A Comparative Study on One-node and Two-node Nodal Methods for GPU-accelerated Fine-mesh SP3 Calculations

Hyunsik Hong [a*], Hwanyeal Yu [a], Wi-Soo Jeong [a], Jooil Yoon [b]
[a] *KEPCO Nuclear Fuel Co. Ltd., 242, Daedeok-daero 989beon-gil, Yuseong-gu, Daejeon, 34057, Republic of Korea*
[b] *KEPCO International Nuclear Graduate School, 658-91 Haemaji-ro, Seosaeng-myeon, Ulju-gun, Ulsan, 45017, Korea*
[*] *Corresponding author: hshong0723@knfc.co.kr*

***Keywords : SP3, GPGPU, Semi-transport, two-step core calculation**

1. Introduction

For the last decade, many studies in the computational reactor physics field have introduced General-Purpose Computing on Graphics Processing Units (GPGPU) technology to improve practicality of the direct whole core calculation (DWCC) method. This trend led to the industrial application of DWCC, represented by the Monte-Carlo code PRAGMA [1] used in the innovative Small Modular Reactor (iSMR) research. Nevertheless, expanding the application of DWCC within domestic industries faces practical difficulties, primarily due to the global shortage of high-performance GPUs and memory semiconductors. Such constraints necessitate a revisit to the innovative two-step calculation (TSC) method employing the diffusion or semi-transport theory and nearly pin-cell level homogenized multigroup (MG) cross sections (XSs). This proposed method serves as a cost-effective alternative to DWCC, while achieving notably enhanced accuracy over conventional TSC.

Previous studies [2, 3] indicated that using eight or more energy groups requires a higher-order solution method. For PWR analysis, a nodal solver based on the simplified P3 (SP3) theory is an appropriate choice, but MG SP3 nodal calculations for the pin-cell level fine-mesh (FM) have been considered burdensome. Recently, Ref. [4] demonstrated the feasibility of accelerating the one-node source expansion nodal method using a GPU, assuming that its compute-bound nature is suitable for the GPU architecture. However, a detailed comparative analysis with the memory-bound two-node scheme was not performed. Although the memory-bound nature is generally not suitable for the GPU, the two-node scheme is very stable and widely used in many CPU-based codes, and GPUs have high memory bandwidth. Therefore, the optimality of the one-node scheme needs to be verified.

In this study, both one-node and two-node SP3 nodal expansion method (NEM) solvers were implemented to compare the computational performance in a GPGPU environment. Section 2 provides a brief description of the two solvers, and Section 3 presents the simple test results performed in a PC environment.

2. Implementation of the SP3 NEM Solvers

Both one-node and two-node SP3 NEM solvers were implemented using CUDA C/C++ 12.6. The nodal kernels are based on FP32 (single precision) arithmetic for two primary reasons. First, the FP32 performance of commercial GPUs is significantly higher than their FP64 (double precision) performance. Second, FP32 is more advantageous for efficient utilization of the limited VRAM. To suppress precision loss from the reduced number of significant figures, Legendre polynomials up to the fourth order were adopted as the basis polynomials for the NEM, replacing the standard basis.

Additionally, CPU-based versions of the two solvers were developed to provide reference solutions for the GPU solvers. The CPU versions are also utilized to evaluate the parallel scalability of each scheme, which is more readily analyzed in a multi-core CPU environment than in a GPU architecture.

2.1. One-node solver

Since the one-node NEM is a well-established method, this section focuses on the modifications made in this study. According to the SP3 theory, the partial odd moments at the node boundaries can be written as Eq. (1), where $\hat{\phi}_0$ and ϕ_2 represent the summed flux and the second moment. The superscript plus (+) and minus (-) denote the outgoing and incoming directions. Additional superscripts l and r indicate the left and right boundaries in the $u = x, y, z$ directions.

$$J_1^{\pm} = \frac{1}{4}\hat{\phi}_0^{\text{surf}} \pm \frac{1}{2}J_1 - \frac{3}{16}\phi_2^{\text{surf}}$$
$$J_3^{\pm} = \frac{7}{16}\phi_2^{\text{surf}} \pm \frac{1}{2}J_3 - \frac{1}{16}\hat{\phi}_0^{\text{surf}} \quad (1)$$

