IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 16, 2019, accepted October 5, 2019, date of publication October 11, 2019, date of current version November 6, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2946684

Real-Time Simulation and Optimization of Elastic Aircraft Vehicle Based on Multi-GPU Workstation

BINXING HU AND LI XINGGUO

Shaanxi Key Laboratory of Aerospace Flight Vehicle Design, Northwestern Polytechnical University, Xi'an 710072, China
School of Astronautics, Northwestern Polytechnical University, Xi'an 710072, China

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

This work was supported in part by the National Natural Science Fund of China under Grant 11672235, and in part by the National Defense Basic Research Project of China under Grant A0420132102.

ABSTRACT Modern aircraft such as missile and rocket, due to the large slenderness ratio of slender body vehicles, the influence of elastic deformation and vibration on navigation, guidance, and engine modules in simulation can not be ignored. For the problems of slow calculation speed and incapability of real-time simulation for time-domain simulation, by analyzing the time proportion of each calculation step under different computing scale, the dynamic parallel construction of octree is used to represent the aerodynamic parameter table under the environment of single and multi GPU. Meanwhile, an innovative parallel algorithm of element stiffness matrix based on finite element model is designed in GPU architecture. Accordingly, the optimized performance is enhanced through the adaptive hardware resources and rational use of shared memory. Furthermore, A multi-threaded asynchronous framework based on task queue and thread pool is proposed to realize the parallel task calculation with different granularities. The numerical result shows that the acceleration ratio of about 20 times in the single GPU condition can be obtained, and the acceleration ratio of at least 30 times can be obtained by the parallel computing of dual GPUs, enabling the real-time simulation of the flexible aircraft with 1200 elements within 20ms.

INDEX TERMS Flexible aircraft, parallel computing, parallel algorithms, heterogeneous computing, Octree.

I. INTRODUCTION

With the increasing requirement for the performance of aircraft, the self-weight ratio of the aircraft gradually reduced, and the coupling problem of elastic structure with aerodynamics and control system is becoming more and more serious. The vibration characteristics of aircraft during flight have a great impact on its dynamic performance, and then the accurate calculation of vibration characteristics has gradually become one of the core technical contents of the aircraft design. Because of the large slenderness of rockets, the conventional analysis method used in the past is to simplify the rocket into a flexible beam. Even for strap-on rocket, the booster is still regarded as a spatial beam system composed of one-dimensional beam and core stage, and the influence of elastic vibration on control system is analyzed based on a specific small deviation incremental model [1]. Free beam or cantilever beam modeling is often used to analyze the coupling relationship with aerodynamic and propulsion systems in the analysis of new hypersonic vehicles. In the design verification phase, the vehicle is usually discretized into several beam elements or spring-mass (site) models, based on which the aircraft modal information is established and bound to the simulation computer in advance [2]. In engineering applications, it is impossible to accurately express the elastic characteristics of the vehicle itself due to the nonlinearity of materials, the irregular geometry of the aircraft and other errors caused by simplified models. While the number of nodes is too large to meet the real-time requirements of simulation.

In the case that traditional serial computing can not meet the existing computing capacity, high performance computing (HPC) has been developed that brings computing power together to provide higher performance than general workstations. HPC embodies three parallel computing technologies at different granularity levels, namely, cluster, multi-core and heterogeneous platforms, represented by MPI (Message Passing Interface), OpenMP (Open Multi-processing) and GPU.

The associate editor coordinating the review of this manuscript and approving it for publication was Gang Mei.

