangle is obtained, numerical integration on space and solid angle is carried out to obtain the scalar flux that is identical to the procedure used in conventional MOC.

Major features of the GENESIS code are summarized as follows:

- Multi-group transport calculation in 2D geometry (using the conventional MOC) or 3D geometry (using the LEAF method)
- Flexible geometry treatment using the factorial geometry method and the R-function solid modeler, covering all existing fuel bundle designs for LWRs and FBRs [7].
- Cyclic ray trace is used in rectangular and hexagonal geometry with the direct neutron path linking (DNPL) method [9].
- Spatial distribution of incoming and outgoing angular fluxes in a flux region can be expanded up to 2nd order for axial and radial direction, respectively, in the LEAF method [7].
- Spatial distribution of neutron source in a flux region can be expanded up to 2nd and 1st order for axial and radial direction, respectively, in the LEAF method [7].

- •The two-level generalized coarse mesh rebalance (GCMR) or the CMFD method is used for convergent acceleration [10], [11].
- •Numerical stability for highly voided regions including the design extension conditions (DEC) [7][12].

The following capabilities are recently developed and implemented in the GENESIS code [12].

- •Convergence stabilization techniques for acceleration calculation.
- •Anisotropic scattering treatment using the rigorous spherical harmonics method or the simplified Pn method. Anisotropic scattering treatment can be used with symmetry
- •Reflective and/or rotational symmetry for 180° (horizontal, vertical, diagonal), 120°, 90°, 60°, 45°, and 30°.
- •Parallel computation using OpenMP.
- •Object rotation

The validity of the GENESIS code has been verified through various benchmark problems, *i.e.*, C5G7 2D/3D benchmarks without and with void regions, hexagonal version of C5G7 3D, Takeda benchmark No.1 [7], Kobayashi 3D radiation transport benchmark [13], and so on. The verification results indicate that the GENESIS code can provide an accurate solution without numerical instability for various conditions even if large and highly voided regions exist in the geometry.

III. FLUX REGION ASSIGNMENT METHODS

In this section, the flux region assignment methods implemented in the GENESIS code are described. They are the geometry-based assignment, the random assignment, and the ray trace-based assignment. The first one is the conventional implementation based on the factorial geometry method and has been used in the GENESIS code. The second one simulates a worse case. The last one is a new method proposed in the present paper. As discussed in section I, the efficiency of memory access greatly depends on the stride of main memory access. These three assignment methods have different characteristics from the viewpoint of the stride. The methodology and qualitative discussion are given in this section and the quantitative discussion will be described in section IV with numerical results.

III.A. Transport Sweep in the GENESIS Code

In this subsection, the transport sweep algorithm in the GENESIS code is described since access to flux regions depends on it. Figure 5 summarizes the transport sweep algorithm used in the GENESIS code. The GENESIS code utilizes a conventional loop structure for transport sweep, i.e., puts the energy group outermost. The OpenMOC code moves energy and polar angle loop to innermost loop to increase cache efficiency as described in section I. Technically, the GENESIS code can adopt this strategy but it was not used due to the following reasons. Firstly, the GENESIS code does not store the angular fluxes at the top and bottom surfaces of a core. Instead, these angular fluxes are temporarily maintained during a transport sweep for a particular energy group and a polar angle. If the energy group and polar angle loops are moved into innermost, memory storage will substantially increase. Secondly, during the parallel computation, angular fluxes are tallied in the thread independent arrays in order to increase parallel efficiency. When the energy and polar loops are innermost, the temporary array should be preserved for all polar and energy groups, which may be impracticable for large three-dimensional problems.

The `angular_flux_tally_array` and `scalar_flux_tally_array` are one-dimensional arrays whose number of elements is equivalent to the number of flux regions. Assigned flux region IDs (generated prior to the transport sweep) are used for the `angular_flux_tally_array` in Fig.5 since the `angular_flux_tally_array` is very frequently accessed at the innermost loop during transport sweeps. Namely, the access pattern to `angular_flux_tally_array` can dominate the computational performance.

III.B. Geometry-based Assignment Method

The geometry-based assignment is the original flux region ID assignment method in the GENESIS code using the factorial geometry method. In the GENESIS code, the DNPL method with the cyclic ray trace is adopted. The core (whole) geometry is constructed by a repeated arrangement of fuel assemblies having an identical outer shape. A fuel assembly generally consists of a repeated structure of fuel cells. Since complicated heterogeneous geometry exists inside a fuel assembly, the factorial geometry method is applied.

The GENESIS code gives a unique integer number to each flux region in a calculation geometry. The unique number is assigned by the algorithm shown in Fig. 6. Note that the algorithm of Fig.6 is simplified from the original one assuming the treatment in a 3D Cartesian geometry.

As a simple example, flux region IDs in a cell generated by the geometry-based assignment method are shown in Fig.7. They are very different from those in Fig.3, which is generated by a simple empirical rule (from inside to outside, anti-clockwise).

The algorithm shown in Fig.6 naturally generates clusters of flux region IDs. Namely, flux regions within a cell have close flux region IDs. Similarly, flux regions within a fuel assembly have similar flux region IDs though the flux region IDs within a fuel assembly are more scattered than those in a cell. Flux region IDs in different fuel assembly are considerably different.

III.C. Random Assignment Method

Since computational efficiency significantly depends on the stride of memory access, the worst case is an assignment of flux region IDs in which cache miss occurs every memory access. However, since the implementation of the worst case would be difficult, the random assignment of flux region IDs throughout a whole geometry is considered instead. In this case, memory access during a ray trace will be completely random, which makes the average memory access stride very large. As a comparison case, this random assignment is also implemented. In the actual implementation, the unique flux region IDs generated by the geometry-based assignment are randomly shuffled to realize the random assignment. Note that number of the random shuffle is determined as (10×number of flux regions in whole geometry) in order to make the flux region IDs completely random. A simple example of flux region IDs obtained by this method is shown in Fig.8.

