Enhancing Chemical Process Simulation through a GPU-Optimized Framework: Implementation and Validation of Equation-Oriented Methods using CUDA

Shaoyi Yang,Xufei Tian, Shifeng Qu, Zhaoyang Duan, Minglei Yang, Feng Qian, Wenli Du\*

Key Laboratory of Smart Manufacturing in Energy Chemical Process, Ministry of Education, East China University of Science and Technology, Shanghai,200237, China

\*E-mail: wldu@ecust.edu.cn

Abstract

In the realm of chemical engineering, the simulation of complex process systems frequently entails solving interconnected equations on a large scale. Traditional methods are difficult to handle higher-dimensional computing, particularly hose necessitating increased real-time processing capabilities. This study aims to significantly enhance the computational efficiency of equation-oriented (EO) methods by leveraging Graphics Processing Units (GPUs), primarily utilizing the CUDA programming paradigm exploit the parallel processing capabilities of GPUs. We present a novel GPU-based framework tailored for chemical process simulations. This framework segments the thermodynamic computations and other complex tasks within the EO method, using GPU's massive parallel capabilities to simultaneously update each thermodynamic parameter in batches, significantly enhancing the overall efficiency of parameter computation.

This work first demonstrates the tasks split method within the EO structure, followed by the utilization of CUDA graph on GPUs for parallel computation of thermodynamic parameters after the breakdown of tasks. Extensive result analyses are provided, validating that using a consumer-grade GPU achieve nearly a 100-fold enhancement in performance, while preserving identical accuracy.

**Keywords**: Equation-Oriented, GPU, nonlinear solver, CUDA.

* 1. Introduction

In industrial process simulation, the Sequential Modular (SM) method simulates each module separately and resolves them iteratively, whereas the Equation-Oriented (EO) approach achieves synchronous optimization of the process by establishing and solving a set of nonlinear equations. The EO method excels in simultaneously handling all equations and variables, employing efficient large-scale solvers and precise derivative computations, making it particularly suited for complex processes with nested loops and intricate design specifications. Compared to the SM approach, the EO method offers greater accuracy and flexibility in process design and performance optimization.

Although the EO approach has distinct advantages in simultaneous optimization and convergence efficiency, solving large-scale models still requires substantial computational effort. Dowling & Biegler (2015) designed a framework for large scale EO modelling, putting forward a large-scale parallel outlook. The IDAES project (Miller, 2018) demonstrates the power of the EO method when integrated with modern modeling tools.

The duration required to solve equations serves as a crucial performance metric for the solver's efficiency. Despite the possibility of accelerating convergence and reducing iteration numbers through mathematical techniques, each iteration necessitates updating the thermodynamic state for all equations, with the computational speed of the thermodynamic parameters involved in each equation directly impacting the actual execution time of the program. On the other hand, despite the mature applications of GPUs in scientific computing tasks such as deep learning and fluid dynamics, there is almost no generic parallel solution based on GPUs in the field of process simulation. Ma (2016) attempted to accelerate the large sets of equations using multicore processors and GPU but faced limitations due to inadequate computational distribution. This led to excessive computational loads on individual GPU threads and insufficient degrees of parallelism, failing to significantly improve GPU performance. Nikolić (2018) developed CUDA based parallel program for DAE equations, yet it was limited to relatively simple computational tasks and fell short in handing complex chemical simulation. To addressing these gaps, we propose a novel, large-scale parallel framework designed specifically for chemical process simulation based on the GPU architecture. This framework systematically organizes the thermodynamic computations and other complex tasks within the EO method, leveraging the GPU's massive parallel capabilities to simultaneously update each thermodynamic parameter in batches, thereby significantly enhancing the overall computation efficiency.

* 1. Thermodynamic calculations based on CUDA graph

A typical workflow of the EO approach in process simulation and optimization consists of several steps, including initialization, parameter calculation, function evaluation and/or gradient evaluation, iterate updating, and termination judgment. Within this workflow, the computation of thermodynamics parameters is notably time intensive. Conventional platforms such as gPROMS and AspenPlus update parameters serially by calling external thermodynamic interfaces. As depicted in Figure 1, for the large sparse matrices establish by actual processes, computing each thermodynamic equation involves numerous parameter calculations. Given that the independence of these parameter updates, they can be grouped into batches. These batches of thermodynamic parameters awaiting computation are dispatched in parallel to the GPU. Here, the updates occur simultaneously. Upon completion, the updated results are fed back into the sparse matrix, setting the stage for the subsequent iterative resolution of the nonlinear equations system.

