applied sciences

MDPI

Article

Accelerating FVM-Based Parallel Fluid Simulations with Better Grid Renumbering Methods

Huajian Zhang †, Xiao-Wei Guo †, Chao Li, Qiao Liu, Hanwen Xu and Jie Liu *

Institute for Quantum Information & State Key Laboratory of High Performance Computing, College of Computer Science and Technology, National University of Defense Technology, Changsha 410073, China; zhjsag@nudt.edu.cn (H.Z.); guoxiaowei@nudt.edu.cn (X.-W.G.); dirk911@nudt.edu.cn (C.L.); liuqiao@nudt.edu.cn (Q.L.); hanwayxu@foxmail.com (H.X.)

* Correspondence: liujie@nudt.edu.cn

† These authors contributed equally to this work.

Abstract: Grid renumbering techniques have been shown to be effective in improving the efficiency of computational fluid dynamics (CFD) numerical simulations based on the finite volume method (FVM). However, with the increasing complexity of real-world engineering scenarios, there is still a huge challenge to choose better sequencing techniques to improve parallel simulation performance. This paper designed an improved metric (MDMP) to evaluate the structure of sparse matrices. The metric takes the aggregation of non-zero elements inside the sparse matrix as an evaluation criterion. Meanwhile, combined with the features of the cell-centered finite volume method supporting unstructured grids, we proposed the cell quotient (CQ) renumbering algorithm to further reduce the maximum bandwidth and contours of large sparse matrices with finite volume discretization. Finally, with real-world engineering cases, we quantitatively analyzed the evaluation effect of MDMP and the optimization effect of different renumbering algorithms. The results showed that the classical greedy algorithm reduces the maximum bandwidth of the sparse matrix by at most 60.34% and the profile by 95.38%. Correspondingly, the CQ algorithm reduced them by at most 92.94% and 98.70%. However, in terms of MDMP, the CQ algorithm was 83.43% less optimized than the Greedy algorithm. In terms of overall computational speed, the Greedy algorithm was optimized by a maximum of 38.19%, and the CQ algorithm was optimized by a maximum of 27.31%. The above is in accordance with the evaluation results of the MDMP metric. Thus, our new metric can more accurately evaluate the renumbering method for numerical fluid simulations, which is of great value in selecting a better mesh renumbering method in engineering applications of CFD.

Keywords: GRID renumbering; MDMP; computational fluid dynamics; YHACT; cell-quotient; parallel computing

Citation: Zhang, H.; Guo, X-W. ; Li, C.; Liu, Q.; Xu, H.; Liu, J. Accelerating FVM-Based Parallel Fluid Simulations with Better Grid Renumbering Methods. *Appl. Sci.* **2022**, *12*, 7603. https://doi.org/10.3390/app12157603

Academic Editor: Sébastien Poncet

Received: 5 July 2022
Accepted: 21 July 2022
Published: 28 July 2022

1. Introduction

Computational fluid dynamics (CFD) are widely used in aerospace [1], reactor thermal-hydraulics, ship and ocean engineering [2], explosions, etc. However, simulating the delicate structures of the flow field for real-world engineering cases is still forbidden due to the enormous amount of calculation [3]. Thus, parallel performance optimization techniques have received extensive attention in academia and industry [4,5]. The grid numbering directly determines the matrix structure after discretization, then exhibiting great impact on performance of parallel fluid simulations. Nowadays, accelerating fluid simulations through grid renumbering has become a well-established approach.

The traditional metrics such as Bandwidth [6] or Profile [7] have been applied to discriminate the matrix structures of a linear system for decades, which could provide a valuable guidance for choosing an effective ordering method. These indicators were widely used due to the simplicity of calculation. However, calculating the bandwidth of the matrix does

not accurately reflect the performance in complex applications. The structure of sparse matrices determines the order of memory access for parallel computation. This has a significant impact on the performance of modern computer architectures with multiple levels of storage hierarchies. Taking the most widely used Finite Volume Method (FVM) [8] as an example, after numbering the discrete cells of the simulation domain, it is necessary to assemble sparse linear systems in a parallel fashion. The assembly process is based on the parameters and numbers of connected grid cells. There are many well-known numbering algorithms, which can be found in many literature works, such as Gibbs [9], Cuthill and McKee [10], Sloan [11], Akhras and Dhatt [12], and the Reverse Cuthill-Mckee by George et al. [13] It is worth noting that optimizing the numbering of a sparse matrix is a formidable combinatorial problem and is therefore not a viable option to find the true optimal solution. Therefore, these published methods all used some kind of greedy-based local optimization algorithm. The heuristic algorithm [14–16] is another type of numbering algorithm proposed in recent years, but it does not perform well in terms of the overall iterative efficiency of numerical simulations.

These classical methods have proved their stability, rapidity, and efficiency in simple benchmarks. It has become a consensus that grid renumbering can effectively improve computing performance. Thus, we believe that refocusing this classical topic and providing improved methods for grids numbering of complex cases are of great significance. Nevertheless, in fluid simulation of complex engineering problems, choosing a better numbering method still faces great challenges. Firstly, these simple indicators such as Bandwidth and Profile cannot accurately evaluate the computational performance of sparse matrices corresponding to complex applications. In complex application scenarios, lower bandwidth does not always lead to better performance. Secondly, the existing grid numbering algorithms are mainly focused on the optimization of solving linear systems. However, the equation discretization process also occupies a large proportion in a CFD simulation. It is necessary to quantitatively analyze the effect of mesh numbering on the complete calculation process with complex fluid simulation cases.

To address the above problems, we proposed an improved indicator for more accurately evaluating the structure of a sparse matrix, and further designed a novel grid renumbering algorithm to reduce the key metrics of the matrices in FVM-based fluid simulations. Complex engineering cases have been tested in parallel and analyzed to validate the proposed algorithm and indicator.

This paper is organized as follows; Section 2 describes the finite volume methods and aims to demonstrate why the grid numbering is important for FVM-based simulations. Section 3 presents the improved indicator MDMP for more accurately discriminating the structure of a sparse matrix. Section 4 proposes the Cell–Quotient (CQ) renumbering algorithms and the implementation framework based on an in-house CFD software. Section 5 presents simulation results and performance for validating the proposed algorithm and software. Conclusions are contained in Section 6.

2. Numerical Fluid Simulations Based on Finite Volume Method

As the most widely used numerical approach, the finite volume method (FVM) [17–19] has been implemented in a large number of commercial and open-source CFD software. For the completeness and clarity of the paper, we provide a brief introduction to the FVM and illustrate the role of grid numbering in a FVM-based fluid simulation in the following.

2.1. The Basic Idea of a Finite Volume Method

The finite volume method is also called the control volume method. Its basic idea is to divide the computational area into discretized non-repeating control volumes. A set of discrete equations is obtained by integrating the differential equations (governing equations) to be solved over each control volume. The coefficients of discrete equations are calculated by values situated in the cell center and its neighbors, which is why it is important to keep the neighboring cell numbers as close as possible.

In a concise summary, the basic idea of FVM can be expressed in Figure 1. The FVM acts like a transformer in fluid dynamics simulations, taking the control equations with boundary and mesh configurations as inputs [20]. The resulting outputs contain discretized sparse linear systems [21].

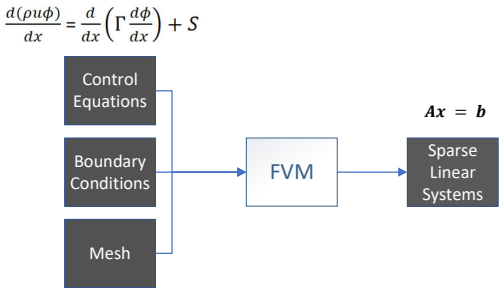


Figure 1. A schematic diagram of FVM. The control equations with boundary and mesh configurations are discretized into sparse linear systems by FVM.

In order to assemble a linear system $\boldsymbol{Ax} = \boldsymbol{b}$, the continuous governing equation has to be solved discretely on a grid. Usually, the input mesh file contains the unstructured grids. It means that the internal meshes in the interesting domain do not have exactly the same adjacent cells. As shown in Figure 2, (a) represents a structured grid and (b) represents an unstructured grid. The unstructured grid is used for this paper's examples and experimental cases.