III.D. Ray trace-based Assignment Method

An array to sum up the angular flux is continuously accessed during a transport calculation following the ray traces. Thus, in order to minimize the stride of memory access to a flux array, flux region IDs should be numbered based on the sequence of ray traces. In the ray trace-based flux region ID assignment method, flux region IDs are numbered using the ray trace information as shown in Fig.9. The algorithm in Fig.9 is based on the first-encountered basis. Namely, if a new flux region is encountered during a transport sweep, a new fuel region ID is assigned. Contrary, if an encountered flux region already has the flux region ID, assignment of new flux region ID is skipped. Figure 10 represents examples in a cell. It should be noted that the flux region ID assignment is carried out for a whole core geometry in practical implementation. Figure 10 indicates that the flux region IDs are set based on the access sequence of the ray trace. It is interesting to see that the result of the geometry-based assignment shown in Fig.7 is similar to Fig.10 (azimuthal angle: $\pi/32$). Actually, the geometry-based assignment utilizes the factorial geometry method that uses the ray trace information. Since the first azimuthal angle ($\pi/32$ in this case) is used for the ray trace in the factorial geometry method, they become similar.

In this method, assigned flux region IDs naturally depend on the azimuthal angle of the ray trace. It means that multiple assignment sets of flux region IDs should be kept during a transport sweep. Since the size of flux region IDs is the same as the number of total flux regions, that would reach 10^7 in a large problem, extra memory storage is necessary. However, as discussed in section IV, the extra memory requirement is not excessive even for a large problem.

In the GENESIS code, variables in a flux region (*e.g.*, flux region volume, neutron source, and total cross section) are copied to a large one-dimensional array in order to increase memory access speed. In the conventional geometry-based assignment, the flux region ID for each flux region is fixed during the ray trace. It means that the copy of the variables to the large one-dimensional array should be done once per inner iteration. However, in the present ray trace-based assignment, flux region IDs depend on azimuthal angle, and thus copy to the one-dimensional array should be carried out for every azimuthal angle sweep. This preparation takes computation time, though the stride to memory access during ray traces can be reduced. Quantitative discussion on these issues will be given in section IV.

IV. NUMERICAL RESULTS AND DISCUSSIONS

IV.A. Analysis of Access Pattern to Flux Region Array

The flux region ID assignment methods are implemented in the GENESIS code and the access pattern to the flux region array integrating the angular flux is analyzed. Calculation conditions are as follows:

•2D C5G7 benchmark problem [14]

•5 annular and 16 azimuthal mesh divisions (total 80) for fuel cells, 6×6 mesh divisions (total 36) for reflector cells. Total number of flux region is 144,500 and number of segments for particular azimuthal and polar angles is 152,966.

•32 azimuthal angles (for 2π), no symmetry

•2 polar angles (for $\pi/2$) using the Tabuchi-Yamamoto quadrature set [15]

•0.2 cm ray trace width, equidistant cyclic ray trace with the DNPL method

Note that the spatial and angular discretization would be too coarse to obtain the accurate and converged result. However, since the analysis of the access pattern is our main interest here, the above discretization is adopted.

Results are shown in Figs. 11, 12, and 13. Each upper figure indicates the access pattern to the flux regions during a transport sweep on ray traces in a particular azimuthal direction. Each lower figure indicates the first 2000 accesses to flux regions during a transport sweep. Figures 11, 12, and 13 clearly show different access pattern to flux regions by the three methods. The geometry-based assignment method shows some periodic oscillation. In the geometry-based assignment, flux region IDs within a fuel assembly are similar. The C5G7 benchmark problem consists of 3×3 fuel assemblies and thus a ray trace passes through 3 fuel assemblies at maximum. The step feature shown in the lower figure of Fig.11 reflects this characteristic, *i.e.*, a ray trace is passing within a fuel assembly at a plateau region, entered another fuel assembly with a discontinuous jump on flux region ID. The access pattern by the geometry-based assignment is structured but contains many large strides. Note that stride is defined by the difference between two successively accessed flux region IDs.

The random assignment method shows no structured pattern as expected. Large stride appears at almost all accesses. This characteristic will significantly degrade cache efficiency.

The access pattern by the ray trace-based assignment method shows smoother behavior than the other methods as expected. Ideally, flux region IDs monotonically increase as the ray trace proceeds, but small dips are observed as shown in the lower figure of Fig.13. In the very first ray trace, contiguous flux region ID can be assigned since no flux region ID has been issued (segment 1-

337 in Fig.13). However, in the second or latter lay traces, some of the encountered flux regions already have their IDs, which are given by the previous ray traces. Thus these small dips are observed.

The average stride per segment can be considered as an index for cache efficiency. From the viewpoint of cache efficiency, smaller stride per segment is desirable. The average strides per segment for the geometry-based, the random, and the ray trace-based assignment method are 210, 47989, and 76, respectively. Clearly, the ray trace-based method gives the smallest average stride per segment.

From the viewpoint of cache efficiency, the average stride per segment should be as small as possible, ideally, 1. However, due to the geometric limitation, the average stride cannot be 1 in the actual situation. The problem is that how the average stride per segment can be reduced. In order to address this issue, a combinatorial problem to assign a unique flux region number to each flux region should be solved. Though this combinatorial problem would be tough, an alternate approach to assign flux region numbers might be developed by addressing this problem. From the viewpoint of optimization, the present method (the ray trace-based method) utilizes a simple approach, i.e., the first in first out basis. However, in order to obtain a global (near) optimal solution, which has small average stride per segment, use of a global combinatorial optimization method would be necessary.

IV.B. Comparison of Computational Efficiency

Performances of the three flux region ID assignment methods described in section III are compared using four different configurations. Calculation conditions are summarized in Table I.

Specifications of computers used in the present calculation are as follows:

- •Core i7-5820K
- •32GB main memory, (32 KB+32 KB)×6, 256 KB×6, 15 MB for the L1, L2, and L3 caches, respectively. Note that there are six cores in this CPU and the L1 and L2 caches are implemented in each core.
- •6 cores, 12 threads (only single thread with no parallel computation is used in the present study)
- •Windows10 with Intel c++ compiler version 18.0

Calculations are carried out by the GENESIS code. Calculation results using the three methods are identical including convergence behavior since only the assignment of flux region IDs is different among these methods.