图片包含 图形用户界面

描述已自动生成

Figure 1. The calculation process and calculation elements of EO method

Figure 2 illustrates the workflow and execution strategy of large-scale parallel computation of thermodynamic parameters on GPUs. This process entails breaking down the thermodynamics calculations into specific functional operators, which are then scheduled for parallel execution via a computational graph. In practical applications, state parameters such as component fractions, temperature, and pressure are extracted from each equation and fed into the parallel program, allowing for the simultaneous resolution of all thermodynamic parameters required by each equation.

Taking the cubic state equation of state SRK as an example, the basic workflow for the computation of thermodynamic parameters, such as the fugacity coefficient, can be decomposed into the following steps: calculating base state parameters, computing mixing rules, solving the cubic equation for the compression factor, sorting the compression factors, and calculating the fugacity coefficients. These operators are individually written as CUDA kernel functions, tailored with specific computational tasks. During execution, CUDA Graphs are utilized to schedule each kernel, enabling the distribution of computational tasks across the GPU threads. Thermodynamic calculations require access to certain fundamental physical properties, which remain read-only during computation and therefore are loaded into shared memory for more efficient access by all threads within the same block, as depicted by the memory structure in Figure 2. This approach maximizes the efficient utilization of the GPU, and the execution of each kernel is carried out by the GPU’s Streaming Multiprocessors. Leveraging the GPU's architectural design, our program significantly outperforms multicore CPU parallelism in terms of acceleration ratios.

图示

描述已自动生成

Figure 2. The calculation process and calculation elements of EO method

In this paper, the SRK Thermodynamic model is coded in C++, the GPU version program is developed using CUDA. The implementation has been conducted on both an Intel multi-core-processor platform and an NVIDIA GPU GTX4060 platform.

* 1. Numerical examples for the parallel framework

We use the calculation of the fugacity coefficient for key components in the ethylene production process (methane, ethylene, ethane, propane) as our numerical test example.

* + 1. Program acceleration ratio

Program acceleration ratio is a crucial metric for evaluating parallel program acceleration. Figure 3 illustrates the acceleration ratio achieved by GPU computation relative to CPU execution time, across various numbers of thermodynamic parameters. The acceleration ratio is calculated by dividing the CPU execution time by the GPU execution time, serving as an indicator of the performance enhancement afforded by GPU parallelism over traditional CPU serial computation. It is evident that the GPU delivers obvious acceleration effect, regardless of the scale of the task. As the increase of the computing scale, the overall acceleration ratio significantly improves, and different kernel functions have different acceleration performance. Observing the trends, we can categorize the calculations into two distinct groups based on their acceleration profiles: consistent acceleration group and variable acceleration group.

* + - 1. Consistent Acceleration Group

Consistent acceleration group includes the Base Parameter, Cubic Compression Factor, and Sort Cubic Roots computations. These three metrics demonstrate a relatively uniform trend in the acceleration ratio as the number of parameters increases. The slight upward trajectory suggests that while the GPU provides a consistent computational advantage over the CPU for these tasks, the extent of the advantage does not dramatically increase with the task complexity. This indicates that the program reaches the memory and computing limits of the GPU prematurely and fails to fully harness its performance potential.

* + - 1. Variable Acceleration Group

The Mixing Rule and Fugacity Coefficient calculations, on the other hand, exhibit distinct profiles:

**Mixing Rule**: The acceleration ratio for the Mixing Rule calculation maintains a stable profile as the number of thermodynamic parameters increases. With the expansion of scale, the acceleration ratio further improves. Although it exhibits a greater acceleration ratio than other kernel functions, it can still be clearly found that the calculation has reached its performance bottleneck.

**Fugacity Coefficient:** In stark contrast, the Fugacity Coefficient computation shows a significant upward trend, indicating a notable performance improvement when transitioning from CPU to GPU as the number of parameters grows. This suggests that the task is highly parallelizable, and the program has not yet reached the memory and computing limits.

