IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 19, 2020, accepted April 25, 2020, date of publication April 28, 2020, date of current version May 7, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2991099

Strength Check of Aircraft Parts Based on Multi-GPU Clusters for Fast Calculation of Sparse Linear Equations

YUHUA ZHANG[1] AND BINXING HU[2]
[1]School of Computer Science and Technology, Baoji University of Arts and Sciences, Baoji 721016, China
[2]Aerospace System Engineering Shanghai, Shanghai 201109, China

Corresponding author: Binxing Hu (376898978@qq.com)

This work was supported in part by the Special Scientific Research Project of Shaanxi Provincial Education Department 'Research on Software Testing Service in Cloud Computing Environment'' under Grant 17JK0044, and in part by the Key Projects of School Level Scientific Research Plan of Baoji University of Arts and Sciences ''Research on Privacy Protection in Big Data Environment'' under Grant ZK2018097.

ABSTRACT In order to improve the cost-effectiveness ratio, the next-generation vehicle needs to meet the requirements of reuse, while adopting a lighter structural weight, so it is necessary to realize the strength calculation and condition monitoring of key components in the digital twin. Most of the current monitoring methods are based on the characteristics of various data acquisition systems, but they need the support of a large number of flight data. The disadvantages of the above strategy can be avoided by reducing the structure of aircraft components to a finite element model and quickly checking the key components in the health management system. In order to solve the problem of fast calculation of the finite element model of the key components of the aircraft, a parallel algorithm and framework of large-scale sparse matrix preprocessing conjugate gradient method based on CUDA(Compute Unified Device Architecture) technology is proposed in the multi GPU(Graphics Processing Unit) workstation cluster environment. Once the sparse matrix is too large to be processed in a single workstation, this paper discusses how to realize the optimized data segmentation in the distributed multi-GPU computing environment. For the problem of iterative solution of matrix preprocessing, two preprocessing strategies of matrix bandwidth reduction parallelization and incomplete Cholesky decomposition are proposed, and asynchronous task concurrency and load balancing strategies are designed on the architecture. The calculation of some examples in the standard sparse matrix database shows that the algorithm and architecture proposed in this paper have the ability to solve large-scale sparse matrix quickly and efficiently, and can complete the fast strength verification of vehicle components.

INDEX TERMS Sparse matrix, CUDA, RCM, PCG, architecture.

I. INTRODUCTION

With the gradual reduction of the weight ratio of the aircraft, and the transformation from a slender body to a composite body and a lifting body, the vibration of the aircraft during flight has an increasingly significant impact on its dynamic characteristics [1]. Due to the reduced stiffness of the aircraft, real-time calculation of the downlink structural strength data is of great significance to ensure flight safety under time-varying conditions [2]. Digital twin system is the basis for the system-level fault tolerance of aircraft. For the fault detection and instruction reconstruction that may occur in the flight process, the computer on the rocket cannot carry out complex calculation due to the limitation of hardware resources. The common method is to send the sensor data back to the ground, and make complex and accurate judgment through the workstation with stronger computing power of the ground system. At present, the spacecraft's health management system is mainly aimed at the two modules of heat protection and engine, and has achieved very good results. In terms of thermal protection, the bending and buckling of Integrated Thermal Protection System (ITPS) are usually considered [3]. The fatigue damage caused by temperature gradient is mainly considered for the possible failure of engine in operation. For the above two kinds of stress-strain problems, the finite element method is usually used to calculate [4]–[6]. It can

The associate editor coordinating the review of this manuscript and approving it for publication was Seok-Bum Ko.

be seen that the finite element modeling of key components and the rapid solution of corresponding statics equations are the focus of the aircraft health management system. In order to ensure the real-time performance of computation, parallel computing is bound to be adopted. With the development of computer technology, the number of transistors in GPU (graphics processing units) is increasing rapidly, which has realized the computing power far beyond that of multi-core CPUs. While the bottleneck of multi-core CPU can't be solved due to the temperature wall and power wall, GPU's computing power develops rapidly due to the continuous progress of semiconductor technology. At the same time, the concept of GPGPU (general purpose computing on graphics processing units) and the CUDA released by NVIDIA make it more convenient to use GPU for high-performance computing, and become an important branch in the field of high-performance computing. Programmers can take advantage of the C/C++ interface provided by NVDIA for parallel programming and achieve several times the performance of the CPU on some applications [7], [8].

Many scholars have done related research on the parallel calculation of element stiffness matrix [8]–[10]. At the same time, for the solution of sparse linear equations, GPU parallel solving algorithms based on acoustics [11], electromagnetics [12], and fluid mechanics [13] based on finite element or finite volume methods have also been applied. Basermann *et al.* [14] elaborated the implementation strategy of sparse matrix preprocessing conjugate gradient method on large-scale parallel machine in 1995, and discussed the data storage and segmentation problems in detail. In 2003, Bolz *et al.* [15] and others realized the solver design of conjugate gradient method on GPU, that is to say, the iterative solution of large-scale linear equation was realized by using image API before the general calculation of GPU. However, CUDA appeared in 2008, Bell and Garland [16] and others completed the algorithm design of sparse matrix vector multiplication (SpMV), which is the most important algorithm in the application of iterative method. Zaharovits *et al.* [17] designed a shared memory parallel conjugate gradient method solver for the topology optimization of the finite element method. Amorim *et al.* [18] proposed the Meshless Local Petrov Galerkin Method (MLPG) algorithm to assemble the element stiffness matrix in 2019, and compared the speed difference between the BICG method and related methods of the conjugate gradient method family. Previous research experience shows that if the dimensions of the sparse equations considered are small, using a GPU has no obvious advantage over the CPU. Due to the data transmission and instruction sending between the CPU and the GPU, running the direct method on the CPU is even faster than implementing the iterative method on the GPU in this scenario. When the matrix dimension is large, the direct solution takes a non-linear increase in time and requires too much CPU memory. At this time, the advantages of the GPU become obvious. In this situation, the iterative solver becomes the only choice. However, it is worth noting that in order to use GPU with many threads to work in parallel, the data must be stored in GPU memory. In order to ensure the validity of the results, the strength check calculations involved in this paper usually use double precision floating point arithmetic. Generally, the Tesla series GPU is used as a computing card, and its graphics memory ranges from 8G (P4) to 32G (V100). Compared to the fact that the workstation motherboard can hold hundreds of GB, the amount of graphics memory is not enough to accommodate all the data involved in large-scale computing. Therefore, in order to avoid memory overflow, the matrix is usually decomposed into several sub-matrices and transferred to a GPU for calculation in batches. Most previous studies on accelerated FEM or sparse linear solvers have focused on algorithms designed on a single GPU, without considering the large-scale calculations where the dimensions of the matrix exceed the graphics memory. The computing environment of this paper is two workstation clusters with multi-core CPU and dual GPU, which is regarded as a hybrid Symmetrical Multi-Processing (SMP)/ distributed memory system. Because the distributed characteristics of GPU computing are similar to cluster computing, but there are significant differences in computing environment, so we need to design different communication, distribution and synchronization strategies for the two architectures.

This paper discusses the problem of sparse linear algebra solution based on GPU accelerated finite element model of aircraft. Based on the two most important libraries released by NVIDIA, cuBLAS [19] and cuSPARSE [20], focusing on the most time-consuming iterative solution steps of finite element matrix, including the overall analysis of bandwidth optimization and iterative solution of large sparse matrix. Since the conjugate gradient method has been studied deeply in the related literature, the bandwidth optimization and data segmentation based on GPU are described in detail here. The bandwidth optimization of sparse matrix can not only reduce the number of iterations of conjugate gradient method to reduce the computation, but also decrease the block of sparse matrix, so as to minimize the communication between computing nodes.

II. DESIGN OF PARALLEL EIGENVALUE SOLVING ALGORITHM FOR LARGE SPARSE MATRIX

Both the analysis of aero-elastic coupling of aircraft and the calculation of yield strength after component damage are essentially eigenvalue problems of linear equations:

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{1}$$

$\boldsymbol{A} \in \mathrm{R}^{\mathrm{nxn}}, \boldsymbol{b} \in \mathrm{R}^{\mathrm{n}}$, $\boldsymbol{x}$ is the position vector. The solution $\boldsymbol{x}^{(1)}$ can be obtained by the equivalent equation $\boldsymbol{M}\boldsymbol{x} = \boldsymbol{b}$, where $\boldsymbol{M}$ is the approximation of $\boldsymbol{A}$. It can be deduced that the error $\Delta\boldsymbol{x}$ between the above formula and the original equation is:

$$\boldsymbol{A}(\boldsymbol{x}_* - \boldsymbol{x}^{(1)}) = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(1)} \tag{2}$$

Since the direct method is difficult to solve, it can be converted into approximate equation:

$$\boldsymbol{M}\Delta\boldsymbol{x} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(1)} \tag{3}$$

An approximate correction amount $\overline{\Delta}x$ is obtained, so the corrected approximate solution is:

$$\boldsymbol{x}^{(2)} = \boldsymbol{x}^{(1)} + \tilde{\Delta}\boldsymbol{x} = \boldsymbol{x}^{(1)} + \boldsymbol{M}^{-1}(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(1)}) \tag{4}$$

Repeat the above steps if the accuracy requirements are not met. This gives the general formula for the iterative method:

$$\boldsymbol{x}^{(k)} = \boldsymbol{x}^{(k-1)} + \boldsymbol{M}^{-1}(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(k-1)}), \quad k = 2, 3, \ldots \tag{5}$$

If $\boldsymbol{M} = \boldsymbol{I}$, the famous Richardson iteration will be obtained after standard splitting:

$$\boldsymbol{x}^{(k)} = \boldsymbol{b} + (\boldsymbol{I} - \boldsymbol{A})\,\boldsymbol{x}^{(k-1)} = \boldsymbol{x}^{(k-1)} + \boldsymbol{r}^{(k-1)} \tag{6}$$

where $\boldsymbol{r}^{(k-1)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(k-1)}$ is the residual of the previous step. As mentioned in the previous chapter, in large-scale numerical simulation calculation, the solution of sparse linear system occupies a considerable part of calculation time and resources, so higher requirements are imposed on the solution speed of sparse linear system. The following instruction will be divided into two parts to describe the parallel design of RCM algorithm. The first part is the introduction of RCM algorithm and pseudo peripheral vertex. The second part is the parallel design of CUDA algorithm based on linear algebra.

A. DESIGN OF PARALLEL BANDWIDTH OPTIMIZATION ALGORITHM FOR LARGE SPARSE MATRIX

Section II.A is divided into two parts in the revised manuscript. The first part is the introduction of the RCM algorithm and the calculation process of pseudo-peripheral vertex. The second part is the design of CUDA parallel algorithm based on linear algebra.

1) DEFINITIONS AND NOTATIONS

In the stress and strain analysis of the refined analysis and flight simulation of large spacecraft, the number of elements and nodes is constantly changing due to the fuel consumption and the separation of each component, so the equations need to be renumbered and reorganized for calculation [21]. As far as the strength check calculation is concerned, the linear algebra calculation needs to be completed every few hundred milliseconds, so the timeliness of calculation is very high. Bandwidth reduction can significantly reduce the amount of calculation, enhance the stability of the calculation, and speed up the iterative method [22]. Related research is done on parallel computing of bandwidth optimization problems. For the finite element model of the aircraft, the stiffness matrix is a symmetric matrix, and the bandwidth reduction is to find the displacement matrix $\boldsymbol{P}^{\mathrm{T}}$, which makes $\boldsymbol{P}\boldsymbol{A}\boldsymbol{P}^{\mathrm{T}}$ bandwidth smaller. At present, the essential algorithms include Cuthill-McKee (CM) [23], Reverse Cuthill-McKee (RCM) [24], Approximate Minimum Degree (AMD) [25], [26] etc.