As described in Ref. [5], the derivation of the one-node SP3 nodal kernel can be simplified by employing the artificial partial moment concept. This approach removes the coupling terms between the partial first and third moments, as shown in Eq. (2). Given that the artificial partial moments are obtained by simply truncating Eq. (1), the net odd moments remain unchanged.

$$\tilde{J}_1^{\pm} = \frac{1}{4}\hat{\phi}_0^{\text{surf}} \pm \frac{1}{2}J_1, \qquad \tilde{J}_3^{\pm} = \frac{7}{16}\phi_2^{\text{surf}} \pm \frac{1}{2}J_3 \quad (2)$$

It must be noted that careful handling of the artificial partial incoming odd moments is required at the system boundaries. For a reflective boundary, the treatment of real and artificial partial moments is identical. In contrast, for a vacuum boundary where the real partial incoming

currents are zero, the artificial partial moments are non-zero, as expressed in Eq. (3).

$$\begin{aligned}\tilde{J}_1^- &= \frac{3}{109}(\tilde{J}_1^+ + 8\tilde{J}_3^+)\\ \tilde{J}_3^- &= \frac{1}{109}(14\tilde{J}_1^+ + 3\tilde{J}_3^+)\end{aligned} \tag{3}$$

By employing the Legendre polynomial basis and the artificial partial moment concept, the node-averaged even moments ($\hat{\phi}_0$, ϕ_2) and the artificial partial outgoing moments are derived as Eqs. (4) and (5). The superscript u indicates the x, y, z directions, while the subscript m $(= 0,2)$ denotes the corresponding even moment for each coefficient. Here, $q_{m,i}^u$ is the source coefficient. The flux coefficient $a_{m,i}^u$ is calculated by the procedure in Eq. (6), and $c_{m,i}^u$ is defined by Eq. (7).

$$\begin{aligned}\hat{\phi}_0 &= \hat{\phi}_0, \qquad \hat{\phi}_2 = \phi_2,\\ \Sigma_{r,0} &= \Sigma_r, \qquad \Sigma_{r,2} = \frac{4}{3}\Sigma_r + \frac{5}{3}\Sigma_t,\\ \hat{\phi}_m &= \frac{\bar{Q} + \sum_{u=x,y,z} C_m^u}{\Sigma_{r,m} + \sum_{u=x,y,z} 40c_{m,1}^u/h^u},\\ C_m^u &= \frac{1}{h^u}\begin{bmatrix}\left(1 - c_{m,2}^u - c_{m,3}^u\right)\left(\tilde{J}_{m+1}^{u,r-} + \tilde{J}_{m+1}^{u,l-}\right)\\ -28c_{m,1}^u a_{m,2}^u\end{bmatrix}\end{aligned} \tag{4}$$

$$\begin{aligned}\tilde{J}_{m+1}^{u,l+} &= c_{m,1}^u\left(20\hat{\phi}_m + 14a_{m,2}^u\right)\\ &\quad + c_{m,2}^u\tilde{J}_{m+1}^{u,l-} + c_{m,3}^u\tilde{J}_{m+1}^{u,r-} - c_{m,4}^u a_{m,1}^u,\\ \tilde{J}_{m+1}^{u,r+} &= c_{m,1}^u\left(20\hat{\phi}_m + 14a_{m,2}^u\right)\\ &\quad + c_{m,3}^u\tilde{J}_{m+1}^{u,l-} + c_{m,2}^u\tilde{J}_{m+1}^{u,r-} + c_{m,4}^u a_{m,1}^u\end{aligned} \tag{5}$$

