Preliminary Study of GPU Based FDM Acceleration Using Python CUDA

Gihyeon Kim [a], Jeong Ik Lee[a*]
[a]*Dept. Nuclear & Quantum Eng., KAIST, 373-1, Guseong-dong, Yuseong-gu, Daejeon, 305-701, Republic of Korea*
[*]*Corresponding author: jeongiklee@kaist.ac.kr*

Keywords : GPU acceleration, Finite Difference Method

1. Introduction

Small modular reactors are manufactured in modules at factories and installed at the site near where demand exists. This reduces costs and enables construction in a wider range of areas where large conventional nuclear power plants could not be constructed, such as remote areas and islands. Additionally, flexible operation is required to accommodate fluctuations in output at the demand site. Therefore, there is a cost burden associated with the operational workforce for small modular reactors [1], [2], [3]. To address this, methods such as remote operation or autonomous operation have been proposed. Among these, autonomous operation aims to minimize human intervention in all cases, including normal operation, load-following operation, and accident operation, by combining artificial intelligence [4].

A Deep Reinforcement Learning (DRL)-based controller has been proposed for autonomous operation of small modular reactors. To train the DRL controller, a learning environment that accurately simulates the thermal-hydraulic phenomena of the reactor is required. While existing thermal-hydraulic analysis codes can be used for controller training, their speed has been identified as an issue [5], [6]. Therefore, the acceleration of existing codes is necessary for efficient controller training.

To analyze thermo-hydraulic system, numerical analysis methods based on finite element methods such as computational fluid dynamics and finite element analysis can be used, but these methods require excessive computation and take extended time to calculate, clearly limiting their use in analyzing large systems. Therefore, most thermal-hydraulic codes currently used for analyzing nuclear reactor thermal-hydraulic systems, including MARS-KS and GAMMA+, employ the Finite Difference Method (FDM) for evaluating heat transfer and fluid flow. In FDM-based codes, the fluid and thermal structure are divided into a finite number of nodes, and the governing equations, which are partial difference equations, are converted into matrix form for computation. Thus, the primary role of thermal-hydraulic codes is to construct a matrix appropriate for the governing equation and problem situation and perform calculations on it. By accelerating matrix construction and calculations, it is possible to improve the speed of thermal-hydraulic codes. Methods for accelerating matrix construction and calculations have been well researched.

Currently, General-purpose computing on graphics processing units (GPGPU) is gaining attention as a method for accelerating numerical analysis methods, including matrix operations. Among these, CUDA is a representative platform that provides general-purpose acceleration libraries and supports various programming languages such as C, C++, Fortran, and Python. In this study, Python CUDA was used to preliminarily evaluate the applicability of GPU-based acceleration in nuclear engineering.

2. Methods and Results

This section describes and compares the results of the Python based FDM, including the Python CUDA libraries used, FDM test case, computing conditions, and the results of the code performance evaluation and comparison using CUDA.

2.1 Test Case and Code Acceleration Method

A simple test case was set up to pre-evaluate the acceleration of the FDM using GPU. Although more complex problems were also possible, a one-dimensional heat transfer problem with an analytical solution was selected to determine the consistency of the code. Furthermore, in the case of heat transfer problems, thermal structure nodes account for a significant proportion of actual nuclear analysis codes and are an important factor in heat exchanger and core analysis, making them an appropriate choice for this problem.

The selected problem involves calculating the temperature distribution of a cylindrical nuclear fuel rod composed of fuel and cladding surfaces, with boundary conditions set at the cladding surface temperature. Assuming a steady state for comparison with the analytical solution, uniform volumetric heat generation and constant thermal properties within the fuel region were assumed. Figure 1 briefly illustrates the problem situation.

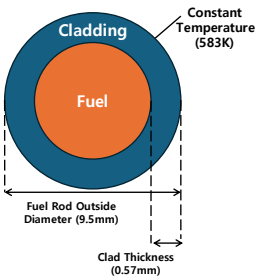

Fig. 1. Simple test case for FDM acceleration

Python code was written for both CPU-based and GPU-based to solve the test problem. By redesigning the code based on the GPU, it is more advantageous for parallelized calculations than CPU-based code. This is because the designs of the GPU and CPU are different. The CPU is optimized to execute sequenced tasks (threads) as quickly as possible, and in general, it can execute several dozen threads in parallel. On the other hand, the GPU is optimized to execute thousands of threads in parallel, even though its single-thread performance is slow [7]. Therefore, unlike the CPUs used in existing thermohydraulic analysis codes, GPUs offer significantly higher instruction throughput and memory bandwidth than CPUs. In other words, GPUs are specialized for parallelized calculations and allocate more transistors to data processing than to data caching and flow control. This allows memory access delays to be hidden through computation when handling large-scale parallelized calculations.