Algorithm 1 RCM Algorithm

1. Initialize the empty queue $\boldsymbol{r}$, $v_i \leftarrow r$
2. Do $i \in [1, n]$
3. Find all unidentified neighbors of the vertex v_i
4. The remaining vertex marked in increasing order of degree
5. End Do
6. The RCM ordering is obtained, and the resulting queue is a feasible solution for bandwidth reduction

As a variant of the CM algorithm, the RCM algorithm is considered to be one of the most promising low-cost heuristics for reducing bandwidth. RCM algorithm transforms matrix into adjacency matrix and node undirected graph from graph theory. The pseudo-vertex is used as the root node, the vertices in the graph are labeled in increasing order, and the vertices with the same distance from the pseudo-vertex are divided into one layer. The structural nodes are divided into several layers to form a tree structure, as shown in Algorithm 1. Previously, Oliveira and Abreu [27] compared 7 search methods for pseudo-peripheral vertices, and verified by examples that George and Liu [28] algorithm is more suitable for symmetric sparse matrices. Nascimento Rodrigues *et al.* [29] realized the RCM algorithm of sparse matrix reordering by using OpenMP technology, and obtained up to 50% acceleration on the small matrix of millisecond time scale. Azad *et al.* [24] and Nascimento Rodrigues *et al.* [29] has done related work on distributed computing of RCM algorithms in the MPI-OpenMP environment. This section draws on previous research results and designs RCM algorithm applicable to heterogeneous computing platforms in combination with CUDA.

The dimension of symmetric matrix $\boldsymbol{A}$ is n, and $f_i(\boldsymbol{A})$ is the first non-zero element in row i, which is:

$$f_i(\boldsymbol{A}) = \min\{j | a_{ij} \neq 0\}, \quad 0 < i,\ j < n - 1 \tag{7}$$

Due to the symmetry of matrix $\boldsymbol{A}$, the bandwidth of each row is $\beta_i(\boldsymbol{A}) = i - f_i(\boldsymbol{A})$, so the matrix bandwidth can be expressed as a formula:

$$\beta_i(\boldsymbol{A}) = i - f_i(\boldsymbol{A}) \tag{8}$$

The set of all non-zero elements is:

$$Env(\boldsymbol{A}) = \{\{i, j\} | 0 < j - i < \beta_i(A), 0 < i, j < n - 1\} \tag{9}$$

The undirected graph of $\boldsymbol{A}$ is expressed as $G(\boldsymbol{A}) = (\boldsymbol{V}, \boldsymbol{E})$, where $\boldsymbol{V}$ is the set of vertices, $\boldsymbol{E}$ is the set of edges. Denote the number of vertices by $n = |\boldsymbol{V}|$, $m = |\boldsymbol{E}|$ is the number of edges. Let $d(v, w)$ be the distance between any two vertices v and w, the farthest path between the two points in the graph is represented as $l(v) = \max\{d(v, w) | w \in \boldsymbol{V}\}$. The root hierarchy of the vertex $v \in \boldsymbol{V}$ that $\boldsymbol{V}$ satisfies the partition of $\mathcal{L}(v)$:

$$\mathcal{L}(v) = \{L_0(v), \quad L_1(v), \quad \ldots \quad L_{l(v)}(v)\} \tag{10}$$

where $L_0(v) = \{v\}$, $L_i(v) = Adj(L_{i-1}(v)) \setminus L_{i-2}(v)$ $i = 2, 3, \ldots l(v)$, so the length of $\mathcal{L}(v)$ equal to $l(v)$.

RCM repeatedly marks all adjacent nodes of the current vertex, as shown in Algorithm 1. It is essentially a breadth-first-search (BFS) algorithm, makes the tree deep enough to reduce the number of nodes in each layer in a disguised way, so as to reduce the bandwidth of the matrix [30], [31]. It is based on the CM method to sort the nodes in reverse order and take the smaller of the sum of the column heights in the positive order and the reverse order.

How to select pseudo-peripheral vertex is the focus of RCM algorithm. Previous experience shows that the first vertex has a significant impact on the effectiveness of bandwidth optimization, and it is advisable to choose a vertex with the longest path and a deeper one. Because it costs a lot to find the most peripheral vertex, so there is a so-called pseudo peripheral vertex instead: the longest path depth of the pseudo peripheral vertex should be as close to the diameter of the graph as possible. Usually, any vertex is found as the root node in the vertex set $\boldsymbol{V}$, and the corresponding tree structure is calculated to obtain the deepest vertex. This vertex is selected as the root node and the tree graph is reconstructed until the maximum depth no longer changes, which is used as the convergence judgment condition. The whole process is as Algorithm 2.

Algorithm 2 Pseudo Peripheral Vertex r Calculation Process

1. Select any vertex $r \in \boldsymbol{V}$
2. Get root level list $\mathcal{L}(r) = \{L_0(r), L_1(r), \ldots L_{l(r)}(r)\}$
3. Record the current number of layers nlvl$\leftarrow l(r) - 1$
4. Do Determine if the number of new layers is greater than the number of previous layers.
5. If the condition is met, assign a new record to nlvl
6. Select the minimum degree of freedom vertex v from $\mathcal{L}_{l(r)}(r)$
7. Calculate primary level list $\mathcal{L}(v)$, replace r by v
8. End Do

2) THE DESIGN OF CUDA PARALLEL ALGORITHM BASED ON LINEAR ALGEBRA

Combined with CPU-GPU architecture, the sorting algorithm and pseudo peripheral vertex index algorithm in RCM are redesigned, and the parallel design of RCM algorithm is completed by using the idea of linear algebra, as shown in Algorithm 3. Take the sparse matrix $\boldsymbol{A}$ and the pseudo-peripheral vertex r as inputs, and return the dense matrix $\boldsymbol{R}$ to represent the ranking of the RCM algorithm. The first step is to initialize the element in $\boldsymbol{R}$ to -1 and modify it after accessing the ith vertex. $\mathcal{L}_{cur}$ and $\mathcal{L}_{next}$ are vertex sets at the current and next BFS levels respectively, where $\mathcal{L}_{cur}$ is called the BFS boundary of the current active vertex. The algorithm first marks the pseudo peripheral vertex $\boldsymbol{R}$ and inserts it into $\mathcal{L}_{cur}$.In the loop iteration, each time iterates through the nodes $\mathcal{L}_{next}$ that $\mathcal{L}_{cur}$ has not visited and marks the vertices in

Algorithm 3 RCM Algorithm Based on Linear Algebra

1. A vector that stores the order of all vertices, $\boldsymbol{R} \leftarrow -1$
2. Assign values to current and next level BFS layers $\mathcal{L}_{cur} \leftarrow \{r\}$,$\mathcal{L}_{next} \leftarrow \phi$
3. Initialize the current vertex count $\boldsymbol{R}[r] \leftarrow 0$, $nv \leftarrow 1$
4. While Judge whether the element in the current level is empty $\mathcal{L}_{cur} \neq \phi$ Do
5. Obtain the BFS boundary of the current active vertex
$$\mathcal{L}_{cur} \leftarrow SetDenV(\mathcal{L}_{cur}, \boldsymbol{R})$$
6. Access the adjacent units of the boundary layer
$$\mathcal{L}_{next} \leftarrow SpMV(\boldsymbol{A}, \mathcal{L}_{cur}, \boldsymbol{SR} = (\text{select2nd, min}))$$
7. Obtain the set of unreachable vertices
$$\mathcal{L}_{next} \leftarrow SelectSpV(\mathcal{L}_{next}, \boldsymbol{R}, \boldsymbol{R} = \text{-1})$$
8. Sort the vertices on the next boundary in ascending order to obtain the sparse vector
$$R_{next} \leftarrow SortPerm(\mathcal{L}_{next}, \deg ree)$$
9. Update vertex $R_{next} \leftarrow R_{next} + nv$
10. Update total vertex count $nv \leftarrow nv + nnz(R_{next})$
11. Set new vertex order to be accessed
$$R \leftarrow SetDenV(\boldsymbol{R}, \boldsymbol{R}_{next})$$
12. Skip to the next level and go back to step 4
13. End While

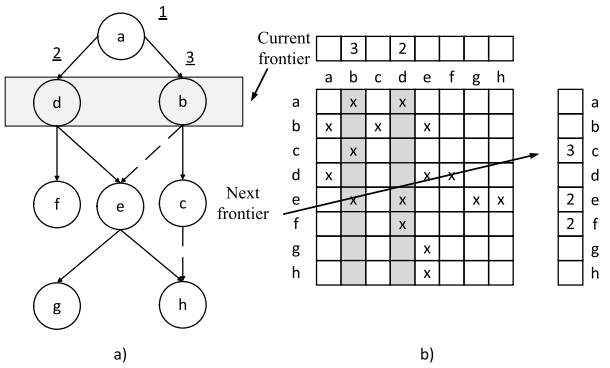

FIGURE 1. Matrix multiplication diagram based on (select2nd, min).

$\mathcal{L}_{next}$, and explores all the vertices in a layer-by-layer iterative manner until the boundary $\mathcal{L}_{cur}$ is empty.

At beginning of the algorithm loop, sparse vector $\mathcal{L}_{cur}$ is assigned as the vertex label. Then, $\mathcal{L}_{next}$ is obtained by multiplication of the current boundary layer $\mathcal{L}_{cur}$ and the adjacency matrix $\boldsymbol{A}$ within a radius(*select2nd*, *min*). The overloaded multiply operation *select2nd* passes the parent class label to the child class, and the overloaded min function ensures that each vertex in $\mathcal{L}_{next}$ becomes the child node of the smallest numbered vertex in $\mathcal{L}_{cur}$. FIGURE 1 is an example of how to mark the parent vertex number of the next level vertex in the form of *SpMV* (Sparse Matrix-Vector

Multiplication). FIGURE 1a) is a BFS tree with root node a, the current level is the set {b, d}, and the level of the parent node to be calculated is the set {c, e, f}. FIGURE 1b) shows the *SpMV* corresponding to the tree view, and the right is the output vector, and the value in the element represents the number of the parent vertex. *SpMV* traverses the sparse vector to get the value corresponding to the non-zero value in the gray column and writes the parent node number in this position. After the record is completed, (select2nd, min) function ensure that any vertex can find its parent vertex number and keep the minimum value. For example, On the right in FIGURE 1, the current layer node e has two parent nodes with numbers of 3 and 2 respectively. Function (select2nd, min) ensures that the corresponding position is recorded as 2 when output. Once finding the point of the next level, the previously marked vertex is removed from it for the next iteration. Due to the existence of *SpMV* step, compared with the traditional BFS algorithm, the RCM algorithm based on linear algebra hides its complexity and is smooth to parallelization.

After obtaining the next level vertex set through *SpMV* function, the previously marked vertices have been excluded from $\mathcal{L}_{next}$. The sorting of vertices in $\mathcal{L}_{next}$ is performed by the *SortPerm* function, which is based on the criteria of the parent vertex number and its own degrees of freedom. The vertices with the smaller parent vertex number are sorted first. Once the parent vertex numbers are the same, they are sorted according to their own positions. For example, in the figure above, the vertices e and f should be ranked first because of the parent vertex number, and they should be placed before c. And e is in front of f, so the sort of this layer after *SortPerm* function is {e, f, c}. This operation can adopt the idea of bucket sort, it can decompose the extremely time-consuming global sorting task into several small sorting tasks, so it is very easy to realize the parallel acceleration of CUDA when the matrix is large enough.