$$\begin{aligned}\mu_0 = \frac{1}{4},\ \ \mu_2 = \frac{7}{16},\ \ \beta_m^u = \frac{D_m}{h^u},\ \ \Sigma_{D,m}^u = \frac{4\beta_m^u}{h^u},\\ \hat{\phi}_m^{u,l} = \frac{1}{2\mu_m}\left(\tilde{J}_{m+1}^{u,l-} + \tilde{J}_{m+1}^{u,l+}\right),\\ \hat{\phi}_m^{u,r} = \frac{1}{2\mu_m}\left(\tilde{J}_{m+1}^{u,r-} + \tilde{J}_{m+1}^{u,r+}\right),\\ a_{m,1}^u = \frac{2q_{m,1}^u + 15\Sigma_{D,m}^u\left(\hat{\phi}_m^{u,r} - \hat{\phi}_m^{u,l}\right)}{2\left(15\Sigma_{D,m}^u + \Sigma_{r,m}\right)},\\ a_{m,2}^u = \frac{2q_{m,2}^u + 35\Sigma_{D,m}^u\left(\hat{\phi}_m^{u,r} + \hat{\phi}_m^{u,l} - 2\hat{\phi}_m\right)}{2\left(35\Sigma_{D,m}^u + \Sigma_{r,m}\right)},\\ a_{m,3}^u = \frac{-2q_{m,1}^u + \Sigma_{r,m}\left(\hat{\phi}_m^{u,r} - \hat{\phi}_m^{u,l}\right)}{2\left(15\Sigma_{D,m}^u + \Sigma_{r,m}\right)},\\ a_{m,4}^u = \frac{-2q_{m,2}^u + \Sigma_{r,m}\left(\hat{\phi}_m^{u,r} + \hat{\phi}_m^{u,l} - 2\hat{\phi}_m\right)}{2\left(35\Sigma_{D,m}^u + \Sigma_{r,m}\right)}\end{aligned} \tag{6}$$

$$\begin{aligned}c_{m,1}^u &= \frac{\beta_m^u\mu_m}{10\beta_m^u + \mu_m},\\ c_{m,2}^u &= \frac{\mu_m^2 - 60(\beta_m^u)^2}{(10\beta_m^u + \mu_m)(6\beta_m^u + \mu_m)},\\ c_{m,3}^u &= -\frac{4\beta_m^u\mu_m}{(10\beta_m^u + \mu_m)(6\beta_m^u + \mu_m)},\\ c_{m,4}^u &= \frac{10\beta_m^u\mu_m}{6\beta_m^u + \mu_m}\end{aligned} \tag{7}$$

The CPU version of the one-node solver is based on the same formulation as the GPU version, yet they differ in specific implementation details. The CPU solver employs FP64 arithmetic, as there is no critical need to maximize FP32 performance or consider strict VRAM limitations. Furthermore, OpenMP parallelization was implemented to ensure computational efficiency.

2.2. Two-node solver

The procedure for obtaining the net first and third moments using the SP3 NEM on each surface of a node is essentially identical to determining the net current in diffusion theory. By utilizing J_{m+1}^{NEM} obtained by the two-node NEM, the correction factor of Coarse Mesh Finite Difference (CMFD) method $\widehat{D}_m$ can be calculated as in Eq. (8). Here, $\widetilde{D}_m$ is the coupling coefficient of Finite Difference Method (FDM), defined by the diffusion coefficients and pitches of the nodes on both the left (L) and right (R) sides of the surface. A CMFD system is then constructed by incorporating $\widehat{D}_m$ into the FM SP3 FDM system, and the even moments are calculated using an efficient linear solver.

$$\begin{aligned}J_{m+1}^{FDM} &= -\widetilde{D}_m\left(\hat{\phi}_m^R - \hat{\phi}_m^L\right),\\ \widehat{D}_m &= \frac{J_{m+1}^{FDM} - J_{m+1}^{NEM}}{\hat{\phi}_m^R + \hat{\phi}_m^L}\end{aligned} \tag{8}$$

However, a numerical issue arises because the second moment can be negative. Consequently, the denominator term, $\phi_2^R + \phi_2^L$, may take a very small value near zero or even become zero. This leads to an anomalous increase in $\widehat{D}_m$, which significantly degrades the stability of the linear system or causes the calculation to diverge.

$$C_{\text{Fixup}} = \ \text{Max}(|\phi_2|) + \alpha\ \text{Ave}(\phi_0) \tag{9}$$

To address this, an effective solution was proposed in Ref. [6], which ensures that the second moment always maintains a positive value by adding a sufficiently large constant C_{Fixup}. This method is referred to as FF-CMFD in Ref. [6]. The constant C_{Fixup} is calculated according to Eq. (9), where $\text{Max}(|\phi_2|)$ is the maximum absolute value of ϕ_2, $\text{Ave}(\phi_0)$ is the average value of ϕ_0, and α is an arbitrary positive constant. In this study, α was set

to 2, but the final calculation results remain identical regardless of the α value.

