Efficient Parallelization Strategy of STREAM for Three-dimensional Whole-core Neutron Transport Calculation

Sooyoung Choi and Deokjung Lee*
Ulsan National Institute of Science and Technology (UNIST), 50 UNIST-gil, Ulsan, 44919, Republic of Korea
**Corresponding author: deokjung@unist.ac.kr*

1. Introduction

An efficient parallel algorithm has been developed for three-dimensional (3D) neutron transport code STREAM. Recently, the 3D neutron transport calculation is becoming more attractive for reactor analysis. The 3D calculation can simplify the calculation procedure and get rid of approximations of the conventional two-step method. In the former work, the 2D/3D method was implemented in STREAM and verified against the C5G7 problem [1], which showed accurate results in eigenvalue and pin power calculations. The work was done for serial calculation taking a long time for 3D problems. The parallel calculation is essential because of a lot of computational burden such as a long computing time and a lot of memory requirement.

STREAM has been parallelized to solve large sized problems, such as a 3D whole core problem. A hybrid MPI/OPENMP parallel algorithm has been developed for the 2D/3D method in STREAM. The 2D/3D method is presented first, and then the parallelization strategy describes how the domain decomposition and hybrid MPI/OPENMP parallelization are applied with a detail algorithm. At the end of this paper, preliminary results for a 3x3 mini-core problem are presented.

2. Methods and Results

2.1. 2D/3D Method in STREAM

STREAM adopts the 2D/3D method which uses 2D method of characteristics (MOC) in the radial direction and the discontinuous Galerkin method in the axial direction. The 2D/3D method is introduced briefly in this section.

The angular flux, scalar flux, and neutron source are expressed as a combination of the radial component and axial component as follows:

$$\begin{cases} \hat{\Phi}^g_{i,j,k}(s,z) \approx \sum_n \psi^g_{i,j,k,n}(s) b_n(z) \\ \phi^g_m \approx \sum_n \phi^g_{m,n} b_n(z) \\ \hat{Q}^g_{i,j,m} \approx \sum_n Q^g_{i,j,m,n} b_n(z) \end{cases}, \qquad (0)$$

where i is the index of the azimuthal angle; j is the index of the polar angle; k is the index of the MOC segment; s is the position in the radial plane; g is the index of the energy group; z is the coordinate in the axial direction; b_n is the axial basis function; and n is the order of the axial basis function.

The second order basis function is used in the method. In other words, the basis function is composed of a constant function and a linear function. The neutron transport equation with these expressions is as follows:

$$\Omega \cdot \nabla \hat{\Phi}^g_{i,j,k}(s,z) + \Sigma^g_{tr,m} \hat{\Phi}^g_{i,j,k}(s,z) = \hat{Q}^g_{i,j,m} \ , \qquad (0)$$

where m is the index of the flat source region (FSR).

By using the orthogonal properties of the basis function and integrating over the discretized axial domain, the following equation is derived which is suitable to be solved by a traditional 2D MOC solver.

$$\cos\bar{\theta}_j \frac{d\psi^g_{i,j,k,n}(s)}{ds} + \tilde{\Sigma}^g_{tr,m,n} \psi^g_{i,j,k,n}(s) = S^g_{i,j,m,n} \ , \qquad (0)$$

where the modified transport cross-section and the sources are

$$\begin{cases} \tilde{\Sigma}^g_{tr,m,1} = \Sigma^g_{tr,m} + \dfrac{2\sin\bar{\theta}_j}{\Delta z} \\ \tilde{\Sigma}^g_{tr,m,2} = \Sigma^g_{tr,m} \\ S^g_{i,j,m,1} = Q^g_{i,j,m,1} + \dfrac{2\sin\bar{\theta}_j}{\Delta z} \hat{\Phi}^{g,\mp}_{i,j,m} \\ S^g_{i,j,m,2} = Q^g_{i,j,m,2} \end{cases}, \qquad (0)$$

and $\hat{\Phi}^{g,\mp}_{i,j,m}$ denotes the axial surface sources from an adjacent plane.