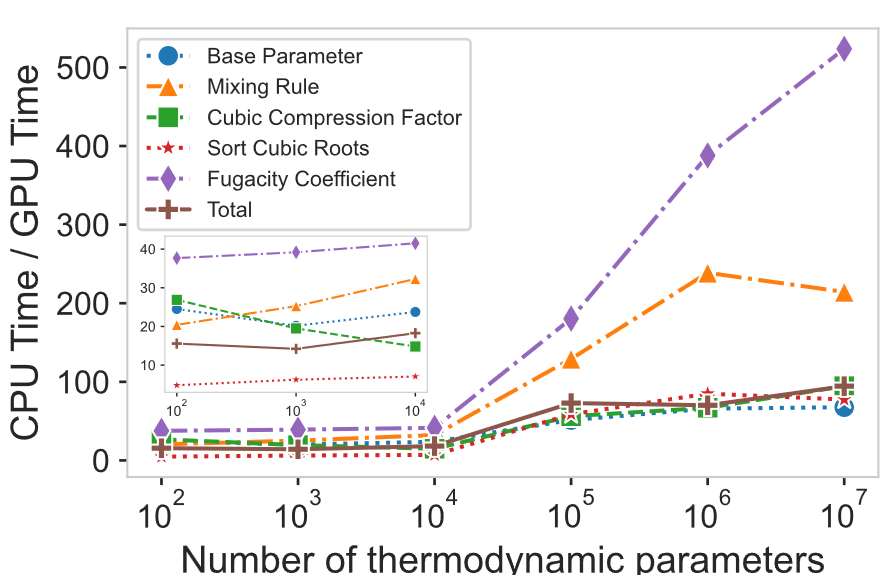


Figure 3. The calculation process and calculation elements of EO method

Overall, Figure 3 suggests that while GPUs can provide computational benefits for all types of tasks, the extent of these benefits varies depending on the nature of the computations involved. Tasks with greater inherent parallelizability exhibit more substantial acceleration, which highlights the importance of aligning computational tasks with the most suitable hardware architecture to maximize performance efficiency.

* + 1. Time-consuming analysis of different kernel functions

Due to the sequential execution of the calculation graph, the most time-consuming function will limit the execution of the whole system as a shortcoming for the whole process, so it is important to analyze the time-consuming of specific programs.