Eq. (10) represents the SP3 equation where the 0th and 2nd moments are coupled. By applying FF-CMFD, the moments are decoupled, and C_{Fixup} is added to ϕ_2 as shown in Eq. (11). This modified equation is referred to as the shifted 2nd moment equation. Since C_{Fixup} is also added to the source term on the right-hand side, ϕ_2 can be correctly recovered by subtracting C_{Fixup} from the solution of the shifted 2nd moment equation. Because $\phi_2^R + \phi_2^L + 2C_{\mathrm{Fixup}}$ is maintained larger than zero, the anomalous behavior of $\widehat{D}_2$ can be prevented.

$$\begin{bmatrix} -D_0\nabla^2 + \Sigma_{r,0} & -2\Sigma_{r,0} \\ -\frac{2}{3}\Sigma_{r,0} & -D_2\nabla^2 + \Sigma_{r,2} \end{bmatrix} \begin{bmatrix} \hat{\phi}_0 \\ \phi_2 \end{bmatrix} = \begin{bmatrix} q_0 \\ -\frac{2}{3}q_0 \end{bmatrix} \quad (10)$$

$$\begin{aligned} -D_0\nabla^2\hat{\phi}_0 + \Sigma_{r,0}\hat{\phi}_0 &= q_0 + 2\Sigma_{r,0}\phi_2 \\ -D_2\nabla^2\phi_2 + \frac{5}{3}\Sigma_t\left(\phi_2 + C_{\mathrm{Fixup}}\right) &= -\frac{2}{3}q_0 + \frac{2}{3}\Sigma_r\phi_0 + \frac{5}{3}\Sigma_t C_{\mathrm{Fixup}} \end{aligned} \quad (11)$$

As a rigorous and efficient linear system solver, the BiCGSTAB algorithm with ILU0 preconditioner was implemented using the cuBLAS and cuSPARSE libraries. cuBLAS and cuSPARSE are high-performance linear algebra and sparse matrix libraries natively included in the CUDA toolkit. Because these libraries are highly optimized by the hardware manufacturer, they are expected to exhibit superior computational performance compared to custom implementations based on element-by-element operations.

The CPU version of the two-node solver is also based on the same formulation as the GPU version but employs FP64 arithmetic. To handle the large sparse matrix efficiently, the Iterative Sparse Solver (ISS) from the Intel Math Kernel Library (MKL) was employed. Intel MKL provides highly optimized implementation of ILU0 and FGMRES. The combination of these two is considered one of the most robust and high-speed linear solvers available for PCs equipped with Intel CPUs.

Unlike the GPU version of the FM CMFD solver which uses only the node-major (NM) ordering scheme, both the node-major and group-major (GM) schemes were implemented for the CPU version. Despite NM having been the conventional choice for CPU-based TSC codes, GM is worth evaluating for its potential to improve memory coherence and parallel efficiency. The scalability of both schemes is compared in Section 3.

2.3. Assembly-level CMFD acceleration

To achieve fast convergence of the one-node and two-node FM SP3 calculations, an assembly-level CMFD acceleration scheme based on diffusion theory is employed. Because the reference net current at each surface is calculated by the FM SP3 solver, the accuracy of the coupling coefficient is guaranteed. Therefore, a diffusion solver offering lower computational cost and more robust convergence is a superior choice. The subsequent update of the FM MG flux and fission source using the assembly-level CMFD solution follows the standard procedure.

Notably, for the one-node solver, the *partial current update scheme with an assembly-level CMFD solver* described in Ref. [4] was adopted. This method utilizes the concept of mp-CMFD. By predetermining the FDM coupling coefficients ($\widetilde{D}$) and partial correction factors ($\widehat{\widetilde{D}}^{+}$) for each FM surface before the assembly-level CMFD calculation, the incoming partial first moments ($\tilde{J}_1^-$) at the FM boundaries can also be updated using the assembly-level flux solution. This approach enhances the convergence speed of the solver.