Buluc and madduri [30] and Azad *et al.* [24] and others pointed out in the relevant literature that *SpMV*(select2nd, min) and *SortPerm* are the most time-consuming two steps in the RCM algorithm, while the *SpMV* operation correlation library provided by CUDA has no relevant API function, so it needs to be specially designed for RCM.

The storage format of large sparse matrices has been introduced in the previous chapter. In view of the symmetry of the matrices generated by the finite element model, the CSR format and the CSC format are consistent, so the popular and easy to calculate CSR format is selected here. There are two ways of parallelization: in the calculation as shown in FIGURE 1, one method is to calculate the columns in the sparse vector for each thread. Due to the symmetry of the matrix, only the matrix data of the current row needs to be taken and put into the global output vector. The calculation steps of this method in each thread are relatively simple, but if each thread writes to the same position in the output vector, it will cause errors in the calculation results. Therefore, it can either be solved by atomic operation, or after each thread completes the calculation, an additional kernel function is called to achieve the calculation of the minimum value on each row (column). Another strategy is to calculate a row in matrix $\boldsymbol{A}$ for each thread. After obtaining the column index, judge whether the position is a non-zero number in the input sparse vector $\boldsymbol{x}$; if it is a non-zero number, call the *Min* function or the ternary number judgment once, that is, each thread gets an element of the output vector. The disadvantage of this strategy is that each thread needs to determine whether the corresponding position in the input sparse vector $\boldsymbol{x}$ is non-zero after getting an element in $\boldsymbol{A}$. In the unsatisfactory case, each thread needs to perform a *log2nnzA* judgment and address operation, which may cause bandwidth congestion due to the disordered access of each thread. However, because of the use of CSR format storage, the input sparse vector $\boldsymbol{x}$ is to be copied to each machine, and the adjacency matrix $\boldsymbol{A}$ can be split to each machine. After the output vector calculation of each computer point is completed, it can be summed up to one computing node for data combination, which is more suitable for Multi-machine and Multi-GPU architecture.

B. PARALLEL DESIGN OF PREPROCESSING CONJUGATE GRADIENT METHOD BASED ON INCOMPLETE CHOLESKY DECOMPOSITION

Incomplete decomposition is a more general preprocessing method, which is suitable for matrices with diagonally dominant [32]. The Cholesky decomposition is applied to the preprocessing equation of CG method. The standard preprocessing processes such as Algorithm 4. $\boldsymbol{M}$ is a nonsingular matrix of approximate $\boldsymbol{A}$, which is called preprocessing matrix. The addition of the matrix changes the spectral properties of the original matrix, reduces the condition, ensures the aggregation of eigenvalues, and improves the convergence speed of the iterative method.

If $\boldsymbol{M}=\boldsymbol{I}$, PCG method degenerates to CG method. When the residual vector $\boldsymbol{r}^{\boldsymbol{j+1}} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{\boldsymbol{j+1}}$ meets the termination criteria of CG method, the iteration can be terminated. From the basic steps, it can be known that each iteration of the PCG method consists of one-time solution of the linear equations (step 3), two-time inner products (steps 4, 6), one-time SpMV (step 6), three-time triplet operations (step 5, 7, 8), and two-time scalar division operations (steps 4,6). Compared with the CG method, there is one more operation for solving the system of equations, that is, step 3. If the preprocessing matrix $\boldsymbol{M}$ is not selected properly in this step, solving the extra equations is also time-consuming. Even if the number of iterations is reduced, but the time of each iteration of PCG method is much longer than that of CG method, it can not guarantee that the total solution time of PCG method is less than that of CG method. Therefore, at the beginning of the construction of M, it should be clearly realized that the use of this preprocessing technology is only meaningful if it can reduce the number of iteration steps.

$\boldsymbol{M}$ is constructed by the incomplete Cholesky decomposition of symmetric positive definite matrix $\boldsymbol{A}$, that is $\boldsymbol{A} \approx \boldsymbol{M} = \boldsymbol{L}\boldsymbol{L}^T$. However, the Cholesky decomposition of symmetric positive definite matrix $\boldsymbol{A}$ will introduce a large

number of non-zero elements, making the sparsity of $\boldsymbol{L}$ worse than that of $\boldsymbol{A}$. Under the premise of ensuring the symmetric positive definiteness of $\boldsymbol{L}^{-1}\boldsymbol{A}\boldsymbol{L}^{-\mathrm{T}}$, the sparsity of $\boldsymbol{L}$ is used to force the elements at some positions to be 0, so that there is a formula:

$$\boldsymbol{A} = \boldsymbol{M} + \boldsymbol{R} = \boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^{T} + \boldsymbol{R} \tag{11}$$

where $\boldsymbol{R}$ is the error matrix introduced by incomplete decomposition, $\boldsymbol{L}$ is the unit lower triangular matrix, and $\boldsymbol{D}$ is the diagonal matrix. Proper adjustment makes more zero elements in $\boldsymbol{R}$ to ensure that $\boldsymbol{L}\boldsymbol{L}^{\mathrm{T}}$ is close to $\boldsymbol{A}$. If $\boldsymbol{M}=\boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^{\mathrm{T}}$ is selected here, the corresponding pretreatment CG method is called Incomplete Cholesky decomposition conjugate gradient method, which is abbreviated as ICCG (Incomplete Cholesky conjugate gradient method).

$$\begin{aligned} \boldsymbol{L}\boldsymbol{y} &= \boldsymbol{r}^{\boldsymbol{j}} \\ \boldsymbol{L}^{T}\boldsymbol{z}^{\boldsymbol{j}} &= \boldsymbol{D}^{-1}\boldsymbol{y} = \boldsymbol{y}^{*} \end{aligned} \tag{12}$$

The incomplete decomposition of the vectorization operation is only calculated once and can be completed before iteration. In the dynamic analysis of elastic vehicle, $\boldsymbol{L}$ and $\boldsymbol{D}$ can be stored in memory according to the same storage format as $\boldsymbol{A}$. According to Algorithm 4, PCG parallel algorithm is mainly composed of SpMV, vector dot multiplication and vector arithmetic. The above three kinds of calculation can realize GPU acceleration by calling API functions [19], [20] such as *cusparseDcsrmv*, *cublasDdot* and *cublasDaxpy*. Considering that NVIDIA has provided a wealth of GPU internal parallel libraries, and the efficiency is usually higher than that of handwritten kernel functions, the memory level and basic algorithms of GPU are not introduced here. This paper mainly describes the design idea of the software from the perspective of the mixed parallel strategy within the node and the highly scalable multi-node communication architecture.

Algorithm 4 PCG Parallel Computing

1. Select $\boldsymbol{x}^{\boldsymbol{0}} \in R^{n}$ and calculate $\boldsymbol{p}^{\boldsymbol{0}} = \boldsymbol{r}^{\boldsymbol{0}} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{\boldsymbol{0}}\, j = 0$ $\beta_0 = 0$
2. Do $j = 0, 1, \ldots$ judge if convergence
3. Solving linear algebra $\boldsymbol{M}\boldsymbol{z}^{\boldsymbol{j}} = \boldsymbol{r}^{\boldsymbol{j}}$, Get residual vector $\boldsymbol{z}^{\boldsymbol{j}}$
4. Get the correction factor $\beta_j = \left(\boldsymbol{r}^{\boldsymbol{jC1}}, r^{jC1}\right) / \left(\boldsymbol{r}^{\boldsymbol{j}}, r^{j}\right)$
5. Get search direction vector for next iteration $\boldsymbol{p}^{\boldsymbol{jC1}} = \boldsymbol{z}^{\boldsymbol{jC1}} + \beta_j \boldsymbol{p}^{\boldsymbol{j}}$
6. Obtain the correction factor $\alpha_i = \left(\boldsymbol{r}^{\boldsymbol{j}}, \boldsymbol{z}^{\boldsymbol{j}}\right) / \left(\boldsymbol{p}^{\boldsymbol{j}}, A\boldsymbol{p}^{\boldsymbol{j}}\right)$
7. Get the next iteration vector $\boldsymbol{x}^{\boldsymbol{jC1}} = \boldsymbol{x}^{\boldsymbol{j}} + \boldsymbol{\alpha}_{\boldsymbol{j}}\boldsymbol{p}^{\boldsymbol{j}}$
8. Obtain the residual vector $\boldsymbol{r}^{\boldsymbol{jC1}} = \boldsymbol{r}^{\boldsymbol{j}} - \alpha_j \boldsymbol{A}\boldsymbol{p}^{\boldsymbol{j}}$
9. End Do

III. DESIGN AND IMPLEMENTATION OF MULTI-GPU WORKSTATION CLUSTER ARCHITECTURE

The research background is that the small-scale computing cluster completes the strength check of key components in a limited number of milliseconds, and ensures the task decision-making under the condition that the control system does not diverge. Therefore, from the perspective of computing scale, the current mainstream computing cluster frameworks such as Hadoop MapReduce and Spark are not suitable for real-time computing or millisecond lightweight cluster interaction required in this paper due to the problems of programming language and architecture volume. Based on Java, the mechanism of managing memory through GC (garbage collection) mechanism affects the calculation stability and efficiency [33]. Zhou *et al.* [34] and Kaur *et al.* [35] have proposed some improvement schemes for this framework, Wakde *et al.* [36], Hazarika *et al.* [37] *et al.* tested the efficiency differences between Spark and Hadoop with different data sets in their literature, the research shows that both of them are competent in traditional counting and iterative solving problems, but they are not suitable for the calculation scenario of spacecraft component strength check in millisecond level in this paper. The API of MPI is not flexible enough, and it is easy to cause the whole system to crash when a single node fails, which is unacceptable in the space launch mission. Its use of communicator to connect process groups may greatly increase the running time of the algorithm [38]. Pellegrini *et al.* [39] demonstrated that the Boost based design is more flexible and faster in transmission speed through experiments; Fukuoka *et al.* [40] indicate that MPI has performance bottlenecks in multi-threaded applications. In millisecond computing intensive computing, the efficiency of OpenMP is much lower than CUDA. Guo *et al.* [41] proved that CUDA has a higher acceleration ratio than OpenMP, and proved this view in the application of target tracking; Sivanandan *et al.* [42] compared and analyzed the application of MPI, OpenMP and CUDA in heat conduction calculation, and the results showed that the efficiency of OpenMP is far lower than CUDA under the condition of single machine parallel.

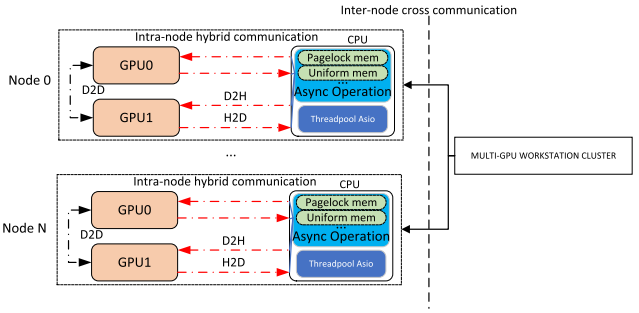

FIGURE 2. Hardware model of cluster communication.

