2025.10.30 Transactions of the Korean
Nuclear Society Autumn Meeting

Python CUDA 이용 GPU 기반 FDM 가속 기초 연구

Preliminary Study of GPU Based FDM Acceleration Using Python CUDA

Author: Gihyeon Kim, Jeong Ik Lee

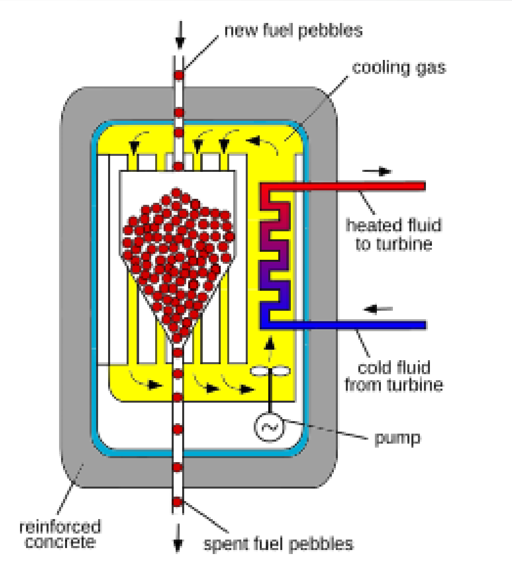
발표자 : 김기현
E-mail : orca2005@kaist.ac.kr

Korea Advanced Institute of Science and Technology
Dept. of Nuclear and Quantum Engineering

Introduction

Pebble Bed Reactor

- **Next-generation reactors** have a more **complex thermal structure** compared to conventional PWRs.
  - **Molten Salt Reactor** : Molten salt thermal properties and increased number of loops
  - **Sodium-cooled Fast Reactor** : High thermal conductivity of coolant
  - **Pebble Bed Reactor** : Heat conduction between many irregularly arranged particles
- For pebble bed reactors, it is necessary to analyze heat transfer between irregularly arranged particles, resulting in **large-scale complex heat conduction** problems.
- In **next-generation reactor analysis**, solving massive **heat transfer problems efficiently** can **enhance overall code performance.**



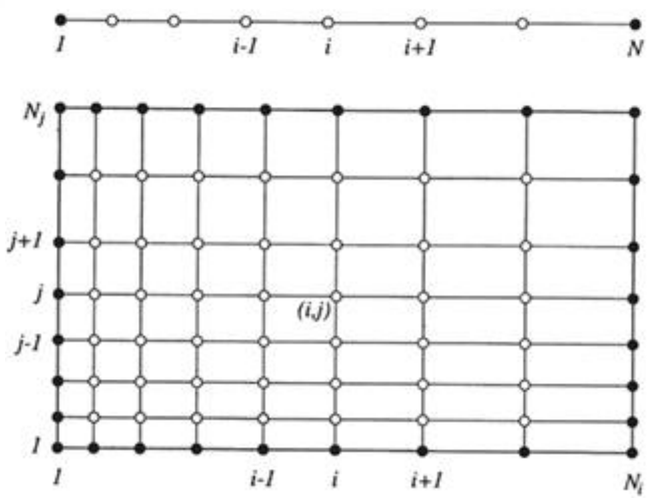
Pebble Bed Reactor

Pile of Pebbles in Pebble Bed Reactor

Introduction

Finite Difference Method (FDM)

- Most nuclear reactor thermal-hydraulic analysis codes, including MARS-KS and GAMMA+, employ the **Finite Difference Method (FDM)** for evaluating thermal structure.
- In FDM-based codes
  - Thermal structure : divided into a **finite number of nodes**
  - Conduction governing equations : **partial difference equation**
  - Both **converted into matrix** form for computation
- The thermal-hydraulic codes
  - **Construct a matrix** appropriate for the governing equation and problem situation
  - Perform **matrix calculations**
- By **accelerating matrix construction and calculations**, it is possible to **improve the speed** of thermal-hydraulic codes.