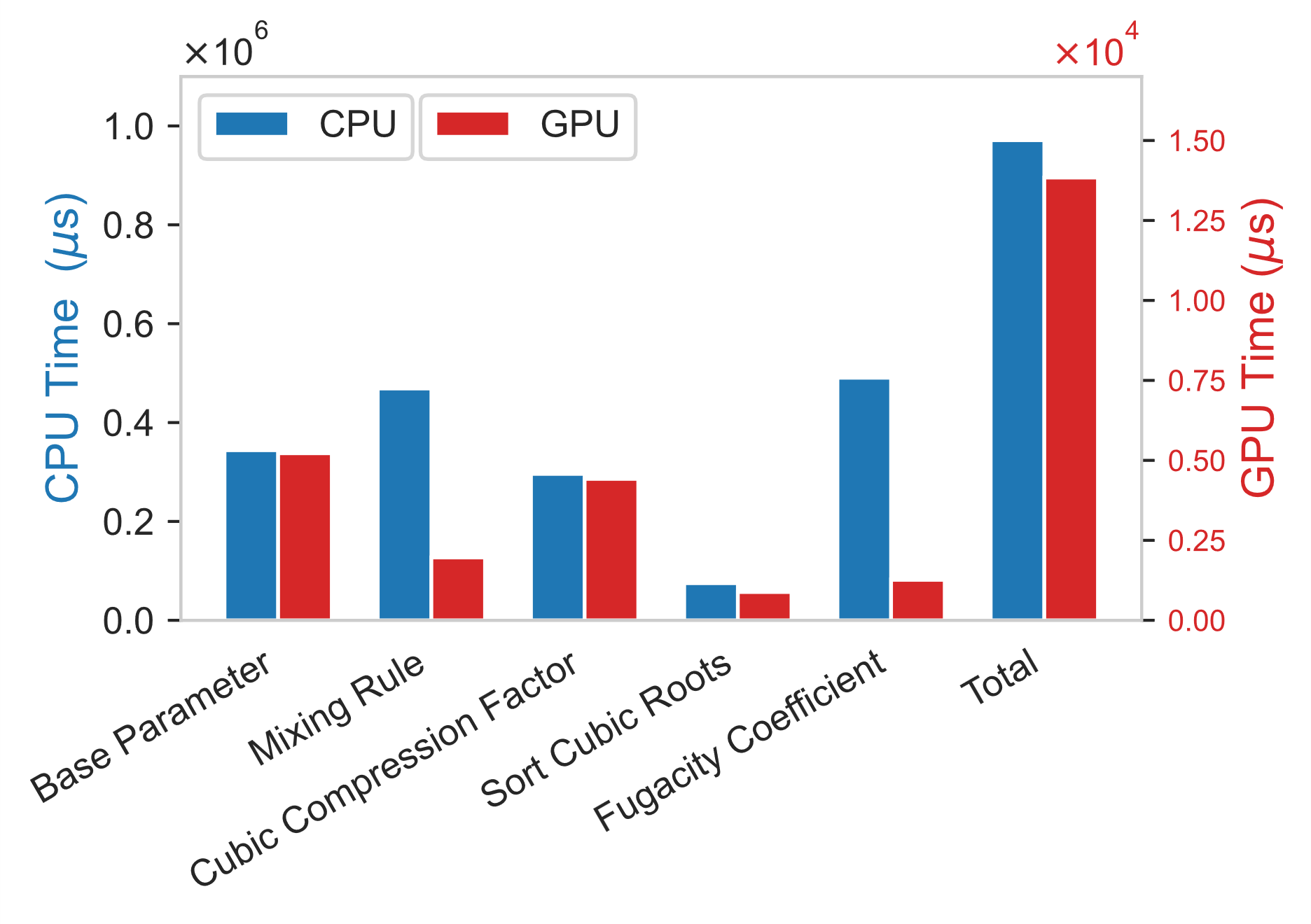


Figure 4. Specific time-consuming analysis of the program

Figure 4 illustrates the computational times, measured in microseconds, for various functions executed on CPUs and GPUs. The left y-axis corresponds to CPU time (), while the right side corresponds to GPU time(), because of the GPU times are significantly lower, hence the need for a separate scale to accommodate the smaller values. Fugacity Coefficient and Mixing Rule calculation spend the most time in CPU mode, however these two functions have the largest acceleration ratio on the GPU, this indicates that the significant improvement offered by the GPU is also significant for the overall operation. In addition, it is important to highlight that the total time of each function is not equal to the time consuming of the whole calculation graph Total because the compiler will fuse the assembly code to reduce unnecessary data replication.

* 1. Optimize GPU execution efficiency.

The block dimension (blockdim) is a pivotal configuration parameter in GPU programming that directly influences the granularity of parallelism and the efficiency of resource allocation within the GPU's architecture. The selection of an appropriate blockdim is essential to achieve an optimal balance between parallel workload distribution and hardware utilization. To illustrate this, experiments have been conducted to observe the effect of different blockdim on program acceleration under different model sizes.

The figure 5 illustrates the acceleration ratio across varying computational scales, with the horizontal axis representing different block dimensions (blockdim), and the vertical axis depicting the corresponding acceleration ratio. The data is segmented into multiple graphs representing different values of model size (n), denoting the number of thermodynamic parameters being computed. These graphs collectively demonstrate the performance impact of block dimension on GPU execution time relative to CPU time for several thermodynamic calculation tasks.