The clustering technology represented by MPI is to realize parallel computing through multi-machine collaborative work. Task data is decomposed into multiple workstations by dividing and conquering idea, and the data blocks are returned to the host to complete the reduction work after each computing node calculates the data blocks. Hadoop and Spark are two typical frameworks that apply this idea and technical route. The two general parallel frameworks are written in Java and support other languages in the form of streams, providing map, reduce, filter, join and other operators. It strips away the user's attention to task allocation and data storage in cluster computing, so that users only need to focus on the application itself. In terms of the underlying applications, Lu *et al.* put forward the idea of MPI accelerating Hadoop [3], explained the key points of communication delay, technical difficulty and performance improvement in his paper, and pointed out that the HPC area seems to have a more efficient communication infrastructure, including high-performance interconnects such as the Infiniband and high-performance communication libraries. In contrast, the Hadoop software suite realizes communication based on traditional network protocols. Bai used a hybrid architecture of MPI and Hadoop for molecular dynamics simulation [4]. The simulation results verified that MPI technology has no obvious acceleration ratio under the same conditions of processors, and the time consuming of Hadoop+MPI is significantly reduced when the number of iterations is higher, which further illustrates that MPI technology is more suitable for compute-intensive models. Dao also proposed using MPI to accelerate Hadoop and HPC-Reuse technology to avoid the continuous creation and destruction of JVM processes [5]. The results of numerical examples show that the computing efficiency of data-intensive programs can be significantly improved. Asaadi *et al.* [6] illustrates the differences between high performance computing and large data in detail through examples, and compares the differences between data-intensive and compute-intensive models by using different frameworks. Kwedlo and Czochanski [7] also proposed a K-means clustering algorithm based on MPI/OpenMP hybrid, focusing on improving parallel efficiency through triangular inequalities, but did not elaborate on the time-consuming of each example. Generally speaking, although MPI has good scalability, it is suitable for fast retrieval, clustering and very large-scale data-intensive computing due to the communication mechanism between processes and the high latency of communication between different workstations, but not for real-time computing in millisecond level or compute-intensive models in limited time.

Similar to MPI, the multi-core technology represented by OpenMP adopts the idea of divide-and-conquer. The former distributes tasks to multiple computing nodes, while the latter distributes tasks to different threads within the CPU of a single computing node. Peng and Wang [8] published the cloud computing model based on MPI and OpenMP, Hadoop combined with MPI/OpenMP was designed, but it was not validated by numerical examples. Ediger and Bader [9] proposed that OpenMP should be combined with Hadoop to solve the problem of fast graph analysis,. Gómez *et al.* [10] pointed out that high performance computing can achieve the highest parallel efficiency only if it combines the parallel MPI technology between nodes with OpenMP-like multi-core technology within nodes. Sun *et al.* [11] designed a real-time planning and task analysis system based on task binding, and divided the tasks into sections with the idea of fork-join. However, OpenMP technology can only provide the maximum acceleration performance of the current computer CPU core number, which is greatly affected by thread creation and task distribution. With the gradual improvement of GPU performance and the abundance of interfaces and libraries, OpenMP has become a subordinate position in high performance computing.

As CPUs no longer keep pace with Moore's law, multi-core devices such as GPU will be more widely used. Since NVIDIA officially released the parallel computing technology of CUDA (Compute Unified Device Architecture) in 2007, it has been able to reduce the development threshold by using the standard C language extended API form [12], [13]. In recent years, it has been widely used in explicit dynamics [14], fluid mechanics [15], [16], structural mechanics [17], molecular dynamics [18], and has become an important branch in the field of HPC. Compared with CPUs cluster computing, GPUs has higher bandwidth, lower latency, and lower power consumption per unit floating-point operation [19]. Because GPU is a SIMD(Single Instruction Multiple Data) architecture, which is not suitable for complex instructions and branch logic in most compute-intensive models, and MIMD(Multiple Instruction Multiple Data) architecture can be realized by combining cluster computing [20]. Agarwal and Becchi [21] proposed a high performance cluster framework of MPI-CUDA, which tested the performance differences between locked page memory and non-locked page memory under the condition of different computing sizes and communication volumes according to the standard test set. Zhu *et al.* [22] elaborates four methods of combining CUDA with Hadoop, and makes a detailed comparison in performance, complexity, program translation and test difficulty. Reza *et al.* [23] proposed using CUDA technology to accelerate sparse matrix-vector multiplication under Hadoop architecture. The results show that the performance of single-precision floating point operation is not improved significantly. Khare *et al.* [24] proposed HCudaBLAST for protein sequencing in 2017, which actually refines and encapsulates Hadoop and CUDA without high acceleration ratio. Kloh *et al.* [25] compared the performance of X86 architecture, ARM architecture and Tesla architecture in three standard test sets and two real applications. By comparing and analyzing the medium-sized HPC system and mobile-based Jetson TX2 development board, it shows that a deep understanding of the application background, underlying framework and programming model is essential

for performance and energy consumption. It also indicates that MPI + OpenMP technology will be implemented in the next step.