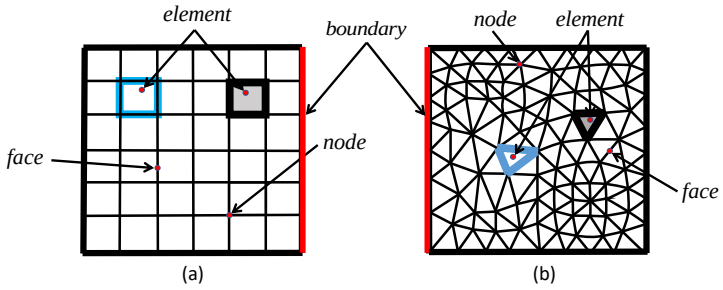


Figure 2. (**a**) Domain discretized using a uniform grid system, and (**b**) domain discretized using an unstructured grid system with triangular elements.

After the domain discretization process using unstructured grids, the following four geometric elements can be obtained:

- Node: The geometrical position of the unknown quantity is to be solved;
- Control volume: The smallest geometric unit to which a governing equation or conservation law applies;
- Face: Interface positions are used to segment control volumes corresponding to different nodes;
- Grid line: A cluster of curves formed by joining two adjacent nodes.

In order to identify each element, a number or id has to be assigned to every node, volumes/cell, and face. The number determines the sequence of assembling and the resulted matrix structure.

2.2. Matrix Assembly for Governing Equations

Considering a simple case with a one-dimensional unstructured grid, the continuity equation, momentum equation, and energy equation can all be written in the following general formula:

$$\underbrace{\frac{d(\rho u\phi)}{dx}}_{\text{convection term}} = \underbrace{\frac{d}{dx}\left(\Gamma\frac{d\phi}{dx}\right)}_{\text{diffusion term}} + \underbrace{S}_{\text{source term}} \quad (1)$$

Equation (1), which represents the diffusion equation for a single time step of FVM., contains the convection term, diffusion term, and source term. ϕ in the equation is a generalized variable, which can be velocity, temperature or concentration, and other physical quantities to be obtained. Γ is the generalized diffusion coefficient corresponding to ϕ, and S is the generalized source term. The boundary condition values of the variable ϕ at endpoints A and B are known.

The above is the conserved form of the equation, which is necessary to establish the discrete equation using the finite volume method. The critical step of the finite volume method is to apply integration in the interval of control volume of the control equation, and one gets discrete equations on the control volume nodes. Next, taking the Equation (1) as an example to perform domain discretization:

$$\int_{\Delta V} \frac{d(\rho u \phi)}{dx} dV = \int_{\Delta V} \frac{d}{dx}(\Gamma \frac{d\phi}{dx}) dV + \int_{\Delta V} S dV \tag{2}$$

Equation (2) represents the integration of the fluid domain over a single time step for the control unit after regional discretization.ΔV is the volume value of the control volume. When the control volume is minimal, ΔV can be expressed as $\Delta \bullet AV$, where A is the area of the interface of the control volume. Both convection and diffusion terms have been transformed into values on the control volume interface. One of the most significant characteristics of the finite volume method is that the equation has explicit physical interpolation. Physical quantities at nodes represent physical quantities at the interface through interpolation.

In order to calculate the physical parameters on the interface, it is necessary to carry out approximate distribution among nodes. It can be imagined that linear approximation is the most straightforward method to calculate the characteristic values, called the central difference. The conservation equation on a control volume can be expressed as the following:

$$a_P \phi_P = a_H \phi_H + a_E \phi_E + b \tag{3}$$

where a_P,a_H,a_E,b are the unknown coefficient. For the one-dimensional problem, area A at the control volume interface e and w are both 1, namely unit area; consequently, $\Delta V = \Delta X$.

Equation (3) demonstrates the tight connection between the unit to be solved and the neighboring units, which also reflects the memory access order of the finite volume method. The discretized system of equations is stored in a numerical simulation framework in the form of a sparse matrix [8]. The rows represent the coefficients of the discretized equations, and the off-diagonal coefficients represent the interaction between adjacent elements. The number of non-zero elements in each row is the same as the number of elements associated with that row.

The input of a linear solver is a system of equations generated by the discretization process, which can be written mathematically as:

$$\boldsymbol{A}\phi = \boldsymbol{b} \tag{4}$$

where $\boldsymbol{A}$ is the matrix of the coefficients of the element a_{ij}, ϕ is the vector of unknown variables ϕ_i, and b is the vector of sources b_i. Substitute a system of equations consisting of a series of discrete Equations (3) into matrix (4) and expand:

$$\begin{bmatrix} a_{11} & a_{12} & & & \\ a_P & -a_H & -a_E & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & a_{n-1n} \\ & & & a_{nn-1} & a_{nn} \end{bmatrix} \begin{bmatrix} \vdots \\ \phi_P \\ \phi_H \\ \phi_E \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ b \\ \vdots \\ \vdots \end{bmatrix} \tag{5}$$

The solution conditions required by the target element are usually only related to its neighbors. The coefficient matrix is always sparse since an element is only connected with a few neighbors. The techniques for solving algebraic equations can be roughly divided into direct and iterative methods. The direct method requires the inverse of a sparse matrix. However, when the matrix is large, the cost of calculating its inverse will be substantial, and a lot of memory will be needed. Therefore, the iterative method is usually adopted in the CFD numerical simulation framework. Iterative rules require multiple applications of the solution algorithm until the desired level of convergence is achieved without obtaining a completely convergent solution. Many standard methods have been developed [22]; for example, Gauss Elimination, LU Decomposition, Gauss–Seidel Method, Incomplete LU (ILU) Decomposition, the Conjugate Gradient Method, and the Multigrid Approach.

Further analyzing the distributions of non-zero elements in linear system (5), the position of the non-zero elements exactly corresponds to the element itself and its neighbors. Therefore, in this process, the numbering of neighboring cells significantly affects the structure of the matrix. Furthermore, simply reordering the input grid data without changing the program [23] can make the numbering of adjacent cells more concentrated and assemble more diagonal-dominant matrices.

3. MDMP: An Improved Matrix Structure Indicator Based on the Mean Distance of Median Points in Sparse Matrix

3.1. Traditional Structure Indicators of Sparse Matrices

It is well known that implicit discretization of the control partial differential equations by the finite volume method yields a set of large sparse matrices. The system of linear equations $\boldsymbol{Ax} = \boldsymbol{b}$ is then solved by iterative or direct methods. The sparsity or structure of a matrix $\boldsymbol{A}$ has a crucial effect on the memory access efficiency. The mostly widely used indicators for the structure of $\boldsymbol{A}$ include the maximum bandwidth and the profile. In a matrix $\boldsymbol{A} = (a_{ij})$ of order N($a_{ij} \neq 0, i = j$), let the first non-zero element of the i-th row be a_{ij}; then,

$$\boldsymbol{\beta_i} = \max\{|i - j| \;\; a_{ij} \neq 0\} \tag{6}$$

is called the bandwidth of the i-th row. By analogy, the maximum bandwidth of matrix $\boldsymbol{A}$ could be defined as

$$\boldsymbol{\beta(A)} = \max_{1 \leq i \leq n} \beta_i = \max_{1 \leq i \leq n} (\max\{|i - j| \;\; a_{ij} \neq 0\}) \tag{7}$$

The smaller the maximum bandwidth, the better the quality of the sparse matrix represented, show in Figure 3.

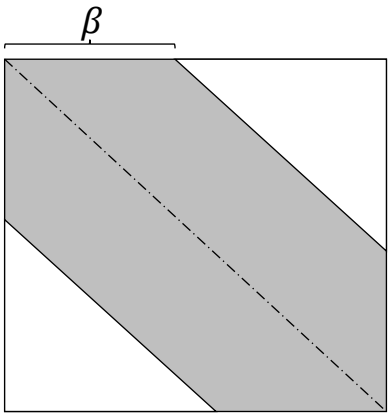


Figure 3. Schematic representation of the maximum bandwidth, where B represents the maximum bandwidth of the matrix shown.

Another important metric is the matrix profile. In order to calculate profile, we firstly locate the leftmost non-zero element of each row of the matrix in Equation (8).

$$f_i = min \ \ j \ \ |a_{ij} \neq 0 \tag{8}$$

Now define the distance from the above non-zero element to the diagonal of the corresponding row:

$$\delta_i = i - f_i \tag{9}$$

Then, the final profile is defined as

$$\sum_{i=1}^{n} \ \delta_i \tag{10}$$

This value portrays the sum of the bandwidth per row of the matrix, which takes into account the overall nature of the sparse matrix more than the bandwidth. However, after extensive testing, these two indicators are actually positively correlated in most cases.

