A Task Graph Parallel Computing Architecture for Distillation Column Simulation on Flash Granularity

Shifeng Qu,a Shaoyi Yang,a Wenli Du,a Feng Qiana

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

Abstract

Simulations of distillation columns stand as the most time-consuming unit operation calculations, profoundly affecting both the efficiency and practical feasibility of process simulation, design and optimization. The complexity of distillation columns stems from the involvement of multiple chemical components as well as extensive energy and mass transfer stages. In this paper, a novel parallel computing architecture is presented specifically for the distillation process. The architecture revolutionizes distillation calculations by decomposing them into smaller, independently and concurrently computable subtasks, i.e., flash calculations and solving tridiagonal matrices for material balance. At the core of the architecture is the creation of an executable task graph, which maps out dependencies between subtasks and incorporates conditional tasking for efficient control-flow management during iterative material and energy balance calculations. The proposed architecture was applied to distillation cases with different computation intensities to ascertain its effectiveness. The results reveal that under optimal parallel granularity, parallelization substantially improves the efficiency of distillation simulation.

**Keywords**: parallel computing, process simulation, distillation column, task graph computation

* 1. Introduction

Process simulation employs mathematical equations and physical models to emulate the behavior and interactions of various components and unit operations under specified conditions. Utilizing process simulation software for distillation processes empowers engineers to comprehend and forecast component separation efficiencies. This facilitates the design and scaling of distillation towers and associated equipment to meet specific production demands and economic objectives. Simulating distillation processes further allows for the optimization of operating conditions, offering vital theoretical and data support for production efficiency, cost control, and safety in the chemical and petroleum industries.

However, distillation columns are the most computation-intensive units in process simulations based on the sequential-modular (SM) approach, demanding extensive iterative calculations. The modular nature of the SM approach facilitates individual module solving and debugging, simplifying error identification and rectification. Despite its limitations in handling processes with nested iterations or complex topologies, the SM approach remains indispensable and is widely adopted in commercial simulation software (Pantelides and Renfro, 2013). In this study, we refined the distillation calculation architecture within the SM framework, integrating parallel computation to speed up distillation simulations. The novel architecture involves dissecting the material balance calculation of each component, bubble point calculation on each stage, and enthalpy conservation calculations into separate subtasks, with their interdependencies mapped according to the distillation algorithm. Based on the dependencies among subtasks, our parallel architecture then generates the computable task graph for distillation simulation, laying the groundwork for parallel processing of material balance equations solving and flash calculations at the components and stages level, respectively. Loop control flow is supported through the integration of conditional task nodes into the task graph, thereby facilitating the control decisions for inner and outer iterations during distillation simulation (Huang et al., 2021). In Section 3, we employed a toy column example and a practical de-butanizer column to substantiate the effectiveness of the proposed architecture.

* 1. Parallel computing architecture for distillation simulation

In chemical engineering, distillation stands out as a prevalent separation technique that capitalizes on the varying boiling points of mixture components to effectively separate lighter and heavier elements. The parallel computing architecture introduced in this paper is intended for parallelizing the bubble-point (BP) distillation algorithm. The BP method is based on equilibrium stages, involving the iterative solving of the MESH equations, namely, **M**aterial balance for each component, phase-**E**quilibrium relation for each component, mole-fraction **S**ummations and energy (ent**H**alpy) balance. Specifically, the vapor phase composition *yi,j* in the material balance equations is eliminated using the phase equilibrium equations, denoted as follows:

|  |  |
| --- | --- |
|  | (1) |

where *K* signifies the phase equilibrium constant, *x* refers to the composition of the liquid phase, with *i* indicating the index of the component and *j* representing the index of the stage in the column. After elimination, the original material balance equations become M-K equations. Each equation within the M-K system can be uniformly expressed in the following form:

|  |  |
| --- | --- |
|  | (2) |

The coefficients in the above equations can each be represented as follows:

|  |  |
| --- | --- |
|  | (3) |

where *Vj* and *Lj* are the vapor and liquid flow rates leaving the *j*th tray, respectively. *F* represents the feed flow rate, while *G* and *S* are the sidestream flow rates for the vapor and liquid phases, respectively. Consequently, the M-K equation set for the *i*th component can be expressed as the following tridiagonal matrix:

|  |  |
| --- | --- |
|  | (4) |

Initial values for *Vj* on each stage and *Ki,j* for each component are set to linearize the M-K equations before solving the distillation system. Then, the M-K tridiagonal matrix of each component is solved to obtain its liquid phase composition *xj* on each stage. After normalizing the *xi,j*on each stage, the bubble point temperature and phase equilibrium constants for the liquid phase of each stage are computed through property calculations. We adopt press-vapor fraction (PV) flash calculation to supersede the bubble point calculation. This is achieved by assigning a near-zero value to the vapor fraction to determine an approximate temperature for the bubble point. The results of PV flash calculations are used to update the temperature, component phase equilibrium constants, and vapor phase composition for each stage. After each update, the liquid composition is checked to ensure it satisfies the S equations, namely:

|  |  |
| --- | --- |
|  | (5) |

where is the tolerance, determined following user-defined computational precision. If not all stages meet the above criteria, the material balance tridiagonal matrix is re-solved using the *Ki,j​* values obtained from the PV flash calculations. This control flow is referred to as the K loop. The loop is executed multiple times until the S equations are satisfied, moving to the energy balance computation in the outer loop to calculate the updated values of *Vj*. Convergence in the outer loop is confirmed by evaluating if the variation in *Vj* across successive iterations falls below a predefined threshold. If the above conditions are not met by all stages, the program will cycle back to the K loop (inner loop), where a new round of material balance and bubble point calculations are performed using the updated *Vj​* values, continuing until the realization of energy balance.

In each iteration of the K loop, bubble point calculations are conducted based on the liquid phase composition *xi,j​*and pressure *Pj​* of each stage to update the stage temperature *Tj*​. The bubble point calculations for stages are independent subtasks, indicating no dependency or communication overhead between them, and thus can be executed simultaneously without waiting for results of each other. Likewise, the solving of the M-K tridiagonal matrix for components can also be performed in parallel. Figure 1 illustrates the revised BP algorithmic diagram in our proposed parallel architecture. As shown in the diagram, bubble point calculations and the solving of tridiagonal matrices are parallelized across all the stages and components, respectively. In our architecture, the simulation of distillation is divided into subtasks with distinct functionalities. These subtasks, along with their dependencies as determined by the distillation algorithm shown in the diagram, are added to the executable task graph. The task scheduler sequences the tasks according to their interdependencies, ensuring that all prerequisite tasks are completed before the commencement of a new task. The workflow of the proposed architecture is shown in A black background with white rectangles

Description automatically generatedFigure 2.

Figure 1: Algorithm diagram of the bubble point method after parallelization.

Notably, it is safe for multiple threads to simultaneously read data from the same memory, provided no thread is writing to that data item at the same time. Data races occur when two or more threads access the same memory location concurrently, and at least one of the accesses is for writing. Data races may lead to inconsistent read results, as different threads might observe various ‘intermediate’ states of the data item being written (Dolan et al., 2018). In our architecture, the distillation column configurations set on the user interface, which remain unchanged throughout the simulation process, are thus shared for reading by multiple threads. Variables that are updated during the inner and outer iteration, like *xi,j*​, *Vj*​, *Ki,j*​, are subject to data races due to repeated modifications by the parallelized subtasks. Therefore, private copies of these data items are created for each subtask to avoid interference. The subtasks process their own data copy and merge the results after parallelization.



Figure 2: Architecture of distillation parallel simulation.

In addition to static tasks such as PV flash calculations and tridiagonal matrix solving, the algorithm also requires tasks consistently monitoring the convergence status of the inner and outer loops and exiting them appropriately. In our architecture, the distillation task graph is designed for dynamic execution, supporting loop control flow decisions without being torn into a flat directed acyclic graph. Specifically, conditional task nodes are introduced to the task graph to determine the subsequent static tasks for execution based on the returned value of convergence evaluation, and thus guide the executor to move between the inner and outer loops.

* 1. Case study

We validated the effectiveness of the proposed task graph parallel architecture on two columns with different computation intensities. One is a toy case for ethylene-ethane separation, with only 3 components and 10 stages. The other is a de-butanizer column from a practical cracked gas separation process, primarily tasked with separating butane (C4 hydrocarbons like butane and isobutane) from heavier hydrocarbons in cracked gas. The de-butanizer involves 32 components and 44 stages. Both columns were simulated using the Soave–Redlich–Kwong equation of state. Experiments were conducted on a Windows 11 x86 64-bit machine equipped with a 12th Gen Intel(R) Core(TM) i5-12490F CPU at 3000 MHz, featuring 6 cores and 12 logical processors, and 32 GB of memory. The whole project was compiled using Ninja with C++19 standards. Notably, we were unable to conduct a comparison of the simulation time for the distillation column across various commercial software. It is impossible to perform time profiling on the distillation code of these software programs due to the inability to access their source code.

A graph with numbers and lines

Description automatically generated

Figure 3: Execution time of columns over an increasing number of threads.