Combined with the characteristics of GPU hardware, the architecture of the asynchronous parallel computing of the slender body elastic aircraft is designed on the multi-GPUs workstation, and the dynamic performance optimization based on GPU algorithm is proposed. The paper is organized as follows. In section II, the composition of the elastic vibration module of aircraft is described, and the key points to realize real-time simulation of the improved algorithm are obtained through time consuming analysis. Section III details the design of the most time-consuming parallel algorithm for aerodynamic coefficient interpolation. In order to facilitate the fast indexing of data, an index octree of dynamically allocated thread blocks is proposed to reduce the time-consuming memory operation by reducing the height. Combining with the fact that the pneumatic coefficient table is not large, the table is put into shared memory to improve the reading efficiency, and the optimal parameters of kernel function are obtained by combining with the limitation of hardware resources. At the same time, a parallel element stiffness matrix algorithm based on GPU is established to simulate the flight state of aircraft in fault mode, and the speed and accuracy of the algorithm are verified by comparison. In section IV, an asynchronous heterogeneous computing framework is described. The concept of concurrent tasks in thread pool is proposed, which can further reduce computing time in multi-task mode after task decomposition. Finally, in section V, the performance analysis of aircraft models with different computational scales is carried out.

II. FORMULATION OF VIBRATION RESPONSE SOLUTION FOR SLENDER VEHICLE

In order to illustrate the parallel implementation of the vibration response algorithm for flexible slender vehicle, the calculation process is analyzed in combination with the algorithm, and the calculation amount and proportion of each step are analyzed in order to determine the key points of the parallel algorithm design. The calculation process of the elastic module in the whole flight process is shown in Fig. 1. The elastic module is mainly divided into two parts: the solution of vibration equation and the calculation of the additional amount of elastic deformation. The solution of vibration equation mainly involves the relevant calculation of vibration mode and aerodynamic force of elastic aircraft, as shown in steps 2, 3 and 4 in Fig. 1 below. In the simulation process, the aerodynamic coefficients of a certain time step are obtained by interpolation of the aerodynamic coefficients table [26]. Aerodynamic coefficients table is a set of data obtained from experiments at a certain numerical interval of angle of attack, velocity and height. Under normal flight conditions, after obtaining the current thrust, attitude and other information from other modules, the aerodynamic coefficients are obtained by interpolation from the bound vibration mode and the vibration mode slope table, once the

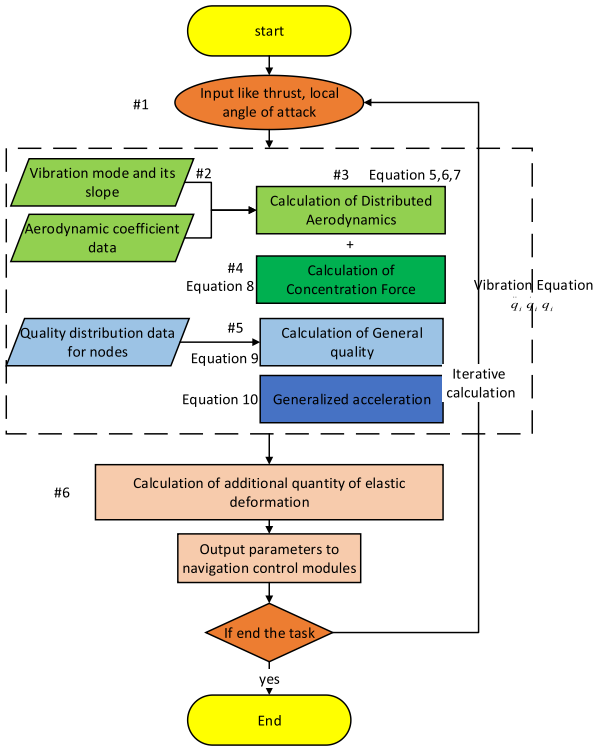

FIGURE 1. Flow chart of the elastic module in flight dynamics simulation.

distribution force are calculated, thus the resultant forces with the concentrated forces are calculated. After the generalized mass calculation in step 5 is completed, the generalized acceleration of the aircraft is obtained by solving the second-order differential equation. The velocity and displacement of each element represent the additional amount of deformation and rate of change, which will be output to other modules, as shown in step 6.