3.2. An Improved Matrix Structure Indicator Based on the Mean Distance of Median Points

The above two traditional sparse matrix quality metrics are very widely used in most renumbering algorithms because of their simple implementation and intuitive clarity. In a computer with hierarchical storage structure, the sparsity of the sparse matrix will greatly affect the cache hit rate during the solution process and thus the overall iteration efficiency of the case, since the sparse matrix is ultimately involved in the CFD solution calculation. However, the above metrics only express the concentration of the outermost nonzero elements of the sparse matrix and do not strictly take into account the dispersion of the inner nonzero elements. For numerical calculations during CFD solving or MPI message passing between different processes, the cache line is so small that the cache data are constantly updated. Therefore, hitting more data in the same cache is the focus of the study. In other words, the presence of more non-zero elements within a smaller distance from the diagonal of the sparse matrix is the desired effect of the renumbering algorithm.

A median point represents the number in the middle of a set of data arranged in order, and equally divides the set of values into two parts. As a statistical variable, the median point values are not easily affected by individual maxima or minima, so they have better stability.

Based on the median points of non-zero elements in a sparse matrix, we define the set of all points whose distance to the diagonal is D . The elements in D are sorted in ascending order and defined as d_k

$$d_k = |i - j|, \ \ (a_{ij} \neq 0 \ \mid \ i \leq j \ \mid \ k = 0, \ldots, n \ \mid \ d_0 < \cdots < dk < \cdots < d_n \ \mid \ d_k \in D) \tag{11}$$

When $k = \frac{n}{2}$, d_k is the distance of the median point and the mean distance of median point, named MDMP, could be expressed as

$$\bar{d} = \sum_{k=1}^{\frac{n}{2}} \frac{d_k}{k} \tag{12}$$

$\bar{d}$ can well characterize the degree of dispersion of this sparse matrix. It was is verified that the degree of dispersion of non-zero elements within the median is an important indicator of the quality of the sparse matrix. The advantages of MDMP over traditional indicators will be analyzed in detail in the Results presented in Section 5.

4. CQ: The Cell-Quotient Grid Renumbering Algorithm for Large-Scale Parallel Fluid Simulations

In this section, we firstly introduce an extensible grid renumbering framework based on an in-house CFD software, then a variant of the widely used greedy method [23]

have been reviewed. Finally, we present the Cell–Quotient (CQ) renumbering algorithm. The details of the framework and the renumbering algorithms for parallel fluid simulations will be presented in the following.

4.1. An Extensible Grid Renumbering Framework for CFD Simulations

The YHACT was designed initially for thermo-hydraulics analysis and implemented as a general-purpose CFD software based on an extensible architecture. As shown in Figure 4, the software adopts a classic hierarchical architecture. Benefitting from object-oriented technology and the modern features of the C++ language, YHACT is more modular and scalable. More details about the design philosophy of YHACT could be found in Ref. [24].

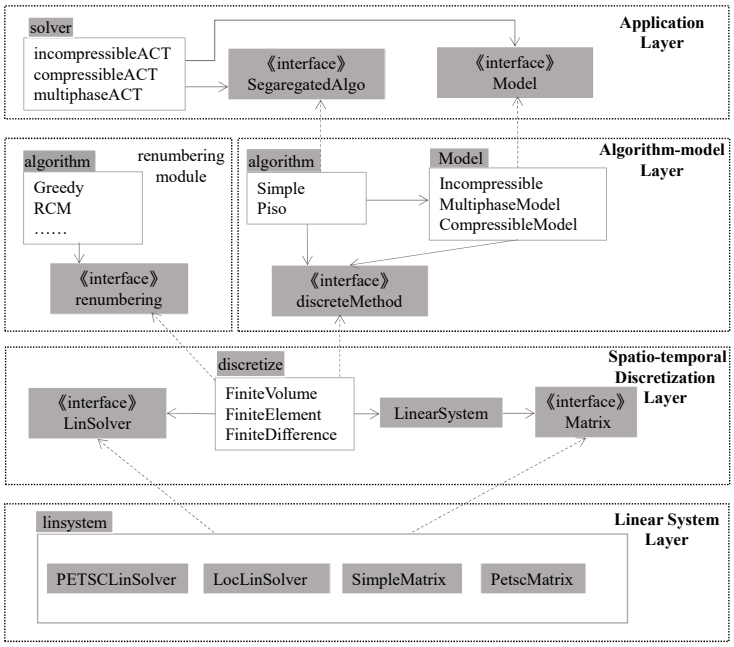


Figure 4. The extensible framework for integrating parallel grid renumbering techniques in YHACT. Different types of renumbering algorithms could be extended by implementing the interfaces of the renumbering module. The Application Layer, Algorithm-model Layer, Spatio-temporal Discretization Layer, and Linear System Layer are the original frameworks in YHACT. The renumbering module is a renumbered framework added in this paper.

In summary, the application layer includes the main program for solving application-specific problems such as incompressible flow, compressible flow, and multiphase flow. Algorithm-model layer, Spatio-temporal discretization layer, and linear system layer mainly accomplish the discretization of physical models, the assembly, and solution of sparse matrices. The renumbering module is crucial for integrating renumbering algorithms into numerical simulations. In a solving algorithm such as *Simple*, the renumbering interface should be called before the *discretize*. Due to the highly scalable software architecture, the higher-layer modules depend only on the interfaces that lower-layer modules could implement. Thus, new algorithms and models can be added continuously.

We integrated the renumbering module and designed the corresponding interfaces from the software architecture perspective. Different numbering algorithms can be seamlessly integrated and tested. In addition, grid renumbering is one of the optimizing procedures before FVM-based simulations from the perspective of parallel computing. As presented in Figure 5, the renumbering algorithms will be executed in parallel after domain decomposing. Then, the FVM-based simulations are the same as the original algorithm.

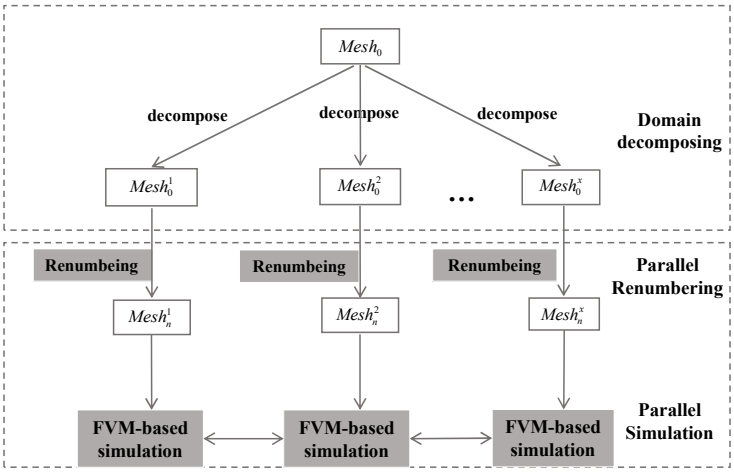


Figure 5. A diagram of parallel grid renumbering and FVM-based simulations. Domain decomposing embodies the meshing step in preprocessing. Parallel Renumbering embodies the idea of parallel grid renumbering. Parallel Simulation indicates a CFD grid for numerical simulation calculations. During the final parallel calculation, boundary information is passed between the processes.

Due to the discrete nature of FVM, the solution of one target cell is only related to its neighbors. In other words, only the data of neighboring cells of different decomposed grids need to be communicated between processors. In the YHACT framework, the boundary cells of all decomposed grids are wrapped with a layer of dummy cells containing the necessary conditions and data for the solution of the boundary cells. As shown in Figure 5, after the decomposing phase, each process calls the renumbering interface for renumbering. In the parallel computation phase, the data of adjacent boundary surfaces between different processes are exchanged by calling MPI functions. Finally, the solution will be obtained by collaborative computing on multiple CPU nodes.