A similar method was developed by Argonne National Laboratory and implemented in PROTEUS-MOC [2]. The methods in STREAM and PROTEUS-MOC are similar in that the discontinuous Galerkin method is used in the derivation. However, the final equation is different. In STREAM, the equation is derived to avoid calculating second order angular flux and scalar flux. Instead, the second order fluxes are implicitly considered by using a recursive expression for a plane interface condition. In other words, the MOC calculation denoted by Eq. (0) is performed for the first order flux only, but the STREAM method still has second order accuracy using the implicit consideration of second order flux.

```
do Outer iteration
  CMFD acceleration
  Update fission source
  do Inner iteration
    do Assemblies ···················· ! MPI parallelization with domain decomposition
      Collect boundary angular flux ·· ! MPI communication with adjacent assemblies
      Update scattering source
      do Upward and downward sweepings ! (Polar: 0~π/2) and (Polar: -π/2~0), respectively
        do Planes
          Get surface source ·········· ! Surface sources from adjacent plane or boundary
          do Azimuthal angles ········· ! OPENMP parallelization
            do Parallel rays ·········· ! Parallel MOC rays in an assembly
              do Segments·············· ! Sequential segments in a MOC ray
                do Energy groups
                  do Polar angles
                    Compute and collect angular flux change
                    Update segment outgoing flux
                  end do
                end do
              end do··················· ! Forward/backward sweeping is omitted
            end do
          end do
          Update surface source ······· ! Surface source for adjacent plane
          Compute scalar fluxes ······· ! Scalar flux of current plane
        end do
      end do
    end do
  end do
  Check convergence
end do
```
Fig. 1. Algorithm of 3D neutron transport calculation in STREAM.

2.2. Algorithm of Neutron Transport Calculation

In the former work, the algorithm for the transport calculation was optimized to reduce memory usage required to store surface sources [1]. The loops for azimuthal angle, polar angle, and energy group were placed outer than loops for plane, MOC ray, and segment. The former algorithm was effective to reduce memory because it does not need to store the surface source for all angles and energy groups at the same time. However, it could lose computational efficiency because of repeating the same calculations to obtain space information.

The new algorithm in the STREAM 2D/3D method is shown in Fig. 1. Basically, STREAM solves a steady-state eigenvalue problem. In order to calculate the eigenvalue, k_{eff}, in which we are interested, the inverse power method is used, and it is composed of inner/outer iteration loops. Before performing the inner iteration loop, the CMFD module is called to accelerate convergence of fission sources and scalar flux. In the inner iteration, a loop for assemblies exists. STREAM uses an assembly-wise MOC modular. The MOC rays and segments are generated for each assembly type and stored in the memory to be used in the MOC sweeping. Here, the assembly indicates not only fuel assembly, but also the reflector region. The outgoing angular fluxes at assembly boundaries come from adjacent assemblies. The loop of the assembly can cause slower convergence rate because sequential MOC ray tracing across assemblies is not possible. The MOC rays between problem boundaries need to be tracked in sequence to improve the convergence rate. However, most MOC codes, including STREAM, use CMFD acceleration which is very powerful to accelerate the global fission source and flux, and the CMFD acceleration can eliminate the disadvantage arising from discontinuous ray tracking (or Jacobi method). The reason for using an assembly loop is related to assembly-wise domain decomposition which is described in Section 2.3.

There are loops for upward/downward sweeping and planes. The upward sweeping is for polar angles from 0 to $\pi/2$ while the downward sweeping is for $-\pi/2$ to 0. The angle-dependent axial surface source comes from an adjacent plane, and a 2D MOC solver is used to solve Eq. (0) for the current plane. After MOC sweeping for the plane, the surface source is updated for next plane. The MOC sweeping is carried out through the azimuthal angle loop to the polar angle loop. In the innermost loop, fractional change of angular flux is calculated and collected for each angle. Because the length of the MOC ray segment projected on the *x-y* plane is the same for energy groups and polar angles, it is not needed to repeatedly calculate the loops for energy groups and polar angles. By placing the loops for energy group and polar angle at the inner of the MOC ray, it is possible to reduce the number of calculations and it is possible to maximize cache usage. In addition, the forward/backward sweeping is adopted in each ray sweeping to reuse data, which is obtained during the forward sweeping, for opposite azimuthal angle sweeping. After the azimuthal angle loop, the angle dependent surface source is calculated using the angular flux for each flat source region. The scalar flux is also calculated by summing up the angular flux.

2.3. Domain Decomposition and MPI Parallelization

The main purpose of developing 3D STREAM is to handle 3D whole core problems in high resolution with pin or sub-pin level and few centimeters height. For this purpose, extremely many flat source regions need to be considered. Therefore, the domain decomposition technique must be applied to reduce a huge amount of memory. The target machine to run is an in-house Linux cluster which has ~400 cores and ~6 TB memory in total. Simply, there are two ways to decompose the space domain. One is decomposing the plane in the axial direction (*i.e.*, plane decomposition), and the other is decomposing the *x*-*y* plane according to the assembly boundary (*i.e.*, assembly decomposition). Both ways can reduce the memory required for scalar fluxes and cross-sections by storing the data in distributed storage nodes with MPI. However, it should be noted that additional interfacing data is required to link the decomposed domains.