Computation time is summarized in Tables II. More specifically, computation time (in nanoseconds) required for a spatial integration of a segment on a ray trace using one CPU core is derived and summarized in Table III. Table II indicates that the ray trace-based flux region assignment method proposed in this study is fastest among the three methods. The random assignment is slowest as expected. The geometry-based method is much faster than the random method but considerably slower than the ray trace-based method. The difference of performance depends on the size of the problem, *i.e.*, the difference in computation time becomes larger as problem size increases. It is natural that the size of a flux array depends on the problem size and the stride tends to be larger for a larger problem. In the PWR full core problems, the ray trace-based method is 20% faster than the geometry-based method, while they are closer in the single PWR assembly problem. The efficiency of spatial integration shown in Table III shows a similar trend to the total computation time shown in Table II. Note that the time per integration per core shown in Table III is comparable to that of OpenMOC (approximately 40 ns) though rigorous comparison is difficult due to the differences of calculation geometry and computer hardware [16].

Table IV summarizes the memory requirement for three methods. Though extra memory storage is necessary for the ray trace-based method, additional memory storage is not excessive even in the PWR full core geometry. Thus, the proposed ray trace-based method is considered to be feasible and practical.

Breakdown of the computation time only for the transport calculation (*i.e.*, excluding ray tracing, acceleration, input processing, editing, and so on) is summarized in Table V. The KAIST-2A benchmark problem is used for Table V. Reduction of computation time for transport sweep of the ray trace-based method is approximately 50% due to improvement in memory access. However, it requires additional computation time for rearranging temporary data for transport sweep since it should be done for every azimuthal angle. The ray trace-based method also requires some additional computation time to sum up angular flux to scalar flux in each flux region, since the arrangement of angular flux in temporary storage is different from that of scalar flux and thus memory stride tends to be larger in this process. The similar trend is also observed in the random flux region assignment.

Table VI summarizes the memory bottleneck that represents the cache efficiency. Parameters in Table VI are measured by Intel c++(version 18.0) compiler for Windows with VTune Amplifier. The KAIST-2A benchmark problem is used for Table VI [17]. The hyper-threading capability is disabled in order to accurately measure the parameters. Note that the L2 cache miss rate is 0.0% for all cases thus it is excluded from Table VI. The CPI rate (clock ticks per instruction rate) is a good indicator for the cache efficiency. The ray trace-based method gives the lowest value, which means average wall clock time per execution of an instruction is minimal among the three methods. Though the fraction of waiting time due to miss or latency of the L1 cache is similar, that of the L3 cache and the DRAM with the ray trace-based method show the smallest value while the random method shows largest. These are the clear and quantitative evidence for the improvement of cache efficiency using the present method.

The theoretical CPI rate for today's CPU is generally 0.25 (four instructions per clock tick) but the CPI rate of 1.0 and 0.75 are considered as the acceptable and the good performances for high-performance computing, respectively [18]. Table VI indicates that the CPI rate measured in the present calculation is less than 1.0 except for the random assignment method. The CPI rate of the ray trace-based method is 0.658, which is generally classified as the good performance. The gap between the theoretical value (0.25) and the measured value (0.658) implies that the importance of alternate new transport calculation method having higher cache efficiency. It is a very interesting topic but out of the scope of the present study.

In the present study, energy iteration is placed at the outer most of the loops, i.e., utilizes the Gauss-Seidel iteration for energy. As described in section I, the Jacobi-iteration can be used for energy to increase the stability of iteration and cache efficiency by moving the energy iteration to the innermost of the loops [5]. The method proposed in the present paper could be used with the Jacobi-iteration, which would further increase the cache efficiency. This is an interesting future study.

A two-dimensional case is used as the benchmark problem in this study. In a three-dimensional case, the GENESIS code utilizes the LEAF method in which computational burden is much higher than MOC since two-dimensional transport calculation in a characteristics plane is performed instead of one-dimensional neutron transport along a characteristics line [7]. It means that the significance of cache efficiency would be smaller than that of the two-dimensional case since the fraction of memory latency will be smaller. The present method can be used for the three-dimensional case, but the reduction of computation time would be smaller.

In the case of direct three-dimensional MOC, computational burden per segment on a ray trace is equivalent to the two-dimensional MOC while number of flux regions generally increases. The present method can be applied to three-dimensional geometries by considering not only the radial direction but also the axial direction. Cache efficiency tends to decrease in three-dimensional geometries since number of flux regions is large. Therefore, the present method would be useful to minimize the degradation of cache efficiency for three-dimensional calculations.

V. CONCLUSIONS

A new flux region ID assignment method, the ray trace-based algorithm, is proposed in order to improve the cache efficiency during transport sweeps with the method of characteristics. In the ray trace-based method, flux region IDs are defined to minimize the stride of memory access to flux array during transport sweeps.

At first, access pattern to the flux array is analyzed for the 2D C5G7 benchmark problem. Three different flux region assignment methods (the conventional geometry-based method, the random method, the ray trace-based method) show different access pattern to the flux array. The ray trace-based method shows the smallest memory stride as expected.

Performances of these methods are confirmed through four different benchmark problems having different problem size, ranging from a single PWR fuel assembly to a PWR full core. Calculation results indicate that the ray trace-based method can reduce approximately 20% of computation time for large problems while suppressing the increase of memory requirement.

ACKNOWLEDGMENTS

This work was supported in part by JSPS KAKENHI, Grant-in-Aid for Scientific Research (C) (16K06956).

REFERENCES

1. D. G. CACUCI ed., Handbook of Nuclear Engineering, Chapter 9, Lattice Physics Computation, ISBN:978-0-387-98130-7, Springer (2010).
2. Z. WEISS, G. BALL, "Ray-Tracing in Complicated Geometries," *Ann. Nucl. Energy*, **18**, 483 (1991).
3. BLAS (Basic Linear Algebra Subprograms), http://www.netlib.org/blas/.
4. W. BOYD, S. SHANER, L. LI, B. FORGET, K. SMITH, "The OpenMOC Method of Characteristics Neutral Particle Transport Code," *Ann. Nucl. Energy*, **68**, 43 (2014).