As mentioned, the elastic body is simplified as a homogeneous beam model, the bending stiffness is represented by EJ (x), the mass density is represented by m (x), the reference length is represented by l, and the laterally distributed external force acting on the beam element is represented by $q(x, t)$. The differential equation of bending vibration of the beam is obtained based on the bending theory and the moment balance equation, ignoring the rotational inertia of the beam element and damping:

$$q(x,t) = EJ(x)\frac{\partial^4 \omega(x,t)}{\partial x^4} + m(x)\frac{\partial^2 y}{\partial t^2} \tag{1}$$

The general solution of the vibration differential equation is obtained when $q(x, t) = 0$. The special solution can be calculated by the boundary conditions of the free beam. The forced vibration equation of the beam is expanded into the series form of mode, as shown in (2):

$$y(x,t) = y_c(t) + \theta(t)(x - x_c) + \sum_{n=1}^{\infty} f_n(x)\, q_n(t) \tag{2}$$

$q(x,t) = q(t)P(x)$ denotes that distributed aerodynamic forces are the product of unit forces, the equation (2) is introduced into (1) after the integral transformation.

$$\begin{cases} m\ddot{y}_c = Q_y \\ I\ddot{\theta} = Q_\theta \\ m_i\left(\ddot{q}_i + 2\xi_i\omega_i + \omega_i^2\right) = Q_i \end{cases} \tag{3}$$

The generalized force at the right end of the equation is (4):

$$\begin{cases} Q_y = q(t)\int_0^l p(x)\,dx \\ Q_\theta = q(t)\int_0^l P(x)(x - x_c)\,dx \\ Q_i = q(t)\int_0^l P(x)f_i(x)\,dx \end{cases} \tag{4}$$

The first term in (3) represents the centroid equation of motion, and the second term represents the equation of motion of rotation around the horizontal axis. Once the external aerodynamic force is given, the generalized coordinates of the beam bending vibration can be obtained from the last set of equations. Through derivation, the numerical solution steps of the normal and lateral elastic vibration equations of the slender body are summarized as:

$$\begin{aligned} \alpha_s(x_j,t) &= \alpha + \frac{\partial y(x,t)}{\partial x} + \frac{\dot{y}(x,t)}{V} \\ &= \alpha + \sum_{i=1}^{n} f'_{iy}(x_j)q_{iy}(t) + \frac{1}{V}\sum_{i=1}^{n} f_{iy}(x_j)\dot{q}_{iy}(t) \end{aligned} \tag{5}$$

$$Y_a(x_j) = C_n^\alpha(x_j)\alpha_s(x_j)q \tag{6}$$

$$Q_{fayi} = Y_a(x_j) \times f_{iy}(x_j) \tag{7}$$

$$\begin{aligned} Q_{iy} &= Q_{fayi} + \sum_k Q_{fkyi} \\ &= Q_{fayi} + P_y + F_{eiy} + F_{py} + F_{dy} \end{aligned} \tag{8}$$

$$M_i = \sum_{j=1}^{N} m(x_j)f_{iy}^2(x_j) \tag{9}$$

$$Q_{iy} = M_i\left(\ddot{q}_{iy} + 2\xi_{iy}\omega_i + \omega_i^2\right) \tag{10}$$

where i represents the modal order and assumes that the truncated modal order is n; j is the station number for a total of N sites. The local attack angle α_s consists of the following three parts: the equivalent attack angle α when the vehicle is regarded as a rigid body, the cumulative sum of the modal shape $f_{iy}(x_j)$ and lateral generalized displacement q_{iy}, the cumulative sum of the slope of mode $f'_{iy}(x_j)$ and lateral generalized velocity $\dot{q}_{iy}$. The normalized force Q_{iy} of the i-th order is composed of the following parts: the normal distribution of the aerodynamic generalized force Q_{fayi}, the thrust component P_y, the engine swing inertia force component F_{eiy}, the tank sloshing component F_{py}, and the separation disturbance component. M_i denotes the generalized mass of the i-th mode.