Combined with the above literature conclusions, it is a reasonable choice to design an efficient, lightweight and cross platform cluster communication architecture combining CUDA, Boost ThreadPool and Asio. The computing solution of multi GPU workstation cluster is generally divided into three levels: inter node parallel, intra node parallel and GPU parallel [43], as shown in Figure 2. It can be seen from the topology structure that the intra-node communication

includes two parts, concurrent execution of CPU in the node and the parallel execution of GPU. In order to avoid data congestion, the Boost Asio based on threadpool is used in inter-node communication to complete the task of data transmission between nodes with a fixed number of threads. Based on the multi-GPU cluster framework, this section will elaborate the parallel algorithm flow on multiple GPUs, including pseudo-code and multi-GPU communication. The first part is the task allocation of multi GPU based on message bus, which describes the allocation model, data division and message bus design in detail; the second part is the asynchronous communication module based on Asio, which describes the connection process of the server and the topological relationship of each component.

A. MULTI-GPU TASK ALLOCATION OF THREAD POOL BASED ON MESSAGE BUS

It can be seen from the preamble of this section that, for the calculation intensive problem, the performance of code execution on GPU can be much higher than that of CPU by selecting appropriate parallel algorithm. At the same time, due to the limited number of memory and stream processors in a single GPU, now multi GPU cluster is used to achieve higher acceleration ratio [44]. In order to ensure the universality and extensibility, a message-driven component design inspiration is selected here, and the multithreading engine is realized by the threadpool to ensure the concurrent execution of tasks in different cores on the CPU. In the computing node shown in Figure 2, all GPUs are called by the engine module (class named *Engng*). In the sequence diagram shown in Figure 3, it can be seen that the software framework used in this paper is mainly composed of three categories: observer, engine and data communicator. The observer contains specific algorithm and is called by *Engng*. in the initialization stage, the observer registers messages with *Engng*, and after receiving the messages, *Engng* stores the messages in the message queue. *Engng* module, as a message bus, centrally manages the interaction between the computing class and the data communication class, and acts as an intermediary to send and receive messages, so as to reduce the coupling between algorithms and modules. This module is the key to asynchronous heterogeneous level parallel of CPU-GPU. In the initialization stage, it completes the creation of threadpool and the segmentation of calculation data required by each node. During the calculation, *Engng* finds and broadcasts the objects related to the message according to the message type. The data communicator class is responsible for managing data communication between computing nodes which called Inter-node cross communication. The data communicator that invokes this computing node contains Boost Asio, which is responsible for data interaction between nodes.

Once the observer completes the data division required by the calculation of each GPU in the node, it sends a message to inform that the operation has been completed, and *Engng* will put each corresponding function into the threadpool for asynchronous operation, and complete data reduction and message sending in the callback function. After the calculation in one computing node is completed, the data communicator calls the data in the observer and sends it to the master node asynchronously. The send completion message is referenced in the callback function to inform the engine that the data transfer step has been completed. The dotted line represents data input and the solid line represents data output.

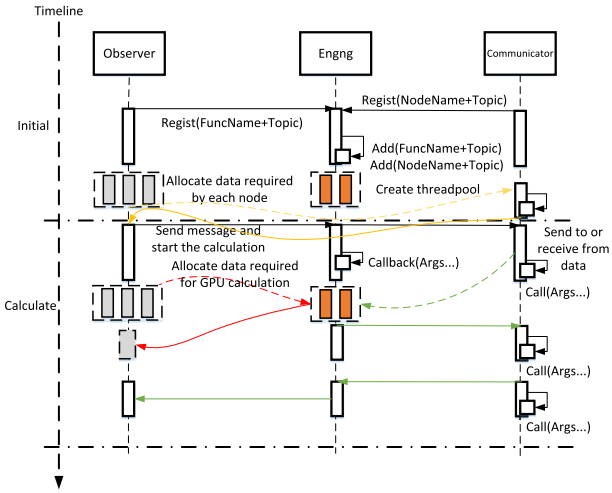

FIGURE 3. Sequence diagram of component calculation.

The most important steps in FIGURE 3 are task partitioning and data division on the CPU. Under the hardware configuration of 2-node and dual-GPUs per node, SpMV is taken as an example to illustrate the task assignment problem of multi GPUs based on asynchronous operation. After the bandwidth optimization in section 2.1, the sparse matrix has been changed into the following banded sparse matrix, which is expressed in the form of block matrix, as showed in FIGURE 4. Computing node 0 is responsible for calculating the first two lines, that is, the sub matrix A11-A23 is put to computing node 0 for calculation. Computing node 1 is responsible for the last two rows, which represents the task level parallelism between nodes. The two GPUs inside the node are respectively responsible for calculating any row to represent the parallel level within the node. When the matrix bandwidth is undersize, each GPU only needs to call the SpMV function once, as shown in FIGURE 4a). Further partitioning is needed when the matrix bandwidth is large, and the computation is carried out in the form of flow [45] to represent the internal parallel level of GPU, as shown in FIGURE 4b).

Note that the number of computing nodes is N, the current node is n, where $n < N$. the number of stream in each computing node is S_n, and the number of GPU in each compute node is i_n, where n<N. If SpMV is calculated in the n-th node, the S_n-th stream on the i-th GPU corresponds to the submatrix $A(\sum_{m=0}^{m<n} i_m + i, s)$, that is, the GPU will calculate all the data in the row where the current submatrix is located. The advantage of this scheme is that it is easy to implement the sum operation

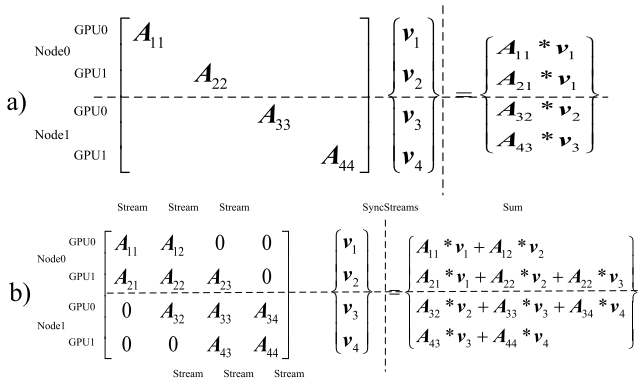

FIGURE 4. Data segmentation and task allocation in SpMV.

Algorithm 5 GPU Pseudo-Code Based on Multiple Streams

1. *Create task structure and contents*
2. *cudaHostAlloc && cudamalloc*
3. *For i=0:WorkloadNum*
4. *cudaMemcpyAsync(H2D)*
5. *kernel ≪< grid, block, sharedSize, stream ≫>(...)*
6. *put function into threadpool*
7. *cudaMemcpyAsync(D2H)*
8. *cudaStreamAddCallback(stream,callbackfunc, data,flag)*
9. End *For*
10. Sum the the submatrices of the corresponding rows

of the Thrust library in a GPU, but the load imbalance will occur when the bandwidth of the original matrix changes greatly, and the hardware utilization is not easy to guarantee. Due to the emergence of P2P technology [46], different GPU memory can be directly connected through the PCIE bus, so this paper applies the task structure and adopts a more efficient multi-stream strategy, as shown in Algorithm 5.

Taking the current node n has s submatrix as an example, it can be divided into s streams. The device number of the s-th stream is $s\%i_{\mathrm{n}}$, and the submatrix of the column number s/i_{n} is calculated. In the first step, the parameters of each device in the calculation node are obtained first, and the submatrix is divided according to the size of the matrix to form a number of task structures. The task structure includes the host input data pointer, the device data pointer, the stream number, device number used by the current stream, and position index of current block matrix in primitive matrix. After getting the length of the input data in the task structure, the address space is allocated in the lock page memory asynchronously, and the task is delivered to the threadpool. Step 5-8 complete the kernel function calculation according to the given flow and device number, and the callback function waits for the data to be returned. Step 10 evaluates the sum of the vector by row number through Thrust [47].

For the above strategies, the following mathematical model can be obtained. Assume a total of M streams in each time step and process the data divided into N blocks. In each stage, the calculation time for unit data is represented by t_i, the transmission time is represented by $t_{trans-i}$. If step l takes the most time in the calculation and recorded as t_l, the total execution time t_{total} is as follows. The value of j in the formula is $j \in [0, M] \cup \neq l$.

$$t_{\mathrm{total}} = \sum_{j=0}^{l-1} t_{trans-j} + N \cdot t_l + \sum_{j=l+1}^{M-l} t_{trans-j} \tag{13}$$

The expression of data processing capacity per unit time is as follows.

$$Data_{perStep} = \frac{N}{\sum_{j=0}^{l-1} t_{trans-j} + N \cdot t_l + \sum_{j=l+1}^{M-l} t_{trans-j}} \tag{14}$$

Pipeline delay is defined as the time when data transmission has been completed from the initial stage to the final stage:

$$delay = \sum_{0}^{M-2} t_{\mathrm{trans}-j} \tag{15}$$

Assuming that the operation time of each flow is the same, expressed as t. The previous equation can be simplified as follows.

$$t_{total} = N \cdot t_l + \sum_{j=1}^{M-1} t_{trans-j} = (N + M - 1) \cdot t \tag{16}$$

$$Data_{\mathrm{perStep}} = \frac{N}{(N + M - 1) \cdot t} \tag{17}$$

$$delay = (M - 1) \cdot t \tag{18}$$

A typical example of linear stream is the pipelining problem. It is proved that the pipeline scheduling problem is not suitable for the application of stream, and that the stream should try to deal with the algorithm that the duration of each stage is similar and the data is far longer than the pipeline length. In reference [48], the slender elastic vehicle is simplified as beam element to solve the vibration problem, it is decoupled into an independent calculation process in three directions, realizing the time overlap of memory replication and kernel function execution, as shown in FIGURE 5.

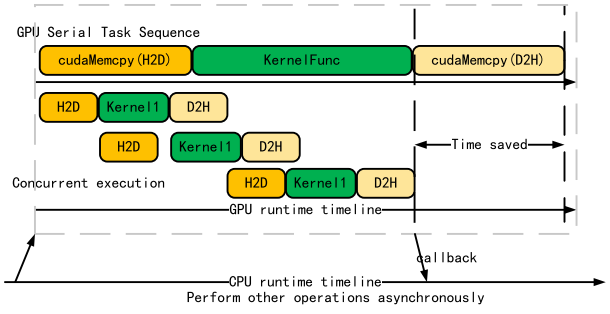

FIGURE 5. The difference of computation time between serial and asynchronous stream.

While the first stream executes the kernel function, the second stream passes data into the GPU. Once the first stream

returns data to the host, the second stream starts executing the kernel function and waits for another stream to return after the completion of the third stream of data into the operation. Compared with the serial execution in GPU, the overall performance can be further improved. Since the Kepler architecture, HyperQ technology makes it possible to run multiple CPU cores or threads to start tasks on a GPU, which improves the utilization of GPU in the form of multiple work queues and avoids pseudo dependence.

B. CLUSTER COMMUNICATION DESIGN AND OPTIMIZATION BASED ON BOOST.ASIO

In order to ensure the reliability and future expansibility of the strength checking calculation framework in the rocket flight mission, the communication module of the software framework in this paper should have the following characteristics. The data communication thread in the node should not affect the stability of the main thread, and the real-time computation is not affected by the communication block, so as to guarantee the computational stability of the node itself. In order to ensure the subsequent larger component strength check, it should have strong expansibility. Due to the need to communicate with multiple nodes and other file servers, should have a high communication efficiency. To meet the above requirements, this paper combines Boost.Asio with Boost.Threadpool. Asio is a cross-platform library that can be used on most operating systems and can support thousands of concurrent connections simultaneously. It provides a set of API that can support TCP socket, UDP socket and IMCP socket. If necessary, it can be extended at any time to support the required protocol.