5. S. STIMPSON, B. COLLINS, B. KOCHUNAS, “Improvement of Transport-corrected Scattering Stability and Performance using a Jacobi Inscatter Algorithm for 2D-MOC,” *Ann. Nucl. Energy*, **105**, 1 (2017).
6. B. KOCHUNAS, T. DOWNAR, “Performance Model Development and Analysis for the 3-D Method of Characteristics,” CASL-U-2015-0162-000, CASL, (2015).
7. A. YAMAMOTO, A. GIHO, Y. KATO, T. ENDO, “GENESIS – A Three-dimensional Heterogeneous Transport Solver based on the Legendre Polynomial Expansion of Angular Flux Method,” *Nucl. Sci. Eng*, **1**, 186 (2017).
8. A. YAMAMOTO, K. KIRIMURA, Y. KAMIYAMA, K. YAMAJI, S. KOSAKA, H. MATSUMOTO, "Angular Dependent Transmission Probability Method for Fast Reactor Core Transport Analysis," *Trans. Am. Nucl. Soc.*, **112**, 736 (2015).
9. S. KOSAKA, E. SAJI, “Transport Theory Calculation for a Heterogeneous Multi-Assembly Problem by Characteristics Method with Direct Neutron Path Linking Technique,” *J. Nucl. Sci. Technol.*, **37**, 1015 (2000).
10. A. YAMAMOTO, “Generalized Coarse-Mesh Rebalance Method for Acceleration of Neutron Transport Calculations,” *Nucl. Sci. Eng.*, **151**, 274 (2005).
11. K. SMITH, J. D. RHODES, “Full-Core, 2-D, LWR Core Calculations with CASMO-4E,” *Proceedings of PHYSOR2002*, Seoul, South Korea (2002). [CD-ROM]
12. A. YAMAMOTO, A. GIHO, T. ENDO, “Recent Developments in the GENESIS Code based on the LEAF Method,” *Nucl. Eng Technol.*, https://doi.org/10.1016/j.net.2017.06.016 (2017).
13. A. YAMAMOTO, A. GIHO, T. ENDO, “Application of the GENESIS Code to the Kobayashi 3D Benchmark Problem,” Trans. Am. Nucl. Soc., **117**, 1403 (2017).
14. *Benchmark on Deterministic Transport Calculations Without Spatial Homogenization, Nuclear Energy Agency*, NEA/NSC/DOC(2003) 16, (2003).
15. A. YAMAMOTO, M. TABUCHI, N. SUGIMURA, T. USHIO, M. MORI, “Derivation of Optimum Polar Angle Quadrature Set for the Method of Characteristics Based on Approximation Error of the Bickley Function,” *J. Nucl. Sci. Technol.*, **44**, 129 (2007).
16. G. GUNOW, S. SHANER, W. BOYD, B. FORGET, K. SMITH, “Accuracy and Performance of 3D MOC for Full-Core PWR Problems,” *Proc. M&C 2017*, Jeju, Korea, Apr. 16-20, 2017, (2017). [USB-DRIVE]
17. N. Z. CHO, *Benchmark Problems in Reactor and Particle Transport Physics*,

http://nurapt.kaist.ac.kr/benchmark/ (2000).

18. *Intel® VTune™ Amplifier 2018 User's Guide*, https://software.intel.com/en-us/vtune-amplifier-help-cpi-rate (2018).

List of Tables

List of Figures

Table I. Calculation conditions

Benchmark	Number of azimuthal angles (2π)	Number of polar angles (π/2)	Ray trace width (cm)	Number of flux regions	Number of segments (1/polar)	Number of energy groups
Single fuel assy.	32	2	0.05	36,984	3,743,424	7
2D C5G7 [14]	32	3	0.05	144,500	20,723,680	7
2D KAIST-2A [15]	32	2	0.05	605,640	70,069,824	7
2D PWR full core	32	2	0.1	7,777,520	418,639,584	7

Note 1: Single PWR fuel assembly is taken as MOX-1 (BA) of the KAIST benchmark problem. 2D PWR full core assumes 4-loop type PWR surrounded by baffle and reflector.

Table II. Total computational time using three flux region ID assignment methods

Method	2D Single PWR fuel assembly	2D C5G7	2D KAIST-2A	2D PWR full core
Geometry-based	42.8 (Reference)	338.2 (Reference)	983.2 (Reference)	3875.5 (Reference)
Random	43.2 1.01	396.3 1.17	1225.3 1.25	5734.4 1.49
Ray trace-based	40.6 0.95	264.5 0.78	747.6 0.76	3098.8 0.80

Note: upper and lower rows indicate the total computation time (including input processing, ray trace, transport sweep, acceleration, edits, and so on) in seconds and the ratio to the reference case, respectively.

Table III. Computational time (nanoseconds) per integration of a segment per CPU core using three flux region ID assignment methods

Method	2D Single PWR fuel assembly	2D C5G7	2D KAIST-2A	2D PWR full core
Geometry-based	31.0 (Reference)	34.6 (Reference)	47.6 (Reference)	43.4 (Reference)
Random	31.4 1.01	40.4 1.17	61.3 1.29	66.2 1.53
Ray trace-based	28.0 0.90	26.0 0.75	34.6 0.73	34.2 0.79

Note: upper and lower rows indicate the computation time per integration of a segment per CPU core in nanoseconds and ratio to the reference case, respectively. Computation time for transport sweep (mainly, ray trace, exponential calculation, tally of partial current) is considerd.

Table IV. Memory requirement using three flux region ID assignment methods

Method	2D Single PWR fuel assembly	2D C5G7	2D KAIST-2A	2D PWR full core
Geometry-based	202 (Reference)	441 (Reference)	1377 (Reference)	3383 (Reference)
Random	203 1.00	442 1.00	1376 1.00	3383 1.00
Ray trace-based	203 1.00	448 1.02	1411 1.02	3827 1.13

Note: upper and lower rows indicate memory (MB) and ratio to the reference case, respectively.

Table V. Breakdown of computational time for transport calculation in the case of KAIST-2A benchmark problem

Method	Transport sweep	Exponential	Sum up average angular flux	Rearrange temporary data	Acceleration calculation	Total
Geometry-based	567.0 (reference)	141.0 (reference)	61.6 (reference)	0.0 (reference)	12.4 (reference)	782.0 (reference)
Random	799.3 1.41	142.2 1.01	81.0 1.32	0.0 -	12.4 1.00	1034.9 1.32
Ray trace-based	284.5 0.50	143.8 1.02	76.7 1.25	55.3 -	11.9 0.96	572.2 0.73

Note: upper and lower rows indicate computation time in seconds and ratio to the reference case, respectively.