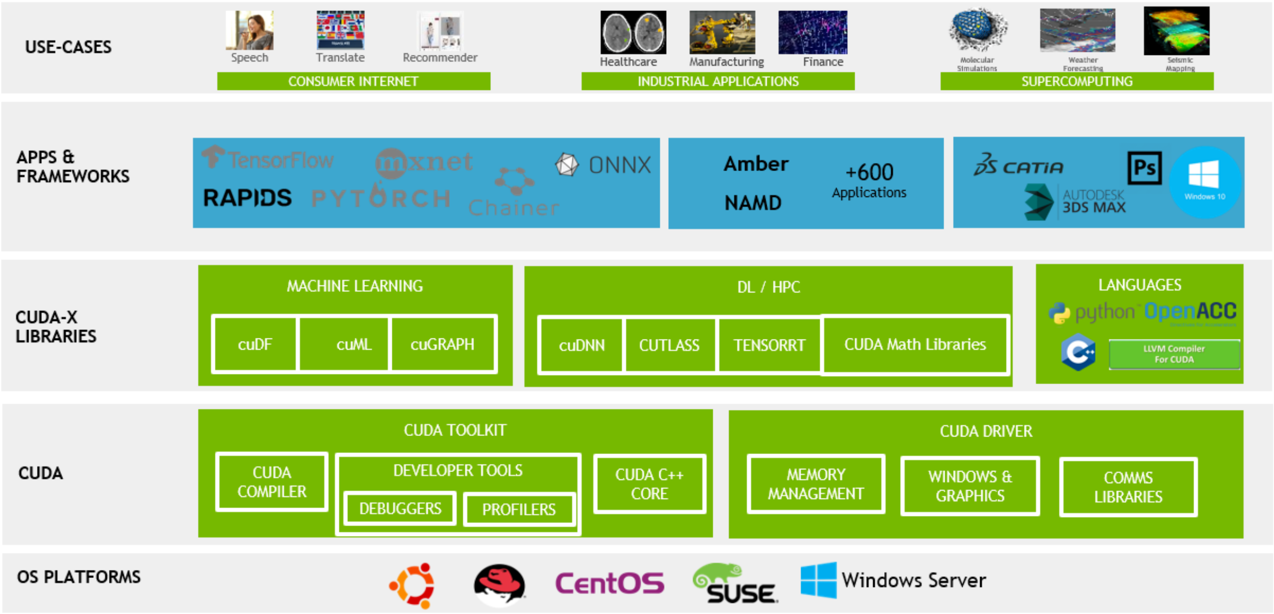


FDM Cartesian Grid

Introduction

CUDA

- Methods for **accelerating matrix** construction and calculations have been well researched.
- **Parallel computing** is gaining attention as a method for accelerating numerical analysis methods.
- **CUDA** is a **parallel computing platform and programming model** developed by NVIDIA.
- CUDA is a representative GPU platform that provides general-purpose acceleration libraries and supports various programming languages such as **C, C++, Fortran, and Python**.
- **Widely adopted** in fields such as deep learning, scientific computing, and high-performance computing



CUDA Overview

Introduction

Python CUDA Library

- NVIDIA originally developed the CUDA toolkit with a focus on C/C++
  - **Official Python support** is added since **2024**
- By using **Python CUDA library**, high performance **GPU computing** is available using **Python alone.**
- Using Python can be beneficial for creating digital twins or reinforcement learning control based on thermal-fluid analysis code in the future.
- To efficiently use CUDA in Python, **choose and use** the **appropriate library** from among the various available libraries.

Why CUDA Python?

CUDA Python provides uniform APIs and bindings for inclusion into existing toolkits and libraries to simplify GPU-based parallel processing for HPC, data science, and AI.