The distributed aerodynamic calculation and generalized mass calculation of the elastic vibration equation are the most time-consuming: the generalized mass has only $2N$ multiplications and N addition operations, which the form is relatively simple. Once the number of nodes is large, the amount of calculation caused by the concentration force which only needs to be calculated once can be neglected. The generalized normal force at the i-th order include the $2n$ multiplication and $2n + 1$ addition required for calculating the local attack angle, as well as the interpolation calculation of the aerodynamic coefficients for each node and the additional 4 multiplications. The aerodynamic coefficient interpolation algorithm is bilinear interpolation for three parameters: height, attack angle and Mach, which needs to complete the index of 8 boundary points and at least 7 basic linear interpolations. Since the interpolation involves complex memory calls on the heap and the management of registers in operations, the operation time cannot be calculated by the time-consuming accumulation of $8^* \log_2^n$ indexes, 14 additions and 7 multiplications under the most ideal conditions. The specific time-consuming situation of interpolation is shown in Table 1, which shows that interpolation is the most time-consuming in CPU operation. When the number of sites is 200 and the number of modal orders is 40, interpolation accounts for 61% of the total time. With the increase of the number of nodes, the interpolation calculation accounts for a gradual increase in the total time-consuming ratio until 90% or more. In view of the fact that the flexible module needs to provide relevant attitude additional information to the control system in each time step, 20ms is chosen as the divergence boundary of the control system based on previous experience, that is, the criterion of real-time simulation. It is known that the demand of real-time simulation cannot be met when the number of sites exceeds 3200.

TABLE 1. Comparison of interpolation computation time of cpu version under different node number.

Node Number	200	800	3200	12800	51200
CPU time consuming(ms)	1.38	2.47	18.6	76.9	269.4
Percentage of total calculated time(%)	61	77	87	91	95

III. PARALLEL ALGORITHM DESIGN BASED ON CUDA

A. PARALLEL INTERPOLATION ALGORITHMS DESIGN FOR AERODYNAMIC COEFFICIENT

Although the GPU accelerated Kriging interpolation algorithm [27], cubic spline interpolation [28] and bilinear interpolation [29] have been studied in the literature, the best linear unbiased estimation capabilities provided by Kriging interpolation brings extra computation. Since the interpolation calculation result of the aerodynamic coefficient only affects the guidance and control system in the current step, the guidance system tracks the standard trajectory during each guidance period, so no cumulative error occurs. In conclusion, Kriging interpolation is not applicable here. Since the effectiveness and accuracy of bilinear interpolation have been verified in previous flight dynamics calculation, considering

the calculation efficiency of hardware, the index time in the interpolation process of aerodynamic coefficient should be reduced as much as possible, so this method is selected for aerodynamic coefficient interpolation. As mentioned in the previous section, if the current aerodynamic parameters need to be indexed $8^* \log_2^n$ times by dichotomy, the increase of GPU addressing times reduces its effective bandwidth. Especially when the scale is small, the program may change from computationally intensive to IO-intensive, resulting in significant loss of computational efficiency. Inspired by the collision detection algorithm, each point in the interpolation table can be regarded as a point in the three-dimensional space, and every eight adjacent points in the table enclose a cube in three-dimensional space. The points to be interpolated may be considered on the surface or within the cube. The first step of bilinear interpolation is to determine the cell in which the point is located, and obtain the coordinates of the eight nodes surrounding the cell and their values.

For such problems, the octree is usually used to describe the data structure in three-dimensional space [30]: compared with the binary tree, the number of arbitrary sub-nodes of octree species is 0 or 9. The number of pointers maintained by each node is changed from two to eight. By judging the boundary value and midpoint of three directions in three-dimensional space, the calculation amount of each index is increased compared with the binary tree, so that the index depth is reduced and the speed is improved. NVIDIA provides CUDA dynamic parallel API and demonstrates how to construct quadtree recursively. Therefore, this algorithm is used as a template to construct octree of aerodynamic coefficient data table.