Figure 3 shows the trend of computation time for the toy case and the de-butanizer as the number of spawned threads increases. All the presented results are an average of 10 runs. It can be seen that the simulation efficiency of the toy case shows limited improvement with parallelization. Compared to the de-butanizer, the toy case features a column with much fewer stages and components, thus its PV flash calculations and tridiagonal matrix solutions entail a much smaller problem size and shorter computation time. When parallelizing inherently small tasks, the overheads - including thread management and context switching - can become a substantial part of the total computation time. As the number of threads increases, it will come to a point where the additional overhead surpasses any performance benefits gained from parallelization. Furthermore, the dependencies between subtasks also significantly impact parallelization effectiveness. For the toy case, the tridiagonal matrix solving task for each inner iteration can be distributed for parallel execution across up to three threads. When the number of spawned threads surpasses three, due to task dependencies, some threads become idle, reducing the overall parallel efficiency. When the thread number reaches ten, the PV flash calculation tasks achieve full parallelism. Spawning additional threads will no longer result in further acceleration of the simulation.

In contrast to the toy case, the de-butanizer with a significantly larger subtask problem size, experiences a remarkable simulation speed increase, approaching almost fivefold. However, it is evident from Figure 3 that the improvement becomes increasingly marginal as the number of threads is close to 12. This is because the thread count approaches the number of CPU logical cores, and simultaneously executing too many parallel tasks under limited resources results in parallel efficiency degradation. Figure 4 displays the time profiling of the de-butanizer execution, with a magnified view of the period between 50 milliseconds and 80 milliseconds. The tridiagonal matrix solving, PV flash calculations, and energy balance calculations are executed sequentially in alignment with the distillation algorithm. The time-consuming tasks of the inner loop are evenly distributed among the threads.

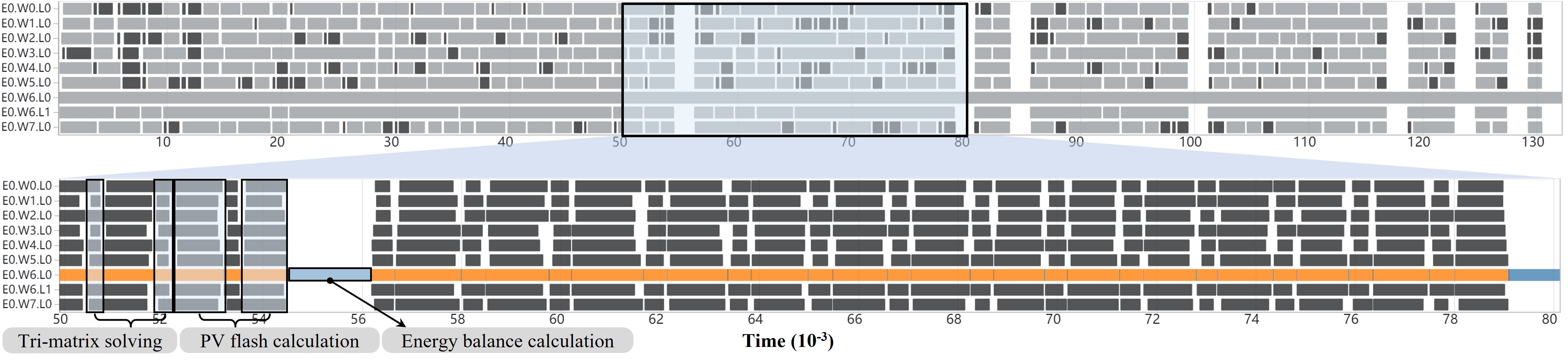


Figure 4: Time profiling of the execution of the column with 32 comps and 44 stages.

* 1. Conclusion

In summary, we have proposed a parallel computing architecture for distillation simulation based on the task graph, which significantly accelerates the material balance tridiagonal matrix solution and bubble point calculation of inner iterations. We facilitate the decision-making process for convergence assessment in a loop control flow by integrating conditional task nodes into the distillation task graph. Experiments show the proposed architecture significantly speeds up the distillation tasks with appropriate parallelism. Owing to the flexibility of the task graph framework, our work can be seamlessly adapted to other distillation algorithms based on MESH equations, such as the inside-out method and the sum-rate method. Given the flexibility in setting task node dependencies within our architecture, the distillation task graph can be involved in assembling any complex chemical process task graph by linking it with other unit operation tasks, realizing parallel simulation of whole process flowsheets.

References

Pantelides, C. C., Renfro, J. G., 2013. The online use of first-principles models in process operations: Review, current status and future needs. Computers & Chemical Engineering, 51, 136-148.

Huang, T. W., Lin, D. L., Lin, C. X., Lin, Y., 2021. Taskflow: A lightweight parallel and heterogeneous task graph computing system. IEEE Transactions on Parallel and Distributed Systems, 33(6), 1303-1320.

Dolan, S., Sivaramakrishnan, K. C., Madhavapeddy, A., 2018. Bounding data races in space and time. ACM SIGPLAN Notices, 53(4), 242-255.