Table I: Estimation of memory requirement for interfacing data: plane domain decomposition.

Category	Values
# of 2D FSRs/assembly	10,000
# of assemblies	300
# of energy groups	72
# of azimuthal angles	48
# of polar angles	6
Real type (byte)	4
Memory / plane domain (GB)	249
# of axial planes	200
Total memory (GB)	49,766

Table II: Estimation of memory requirement for interfacing data: assembly domain decomposition.

Category	Values
# of boundary points/assembly	1,000
# of axial planes	200
# of energy groups	72
# of azimuthal angles	48
# of polar angles	6
Real type (byte)	4
Memory / assembly domain (GB)	17
# of assemblies	300
Total memory (GB)	4,977

In case of plane decomposition, axial surface sources need to be stored. Table I shows the estimation of the memory requirement for interfacing data. Because the surface source is angular dependent and each flat source region has its own surface source, a huge memory is required. For a 3D whole core problem without symmetric modeling, 249 GB is required for each plane domain. When 200 axial plane domains are used, 49,766GB memory is required. Moreover, a large amount of memory can be required because the required memory is proportional to the number of flat source regions. The number of flat source regions can increase due to several possible reasons, *i.e.*, ~15 radial sub-divisions in a gadolinia fuel pellet.

In case of MOC assembly decomposition, angular fluxes for MOC rays at assembly boundaries need to be stored. Table II shows the memory requirement for assembly-wise domain decomposition. About 1,000 boundary points per assembly exist when a 0.05 cm MOC ray spacing condition is used. The boundary point is the point crossed by the MOC ray at the assembly boundaries. For each point, outgoing or incoming angular fluxes exist depending on its position and azimuthal angles. For a 3D whole core problem, around 17 GB is required to store the angular flux at the assembly boundary for each assembly domain. In total, 4,977 GB is required to solve a 3D whole core problem. The amount of memory requirement is much smaller than that of the plane-wise domain decomposition. For this reason, the assembly-wise domain decomposition is applied to the STREAM 3D transport solver. The number of boundary points can be increased with finer MOC ray spacing. For practical applications, however, a 0.05cm ray spacing condition gives a reasonable result and it is rare to use the strict condition. The planes and assemblies can be decomposed simultaneously. However, this way is not considered in this work because the number of domains is sufficiently large, and the number of available computing cores is limited.

The assembly domain decomposition is applied to the MOC solver as described in Fig. 1. The loop for assemblies is parallelized using MPI. A MPI process has one or more assemblies in its domain. Each MPI process stores data fluxes, sources, and cross-sections used in its domain. Neighboring assemblies or decomposed domains exchange incoming and outgoing angular fluxes at the interfacing surfaces with point-to-point communication. This is done by the MPI_SEND and MPI_RECV functions in MPI library. Except for this communication, no MPI communication is required inside the loop for an assembly. After each power iteration, collective communication, which is done by MPI_ALLREDUCE, is needed to calculate the eigenvalue and to check the convergence of sources.

2.4. OPENMP Parallelization

In addition to MPI, OPENMP is applied to reduce the computing time. Each MPI process calls OPENMP threads, and the OPENMP threads distribute work with shared memory allocated for each MPI process. Parallelization is generally most efficient when implemented at the coarsest level as possible if enough work is shared. This is done by parallelizing azimuthal angles as shown in Fig. 1. The loops for upward/downward sweepings and planes are not candidates for OPENMP parallelization because planes need to be swept sequentially (such as Gauss-Seidel) in the assembly-wise domain decomposition. In the

OPENMP parallelization, it is important to consider the data synchronization because they can access the same variables during work sharing. If the azimuthal angle is parallelized, it is not needed to give an effort to synchronize the angular flux because the OPENMP threads use different memory locations with their own azimuthal angle indexes. This is possible because a flux change and outgoing angular flux, which are calculated in innermost loop in Fig. 1, are stored in the angular flux variable with the azimuthal angle index. In other words, the angular fluxes are synchronized implicitly because the angular flux variable has an index of azimuthal angle, and the angle is decomposed by OPENMP threads. The only data that needs to be synchronized within the OPENMP loop is CMFD currents at pin-cell boundaries. The memory size of CMFD currents are much smaller than MOC fluxes, thus synchronizing the current is not a big burden. OPENMP is also uses many subroutines in STREAM, including the azimuthal angle loop in the 3D transport solver, but it is not treated in this paper.