3. Numerical Results and Discussions

To evaluate the computational performance of the two SP3 NEM solvers, a large 3D assembly-array problem was constructed. This problem consists of an 8x8 array of CE-type fuel assemblies with reflective boundaries in the radial direction. In the axial direction, the active fuel length of 381 cm is divided into 20 planes, with 30 cm axial reflectors placed at both the top and bottom. Vacuum boundaries are applied to the axial surfaces. By setting the FM size to the pin-cell level, the total number of FM is 360,448, which is comparable in scale to a 1/4 core of an OPR-1000 PWR or a full core of an iSMR.

The pin-homogenized eight-group XSs was generated using the DWCC code nTRACER [2]. It is noteworthy that all four calculation cases, including both one-node and two-node solvers executed on CPU and GPU, converged to the same k-effective value, 1.094466. This consistency confirms that the use of FP32 arithmetic in the GPU solver does not introduce significant numerical bias in the global eigenvalue calculation.

Performance evaluations were conducted on a PC equipped with an 13$^{\mathrm{th}}$ Gen. Intel i9-13950HX CPU and an NVIDIA GeForce RTX 4060 Laptop GPU. The system is configured with 64 GB of RAM and 8 GB of VRAM. The Intel 13$^{\mathrm{th}}$ Gen. i9 CPU features a hybrid architecture with eight Performance-cores (P-cores) and eight Efficient-cores (E-cores). Because these two types of cores possess distinct execution capabilities, their simultaneous use can introduce significant variability in parallel performance. To ensure a consistent evaluation of the parallel scalability, all E-cores were disabled, and the Hyper-Threading feature of the P-cores was turned off via the BIOS settings. This ensures that all parallel threads are executed on identical physical P-cores.

The convergence criterion was defined as a relative error in the fission source of less than 1.0E-6. For the one-node solver, the nodal sweep was performed using the red-black method, which can be conceptually viewed as a matrix-free implementation of the Gauss-Seidel

iterative solver. Without the assembly-level CMFD, the one-node solver required 7,200 nodal sweeps to reach convergence. This number significantly decreased to 300 sweeps when the assembly-level CMFD was applied, confirming the effectiveness of the scheme.

In the case of the two-node solver, a notable difference was observed in the number of surface moment updates between NM and GM. Because GM performs decoupled calculations for each energy group in a Jacobi manner, its convergence rate is inherently slower than that of NM. Without the assembly-level CMFD, the number of surface moment updates was 196 for NM and 604 for GM. With the assembly-level CMFD, these counts were reduced to 64 for NM and 156 for GM, respectively. Following the completion of the surface moment updates, four to five outer iterations were typically required.

Table I: Comparison of Calculation Times
(One-node, without the assembly-level CMFD)

Case	Elapsed time (s)	Speedup
OMP, 1trd	10556.6	
OMP, 2trd	5649.3	1.9x
OMP, 4trd	3288.6	3.2x
OMP, 8trd	2112.7	5.0x
GPU	44.1	239.4x

Table II: Comparison of Calculation Times
(Two-node, NM, without the assembly-level CMFD)

Case	Elapsed time (s)	Speedup
OMP, 1trd	1734.5	
OMP, 2trd	1223.6	1.4x
OMP, 4trd	1001.4	1.7x
OMP, 8trd	879.9	2.0x
GPU	262.7	6.6x

Table III: Comparison of Calculation Times
(Two-node, GM, without the assembly-level CMFD)

Case	Elapsed time (s)	Speedup
OMP, 1trd	6278.0	
OMP, 2trd	3448.7	1.8x
OMP, 4trd	2215.0	2.8x
OMP, 8trd	1536.1	4.1x

Tables I through III and Fig. 1 summarize the results of the pure FM calculations without the assembly-level CMFD. The CPU-based results are categorized by the number of OpenMP (OMP) threads. As shown in Fig. 1, the two-node NM solver is the most efficient choice when the number of CPU threads is limited. However, this approach exhibits poor parallel scalability. Increasing the number of CPU cores by a factor of eight results in only a twofold speedup. The speedup achieved by the GPU version is also limited to 6.6x.