4.2. Greedy Numbering Algorithm

The Greedy algorithm was originally proposed by Farhat [25] and is derived from the idea of mesh partitioning. Numbering unstructured grids based only on index positions leads to a loss of the read data's spatial locality and temporal locality. This index-based numbering may create a loop that traverses the grid in the direction of the index. Computers use hierarchical storage. A high number of grids may result in frequent memory reads. Some grid data that are adjacent but not end numbers may not be stored in the same or adjacent cache blocks. Therefore, in order to improve the performance of the code, loop-blocking [26] is frequently used. Loop-blocking allows the code to process artificially set implicit grid blocks at once during data reading. In contrast, the numbered block members are brought to the cache for the next operation by setting the corresponding storage list. The general program structure of the algorithm is given in Algorithm 1. "$mesh_o$" represents the initial grid, which is numbered according to the read CGNS mesh file in random order. "$mesh_n$" is the renumbered grid. "*blocksize*" is the size of the implicit block, which indicates that the current loop numbers the "*blocksize*" number of cells first. According to the experience, the size of *blocksize* does not have much influence on the numbering effect and the overall calculation speed of CFD numerical simulation, so it was set to 2000 in this paper. The variables "*nElement*" represent the total number of cells in the current grid . It is critical to select the appropriate initial numbering units as far as possible on the grid boundaries. Steps 1 through 6 are the initialization before the program starts the loop. Step 7 through Step 14 are the program's main loop, with elements numbered in implicit blocks. Step 15 represents the end of renumbering all elements in the case. When the number of grids is large, Step 18 ensures that each implicit block is derived from the interface of the previous block to preserve data locality as much as possible. Step 21 ensures that when the grid is continuous. Suppose the current list of neighbors is exhausted, and the overall grid

is not finished numbering. In that case, we can jump to the relevant position of the previous block and continue numbering the unnumbered neighbors. The numerical simulation of CFD is complex, and disconnected blocks are often generated at different grid components. Therefore, Step 24 ensures that the algorithm can jump to a grid component that does not participate in renumbering in time when the grid is disconnected.

Algorithm 1 Greedy Numbering Algorithm

input: $mesh_o$, $blocksize$
output: $mesh_n$

1: initialise: select nodes to the initially $mesh_o.element(k)$, set all the original cell numbers to -1, $counter = 0$, $I = \emptyset$
2: $mesh_o.element(k).id() = counter$
3: $counter = counter + 1$
4: **for all** the neighbors of $mesh_o.element(k)$ **do**
5: initialise L: Push the unnumbered neighbors of $mesh_o.element(k)$ into list L
6: **end for**
7: **while** $L \neq \emptyset$ and $mod(counter, blocksize) \neq 0$ and $counter < mesh_o.nElement()$ **do**
8: Renumber the first element (as $mesh_o.element(i)$) of the L list with the value "$counter$"
9: update $mesh_n$ with the renumbered element ids
10: $counter = counter + 1$
11: if $mesh_o.element(i)$ exists in containers I and L, remove $mesh_o.element(i)$ from both containers
12: **for** All the neighbors of $mesh_o.element(i)$ **do**
13: Push the unnumbered neighbors of $mesh_o.element(i)$ into list L
14: **end for**
15: **end while**
16: **if** $counter = mesh_o.nElement()$ **then**
17: $BREAK$
18: **end if**
19: **if** $mod(counter, blocksize) = 0$ **then**
20: Assign the first element of container L to $mesh_0.element(k)$
21: **end if**
22: **if** $L = \emptyset$ and $I \neq \emptyset$ **then**
23: Assign the first element of container I to $mesh_o.element(k)$
24: **end if**
25: **if** $L = \emptyset$ and $I = \emptyset$ **then**
26: Find a new unnumbered element for $mesh_o.element(k)$
27: **end if**
28: **if** $L \neq \emptyset$ **then**
29: Insert list L at the end of list I, and goto 2
30: **end if**

4.3. Cell-Quotient (CQ) Numbering Algorithm

The idea of CQ renumbering algorithm proposed in this paper is derived from the AD algorithm [12]. This method belongs to the mathematical induction method. The sparse matrix with good convergence is generated by numbering the CFD grid. Based on the idea of mathematical induction, the final CQ algorithm is formed by summarizing the characteristics of each element such as the sum of neighboring element numbers. Figure 6 shows a comparison of maximum bandwidth and profile of matrix with disordered cell numbers and better cell numbers.

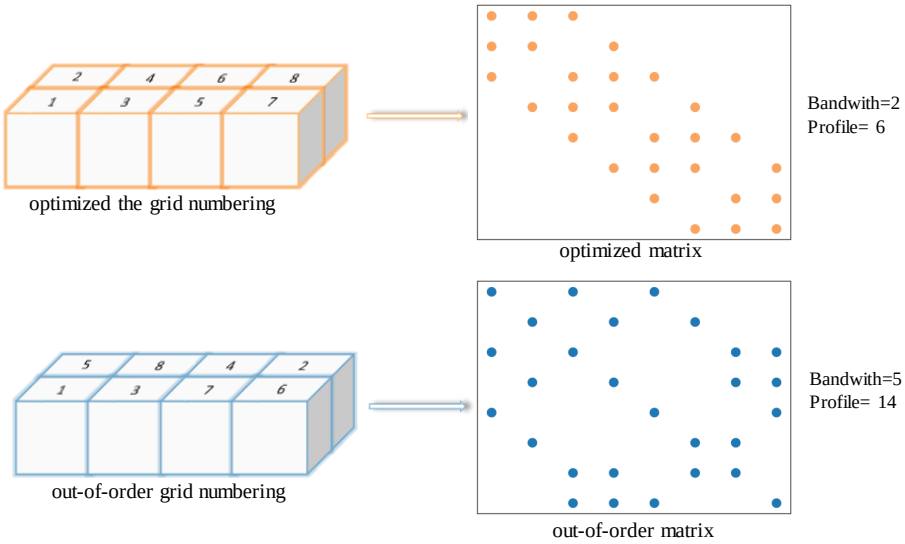


Figure 6. Comparison of maximum bandwidth and profile of matrix with disordered cell numbers and better cell numbers. The upper part shows from left to right the better cell numbered cases, the matrix of the cases and the bandwidth and contours of the matrix, respectively. The lower part shows the disordered counterpart.

Tables 1 and 2 correspond to the better grid numbering, disordered grid numbering of Figure 6, respectively. The CFD grid number has a close relationship with its cell quotient, with a significant positive correlation. Only the cell quotient of the better grid numbering is arranged from smallest to largest. In other words, numbering the mesh according to the ascending sequence of cell quotients will result in better numbered cells, which further results in better sparse matrices. The algorithm proves that it greatly reduces the bandwidth and contour of sparse matrices and may play a great role in matrix multiplication operations such as SpMV in the future. This paper only studies its role in CFD numerical simulations based on the finite volume method.

Table 1. The cell quotient calculation method for the better grid. The first column indicates the cell number of the current grid. The second row indicates the neighboring cells corresponding to the cell. The third column indicates the sum of neighboring cells. The fourth column indicates the cell quotient of the cell, which is obtained from *the quotient of the sum of number and the number of neighboring cells*.

Cell ID	Optimized Neighboring Cell	Sum of Number	CQ
1	2,3	5	2.5
2	1,4	5	2.5
3	1,4,5	10	3.3
4	2,3,6	11	3.7
5	3,6,7	16	5.3
6	4,5,8	17	5.7
7	5,8	13	6.5
8	6,7	13	6.5

Table 2. The cell quotient calculation method for the disorder grid. The meaning of each column is the same as in Table 1.

Cell ID	Disorder Neighboring Cell	Sum of Number	CQ
1	3,5	8	4.0
2	4,6	10	5
3	1,7,8	16	5.3
4	2,7,8	17	5.7
5	1,8	9	4.5
6	2,7	9	4.5
7	3,4,6	13	4.3
8	3,4,5	12	4.0

Different from the traditional AD algorithm, this paper uses the change relationship between the target grid cell and the sum of neighbor numbers as the judgment criterion for the optimization iteration. The specific steps are as follows:

1. Set the convergence index ϵ of the optimization iteration;
2. Calculate the maximum bandwidth β_o under the current number;
3. Calculate the sum of adjacent mesh numbers, the number of related grids, and the number of cell quotient for each grid;
4. Renumber all nodes according to the cell quotient;
5. Calculate the maximum bandwidth β_1 under the current number;
6. If $0 \leq (\beta_o - \beta_1)/\beta_o \leq \epsilon$, abort the optimization iteration process; otherwise, repeat the calculation until convergence.