Since most codes are a mix of parallel and sequential parts, using a GPU does not guarantee improved performance. Additionally, since GPUs have slower memory access speeds, they offer no performance advantages for small-scale calculations. Therefore, evaluating the performance of GPU-based codes using simple examples should precede applying them to more complex cases. For example, if performance improvements are only seen in GPU-based code when there are tens of thousands of nodes, then it is meaningless to use it in nuclear thermohydraulic analysis code with fewer than a few thousand nodes.

$$AX = B \cdots (1)$$

$$A = \begin{bmatrix} \ddots & R_i & \cdots & 0 & 0 & 0 \\ R_i & -(R_i + R_{i+1}) & R_{i+1} & \cdots & 0 & 0 \\ \vdots & R_{i+1} & \ddots & \ddots & \vdots & 0 \\ 0 & \vdots & \ddots & \ddots & R_j & \vdots \\ 0 & 0 & \cdots & R_j & -(R_j + R_{j+1}) & R_{j+1} \\ 0 & 0 & 0 & \cdots & R_{j+1} & \ddots \end{bmatrix},$$

$$X = \begin{bmatrix} T_1 \\ \vdots \\ T_{i-1} \\ T_i \\ T_{i+1} \\ \vdots \\ T_n \end{bmatrix}, B = \begin{bmatrix} -\frac{\dot{q}_1}{2}(r_1^2 - 0) \\ \vdots \\ -\frac{\dot{q}_{i-1}}{2}(r_{i-1}^2 - r_{i-2}^2) \\ -\frac{\dot{q}_i}{2}(r_i^2 - r_{i-1}^2) \\ -\frac{\dot{q}_{i+1}}{2}(r_{i+1}^2 - r_i^2) \\ \vdots \\ -\frac{\dot{q}_n}{2}(r_n^2 - r_{n-1}^2) \end{bmatrix}, R_i = k_i \frac{r_i}{\Delta r} \cdots (2)$$

Both codes perform calculations by constructing the same matrix expression based on the problem conditions for the given number of nodes. The linear equation for the 1D FDM heat transfer problem is shown in Equation 1, and each matrix in Equation 1 is the same as Equation 2. The matrix operations in Equations 1 and 2 were performed using Python libraries. In the CPU-based code, Equation 1 was solved using the numpy library, and no library was used for matrix construction in Equation 2. In GPU-based code, equation 1 was solved using the cupy library with CUDA, and CUDA kernels were just-in-time compiled using the numba library to construct each matrix in equation 2. Therefore, both matrix construction and operations for solving FDM problems were performed on the GPU.

2.2 Code Validation

The temperature and heat flux distributions were calculated using CPU-based FDM and GPU-based FDM for the given test cases. The results obtained from these three calculation methods were compared with the analytical solution. Figures 2 and 3 show the code comparison results in graph. It was confirmed that the FDM results matched the analytical solution for both codes.

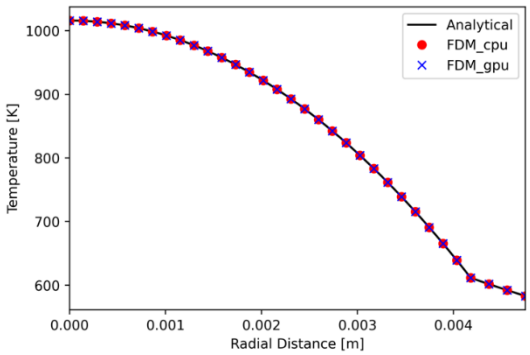

Fig. 2. Temperature distribution

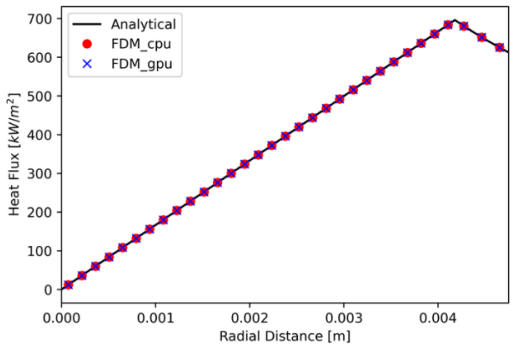

Fig. 3. Heat flux distribution

2.3 Code Speed Comparison

The execution time of CPU-based code and GPU-based code was compared by increasing the number of nodes using Python 3.12 in the computational environment shown in Table 1.