Python CUDA Libraries

Introduction

Python CUDA Library

Numba

- Developed to accelerate numerical analysis using Python
- Enables development of CUDA kernels based on JIT (Just-In-Time) compilation
- Limited set of available functions
  - Array-related functions cannot be used

CuPy

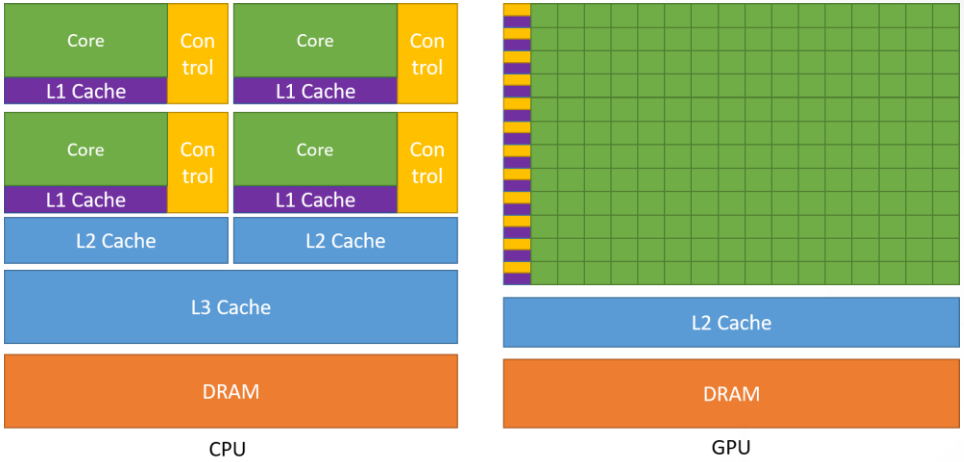
- Provides advanced functionality for scientific and technical computing in Python
- Offers GPU-accelerated libraries compatible with NumPy and SciPy
- Difficulty in writing CUDA kernels primarily at a high level

→ **Using Numba and CuPy complementarily enables fast and efficient development!**

Introduction

CPU vs GPU

- Designs of CPU and GPU
  - CPU
    - Optimized to execute **sequenced tasks** (threads) as quickly as possible
    - Can execute **several dozen** threads **in parallel**
  - GPU
    - Optimized to **execute** thousands of threads **in parallel**
    - **Single-thread** performance is **slow**



CPU v. GPU

Introduction

CPU vs GPU

- **GPU-based code** is more advantageous for **parallelized calculations** than CPU-based code.
  - GPUs offer significantly **higher instruction throughput and memory bandwidth** than CPUs.
  - GPUs allocate more transistors to **data processing** than to data caching and flow control.
  - **Memory access delays** while computation hides for 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**.

→ **The decision to use a GPU should be made based on the type and size of the problem!**

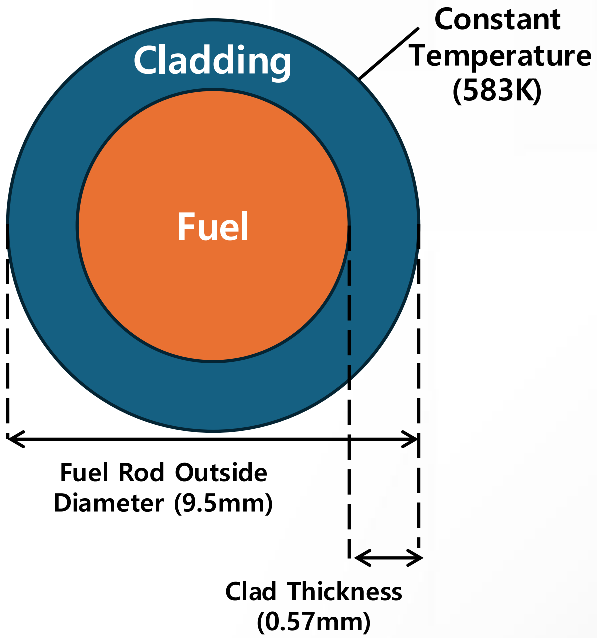
Methodology

FDM Example

Test Case

- A **simple test case** was set up to **pre-evaluate** the acceleration of the FDM using GPU.
- A **one-dimensional heat transfer problem** with an **analytical solution** was selected.
  - Determine the **consistency** of the code
  - Thermal structure node accounts large proportion of actual analysis code
- **Compare analytical solution with the numerical solution calculated using CPU and GPU**.
- Problem Descriptions