The general program structure of the algorithm is given in Algorithm 2.

Algorithm 2 CQ Numbering Algorithm

input: $mesh_o$
output: $mesh_n$
initialise: convergence index $e \in [0.01, 0.05]$,$value = DOUBLE\_MAX$
while $value \geq e$ **do**
  initialise β_o: sparse matrix maximum bandwidth
  for all $mesh_o.element(i)$ in $mesh_o.nElement()$,and $i = 0, 1, \ldots, n$ **do**
    initialise I:Press the sum of all neighbor numbers of $mesh_o.element(i)$ into I
    initialise L: Press the total number of neighboring cells of $mesh_o.elemen(i)$ into L
  end for
  initialise K
  for all $I(i), L(i)$ in I,L, and $i = 0, 1, \ldots, n$ **do**
    $cell\_quotient(i) = I(i)/L(i)$
    Establish the $key - value$ relationship between $mesh_o.element(i)$ and $cell\_quotient[i]$ in K
  end for
  for all menbers in K **do**
    renumbering all the $mesh_o.element(i).id()$ according to the ascending order of $cell\_quotient$
    update $mesh_n$ with the renumbered element ids.
  end for
  initialise β_n: sparse matrix maximum bandwidth after renumbering
  $value = (\beta_o - \beta_n)/\beta_o$
  clear I, L, K
  if $value < 0$ goto step 3,and continue calculation
end while

Step 1 is the initialization of the program before starting the loop. It mainly sets the convergence coefficient ϵ, which is usually set from 0.01 to 0.05. Step 2 starts the main loop of the program. Step 2 and step 18 calculate the maximum bandwidth of the sparse matrix before and after renumbering, respectively. Step 19 updates the current value. Steps 4 to 9 calculate the neighbor numbers of all grids and the number of neighbor grids. Steps 11 to 14 establish the $key - value$ relationship between the cell quotient and the corresponding mesh in container K. Step 15 is the renumbered gird. Step 15 is the key step for renumbering. Step 21 prevents renumbering from becoming ineffective.

However, the algorithm has certain implementation difficulties. (1) Since all the cases in this paper use unstructured grids, the neighbors cannot be traversed directly with indexes. (2) The algorithm is extremely dependent on the initial numbering. If the initial numbering is too chaotic, as the case size gradually increases, the iteration time of the numbering method with this quotient will be very long and the algorithm is unstable, which defeats the original purpose of the efficient renumbering algorithm. Therefore, this paper is the implementation of the CQ algorithm under the premise of Greedy algorithm. The classical Greedy algorithm is used as the initial numbering method, and the CQ algorithm is used for the second renumbering at the end. At the same time, due to the object-oriented development feature of ACT framework adopted in this paper, each unit has a "neighbor" pointer. Therefore, the above two difficulties can be well solved. It was proved that the numbering time of both the Greedy algorithm and the CQ algorithm based on the Greedy algorithm is several seconds in parallel computing, which is negligible in CFD numerical simulation.

5. Results

To validate and analyze the algorithms and framework proposed in previous sections, we build complex testing cases and run typical configurations on a modern high-performance computer.

All results in this section are obtained from parallel simulations on a High-performance computer which consists of around 480 computing nodes. The details of the CPU node are shown in Table 3. The CPU with hardware configuration adopts Intel(R) Xeon(R) E5-2620 v2, which carries 12 cores and 64 G memory in one node. A summary of this HPC system is presented in Table 3.

Table 3. The configuration information of the HPC platform used in this paper.

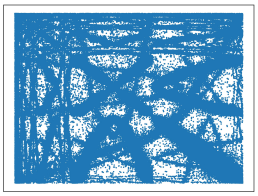
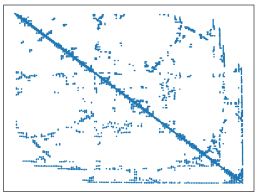
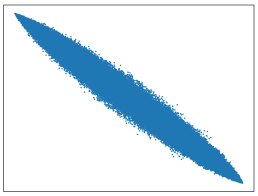
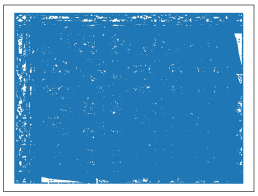
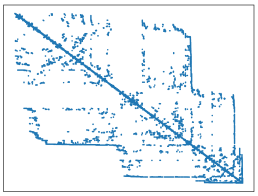
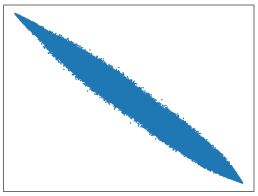
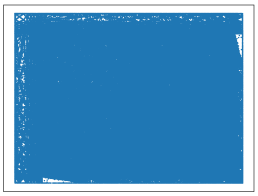
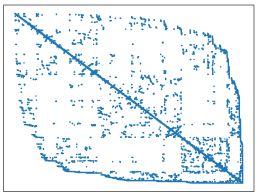
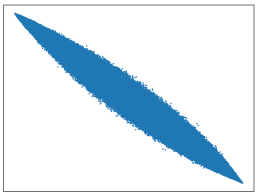
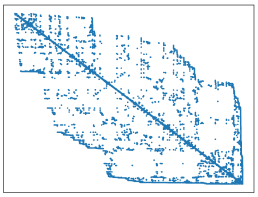
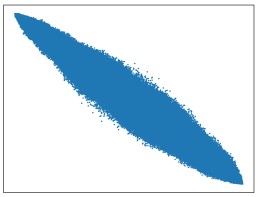
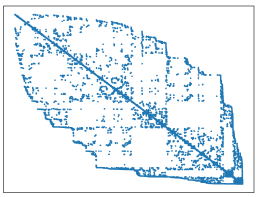
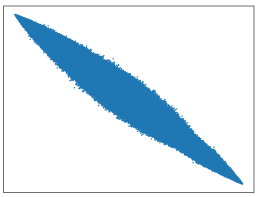
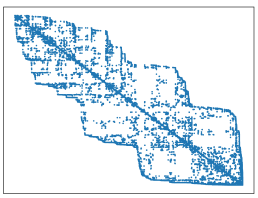
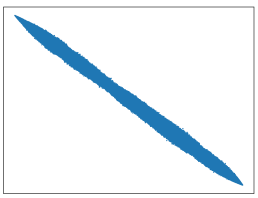
Parameter	Configuration
Processor model	Intel(R) Xeon(R) E5-2620 v2 @2.10 GHz
Number of cores	12 cores
Memory	64 G
OS	CentOS Linux release 7.6.1810 (Core)
Compiler	GNU Compiler Collection (GCC_v8.3.0)

5.1. Optimizing the Coefficient Matrix by Grid Renumbering

In order to demonstrate the effectiveness of CQ method proposed in this paper, we implemented the above two numbering algorithms in parallel. Therefore, the quality of the sparse matrices produced by the original algorithm exported from ICEM CFD using a default setting, renumbered by the greedy algorithm, and the CQ algorithm are analyzed and compared respectively.

The coefficient matrix based on the original and the renumbered grid assembly using YHACT is shown in Table 4. The first column indicates the different meshes sizes in thousands. The second to fourth columns indicate the sparse matrices of the grids without renumbering, processed by the Greedy algorithm, and processed by the CQ algorithm, respectively. The results showed that two renumbering algorithms improve the quality of the grid to a large extent. The non-zero elements are more concentrated near the diagonal after renumbering.

Table 4. The sparse matrices are processed by different numbering algorithms. Discrete points mark the distribution of non-zero elements in a matrix.