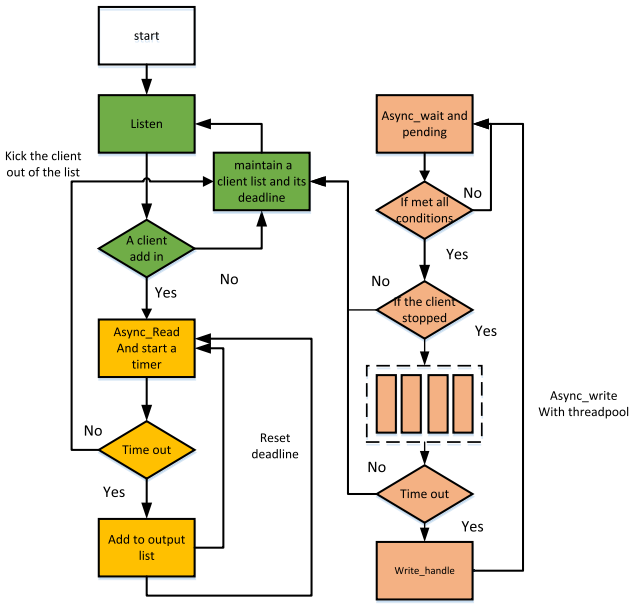

FIGURE 6. Flow chart of communication module.

In the communication module flowchart shown in Figure 6, the server maintains a message bus, which contains the

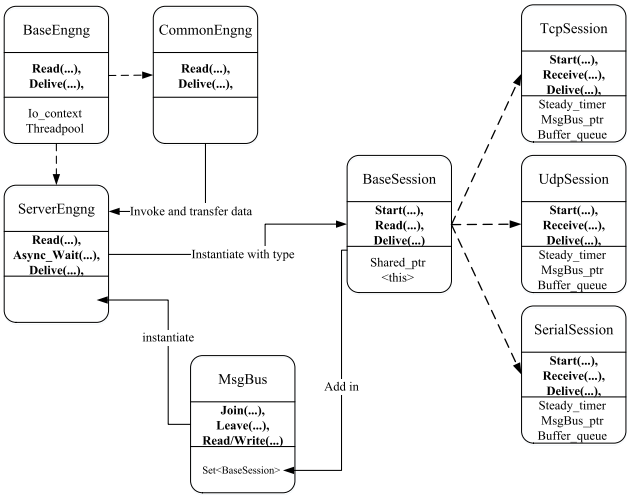

FIGURE 7. Invocation relationships and topology diagrams between components.

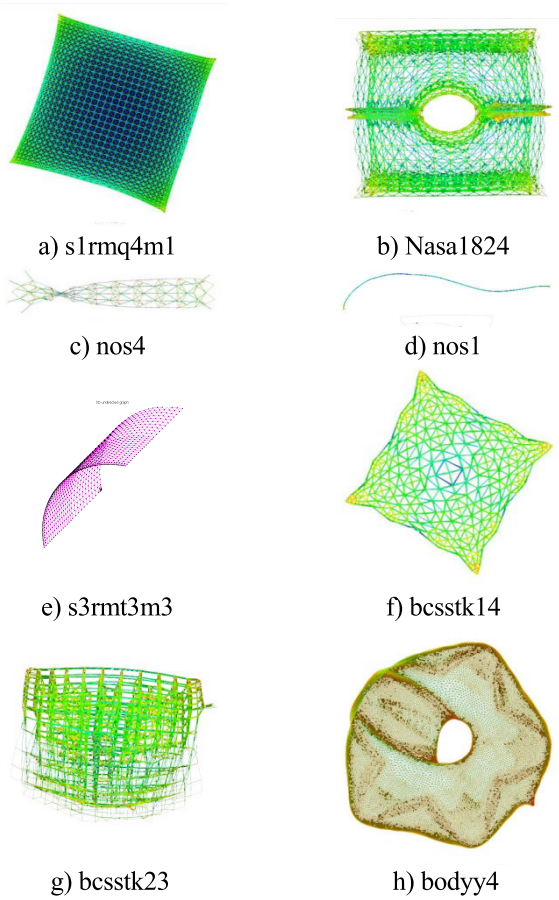

FIGURE 8. Grid structure of partial test cases.

shared_ptr of all TCP/UDP/Serial instances. The server listens asynchronously to a port, keeps listening after connecting to a client, and creates the session class to send and receive data. When other components send a notification to call the function *Read()*, and at the same time call the

TABLE 1. Matrix market.

Matrix number and name	The row number	A non-zero value	Symmetry/ positivity	Condition number
C3D8	122208	10641602	Symmetry/ positivity	6.73e10
2D3N	8076	430909	Symmetry/ positivity	6.1e6
bcsstk03	112	640	Symmetry/ positivity	6.7913e6
bcsstk14	1806	63454	Symmetry/ positivity	1.3e10
bcsstk23	3134	45178	Symmetry/ positivity	2.645e12
bcsstm34	588	24270	Symmetric / nonpositive definite	1.133e6
bodyy4	19366	134208	Symmetry/ positivity	8.05e2
nasa1824	1824	39208	Symmetry/ positivity	1.89e6
nasa4704	4704	104756	Symmetric / nonpositive	Inf
NOS1	237	1017	Symmetry/ positivity	1.9915e7
NOS4	100	594	Symmetry/ positivity	1.578e3
s1rmq4m1	5489	262411	Symmetry/ positivity	1.81e6
s3rmt3m3	5357	207123	Symmetry/ positivity	2.4e10
sts4098	4098	72356	Symmetry/ positivity	2.17e8
nd24k	72000	2.87e7	Symmetry/ positivity	\

TABLE 2. The influence of bandwidth reduction on the bandwidth and solution time of linear equations.

Matrix	original parameter		RCM		approximates the minimum degree of freedom	
	Band width	elapsed	Band width	total elapsed	Band width	total elapsed
bcsstk 03	7	0.3992	3	0.056	5	0.17
bcsstk 14	161	146.85 7	199	34.042	1712	123.302
bcsstm 34	410	87.489	64	29.195	1781	12.155
bcsstk 23	449	1688.8 5	410	367.872	3113	220.143
bodyy 4	16818	836.34 2	248	826.399	17463	128.93
2D3N	7614	527.98 1	223	365.211	7970	79.863
s3rmt3 m3	5302	2603.4 39	212	122.472	5325	98.76
nd24k	3323	1096.9 86	1418	630.809	4096	28.238

run() function of *io_context* to start running the program. The deadline of asynchronous read operation is set as a time step, that is, when the data cannot be accepted in a time step, the callback function will close the communication class instance and delete the client from list. If the receive operation does not timeout, the relevant operation will be carried out, and the *io_context* will be suspended waiting for the next message reading. The logic of write operation is similar to that of read operation, but when calling asynchronous write

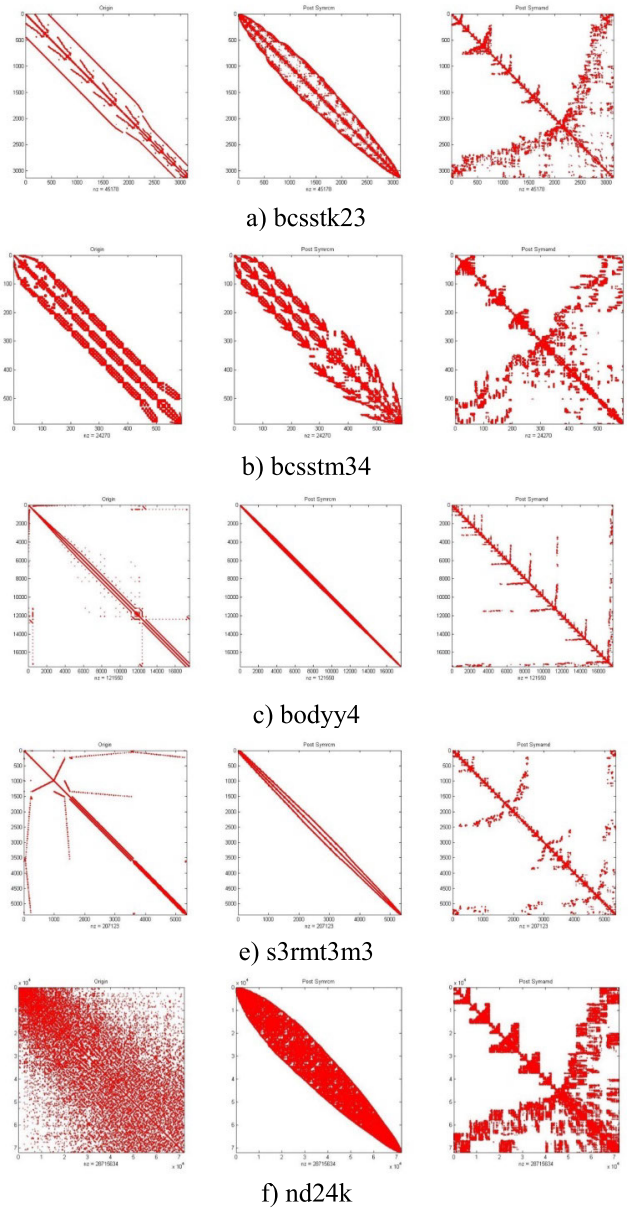

FIGURE 9. Comparison of the distribution of reduced matrix elements in the test set.

function, it needs to consider whether the output queue has been cleared. If not, it is necessary to send the original data before the data transmission.

A basic message framework is given here, as shown in FIGURE 7. Where the dashed lines represent derived relationships and the solid lines represent associations such as invocation or instantiation. It can be seen from the figure that *BaseEngng* and *BaseSession* are both atomic components that derive different types of entity components. *BaseEngng* derived the communication classes *ServerEngng* and *CommonEngng*, and maintained their own *io_context* instance and threadpool instance. *BaseSession* derive various instances of communication classes and realize polymorphism by calling the same function interface. *CommonEngng* is responsible for