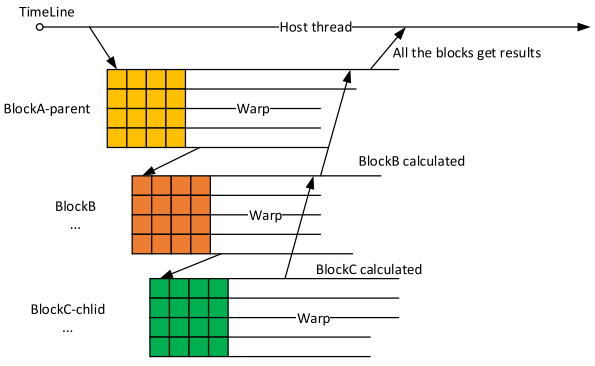

FIGURE 2. Time diagram of dynamic building thread blocks.

The flow chart of dynamically creating blocks in GPU is shown in Fig. 2 as follows. The depth of the octree is obtained after the size of the startup data table is obtained. After the initialization of the tables to be built is completed, the parent block is divided into eight blocks, the number of points in each sub-block is counted with thread warp as the basic unit. After the warp completes its internal correlation calculation, it passes the relevant data to the eight sub-blocks, and recursively until the condition is not satisfied.

After completing the construction of octree, the linear form storage suitable for GPU is selected to improve the index efficiency. There are $8^*n + 8$ elements in the linear table. Each of the eight elements in the table represents the number of all the sub-nodes under a node. If there is no data in a sub-node, it can set -1. It can be seen that it is necessary to traverse all the other seven sub-nodes at the same level before indexing a node, the theoretical efficiency of GPU could be significantly improved by replacing the $3^* \log_8^n$ additional addressing operations generated by the binary tree with up to $8^* \log_8^n$ boolean judgments and $16^* \log_8^n$ conditional judgments. Compared with the storage on the CPU through a linked list, linear storage needs to convert any octree into a complete octree, and the empty nodes need to be filled. But once the tree graph is unbalanced and deep, it will cause a huge waste of memory.

TABLE 2. Time in millisecond for the interpolation computation.

Node Number	200	800	3200	12800	51200
GPU time consuming(ms)	1.047	1.313	1.722	6.2	14.82
SpeedUp	1.3	1.8	10.83	12.42	18.18

The test results are shown in Table 2. By comparison with table I, it can be seen that after using octree to form the interpolation table of aerodynamic coefficients, the parallel interpolation based on GPU is greatly improved compared with CPU by using the method of building thread blocks dynamically. Although the performance of traditional single-core CPU is compared here, according to the most ideal linear acceleration, the eight-core CPU using OpenMP technology can not reach the speed of GPU in 3200 elements. At the same time, OpenMP needs to start threads every time it calculates, which is extremely time-consuming and not very stable. In addition, since GPU is the coprocessor of CPU, CPU can execute other tasks concurrently when GPU performs interpolation and stiffness matrix calculation tasks in subsequent asynchronous architecture.

B. DESIGN OF PARALLEL ALGORITHMS FOR DISTRIBUTED QUALITY

As mentioned in the preceding section, the calculation of generalized mass includes the problem of multiplying two vector elements with length N and reducing the result. After each thread completes the multiplication of elements, the interleaved pairing reduction algorithm is selected. Compared with the adjacent pair reduction algorithm, the crossover step size calculated by each thread is no longer the data of adjacent threads, but starts at half of the thread number and reduces by half per iteration. Therefore, the interleaved pairing algorithm does not change the number of worker threads per iteration, which reduces the stall of thread bundle as much as possible. The results show that the staggered pairing reduction