Quantity (k)	Original Grid	Greedy	CQ
270			
800			
1600			
2240			
4000			
8648			

To further quantitatively compare the renumbering results, we present evaluation data for the measures of maximum bandwidth, matrix profile, and MDMP. Moreover, in the solution environment of the CFD finite volume method, discriminate which metric can effectively filter the numbering quality of both algorithms. Table 5 demonstrates the maximum bandwidth, profile and MDMP of the test as mentioned above grids. The second row is the original grid test data. The third and fourth rows are the grid test data renumbered by Greedy and CQ algorithms. The sub rows then represent the measurement of the different metrics. The degree of improvement of the current data compared to the original grid data is indicated in parentheses. Traditional metrics lead to the conclusion that the CQ algorithm optimizes the maximum bandwidth and profile of the matrix considerably better than the greedy algorithm. It can be seen that the Greedy algorithm has a large improvement in the quality of the sparse matrix. However, the CQ algorithm minimized the bandwidth and

profile of the sparse matrix with a maximum reduction of 92.94% in bandwidth and 98.7% in profile. Therefore, in the context of the CFD finite volume method, the CQ algorithm is better to enhance the speed of solving CFD numerical simulations. However, as far as the MDMP metric is concerned, the quality of the matrix generated after Greedy numbering is better than that of the CQ algorithm. The two numbering algorithms are very highly optimized on MDMP. Because real world cases are generated directly by ICEM library after being discretized by a finite volume method. It does not have any modifications and therefore the results are very poor. However, it could be clearly seen that the MDMP metrics of Greedy and CQ algorithms were not of the same order of magnitude, and the CQ algorithm was of better quality. That is, when the effects of both renumbering algorithms are obvious, the MDMP metric also has the effect of amplifying the quality of the different algorithms. Therefore, this paper considers that Greedy is better to improve the speed of solving CFD numerical simulations. Next, we verify the correctness of the latter.

Table 5. Mesh quality comparison. Each algorithm measures the nature of the sparse matrix using three metrics, bandwidth, profile and MDMP, respectively. The round brackets indicate the degree of optimization of the current data relative to the original grid data.

		Quantity (k)					
Method	**Index**	**270**	**800**	**1600**	**2240**	**4000**	**8648**
Original	bandwidth	272,883	802,837	1,601,796	2,236,304	4,003,663	8,645,851
	profile	31,475,740,701	294,927,759,978	1,130,160,745,627	2,279,170,105,336	6,842,108,134,704	30,723,125,177,693
	MDMP	9131.96	45,719.74	90,351.71	133,808.299	238,231.64	409,337.80
Greedy	bandwidth	228,185	514,128	1,176,540	1,219,768	2,184,311	3,428,878
	(growth)	(16.38%)	(35.96%)	(26.55%)	(45.46%)	(45.44%)	(60.34%)
	profile	2,706,632,365	15,905,731,993	87,668,806,598	110,044,892,719	437,301,559,155	1,420,551,357,616
	(growth)	(91.4%)	(94.6%)	(92.24%)	(95.17%)	(93.6%)	(95.38%)
	MDMP	37.23	41.37	43.95	42.71	46.23	43.73
	(growth)	(99.59226%)	(99.90951%)	(99.95136%)	(99.96808%)	(99.98060%)	(99.98932%)
CQ	bandwidth	47,840	110,648	254,891	560,816	685,326	610,811
	(growth)	(82.47%)	(86.21%)	(84.09%)	(74.92%)	(82.88%)	(92.94%)
	profile	1,023,887,234	8,054,992,527	35,761,499,444	56,943,570,117	206,200,122,641	400,613,572,413
	(growth)	(96.75%)	(97.27%)	(96.84%)	(97.50%)	(96.99%)	(98.70%)
	MDMP	128.10	298.63	767.87	96.31	1070.11	2052.20
	(growth)	(98.59720%)	(99.34683%)	(99.15013%)	(99.92803%)	(99.55081%)	(99.49865%)

5.2. Reducing Parallel Simulation Time of a Real-World CFD Case

There is a consensus that reducing the profile and bandwidth of the matrix can improve the solving speed of the linear system. However, fluid simulations also involve matrix assembly and parallel communication on complex unstructured grids. As described in the previous section, the calculations in FVM-based simulations are directly related to the cell numbers.

Simple metrics about the linear systems can not provide accurate measurement for a complete numerical simulation. Therefore, we build a real-world CFD case on YHACT for quantitatively analyzing the algorithm and the framework. As shown in Figure 7, the flow through a fuel rod bundle (named FuelRodBundleFlow) with positioning lattice is simulated. This case comes from real-world scenarios in a nuclear reactor core. The numerical results contain the essential thermal-hydraulics characteristics in the fluid flow. It is crucial for nuclear core design and optimization.

The stirred wings on the positioning lattice can effectively mix the coolant in the core and reduce the fuel rod surface temperature. It is widely used in engineering practice, and its fluid computational domain and cross-sectional grid model are shown in Figure 7a,b. Figure 7c,d reflect the consistency of velocity and pressure in the z-direction for the case with multiple cores in parallel and single cores in serial. The data consistency of the case FuelRodBundleFlow with different computation modes and numbering methods were analyzed using *Tecplot*. Figure 7c represents the velocity distribution in the z-axis direction for the case in both serial and parallel computation modes. Different numbering methods are used for each computation mode. The velocity distribution of each slice is consistent

with an error of no more than 10^{-5}. Figure 7d shows the pressure distribution in the z-axis direction for different calculation modes and numbering methods. The pressure distribution is kept consistent for each slice with an error of no more than 10^{-3}. Therefore, the accuracy of the experimental results is guaranteed whether serial or parallel computation is used, or different renumbering methods are used for computation. This section mainly focuses on several cases with different grid sizes. This section focuses on testing cases with different sizes of fuel rod bundles. The questions left in the previous section are verified by the numerical simulation time consumption of the cases processed by two different numbering algorithms.

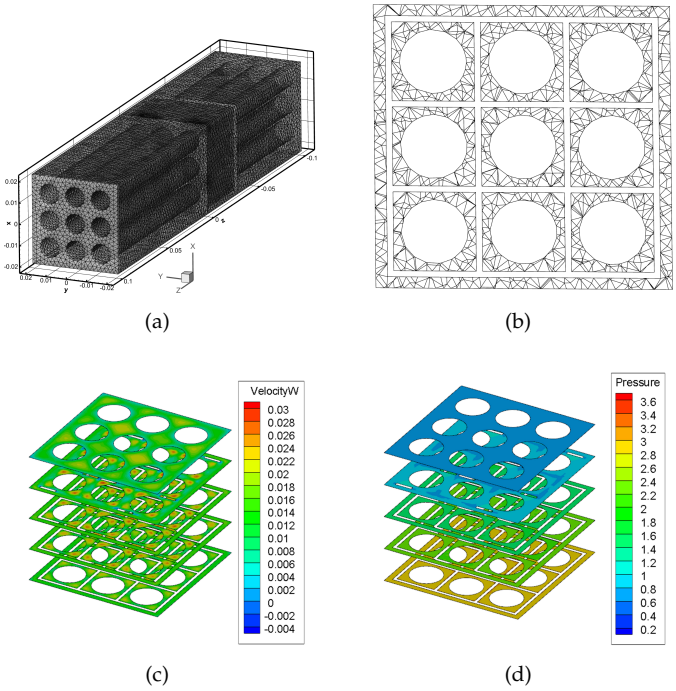


Figure 7. A real-world CFD case (named FuelRodBundleFlow): the fluid flow through a fuel rod bundle with a positioning lattice. (**a**) fluid computing domain; (**b**) sectional mesh model; (**c**) z-direction velocity contour diagram; (**d**) z-directional pressure contour diagram.

To analyze the correctness of the proposed sparse matrix discrimination metric and the scalability of the renumbering algorithm, we run the cases on different numbers of processors and grid sizes. As shown in Table 6, the first column indicates the different grid sizes. The second column indicates the number of parallel processes used for the different grid size cases. Columns 3 through 5 indicate the total time for 100 iterative steps for different grid sizes. It was clear that the CQ algorithm, which is superior in terms of warp bandwidth and profile discrimination, is slower than the Greedy algorithm in numerical simulations. This is clearly inconsistent with the traditional sparse matrix quality discrimination approach. However, it is well established by the average distance of the median points that the Greedy algorithm will outperform the CQ algorithm in the numerical simulations of the finite volume method of CFD. When a 2.24 million scale grid case is used, the numerical simulation will improve up to 38.19% with a degree of parallelism of 40 processes. In addition, it can be seen that the renumbering algorithm can still reduce the time consumption of the numerical simulation after reaching a specific grid size. However, this is not an infinite improvement, and the improvement is always within a certain threshold.

Table 6. Scalability of the grids renumbering algorithms on different processors and grid sizes of *FuelRodBundleFlow* case. Column 1 indicates the grid size. Column 2 indicates the number of parallel processes. Columns 3 to 5 indicate the total time to complete 100 iterative steps with different numbering methods. The time optimization rate after grid renumbering is shown in parentheses.