Table 1. Computational Environment

Components	Specification
CPU	Intel® i9-13900
GPU	NVIDIA GeForce RTX 4090
RAM	SK Hynix DDR5-5600 (16GB) x2
OS	Windows 11

Figures 4 and 5 show the difference in execution time between CPU-based code and GPU-based code depending on the number of nodes. The execution time of the code was compared by averaging the results after performing 1,000 calculations. From 320 nodes onwards, the GPU-based code outperformed the CPU-based code in terms of speed. As the number of nodes increased, the speed difference exceeded 30 times. The reason for the slower GPU speed at lower node counts is likely due to the time required to transfer data to GPU memory. These results may vary depending on the computational environment.

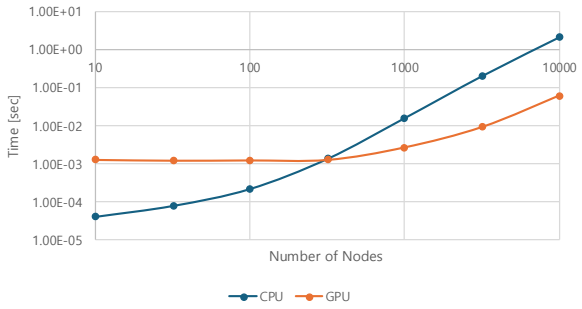

Fig. 4. Execution time comparison between CPU and GPU based Python FDM code

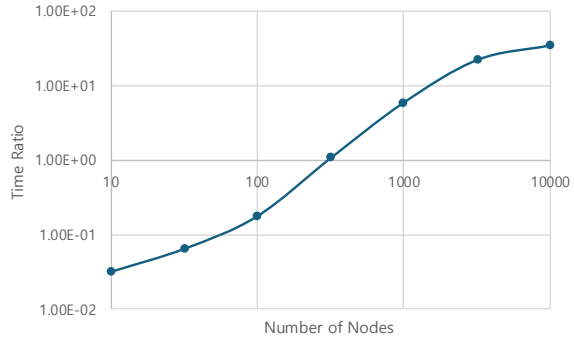

Fig. 5. Ratio of CPU and GPU based Python FDM code execution time

3. Conclusions

This study conducted a preliminary evaluation of the application of GPU-based code to nuclear systems. For the evaluation, CPU-based and GPU-based codes were written in Python to solve a simple thermal-hydraulic example. It was confirmed that the GPU-based code outperformed the CPU-based code in terms of speed when the number of nodes reached several hundred. It is expected that the condition under which the GPU-based code outperforms the CPU-based code in terms of speed will be similar on a logarithmic scale even in more complex systems. This is because the time required for matrix operations was found to be greater than the time required for matrix construction for calculations. Additionally, since thermal-hydraulic analysis codes require matrix calculations of the square of the number of nodes, the value of using GPU-based codes increases as the system grows larger.

In the case of nuclear thermal-hydraulic systems, as the fidelity of the code calculation becomes more important greater number of nodes will be needed, making the use of GPU-based code more meaningful. Furthermore, the fact that GPU-based acceleration is meaningful only when the number of nodes reaches hundreds implies that there is no reason to apply GPU-based computation to programs with a small number of nodes, and that the target for GPU acceleration must be carefully selected. As a future work, GPU acceleration performance tests will be conducted using examples other than heat transfer, and if the results are promising, an attempt will be made to build a general solver.

ACKNOWLEDGEMENTS

"This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. RS-2025-25454059)."

REFERENCES

[1] A. Asuega, B. J. Limb, and J. C. Quinn, “Techno-economic analysis of advanced small modular nuclear reactors,” *Appl Energy*, vol. 334, p. 120669, 2023.

[2] G. Locatelli, C. Bingham, and M. Mancini, “Small modular reactors: A comprehensive overview of their economics and strategic aspects,” *Progress in Nuclear Energy*, vol. 73, pp. 75–85, 2014.

[3] J. Vujić, R. M. Bergmann, R. Škoda, and M. Miletić, “Small modular reactors: Simpler, safer, cheaper?,” *Energy*, vol. 45, no. 1, pp. 288–295, 2012.

[4] J. Kim, D. Lee, J. Yang, and S. Lee, “Conceptual design of autonomous emergency operation system for nuclear power plants and its prototype,” *Nuclear Engineering and Technology*, vol. 52, no. 2, pp. 308–322, 2020.

[5] J. Y. Baek, T. Min, and J. I. Lee, “Robustness of Deep-Reinforcement-Learning Control for Fast Load-Following-Operation of S-CO2 Microreactor,” *Trans Am Nucl Soc*, vol. 131, no. 1, pp. 310–313, 2024.

[6] J. Y. Baek and J. I. Lee, “Development of a Time-Series Surrogate Model for Predicting System Dynamics in KAIST-MMR under Load-following Operation”. *Transactions of the Korean Nuclear Society Spring Meeting Jeju, Korea, May 10, 2024*

[7] NVIDIA Corporation, “CUDA C++ Programming Guide Release 13.0,” 2025.