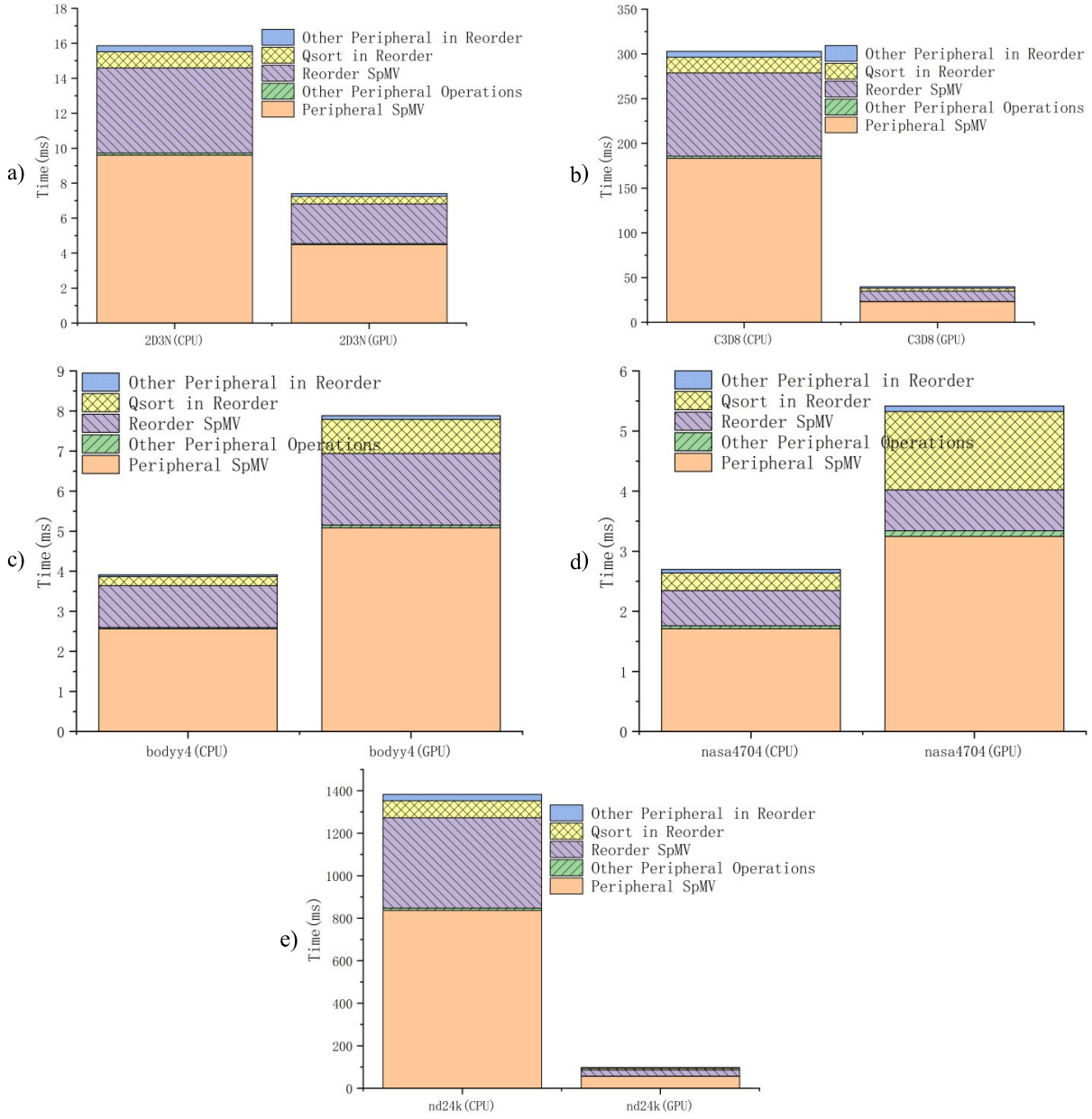

FIGURE 10. Comparison of CPU/GPU time consumption for some test cases.

managing all message buses, while *ServerEngng* maintains a single messagebus for the communication class to manage all instances derived from *BaseSession*.

In the algorithm design, the calculation of element matrix and the parallel solution of linear equations are a series of steps repeated on a data set. When cluster computing is used, the data should be separated along one or more dimensions and distributed to different computing nodes. If there is no dependency between the data, it is a simple problem of data segmentation, which can maximize the potential speedup ratio. Unfortunately, there is data dependency among the calculation nodes when solving the linear equations, and the data stored among the nodes needs to be exchanged in each calculation step. Referring to the existing TCP/IP network characteristics and GPU video memory bandwidth characteristics, the implementation platform is homogeneous and has a single port all two-way communication connection, and the problem parameters are described as follows:

N is the number of cells in each dimension.

k is the number of cells in each group along the decomposition density.

P is the number of calculation nodes.

t_{comp} is the cost of one update in each cell.

t_{comm} is the communication cost between two computing nodes.

t_{start} is delay caused by communication start.

Assuming that a total of $P = N/k$ groups are mapped to this node in the one-dimensional decomposition, and N is divisible by k, the execution and communication costs of a single node at each time step are:

$$comp1D = \frac{N^*}{k} N^* t_{comp} = \frac{N^2}{k} t_{comp} \tag{19}$$

$$comm1D = 2\left(t_{start} + t^*_{comm} N\right) \tag{20}$$

In two-dimensional decomposition, each node processes k^2 cells, $P = N^2/k^2$. The execution and communication costs of a single node in each time step are as follows:

$$comp2D = k^{2*} t_{conp} = \frac{N^2}{P} t_{comp} \tag{21}$$

$$comm2D = 4\left(t_{start} + t^*_{comm} k\right) = 4\left(t_{start} + t^*_{comm} \frac{N}{\sqrt{P}}\right) \tag{22}$$

Similarly, we can deduce the execution and communication cost of each node processing k^3 cells under three-dimensional decomposition

$$comp3D = k^{3*} t_{comp} = \frac{N^3}{P} t_{comp} \tag{23}$$

$$comm3D = 6\left(t_{start} + t^*_{comm} k^2\right) \tag{24}$$

According to the relationship between each dimension calculation and communication cost, the optimal threshold value of each dimension decomposition can be derived:

$$\begin{aligned}
&comm1D + comp1D < comm2D + comp2D \\
&\Rightarrow t^*_{comm} N\left(1 - \frac{2}{\sqrt{P}}\right) < t_{start} \\
&comm2D + comp2D < comm3D + comp3D \\
&\Rightarrow 4\left(t_{start} + t^*_{comm} \frac{N}{\sqrt{P}}\right) + \frac{N^2}{P} t_{comp} \\
&< 6\left(t_{start} + t^*_{comm} k^2\right) + \frac{N^3}{P} t_{comp} \\
&\Rightarrow \left[t^*_{comm}\left(4\frac{N}{\sqrt{P}} - 6^* k^2\right) - 2t_{start}\right] \\
&< \frac{N^2}{P}(N-1)\, t_{comp}
\end{aligned}$$

At the beginning of calculation, N, P and K are known, so when the time of communication and calculation is taken, the optimal geometric decomposition method can be obtained.

IV. NUMERICAL EXAMPLES ANALYSIS

A. TEST PLATFORM AND TEST SUITE

The purpose of this article is to realize the rapid calculation of the strength of aircraft structural components, and to provide a reference for the decision of subsequent flight missions and the reconstruction of guidance instructions. For the components that affect the flight attitude, the digital twin system should ensure that the aircraft system does not diverge during the calculation time of strength check and the subsequent decision-making time. The guidance and control system of aircraft may diverge in several periods. In order to ensure sufficient time for subsequent guidance reconstruction and mission decisions, a guidance cycle is selected here, that is, 25ms as the real-time judgment criterion.

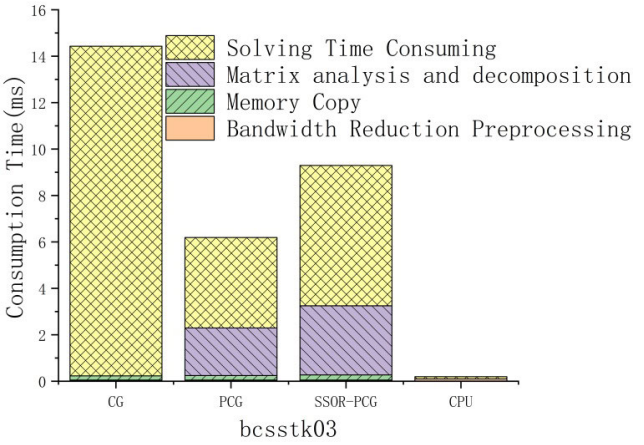

FIGURE 11. Comparison of time consumption of different algorithms for bcsstk03.

The experimental platform is a computing cluster composed of 2 nodes, each of which includes Intel E3 1230 V5 with 3.4G Hz and two GeForce GTX1060 graphics cards. Due to funding constraints, Tesla GPUs with higher double-precision computing capabilities will be used in future tests. In order to illustrate the impact of bandwidth reduction on the speed of solving sparse linear equations and ensure generality, two components (C3D8 and 2D3N) in engineering applications and part of sparse matrices in sparse matrix library [50] are selected as test sets to verify the parallel algorithm of RCM and the parallel algorithm of linear equations in the next section. The selected test matrix attributes are shown in TABLE 1. C3D8 represents a model consisting of a three-dimensional hexahedron and eight-node unit, which is a load-bearing component in the body. The simplified heat-preventing component 2D3N is composed of a two-dimensional three-node element. It should be noted that neither of the above two types of units can accurately simulate non-linear problems. However, the main studio of this article completes the real-time strength check based on CUDA acceleration, focusing on completing the algorithm design and performance testing first. The initial work of the two models, from element division to total stiffness matrix assembly, is completed in ABAQUS, and output through .mtx file.

The above test cases include thin shell and truss structure of slender aircraft, simplified slender beam, truss structure applicable to complex aircraft interior, and thin plate structure similar to simplified wing, etc. Some of the typical structures are shown in FIGURE 8.

B. TEST RESULTS AND ANALYSIS

The experiment first validates the parallel RCM algorithm based on linear algebra, which shows the fastness of

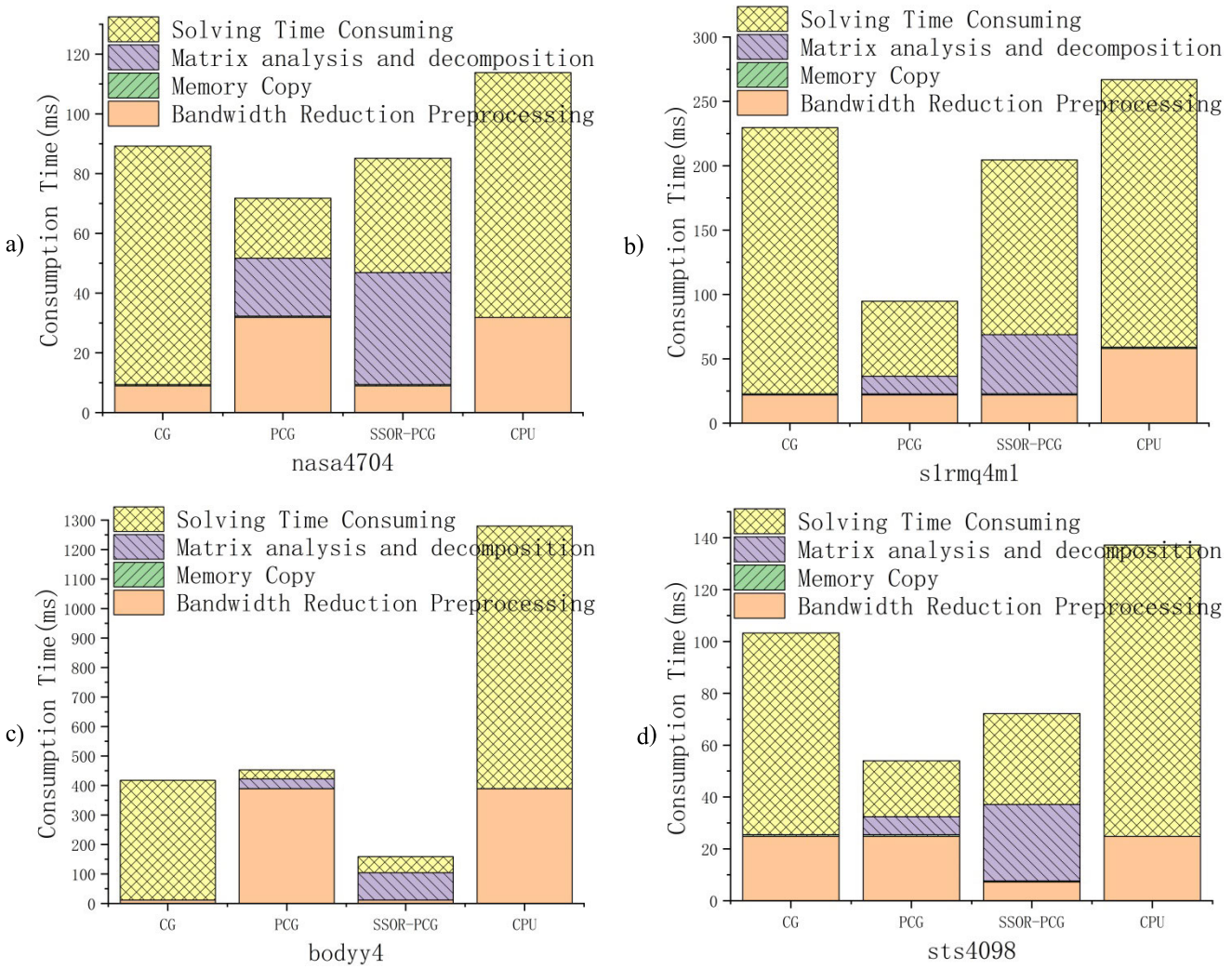

FIGURE 12. Time consuming situation of partial large sparse matrix with different algorithms.

the method. Next, the overall solution time of different matrices is compared, and the components of the solution time are analyzed to illustrate the fastness of the parallel algorithm. Finally, the correctness of the algorithm is verified.