Grid Size	Core	Time		
		Original	Greedy	CQ
270,000	12	367.24 s	302.724 s (17.57%)	322.315 s (12.23%)
	5	436.46 s	397.684 s (8.88%)	401.355 s (8.04%)
800,000	40	389.29 s	306.8 s (21.19%)	320.583 s (17.65%)
	16	974.57 s	668.019 s (31.46%)	708.43 s (27.31%)
2,240,000	100	1229.56 s	1020.05 s (17.04%)	1146.17 s (6.78%)
	40	1943.6 s	1201.31 s (38.19%)	1597.39 s (17.81%)
4,000,000	192	1147.78 s	1020.49 s (11.1%)	1092.3 s (4.83%)
	72	1159.9 s	875.68 s (24.5%)	932.46 s (19.61%)
8,648,000	384	1997.66 s	1662.41 s (16.78%)	1757.01 s (12.05%)
	160	5012.54 s	4264.56 s (14.92%)	4611.35 s (8.0%)

5.3. Impact of the Renumbering Algorithm

This section focuses on the specific impact that the renumbering algorithm described above has on the numerical simulation calculations. In this section, the *Valgrind* third-party tool is used to simulate and test the access process in CFD calculations. *Valgrind* will simulate a large access space during the test; it will greatly increase the numerical simulation time by a factor of ten or more. Here, we select 270,000 grid, 800,000 grid and 2,240,000 grid (with parallelism of 12 cores, 16 cores and 40 cores, respectively) from *FuelRodBundleFlow* case for testing. The above three cases were iterated by *Valgrind* in 10 steps. The six standard test cases in YHACT are also used as test subjects, with grid sizes ranging from 2000 to 70,000. All examples can be found in the ANSYS FLUID DYNAMICS VERIFICATION MANUAL. The above six classic test cases were iterated by Valgrind in the same number of iteration steps. The feasibility of MDMP is further validated.

In YHACT, the original grid numbering of the six test cases is optimal. The three *FuelRodBundleFlow* cases are large real-world engineering cases whose original grid arrangement is chaotic. Therefore, by renumbering these two types of cases, this section further verifies whether the MDMP metrics can correctly reflect the relationship between different numbering algorithms and the overall numerical simulation efficiency. Table 7 shows that the MDMP increases for six test cases and decreases for three *FuelRodBundleFlow* cases after processing by Greedy and CQ algorithms. The corresponding CFD numerical simulation time and cache miss rate also change. Figure 8 visualizes the correspondence between the elapsed time and the cache miss rate for each case iteration of 100 steps. The iteration time and cache miss rate of the original mesh for the six YHACT test cases are the smallest. The iteration times and cache miss rates of the meshes of the three A cases processed by Greedy's algorithm are the smallest. The cache hit rate of CFD computation for all cases after CQ renumbering is lower than the grid after Greedy renumbering, and the computation time is longer. The above is consistent with the judgment of MDMP, that is, the smaller the MDMP of the cases, then the better the renumbering algorithm used. It can be concluded that, in the CFD numerical simulation by finite volume method, the MDMP of the sparse matrix can correctly reflect the cache hit rate during the computation, and thus judge the impact of different renumbering algorithms on the overall CFD numerical simu-

lation. It can be used as one of the pre-processing steps before the CFD mesh is involved in the solution. Therefore, MDMP of sparse matrices is a more appropriate indicator of the quality of sparse matrices than the traditional *bandwidth* and *profile*.

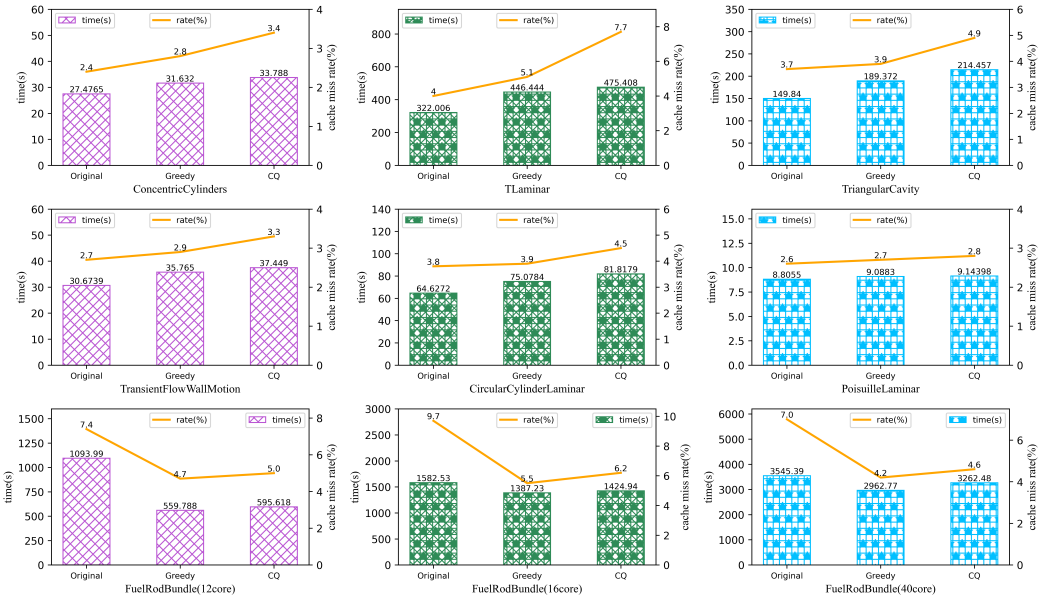


Figure 8. Correspondence between time consumed and cache miss rate for six test cases and three *FuelRodBundleFlow* cases. The line graph of each subplot indicates the cache miss rate, and the bar graph indicates the time consumed by the numerical simulation calculation.

Table 7. Case test. The third column indicates the numbering algorithm used for each case. The fifth and sixth columns indicate the time to compute the same number of steps and the corresponding cache miss rate. The first six rows are data from the ANSYS FLUID DYNAMICS VERIFICATION MANUAL, and the last three rows are data from the real-world *FuelRodBundleFlow* cases.

Case	Grid Size	Method	MDMP	Time	Cache Miss
ConcentricCylinders	7400	original	2.68	27.4765	2.40%
		Greedy	13.39	31.632	2.80%
		CQ	100.42	33.788	3.40%
TLaminar	70,000	original	6.83	322.006	4.00%
		Greedy	16.92	446.444	5.10%
		CQ	495.91	475.408	7.70%
TriangularCavity	32,000	original	2.08	149.84	3.70%
		Greedy	14.51	189.372	3.90%
		CQ	155.58	214.457	4.90%
TransientFlowWallMotion	13,000	original	1.00	30.6739	2.70%
		Greedy	12.46	35.765	2.90%
		CQ	52.28	37.449	3.30%
CircularCylinderLaminar	22,600	original	3.11	64.6272	3.80%
		Greedy	9.39	75.0784	3.90%
		CQ	55.18	81.8179	4.50%
PoisuilleLaminar	2000	original	1.00	8.8055	2.60%
		Greedy	6.05	9.0883	2.70%
		CQ	7.04	9.14398	2.80%

Table 7. *Cont.*

Case	Grid Size	Method	MDMP	Time	Cache Miss
FuelRodBundleFlow(12core)	270,000	original	9131.96	1093.99	7.40%
		Greedy	37.23	559.788	4.70%
		CQ	128.10	595.618	5.00%
FuelRodBundleFlow(16core)	800,000	original	45719.74	1582.53	9.70%
		Greedy	41.37	1387.23	5.50%
		CQ	298.63	1424.94	6.20%
FuelRodBundleFlow(40core)	2,240,000	original	133808.29	3545.39	7.10%
		Greedy	42.71	2962.77	4.20%
		CQ	96.31	3262.48	4.60%

6. Conclusions

Numerical simulation of fluid dynamics is one of the essential applications on supercomputers. Due to the enormous amount of computation, improving the parallel performance is critical for solving real-world CFD cases. In this paper, we refocus on the classic topic of grid numbering for FVM-based fluid simulations. In order to choose a better renumbering method to improve the efficiency of FVM-based CFD numerical simulations, this paper proposed an improved evaluation metric MDMP with sparse matrix properties. We also proposed a CQ renumbering algorithm that greatly optimizes the traditional evaluation metrics (bandwidth and profile). In order to compare and validate the proposed metrics and algorithms, we integrated the CQ and Greedy renumbering algorithms into a generic CFD software YHACT. We also conducted a quantitative study of different numbering algorithms using a real-world case derived from nuclear reactor engineering. Compared to the widely used Greedy algorithm, the CQ algorithm yielded a lower maximum bandwidth (up to 82.19% optimized compared to Greedy) but leads to a lower numerical simulation efficiency (up to 24.8% lower compared to Greedy). In addition, the MDMP of the Greedy algorithm is reduced by 83.43% compared to the CQ algorithm. The Results section verifies that the size of the MDMP is positively correlated with the size of the cache hit rate of the different renumbering algorithms during CFD computation. The results showed that using MDMP can more accurately discriminate the quality of a sparse matrix. In addition, after renumbering the cells exported from commercial grid-generating software, the parallel performance have been greatly improved for simulating hydraulics for a bundle of fuel rods in the nuclear reactor core. More importantly, the algorithms are implemented in a general CFD framework, which has good parallel scalability and could be extended seamlessly by new grid numbering algorithms.