Table VI. Memory bottleneck in computation time (cache efficiency) in the case of KAIST-2A benchmark problem

Method	CPI rate	Fraction of waiting time due to miss or latency of		
		L1 cache	L3 cache	DRAM
Geometry-based	0.860	5.0%	15.3%	24.2%
Random	1.084	6.2%	16.1%	32.7%
Ray trace-based	0.658	6.7%	9.4%	13.6%

CPI rate: Clock ticks per instruction rate. Cloc kticks is a unit of time in CPU. CPI rate is proportional to average wall clock time required for execution of an instruction in CPU.

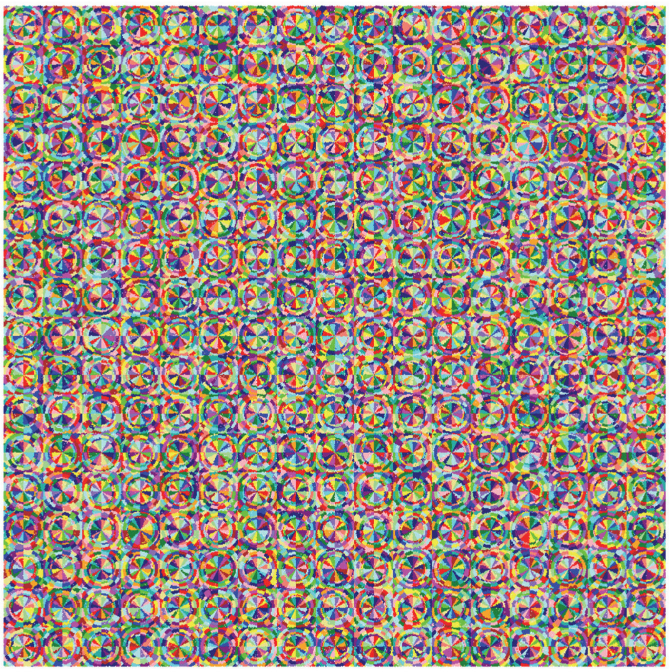


Figure 1. Example of flux region in MOC calculation for a typical PWR fuel assembly

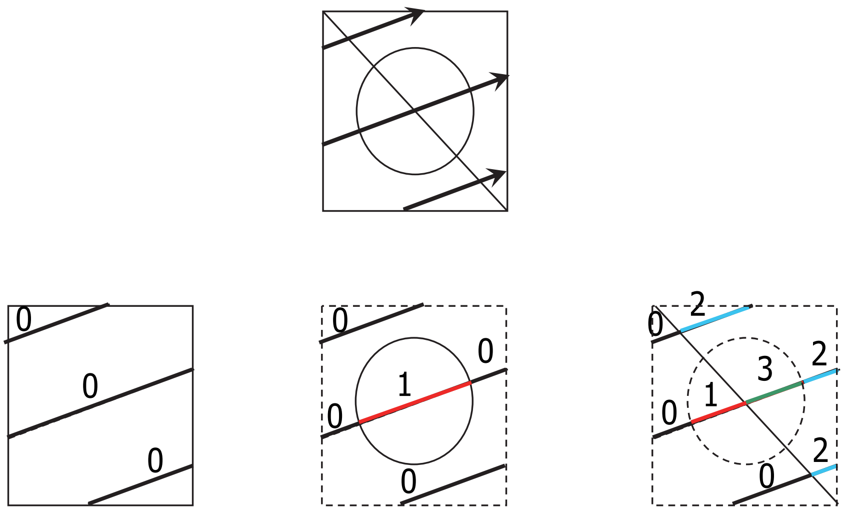


Figure 2. Flux region assignment using the factorial geometry method

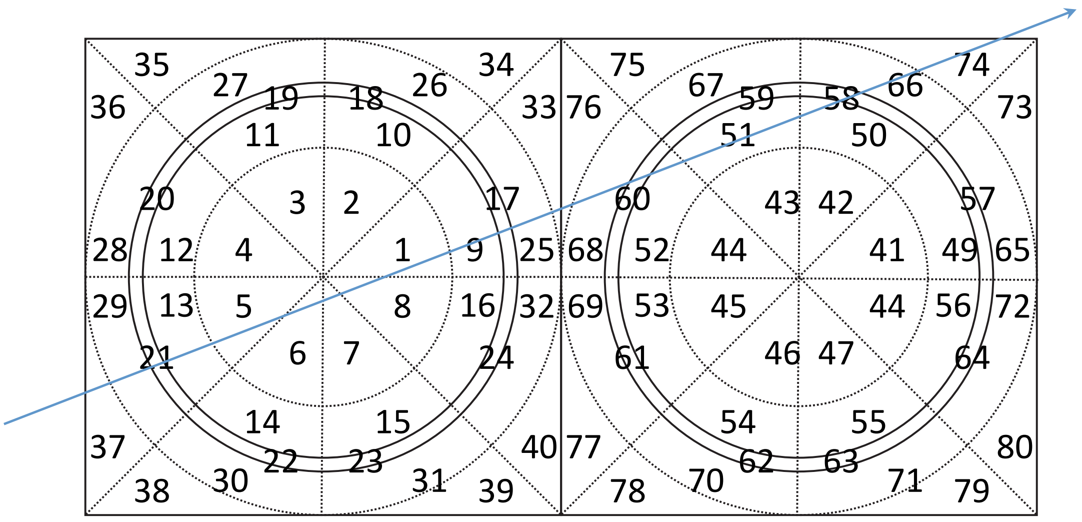


Figure 3. Example of ray trace crossing various fuel IDs. Note that fuel region ID is assigned by a rule that is different from GENESIS.