Taking RCM and the approximate minimum degree algorithm as examples, Cholesky decomposition is adopted for the positive definite matrix and QR decomposition is adopted for the general matrix. The time spent on CPU after bandwidth optimization is compared as shown in TABLE 2. It can be concluded from TABLE 2 that the bandwidth reduction is very effective in reducing the solution time of linear equations. Among them, the row number of matrix No. 12 is smaller than that of matrix No. 13, but the solution time is longer. The main reason is that the sparsity of this matrix is lower. In addition, RCM algorithm is not effective in accelerating the solution of linear equations in several examples. The reason is that the row number of these matrix is massive with poor sparseness. The above part of bandwidth reduction schematic diagram is shown in FIGURE 9. The first column represents the original element distribution, the second column is the matrix element distribution optimized by RCM algorithm, and the third column is the matrix element distribution optimized by AMD algorithm [25], [26], [51].

Considering that GPU-based parallel algorithm does not have a good acceleration effect on small-scale matrix, some test cases close to the structure of aircraft parts in the test set in TABLE 2 are taken as the comparison between CPU operation time and GPU time, as shown in FIGURE 10.

It can be seen from the above figure that for the matrix of bodyy4 and nasa4704, which have small dimensions and a small average bandwidth, it is not appropriate to use the GPU-based RCM parallel algorithm designed in this paper, which takes even more time than the CPU version. The data interaction between the CPU and the GPU requires extra time, and cannot allocate enough threads to hide the overhead of data transmission. For larger-scale examples such as nd24k, the RCM algorithm designed in this paper has great advantages, and the acceleration ratio ranges from 7-14. From the

composition of computation time, *Qsort* and *SpMV* operation take up most of the time of RCM algorithm, and GPU has better acceleration performance for both algorithms. If a three dimensional node has six degrees of freedom, considering the time consumption of other calculation parts, the current acceleration effect can meet the real-time calculation of the aircraft finite element model with about 2000 nodes, but the matrix beyond this scale will not guarantee the real-time calculation with the step size of 25ms. Even so, the parallel bandwidth optimization algorithm can still be applied to the fault diagnosis of aircraft parts in limited time, which can save a lot of time for subsequent calculations.

In order to prove the performance of the algorithm and the architecture, the examples in TABLE 2 are used for time comparison. The serial program adopts Lapack algorithm package and Eigen library as contrast. By using partial parallel library and analyzing the time consuming of various linear equation solving algorithms, the algorithm with the least time-consuming on CPU is selected as the test benchmark. The stable double-conjugate gradient algorithm is not compared with other parallel iterative methods because of its large single-step calculation and low efficiency for symmetric positive definite matrix. Part of the test results of the test set are shown in FIGURE 11-FIGURE 12. The difference in the bandwidth optimization preprocessing time is due to the use of different bandwidth optimization algorithms.

The test results show that when the number of matrix rows is less than 1000, the CPU computing time is within 10 ms, and the conjugate gradient parallel algorithm based on CUDA has no advantage. Especially for small matrix such as bcsstk03, the efficiency of parallel algorithm is much lower than that of CPU due to the data transmission. However, when the number of matrix rows reaches 4000, the parallel algorithm gradually begins to show its advantages, as FIGURE 7 shows. For example, the acceleration ratio can reach 3 when solving sts4098, bodyy4 and some other examples. With the increase of matrix dimension, the time-consuming proportion of matrix analysis and decomposition steps in the total time of preprocessing algorithm is increasing, while the proportion of iterative solution time is decreasing. The reason is that LU and Cholesky decomposition are path dependent. Only after the previous row or column has completed the calculation can the next step be calculated. The available threads also change with the bandwidth, and the time-consuming increases sharply with the expansion of matrix dimension. In addition, the time-consuming results of two parallel preprocessing algorithms show that the preprocessing technology can effectively reduce the time consuming of solving linear equations. When the line equations are larger, the calculation acceleration is more obvious.

As can be seen from FIGURE 13, the pretreatment technology can effectively reduce the initial residual, significantly reduce the number of iterations, and is not prone to divergence. The difference between FIGURE 13b) and FIGURE 13d) examples is obvious. It can be seen from TABLE 2 that the condition numbers of both matrices are

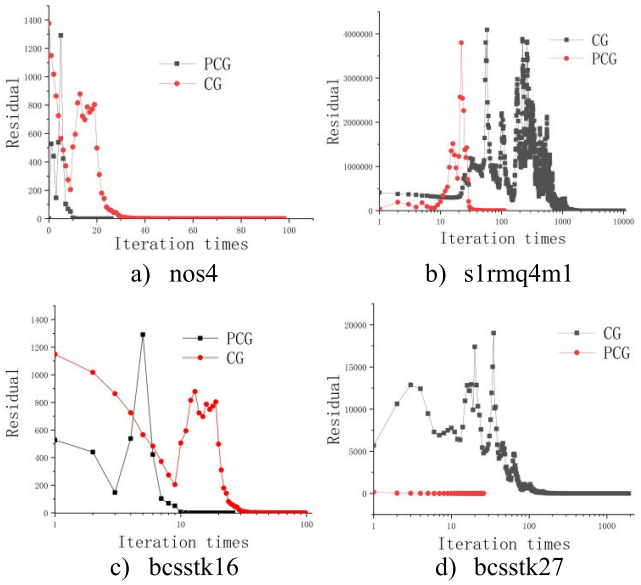

FIGURE 13. Comparison of iteration times of different iteration methods for several examples.

relatively large, so they are sensitive to the calculation truncation error in the iteration. However, the preprocessing technology improves the matrix attributes and computational efficiency by reducing the number of conditions.

V. CONCLUSION AND FUTURE WORK

In this paper, a large sparse linear algebra parallel bandwidth optimization based on RCM algorithm and parallel preprocessing conjugate gradient iteration method is proposed. A data segmentation model is established in a multi GPU Clusters, and the algorithm time-consuming test is carried out with some examples in the sparse matrix library. The specific contents are as follows:

The parallel RCM algorithm based on graph theory is analyzed and designed. By using BFS and graph theory, the matrix is equivalent to undirected vertex graph, and the depth of tree graph is changed to reduce the bandwidth. The custom *SpMV* algorithm realizes the fast parent node marking. An example of the sparse matrix library of the university of Florida, which is close to the parts of the aircraft, is used to analyze the composition of the parallel bandwidth optimization algorithm and verify the rapidity of the parallel bandwidth optimization algorithm.

Aiming at the solution algorithm of the current mainstream large linear equations, the parallel iterative solution algorithm of ICCG is implemented, and the calculation amount of each iteration is analyzed. By comparing the test set with the traditional CPU parallel solution library, the time consumption of each calculation step under different matrix dimensions is analyzed in detail. The results show that the pretreatment technique can effectively reduce the number of iterations and reduce the solution time for conventional structural components.

The asynchronous parallel architecture of multi-GPU clusters is designed, and the structural examples in the sparse matrix library of the University of Florida prove that the design method in this paper can be applied to the rapid calculation of the structural strength of aircraft parts.

REFERENCES

[1] E. J. Oliveira, P. Gasbarri, and I. M. da Fonseca, "Flight dynamics numerical computation of a sounding rocket including elastic deformation model," in *Proc. AIAA Atmos. Flight Mech. Conf.*, Dallas, TX, USA, Jun. 2015, pp. 1–13.

[2] E. Glaessgen and D. Stargel, "The digital twin paradigm for future NASA and U.S. air force vehicles," in *Proc. 53rd AIAA/ASME/ASCE/AHS/ASC Struct., Struct. Dyn. Mater. Conf.*, Honolulu, HI, USA, Apr. 2012, pp. 1–15.

[3] B. Ravishankar, B. Sankar, and R. Haftka, "Homogenization of integrated thermal protection system with rigid insulation bars," in *Proc. 51st AIAA/ASME/ASCE/AHS/ASC Struct., Struct. Dyn., Mater. Conf.*, Orlando, FL, USA, Apr. 2010, p. 2687.

[4] T. D. Skinner, S. Datta, A. Chattopadhyay, and A. Hall, "Biaxial fatigue damage in quasi isotropic laminates," in *Proc. AIAA Scitech Forum*, Orlando, FL, USA, Jan. 2020, pp. 1–10.

[5] J. Smith, K. Hamm, Jr., K. Imtiaz, and I. S. Raju, "Lesson learned from recent space flight assessments," in *Proc. AIAA Scitech Forum*, Orlando, FL, USA, Jan. 2020, pp. 1–19.

[6] K. R. R. Venkatesan, A. Rai, T. G. Stoumbos, D. Inoyama, and A. Chattopadhyay, "Finite element based damage and failure analysis of honeycomb core sandwich composite structures for space applications," in *Proc. AIAA Scitech Forum*, Orlando, FL, USA, Jan. 2020, pp. 1–10.

[7] S. Cook, *CUDA Programming: A Developer's Guide to Parallel Computing With GPUs*. San Francisco, CA, USA: Morgan Kaufmann, 2012.

[8] J. V. T. Risso, M. Bauer, P. R. de Carvalho, U. Rude, and D. Weingaertner, "Scalable GPU communication with code generation on stencil applications," in *Proc. 31st Int. Symp. Comput. Archit. High Perform. Comput. (SBAC-PAD)*, Campo Grande, Brazil, Oct. 2019, pp. 88–95.

[9] H. Binxing, L. Xinguo, Q. Hao, and L. Zenghao, "Accelerated solution of stiffness matrix for isoparametric elements based on CUDA," in *Proc. IEEE Int. Conf. Signal Process., Commun. Comput. (ICSPCC)*, Xiamen, China, Oct. 2017, pp. 1–4.

[10] S. Wang, X. Yan, Y. Zhang, D. Wu, and D. Xie, "Research on EBE-FEM realized by CUDA applying to electromagnetic field analysis," in *Proc. IEEE Student Conf. Electr. Mach. Syst.*, HuZhou, China, Dec. 2018, pp. 1–4.

[11] A. T. Corrigan, A. Kercher, J. Liu, and K. Kailasanath, "Jet noise simulation using a higher-order discontinuous Galerkin method," in *Proc. AIAA Aerosp. Sci. Meeting*, Jan. 2018, p. 1247.