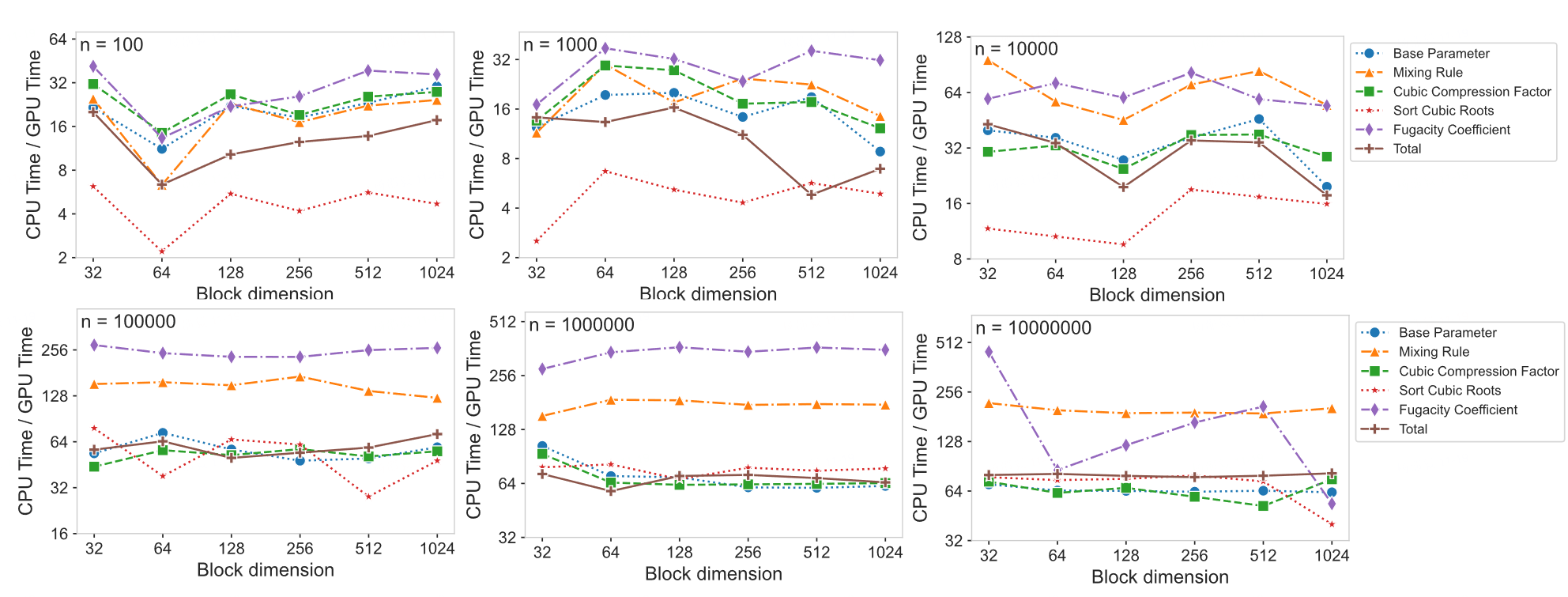


Figure 5. Effect of different Block dimension on program acceleration

As showed in Figure 5, each computational task, from the calculation of base parameters to the evaluation of fugacity coefficients, responds differently to changes in blockdim. Notably, there exists a specific range of block dimensions that maximizes the acceleration ratio for each task, which is indicative of the optimal utilization of the GPU's streaming multiprocessors (SMs) and the minimization of idle time due to thread divergence or memory access bottlenecks. Suboptimal block dimensions can lead to underutilization of the GPU's computational resources, while excessively large block dimensions may cause an oversubscription of resources, leading to contention and reduced throughput. Therefore, the careful tuning of blockdim is instrumental in harnessing the full computational prowess of GPU.

Consequently, the insights underscore the critical role of blockdim in the optimization of GPU programs. The findings serve as a testament to the necessity of deliberate parameter tuning to align with the architecture-specific characteristics of GPUs, thereby ensuring that the parallel computational capabilities are fully leveraged for enhanced scientific computing performance.

* 1. Conclusions

This paper advances the calculation of the EO model by introducing a parallel architecture based on the GPU platform. It proposes a method for large-scale parallel calculation of thermodynamic parameters and conducts a comparative analysis of GPU acceleration performance in different problem dimensions and shows the acceleration effect of the program at the performance bottleneck. Moreover, the execution of the program on the GPU is optimized by blockdim. This work will provide a far-reaching impact on the GPU parallel programming of process simulation.

References

Ma, Y., Shao, Z., Chen, X., & Biegler, L. T. (2016). A parallel function evaluation approach for solution to large-scale equation-oriented models. Computers & Chemical Engineering, 93, 309-322.

Nikolić, D. D. (2018). Parallelisation of equation-based simulation programs on heterogeneous computing systems. PeerJ Computer Science, 4, e160.

Dowling, A. W., & Biegler, L. T. (2015). A framework for efficient large scale equation-oriented flowsheet optimization. Computers & Chemical Engineering, 72, 3-20.

Miller, D. C., Siirola, J. D., Agarwal, D., Burgard, A. P., Lee, A., Eslick, J. C., ... & Gunter, D. (2018). Next generation multi-scale process systems engineering framework. In Computer Aided Chemical Engineering (Vol. 44, pp. 2209-2214). Elsevier.