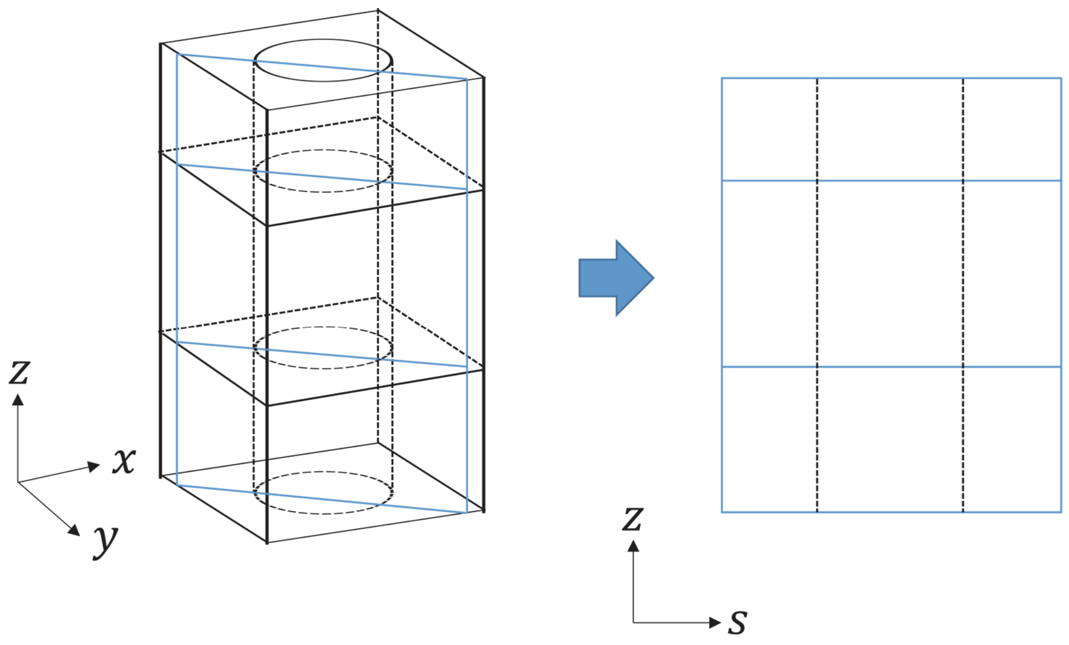


Figure 4. Concept of the LEAF method

```
prepare angular_flux_tally_array, scalar_flux _tally_array
update k-effective
set fission source
for g = 0; g < max_energy_group
   set scattering source
   for p = 0; p < max_polar_angle
      for a = 0; a < max_azimuthal_angle
        for k = 0; k < max_ray_trace
          for l = 0; l < max_segment
             calculate average angular flux
             add average angular flux to angular_flux_tally_array
   for r = 0; r < max_flux_region
      add angular_flux_tally array to scalar_flux_tally_array
```

Figure 5. Algorithm for transport sweep in the GENESIS code

```
flux_region_id = 0      //unique flux region ID starting from 0
set max_iz, max_iy      //number of fuel assemblies for x- and y- direction
for iy = 0; iy < max_iy; iy++         //loop for fuel assembly
  for ix = 0; ix < max_ix; ix++
    set max_icy, max_icx              //number of cells for x- and y- direction
    for icy = 0; icy < max_icy; icy++ //loop for cell within an assembly
      for icx = 0; icx < max_icx; icx++
        set max_freg                  //number of flux regions in a cell
        for ir = 0; ir < max_freg; ir++
          for iz = 0; iz < max_iz; iz++      //loop for axial planes
            set flux_region_id               //set unique flux region ID
            flux_region_id++
```