2.5. Numerical Results

The performance of the parallel algorithm was examined using a 3x3 mini-core problem. The test case has 9 fuel assemblies. The UOX and MOX fuel assemblies, which are same assemblies as in the 3D C5G7 problem [3], are placed in the min-core with a checkerboard configuration. The number of flat source regions in the 2D plane is 41,616, and the number of axial planes is 21. The following MOC conditions are used: 0.05 cm ray spacing, 48 azimuthal angles, and 10 polar angles.

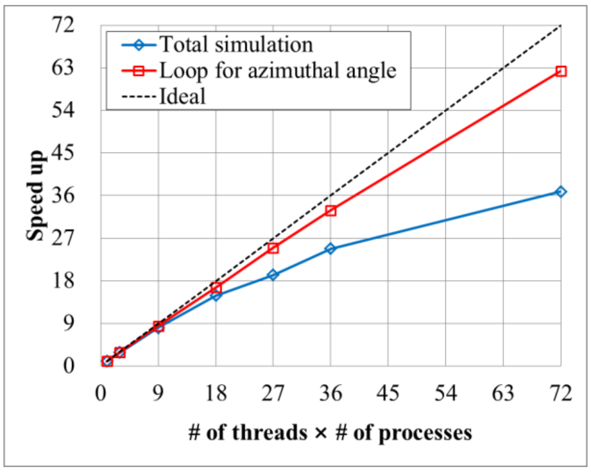

Fig. 2. Speed up factor for 3x3 mini-core problem.

Table III: Elapsed time for 3x3 mini-core problem.

# of threads × # of processes	Simulation time (sec)	
	Total simulation	Loop for azimuthal angle
1	3,298	2,681
9	406	316
72	90	43

The speed up factor and simulation time are shown in Fig. 2 and Table III, respectively. As shown in the figure and table, the loop for azimuthal angle is well parallelized with a speed up factor of 62 when 72 threads and processes (*i.e.*, 9 MPI × 8 OPENMP) are used. The calculation efficiency is about 87 %. On the other hand, the efficiency of the overall simulation is about 51 %. There are several reasons for this degradation. First, the parallel calculation is not fully implemented. The parts for flux calculations outside the azimuthal angle, and routines for CMFD are needed to be improved to utilize the multi cores. From now on, the loop for the azimuthal angle is focused to be parallelized because it is the most time consuming part in the 3D calculation. Second, elapsed time for MPI communication is inevitable part when the MPI is used. STREAM is being developed, and it is expected to be improved in the future.

3. Conclusions

The neutron transport code STREAM has been parallelized to solve a large size 3D problem. The 2D/3D method and its characteristic have been described. The hybrid MPI/OPENMP parallel algorithm has been developed with consideration of the characteristic of the 2D/3D method. STREAM adopts assembly-wise domain decomposition to store required memory in distributed storage. The assembly domains are parallelized in the transport solver. In addition to enhancing performance, OPENMP is applied and work for the azimuthal angles is distributed to each of the OPENMP thread. A 3x3 mini-core problem was solved to examine the performance of the parallel algorithm, and it was confirmed that the developed algorithm was implemented successfully and showed a possibility to be used in 3D whole core problems. In future work, the parallel algorithm will be implemented and improved for the whole code, and larger sized p!troblems, *i.e.*, 3D whole core problem, will be solved.

ACKNOWLEDGMENTS

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT).
(No.NRF-2016M2B2A9A02945205)

REFERENCES

[1] Y. Zheng, S. Choi, D. Lee, A new approach to three-dimensional neutron transport solution based on the method of characteristics and linear axial approximation, Journal of Computational Physics. vol 350, pp. 25–44, 2017.
[2] A. Marin-Lafleche, M.A. Smith, C.H. Lee, PROTEUS-MOC: a 3D deterministic solver incorporating 2D method of characteristics, in: Proc. M&C 2013, Sun Valley, May 5–9, 2013.
[3] M.A. Smith, E.E. Lewis, B.C. Na, Benchmark on Deterministic Transport Calculations without Spatial Homogenization, NEA/NSC/DOC(2005)16, 2005.