1. **Cylindrical** nuclear **fuel rods** composed of **fuel and cladding**
2. **Cladding surface temperature boundary** condition
3. **Steady state** temperature distribution
4. Uniform volumetric heat generation
5. Constant thermal properties
6. **Have analytical solution**



Methodology

FDM Example

Governing Equation and Algorithm

- Both CPU-based code and GPU-based code construct and compute the same matrix equation based on the problem conditions.
  - Equation 1 : Linear Equation for the 1D FDM heat transfer problem
  - Equation 2 : Problem conditions for the given number of nodes

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

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

	CPU-based	GPU-based
Equation 1	NumPy	CuPy
Equation 2	-	Numba

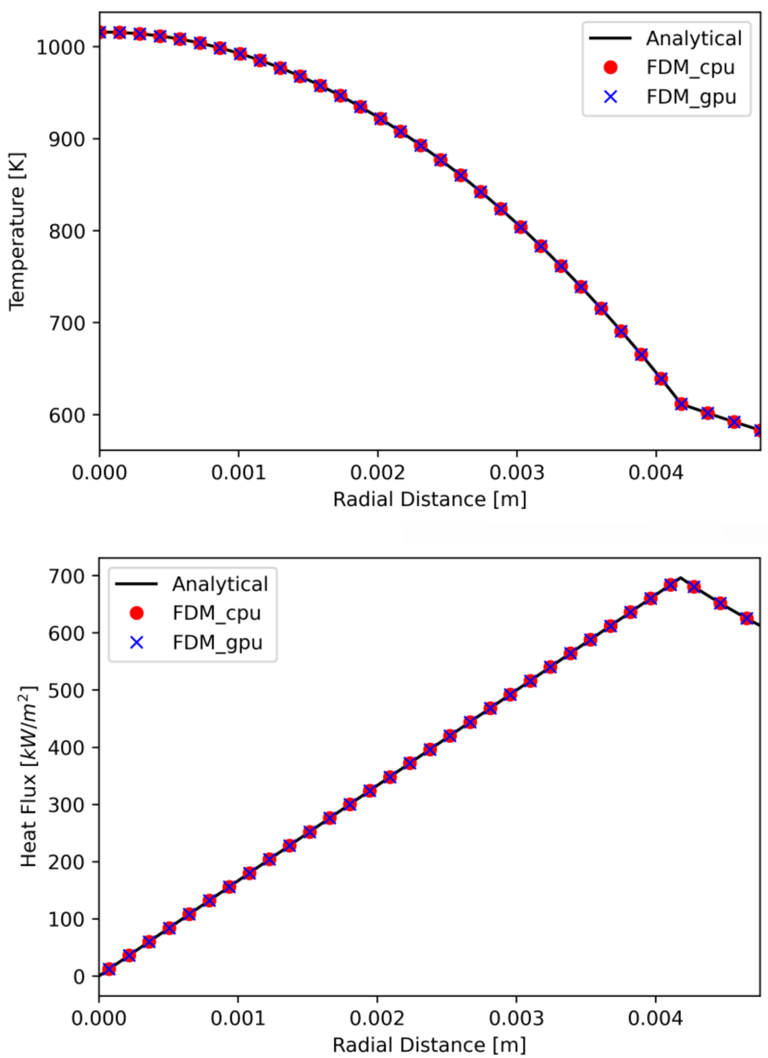
Result

Code Validation

Code Validation Using Analytical Solution

- 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 **FDM codes** were **compared with the analytical solution**.
- Figures show the code comparison results in graph.
- It was confirmed that the **FDM results matched the analytical solution for both codes**.