Figure 6. Algorithm for flux region ID with the geometry-based assignment method

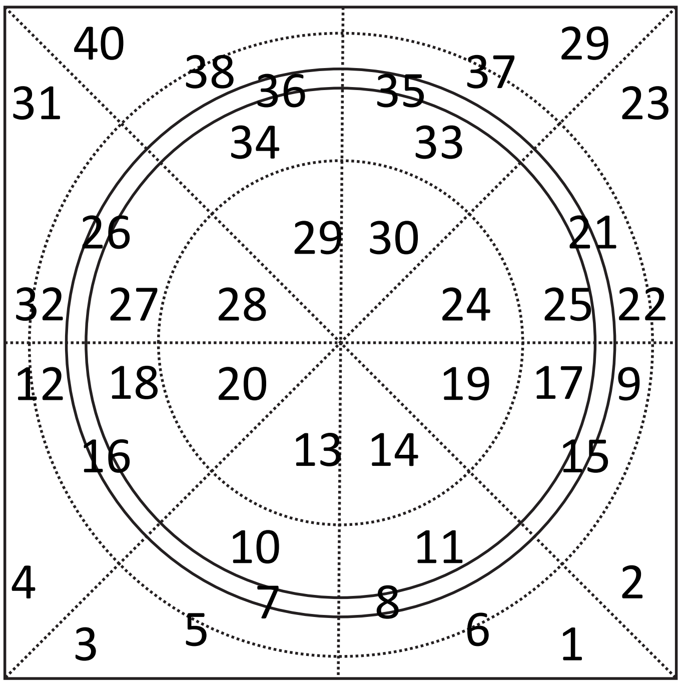


Figure 7. Flux region IDs generated by the geometry-based assignment method

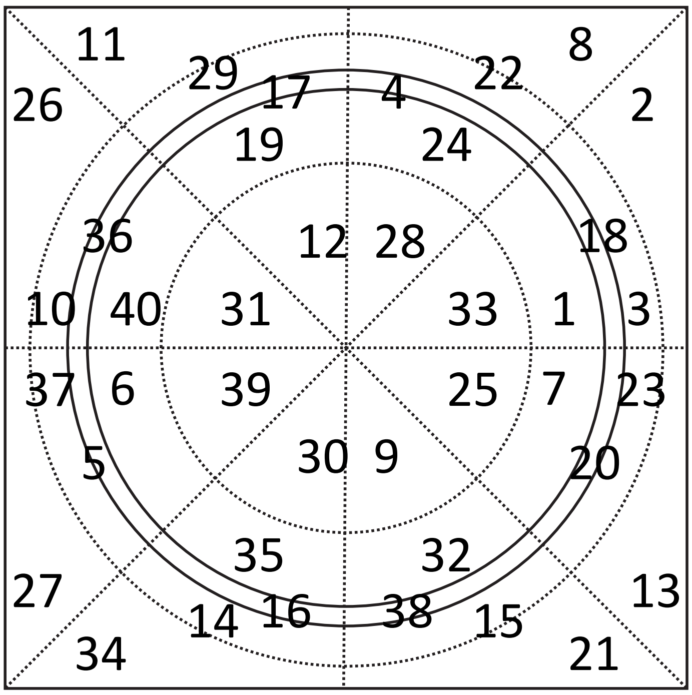
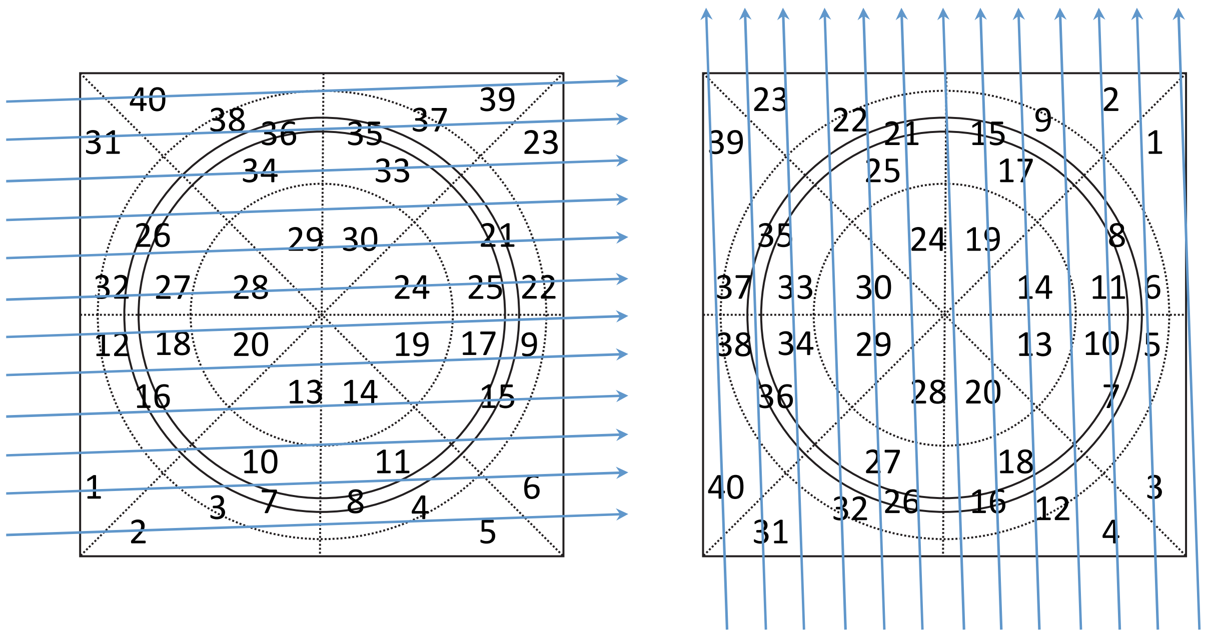


Figure 8. Flux region IDs generated by the random assignment method

```
flux_region_id = 0      //unique flux region ID starting from 0
set max_raytrace        //set number of ray traces for particular direction
for k = 0; k < max_raytrace; k++      //loop for ray traces
  set max_segment                     //set number of segments
  for i = 0; i < max_segment; i++     //loop for segments
    for iz = 0; iz < max_iz; iz++     //loop for axial planes
      if new flux region
        set flux_region_id            //set unique flux region ID
        flux_region_id++
```

Figure 9. Algorithm for flux region ID with ray trace-based assignment method



(azimuthal angle: $\frac{\pi}{32}$) (azimuthal angle: $\frac{17\pi}{32}$)

Figure 10. Flux region IDs generated by the ray trace-based assignment method. Part of the ray traces are shown; finer ray trace is actually used to determine flux region IDs.

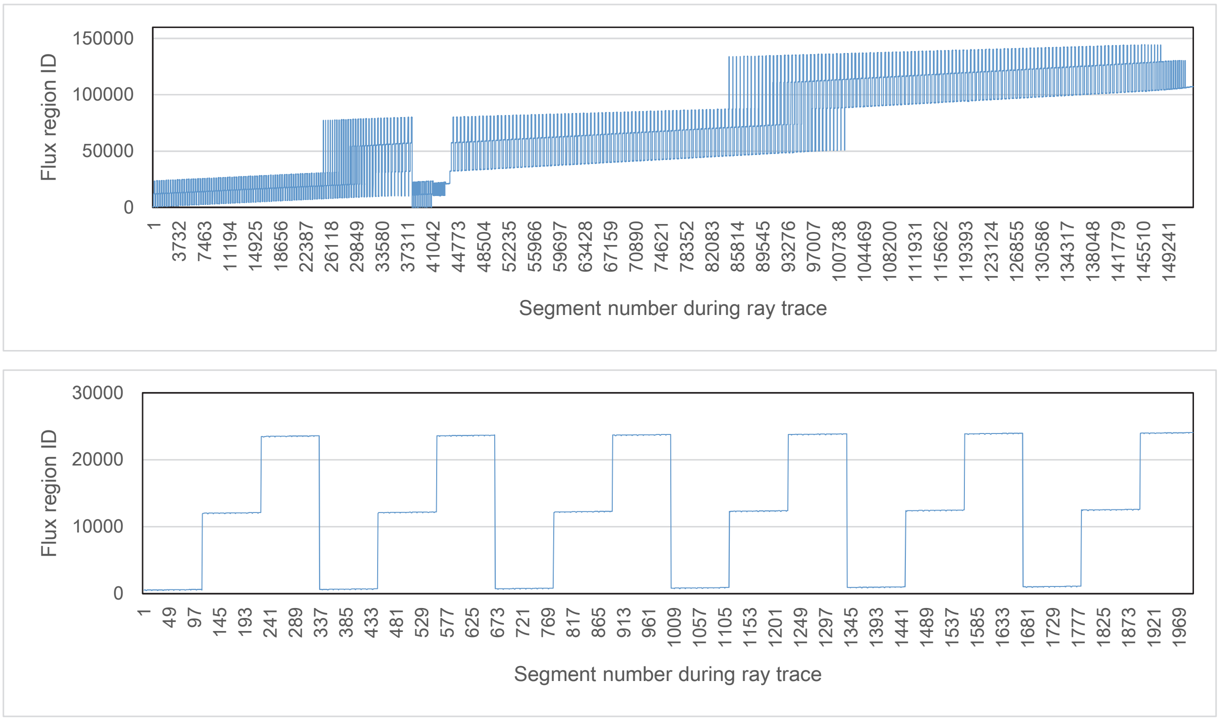
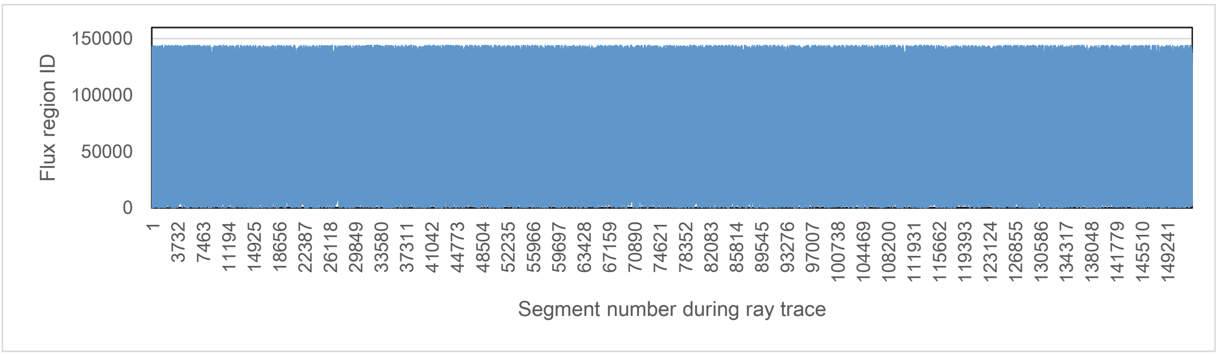