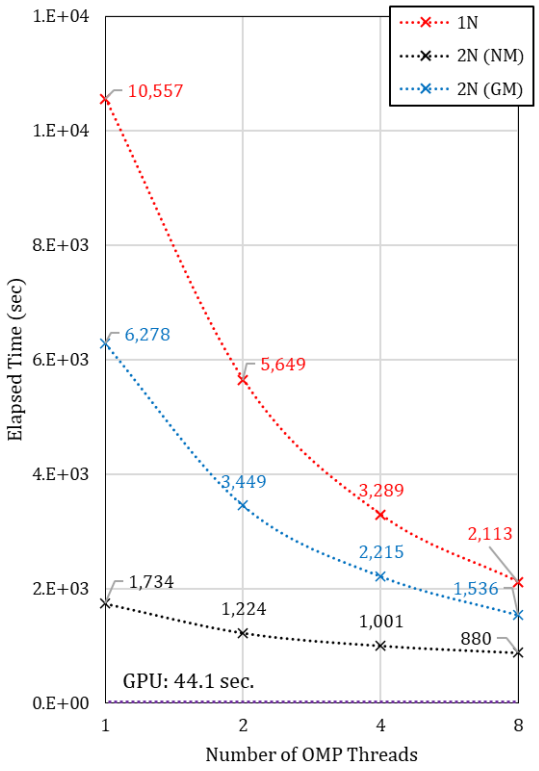

Fig. 1. Comparison of calculation times and parallel scalability without the assembly-wise CMFD

The bottleneck primarily arises from the limited parallelism inherent in ILU0 preconditioning because it requires triangular matrix solves. Although Intel MKL and cuBLAS attempt to parallelize these operations using graph theory-based algorithms, the sparsity of the matrices significantly restricts the parallel efficiency. In addition, the sparse matrix system itself is fundamentally memory-bound. It requires high-frequency memory access for index searching and suffers from low data coherence between the matrix and vectors. These characteristics prevent the solver from fully utilizing the parallel architecture.

While the two-node GM solver shows improved scalability compared to NM in Fig. 1, the improvement is not decisive. Its execution time remains significantly higher due to the increased number of iterations. In contrast, even in a Windows PC environment where computing performance can be inconsistent due to background system processes, the one-node solver demonstrates the most distinct scalability as the thread count increases. This is because the one-node nodal sweep is an embarrassingly parallel task that is compute-bound. Its performance is primarily determined by raw mathematical calculation capacity rather than data transfer speeds. As a result, NSight profiling confirms that the one-node NEM kernels achieve nearly 99% GPU occupancy. This high utilization of the GPU's computational resources results in a massive speedup of 239.4x compared to single-thread CPU result.

Table IV: Comparison of Calculation Times
(One-node, with the assembly-level CMFD)

Case	Elapsed time (s)	Speedup
OMP, 1trd	429.5	
OMP, 2trd	239.4	1.8x
OMP, 4trd	143.0	3.0x
OMP, 8trd	90.1	4.8x
GPU	2.8	153.4x

Table V: Comparison of Calculation Times
(Two-node, NM, with the assembly-level CMFD)

Case	Elapsed time (s)	Speedup
OMP, 1trd	572.1	
OMP, 2trd	408.5	1.4x
OMP, 4trd	341.5	1.7x
OMP, 8trd	323.8	1.8x

Table VI: Comparison of Calculation Times
(Two-node, GM, with the assembly-level CMFD)

Case	Elapsed time (s)	Speedup
OMP, 1trd	1503.0	
OMP, 2trd	893.0	1.7x
OMP, 4trd	558.6	2.7x
OMP, 8trd	414.5	3.6x

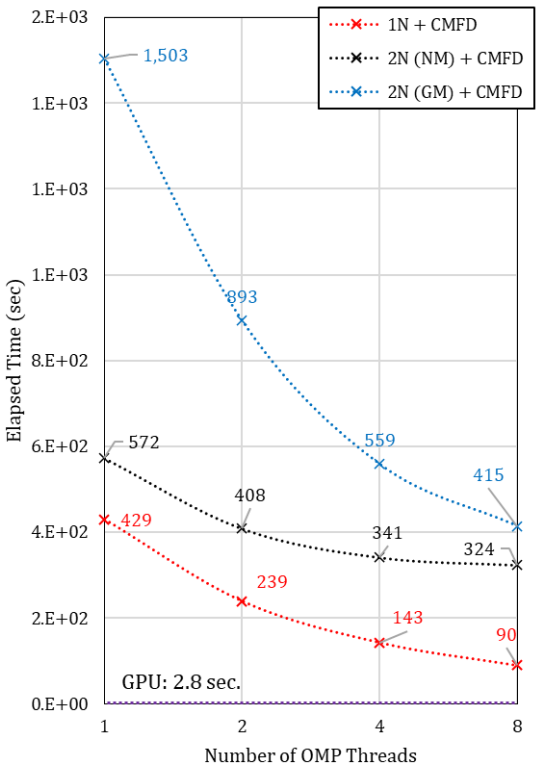

Fig. 2. Comparison of calculation times and parallel scalability with the assembly-wise CMFD