[12] A. Dziekonski and M. Mrozowski, "GPU acceleration of block Krylov methods for FEM problems in electromagnetics," in *Proc. IEEE MTT-S Int. Conf. Numer. Electromagn. Multiphys. Modeling Optim. RF, Microw., THz Appl. (NEMO)*, Seville, Spain, May 2017, pp. 1–3.

[13] L. Mangani, G. Romanelli, A. Gaddai, and E. Casartelli, "Comparison of acceleration techniques on CFD open-source software for aerospace applications," in *Proc. 22nd AIAA Comput. Fluid Dyn. Conf.*, Dallas, TX, USA, Jun. 2015, pp. 1–14.

[14] A. Basermann, B. Reichel and C. Schelthoff, "Preconditioned CG methods for sparse matrices on massively parallel machines," *Parallel Comput.*, vol. 23, no. 3, pp. 381–398, 1997, doi: 10.1016/S0167-8191(97)00005-7.

[15] J. Bolz, I. Farmer, E. Grinspun, and P. Schröder, "Sparse matrix solvers on the GPU: Conjugate gradients and multigrid," *ACM Trans. Graph.*, vol. 22, pp. 171–178, Jul. 2003, doi: 10.1145/1198555.1198781.

[16] N. Bell and M. Garland, "Efficient sparse matrix-vector multiplication on CUDA," NVIDIA, Santa Clara, CA, USA, Tech. Rep. NVR-2008-004, 2008.

[17] A. Zaharovits, S. Stegaru, M. Carabas, E.-I. Slusanschi, and M.-V. Pricop, "Shared memory parallelization of the conjugate gradient linear system solver of a FEM topology optimization eode," in *Proc. 14th RoEduNet Int. Conf.-Netw. Educ. Res. (RoEduNet NER)*, Craiova, Romania, Sep. 2015, pp. 143–151.

[18] L. P. Amorim, R. C. Mesquita, T. D. S. Goveia, and B. C. Correa, "Node-to-node realization of meshless local Petrov Galerkin (MLPG) fully in GPU," *IEEE Access*, vol. 7, pp. 151539–151557, 2019, doi: 10.1109/ACCESS.2019.2948134.

[19] V. Volkov, D. Barbieri, J. Hogg, and A. Charara. CUBLAS Library User Guide, V10.1th ed. NVIDIA, Santa Clara, CA, USA. [Online]. Available: https://docs.nvidia.com/pdf/CUBLAS_Library.pdf

[20] L. W. Chang, P. Valero-Lara, and I. Martínez-Pérez. CUSPARSE Library, V10.1th ed. Santa Clara, CA, USA. [Online]. Available: https://docs.nvidia.com/pdf/CUSPARSE_Library.pdf

[21] W. Su, C. K. King, S. R. Clark, E. D. Griffin, J. D. Suhey, and M. G. Wolf, "Dynamic beam solutions for real-time simulation and control development of flexible rockets," *J. Spacecraft Rockets*, vol. 54, no. 2, pp. 403–416, Mar. 2017.

[22] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," in *Proc. 24th Nat. Conf.*, New York, NY, USA, 1969, pp. 157–172.

[23] A. Shrestha and I. Senocak, "Multi-level parallel algorithm to solve the Eikonal equation with the fast sweeping method," in *Proc. 23rd AIAA Comput. Fluid Dyn. Conf.*, Denver, CO, USA, Jun. 2017, pp. 1–10.

[24] A. Azad, M. Jacquelin, A. Buluc, and E. G. Ng, "The reverse Cuthill–McKee algorithm in distributed-memory," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, Orlando, FL, USA, May 2017, pp. 22–31.

[25] M. Fahrbach, G. L. Miller, R. Peng, S. Sawlani, J. Wang, and S. C. Xu, "Graph sketching against adaptive adversaries applied to the minimum degree algorithm," in *Proc. IEEE 59th Annu. Symp. Found. Comput. Sci. (FOCS)*, Paris, France, Oct. 2018, pp. 101–112.

[26] S. Singh, R. Srivastava, V. Kumar, and S. Agarwal, "An approximate algorithm for degree constraint minimum spanning tree," in *Proc. Int. Conf. Comput. Commun. Technol. (ICCCT)*, Allahabad, India, Sep. 2010, pp. 687–692.

[27] S. L. G. de Oliveira and A. A. A. M. de Abreu, "An evaluation of pseudoperipheral vertex finders for the reverse Cuthill–McKee method for bandwidth and profile reductions of symmetric matrices," in *Proc. 37th Int. Conf. Chilean Comput. Sci. Soc. (SCCC)*, Nov. 2018, pp. 1–9.

[28] A. George and J. W. H. Liu, *Computer Solution of Large Sparse Positive Definite Systems*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1981.

[29] T. N. Rodrigues, M. C. S. Boeres, and L. Catabriga, "A non-speculative parallelization of reverse Cuthill–McKee algorithm for sparse matrices reordering," in *Proc. Federated Conf. Comput. Sci. Inf. Syst.*, Sep. 2017, pp. 527–536.

[30] A. Buluç and K. Madduri, "Parallel breadth-first search on distributed memory systems," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal. (SC)*, Seatle, WA, USA, 2011, pp. 1–12.

[31] K. Ueno, T. Suzumura, N. Maruyama, K. Fujisawa, and S. Matsuoka, "Efficient breadth-first search on massively parallel and distributed-memory machines," *Data Sci. Eng.*, vol. 2, no. 1, pp. 22–35, Mar. 2017, doi: 10.1007/s41019-016-0024-y.

[32] A. Kameari, "Improvement of ICCG convergence for thin elements in magnetic field analyses using the finite-element method," *IEEE Trans. Magn.*, vol. 44, no. 6, pp. 1178–1181, Jun. 2008, doi: 10.1109/TMAG.2007.916501.

[33] T. White, *Hadoop: The Definitive Guide*, 4th ed. Sebastopol, CA, USA: O'Reilly, 2009.

[34] J. Zhou, Y. Chen, W. Wang, S. He, and D. Meng, "A highly reliable metadata service for large-scale distributed file systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 2, pp. 374–392, Feb. 2020, doi: 10.1109/TPDS.2019.2937492.

[35] K. Kaur, S. Garg, N. Kumar, G. S. Aujla, K.-K.-R. Choo, and M. S. Obaidat, "An adaptive grid frequency support mechanism for energy management in cloud data centers," *IEEE Syst. J.*, vol. 14, no. 1, pp. 1195–1205, Mar. 2020, doi: 10.1109/JSYST.2019.2921592.

[36] A. Wakde, P. Shende, S. Waydande, S. Uttarwar, and G. Deshmukh, "Comparative analysis of Hadoop tools and spark technology," in *Proc. 4th Int. Conf. Comput. Commun. Control Automat. (ICCUBEA)*, Pune, India, Aug. 2018, pp. 1–4.

[37] A. V. Hazarika, G. J. S. R. Ram, and E. Jain, "Performance comparision of Hadoop and spark engine," in *Proc. Int. Conf. I-SMAC*, Palladam, India, Feb. 2017, pp. 671–674.

[38] M. Axtmann, A. Wiebigke, and P. Sanders, "Lightweight MPI communicators with applications to perfectly balanced quicksort," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, Vancouver, BC, Canada, May 2018, pp. 254–265.

[39] S. Pellegrini, R. Prodan, and T. Fahringer, "A lightweight C++ interface to MPI," in *Proc. 20th Euromicro Int. Conf. Parallel, Distrib. Netw.-Based Process.*, Feb. 2012, pp. 3–10.

[40] T. Fukuoka, W. Endo, and K. Taura, "An efficient inter-node communication system with lightweight-thread scheduling," in *Proc. IEEE 21st Int. Conf. High Perform. Comput. Commun., IEEE 17th Int. Conf. Smart City, IEEE 5th Int. Conf. Data Sci. Syst. (HPCC/SmartCity/DSS)*, Zhangjiajie, China, Aug. 2019, pp. 687–696.
[41] X. Guo, J. Wu, Z. Wu, and B. Huang, "Parallel computation of aerial target reflection of background infrared radiation: Performance comparison of OpenMP, OpenACC, and CUDA implementations," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 9, no. 4, pp. 1653–1662, Apr. 2016, doi: 10.1109/JSTARS.2016.2516503.
[42] V. Sivanandan, V. Kumar, and S. Meher, "Designing a parallel algorithm for heat conduction using MPI, OpenMP and CUDA," in *Proc. Nat. Conf. Parallel Comput. Technol. (PARCOMPTECH)*, Bengaluru, India, Feb. 2015, pp. 1–7.
[43] Y. Fang and Q. Chen, "A real-time and reliable dynamic migration model for concurrent taskflow in a GPU cluster," *Cluster Comput.*, vol. 22, no. 2, pp. 585–599, Jun. 2019, doi: 10.1007/s10586-018-2866-8.
[44] S. Lin and Z. Xie, "A Jacobi_PCG solver for sparse linear systems on multi-GPU cluster," *J. Supercomput.*, vol. 73, no. 1, pp. 433–454, Jan. 2017, doi: 10.1007/s11227-016-1887-4.
[45] J. Sanders and E. Kandrot, *CUDA By Example: An Introduction to General-Purpose GPU Programming*, 1st ed. Reading, MA, USA: Addison-Wesley, 2010, pp. 38–46.
[46] C. Reaño and F. Silla, "On the support of inter-node P2P GPU memory copies in rCUDA," *J. Parallel Distrib. Comput.*, vol. 127, pp. 28–43, May 2019, doi: 10.1016/j.jpdc.2018.12.011.
[47] A. V. George, S. Manoj, S. R. Gupte, S. Mitra, and S. Sarkar, "Thrust++: Extending thrust framework for better abstraction and performance," in *Proc. IEEE 24th Int. Conf. High Perform. Comput. (HiPC)*, Dec. 2017, pp. 368–377.
[48] B. Hu and L. Xingguo, "Real-time simulation and optimization of elastic aircraft vehicle based on multi-GPU workstation," *IEEE Access*, vol. 7, pp. 155659–155670, 2019, doi: 10.1109/ACCESS.2019.2946684.
[49] G. R. L. Silva, R. R. De Medeiros, B. R. A. Jaimes, C. C. Takahashi, D. A. G. Vieira, and A. De Padua Braga, "CUDA-based parallelization of power iteration clustering for large datasets," *IEEE Access*, vol. 5, pp. 27263–27271, 2017, doi: 10.1109/ACCESS.2017.2765380.
[50] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, pp. 1–25, Nov. 2011.
[51] P. R. Amestoy, T. A. Davis, and I. S. Duff, "Algorithm 837: AMD, an approximate minimum degree ordering algorithm," *ACM Trans. Math. Softw.*, vol. 30, no. 3, pp. 381–388, Sep. 2004.

YUHUA ZHANG received the B.S. degree from Xi'an Technological University, Xi'an, China, in 2013. She is currently a Teacher at the School of Computer Science and Technology, Baoji University of Arts and Sciences. Her current research interests include big data architecture design, parallel algorithm design, and cloud computing.

BINXING HU received the B.S. and M.S. degrees from the Kunming University of Science and Technology, Kunming, China, and the Ph.D. degree from Northwestern Polytechnical University, Xi'an, China, in 2020. He is currently a Researcher at Aerospace System Engineering Shanghai. His current research interests include flight dynamics, reentry guidance, and parallel programming and applications.