- **Both codes have achieved consistency.**



Temperature and Heat Flux Distribution

Result

Code Speed Comparison

Computational Environment

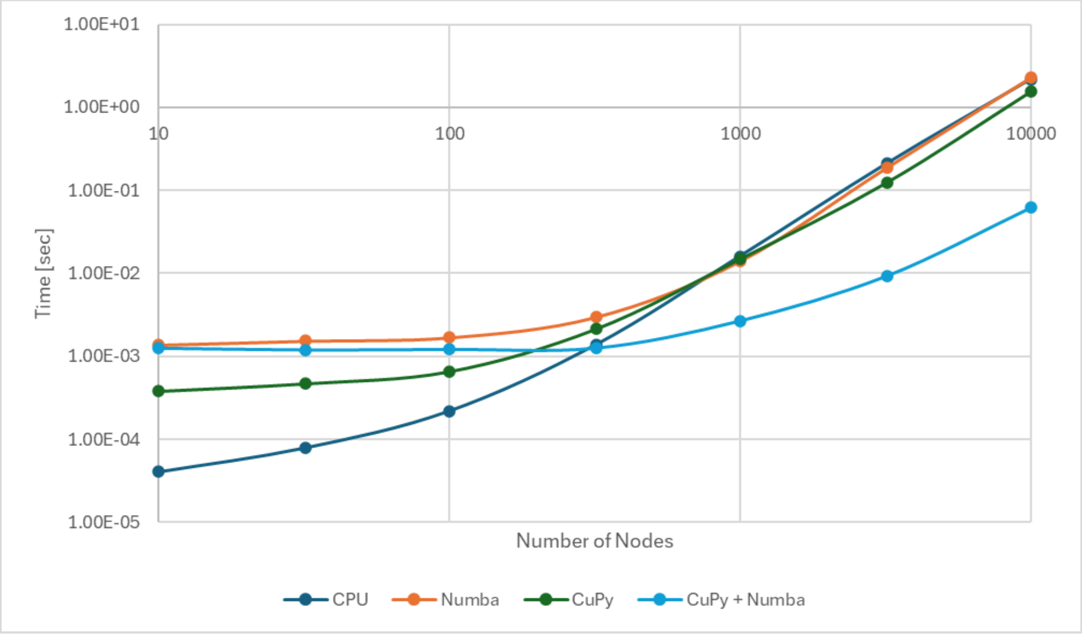
- The **execution time** of CPU-based code and GPU-based **code was compared by increasing the number of nodes** using Python 3.12
- The execution time of the code was compared by averaging the results after performing 1,000 calculations.
- Comparing CPU-based code with three GPU-based codes
  1. **Numba only** : Only **matrix construction** done by **GPU**
  2. **CuPy only** : Only **matrix calculation** done by **GPU**
  3. **CuPy + Numba** : Both **matrix construction and calculation** done by **GPU**
- Speed comparisons were performed in the following computational environment, and **results may vary depending on the environment**.

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

Result

Code Speed Comparison

- The **fastest code varies** depending on the **number of nodes**.
  - **Few nodes : CPU**
  - **Many nodes : CuPy + Numba**
- Code using only Numba and code using only CuPy are slower than the CPU when there are few nodes, and slower than CuPy + Numba code when there are many nodes.
  - **Copy overhead to/from the device due to GPU computation increases the time required.**