Tables IV through VI summarize the calculation results with assembly-level CMFD acceleration. The corresponding performance trends and scalability of each scheme are visually compared in Fig. 2. The results remain consistent with the cases shown in Tables I through III. As illustrated in Fig. 2, the one-node solver consistently outperforms both two-node schemes across all thread counts while demonstrating superior parallel efficiency. Notably, the two-node GM scheme shows a steeper reduction in elapsed time than NM. This confirms the improved scalability of GM, even though NM still maintains a lower absolute execution time within the tested range.

The most remarkable outcome is achieved by the GPU-based one-node solver. The total execution time is reduced to only 2.8 seconds when combined with the assembly-level CMFD. This massive performance gap between the CPU and GPU implementations is clearly highlighted in Fig. 2. From a practical perspective, this level of speed is sufficient for industrial applications, including the commercial reactor designs and analyses.

4. Conclusions

The study aimed to identify the optimal approach between one-node and two-node methods for performing GPU-based FM SP3 calculations. To achieve this, both one-node and two-node SP3 NEM solvers based on the Legendre polynomial basis were implemented in both CPU and GPU versions. The performance was evaluated by focusing on hardware-specific suitability of each scheme. The results demonstrate that the one-node solver is advantageous for GPU-based parallel computing. While the two-node NM approach has been considered a rational choice in the conventional CPU environments due to its stability and fast-converging behavior, its parallel efficiency drops significantly on GPUs due to the bottlenecks inherent in sparse matrix operations. In contrast, the one-node solver effectively maximizes the GPU's computational throughput because of its compute-bound nature. Consequently, a massive speedup of up to approximately 240x was achieved compared to a single-thread CPU execution. The execution time was notably reduced to a mere 2.8 seconds when combined with the assembly-level CMFD.

These findings offer significant implications for the field of computational reactor physics. This study proves that methodologies previously optimized for CPU-based calculations can become highly unsuitable and cause performance bottlenecks in GPU applications. Therefore, to effectively utilize modern GPUs, it is essential to move beyond traditional methods.

Acknowledgments

This work was supported by the Innovative Small Modular Reactor Development Agency grant funded by the Korea Government (MCEE) (RS-2023-00259289).

REFERENCES

[1] N. Choi, K. M. Kim, and H. G. Joo, Initial Development of PRAGMA – A GPU-Based Continuous Energy Monte Carlo code for Practical Applications, in Transactions of the Korean Nuclear Society Autumn Meeting, Goyang, Korea, October 24-25, 2019.
[2] H. Hong and J. Yoon, Solution of OECD/NEA PWR MOX/UO2 Benchmark with a High-Performance Pin-by-Pin Core Calculation Code, Nuclear Engineering and Technology, vol. 56, no. 9, pp. 3654-3667, 2024.
[3] H. Hong and H. G. Joo, Thorough analyses and resolution of various errors in pin-homogenized multigroup core calculation, Annals of Nuclear Energy, vol. 163, 108502, 2021.
[4] S. Jeon, H. Hong, N. Choi, and H. G. Joo, Methods and performance of a GPU-based pinwise two-step nodal code VANGARD, Progress in Nuclear Energy, vol. 156, 104528, 2023.
[5] T. Downar, Y. Xu, V. Seker, PARCS v3.0 U.S. NRC Core Neutronics Simulator Theory Manual, University of Michigan, December 1, 2009.
[6] A. Yamamoto, T. Sakamoto and T. Endo, A CMFD Acceleration Method for SP3 Advanced Nodal Method, Nuclear Science and Engineering, vol. 184, no. 2, pp. 168-173, 2016.