Figure 11. Access pattern of flux region ID using the geometry-based flux region assignment



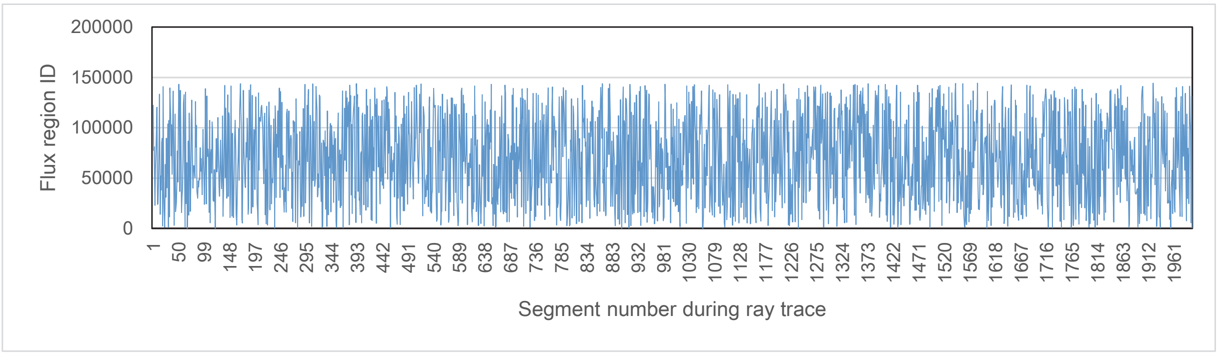
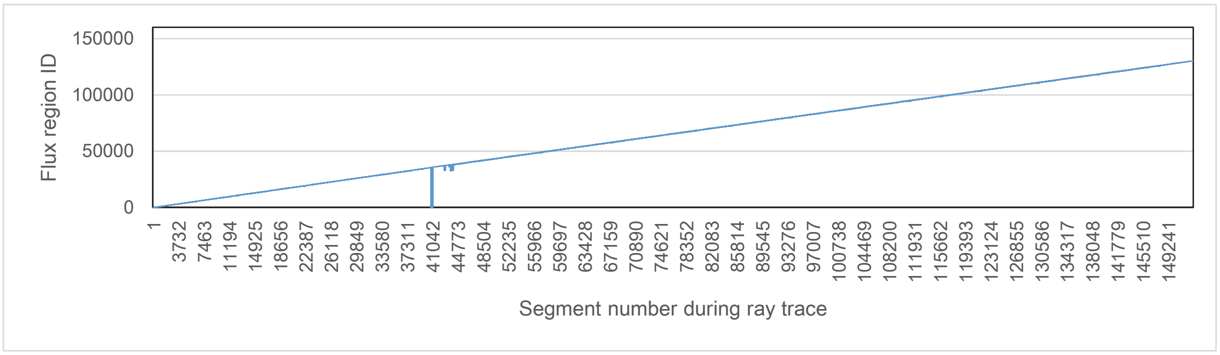


Figure 12. Access pattern of flux region ID using the random flux region assignment



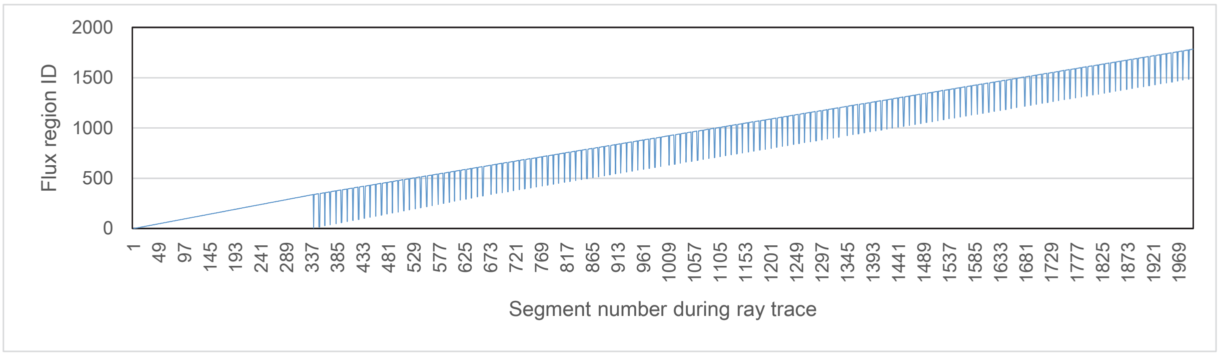


Figure 13. Access pattern of flux region ID using the ray trace-based flux region assignment