This work can provide useful guidance for the design and optimization of an efficient general-purpose CFD software. In the future, we aim to build an accurate performance discrimination model and propose novel numbering algorithms with optimal performance based on this extensible framework for grids renumbering. We also hope that the approaches of this paper and the corresponding CFD software can be open-sourced, and more applications will be carried out in academia and industry shortly.

Author Contributions: Conceptualization, H.Z., X.-W.G. and J.L.; Data curation, H.Z., Q.L. and H.X.; Formal analysis, Q.L. and H.X.; Funding acquisition, X.-W.G., C.L. and J.L.; Methodology, H.Z.; Project administration, X.-W.G.; Resources, J.L.; Software, H.Z., X.-W.G. and C.L.; Supervision, J.L.; Validation, H.Z.; Visualization, H.Z. and X.-W.G.; Writing—original draft, H.Z.; Writing—review and editing, X.-W.G. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially supported by the National Natural Science Foundation of China (Grant Nos. 61902413, 12102468, and 12002380) and the National University of Defense Technology Foundation (Nos. ZK21-02 and ZK20-52).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Xu, C.; Deng, X.; Zhang, L.; Fang, J.; Wang, G.; Jiang, Y.; Cao, W.; Che, Y.; Wang, Y.; Wang, Z.; et al. Collaborating CPU and GPU for large-scale high-order CFD simulations with complex grids on the TianHe-1A supercomputer. *J. Comput. Phys.* **2014**, *278*, 275–297.
2. Wang, J.; Wan, D. Application Progress of Computational Fluid Dynamic Techniques for Complex Viscous Flows in Ship and Ocean Engineering. *J. Mar. Sci. Appl.* **2020**, *19*, 1–16. https://doi.org/10.1007/s11804-020-00124-8.
3. PI, J.S.; PM, A.K.; Alonso, J.; Darmofal, D.; Gropp, W.; Lurie, E.; Mavriplis, D. *CFD Vision 2030 Study: A Path to Revolutionary Computational Aerosciences*; NASA: Washington, DC, USA, 2013.
4. Long, S.; Fan, X.; Li, C.; Liu, Y.; Fan, S.; Guo, X.W.; Yang, C. VecDualSPHysics: A vectorized implementation of Smoothed Particle Hydrodynamics method for simulating fluid flows on multi-core processors. *J. Comput. Phys.* **2022**, *463*, 111234. https://doi.org/10.1016/j.jcp.2022.111234.
5. Guo, X.W.; Li, C.; Li, W.; Cao, Y.; Liu, Y.; Zhao, R.; Zhang, S.; Yang, C. Improving performance for simulating complex fluids on massively parallel computers by component loop-unrolling and communication hiding. In Proceedings of the 2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Yanuca Island, Fiji, 14–16 December 2020. https://doi.org/10.1109/HPCC-SmartCity-DSS50907.2020.00017.
6. Reid, J.K.; Scott, J.A. Reducing the Total Bandwidth of a Sparse Unsymmetric Matrix. *SIAM J. Matrix Anal. Appl.* **2006**, *28*, 805–821. https://doi.org/10.1137/050629938.
7. Hager.; W., W. Minimizing the Profile of a Symmetric Matrix. *SIAM J. Sci. Comput.* **2002**, *23*, 1799–1816. https://doi.org/10.1137/S1064827500379215.
8. Darwish, M.; Moukalled, F. *The Finite Volume Method in Computational Fluid Dynamics: An Advanced Introduction with OpenFOAM® and Matlab®*; Springer: Berlin/Heidelberg, Germany, 2021.
9. Gibbs, N.E.; Poole, W.G., Jr.; Stockmeyer, P.K. An algorithm for reducing the bandwidth and profile of a sparse matrix. *SIAM J. Numer. Anal.* **1976**, *13*, 236–250.
10. Cuthill, E.; McKee, J. Reducing the bandwidth of sparse symmetric matrices. In Proceedings of the 1969 24th National Conference, New York, NY, USA, 26–28 August 1969; pp. 157–172.
11. Sloan, S. An algorithm for profile and wavefront reduction of sparse matrices. *Int. J. Numer. Methods Eng.* **1986**, *23*, 239–251.
12. Akhras, G.; Dhatt, G. An automatic node relabelling scheme for minimizing a matrix or network bandwidth. *Int. J. Numer. Methods Eng.* **1976**, *10*, 787–797.
13. George, A.; Liu, J.W. *Computer Solution of Large Sparse Positive Definite*; Prentice Hall Professional Technical Reference: Englewood Cliffs, NJ, USA, 1981.
14. de Oliveira, S.G.; Silva, L.M. An ant colony hyperheuristic approach for matrix bandwidth reduction. *Appl. Soft Comput.* **2020**, *94*, 106434.
15. Pop, P.; Matei, O.; Comes, C.A. Reducing the bandwidth of a sparse matrix with a genetic algorithm. *Optimization* **2014**, *63*, 1851–1876.
16. Gonzaga de Oliveira, S.L.; de Abreu, A.A.; Robaina, D.; Kischinhevsky, M. A new heuristic for bandwidth and profile reductions of matrices using a self-organizing map. In Proceedings of the International Conference on Computational Science and Its Applications, Beijing, China, 4–7 July 2016 Springer: Berlin/Heidelberg, Germany, 2016; pp. 54–70.
17. Blazek, J. *Computational Fluid Dynamics: Principles and Applications*; Butterworth-Heinemann: Oxford, UK, 2015.
18. Ferziger, J.H.; Perić, M.; Street, R.L. *Computational Methods for Fluid Dynamics*; Springer: Berlin/Heidelberg, Germany, 2002; Volume 3.
19. Versteeg, H.K.; Malalasekera, W. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*; Pearson Education: London, UK, 2007.
20. Zhang, B.; Wan, W.; Bi, L. Improvement on Simulation Methods of Fluid Transient Processes in Turbine Tailrace Tunnel. *J. Press. Vessel. Technol.* **2021**, *143*, 031403.
21. Mosedale, A.; Drikakis, D. Assessment of very high order of accuracy in implicit LES models. *J. Fluids Eng.* **2007**, *129*, 1497–1503.
22. Zou, S.; Yuan, X.F.; Yang, X.; Yi, W.; Xu, X. An integrated lattice Boltzmann and finite volume method for the simulation of viscoelastic fluid flows. *J. Non-Newton. Fluid Mech.* **2014**, *211*, 99–113.
23. Burgess, D.; Giles, M.B. Renumbering unstructured grids to improve the performance of codes on hierarchical memory machines. *Adv. Eng. Softw.* **1997**, *28*, 189–201.
24. Guo, X.; Li, C.; Liu, J.; Xu, C.; Gong, C.; Chen, L. A highly scalable general purpose CFD software architecture and its prototype implementation. *Comput. Eng. Sci.* **2020**, *42*, 2117–2124.
25. Farhat, C. A simple and efficient automatic FEM domain decomposer. *Comput. Struct.* **1988**, *28*, 579–602.
26. Bacon, D.F.; Graham, S.L.; Sharp, O.J. Compiler transformations for high-performance computing. *ACM Comput. Surv. (CSUR)* **1994**, *26*, 345–420.