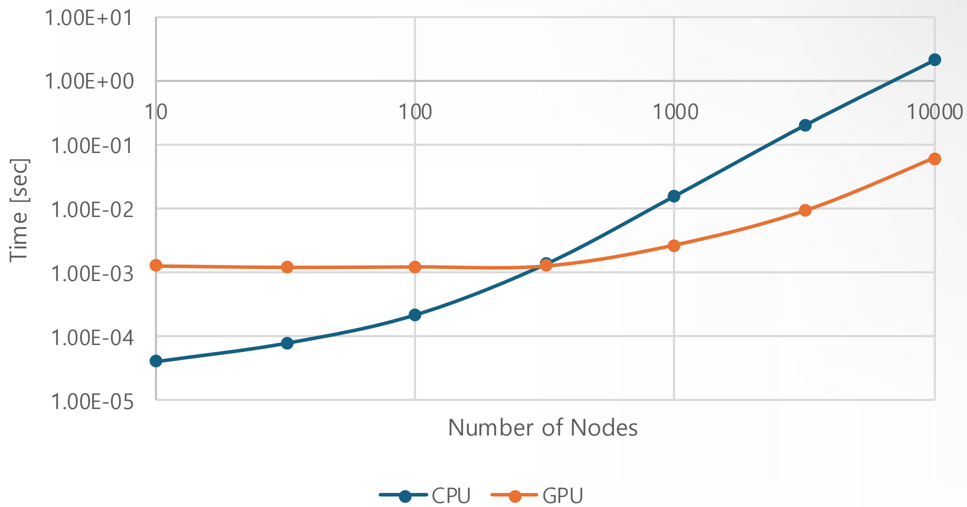


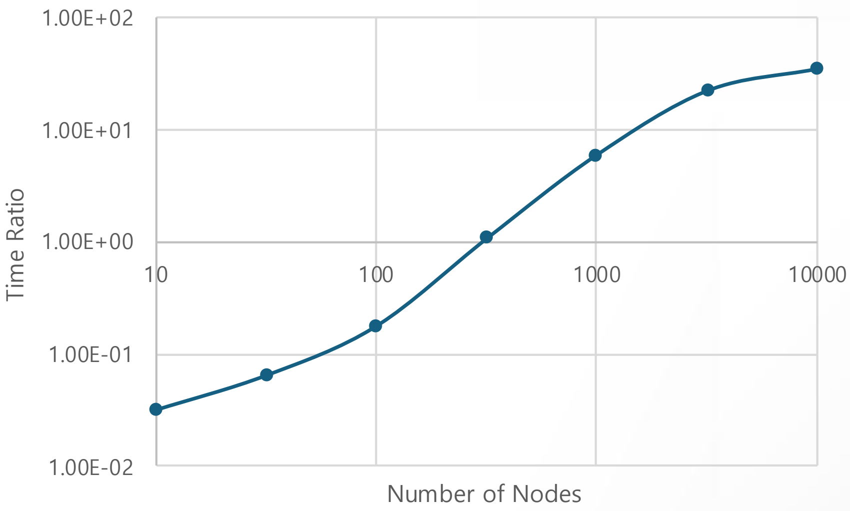
→ **Using a GPU doesn't automatically make things faster; good code structure is necessary.**

Result

Code Speed Comparison

- The results **comparing GPU-based** code using CuPy and Numba with **CPU-based** code are shown in the graph on the right.
- 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**.





Summary And Conclusions

Summary

- Most nuclear reactor thermal-hydraulic analysis codes, including MARS-KS and GAMMA+, employ the **Finite Difference Method (FDM)** for evaluating heat transfer.
- By **accelerating matrix construction and calculations**, it is possible to **improve the speed** of thermal-hydraulic codes.
- By using **Python CUDA library**, high performance **GPU computing** is available using **Python alone.**
- Although **GPU-based code** is more advantageous for **parallelized calculations** than CPU-based code, most codes are a **mix of parallel and sequential** parts, **using a GPU does not guarantee improved performance**.
- **The speed of GPU-based code depends on how the code is built.**
- Even when using a **GPU**, copy overhead **can make it slower than a CPU**, so **careful design is required** when building GPU-based code.
- 1D heat transfer FDM case shows the **GPU-based code outperformed the CPU-based code when the node number is bigger than hundreds**.

Summary And Conclusions

Conclusions

- **GPU-based code can outperform the CPU-based code** in terms of speed when the number of nodes reached **several hundred**.

- 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 goes bigger**.

- **Using a GPU doesn't automatically make things faster; good code structure is necessary.**

- As 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**.

Thank you