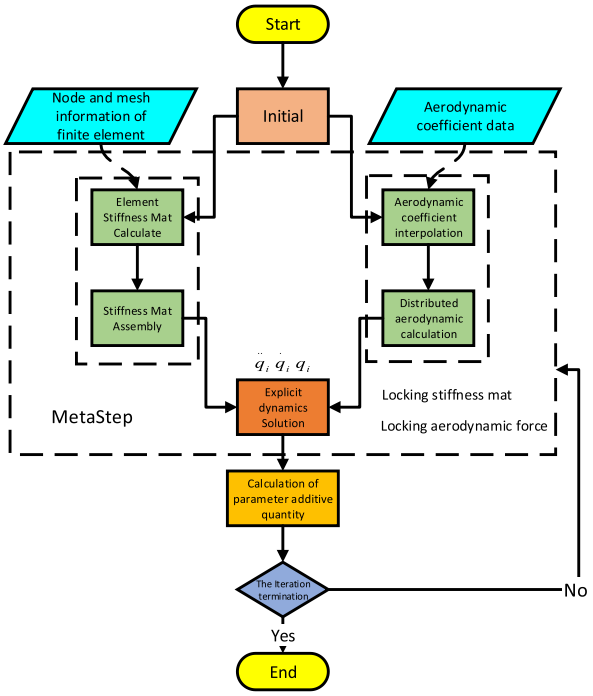

FIGURE 3. Flow chart of elastic module based on FEM.

algorithm has an acceleration ratio of 1.34 to 1.69 times in different computing scales [31]–[33].

C. PARALLEL ALGORITHM DESIGN FOR STIFFNESS MATRIX CALCULATION

As shown in Fig. 1, it is effective to use the preset vibration mode parameters in the normal conditions. Once the aircraft fails in the flight process, such as the engine turbopump speed decreases and the flow rate decrease, the mode will deviate from the original data due to the change of mass. Therefore, a slender aircraft model based on finite element method(FEM) is proposed, and the calculation process is shown in Fig. 3. After introducing the mesh model in the initialization stage, the mass and stiffness matrices of elements need to be calculated for each step. Once the assembly is completed, the displacement, velocity, and acceleration of each node under distributed aerodynamic forces are calculated by explicit dynamics [34]. Referring to the integration algorithm in reference [14], matrix addition and matrix-vector multiplication are performed in each time step, which does not take much time on GPU[35], [36]. According to Amdahl's law, the calculation of element matrices is the bottleneck of real-time simulation performance. The parallel algorithm will be described in detail.

For the purpose of simultaneous calculation of multiple elements, it is conceivable to simultaneously calculate elements at the same position in the stiffness matrix of several beam elements. The expression of the element stiffness matrix of plane beam element in the global coordinate system is given as:

$$K = T^T \overline{K} T \tag{11}$$

The matrix T is the coordinate transformation matrix and $\overline{K}$ is the stiffness matrix in the local coordinate system. The element stiffness matrix can be divided into several parts.

$$K = \begin{pmatrix} K_1 & K_2^T \\ K_2 & K_3 \end{pmatrix} \tag{12}$$

In (12),

$$K_1 = \begin{pmatrix} EA/L & & \\ & 12EI/L^3 & 6EI/L^2 \\ & 6EI/L^2 & 4EI/L \end{pmatrix},$$
$$K_2 = \begin{pmatrix} -EA/L & & \\ & -12EI/L^3 & -6EI/L^2 \\ & 6EI/L^2 & 2EI/L \end{pmatrix},$$
$$K_3 = \begin{pmatrix} EA/L & & \\ & 12EI/L^3 & -6EI/L^2 \\ & -6EI/L^2 & 4EI/L \end{pmatrix}$$

Divide the transformation matrix T:

$$T = \begin{bmatrix} t & \\ & t \end{bmatrix} \tag{13}$$

$$T = \begin{bmatrix} t^T & \\ & t^T \end{bmatrix}, \quad t = \begin{pmatrix} \cos\alpha & \sin\alpha & \\ -\sin\alpha & \cos\alpha & \\ & & 1 \end{pmatrix} \tag{14}$$

Substitute (14) into (11):

$$K = \begin{bmatrix} t^T & \\ & t^T \end{bmatrix} \begin{bmatrix} K_1 & K_2^T \\ K_2 & K_3 \end{bmatrix} \begin{bmatrix} t & \\ & t \end{bmatrix} = \begin{bmatrix} t^T K_1 t & t^T K_2^T t \\ t^T K_2 t & t^T K_3 t \end{bmatrix} \tag{15}$$

Due to the similarity of elements K_1, K_2 and K_3, suppose the submatrix K_1 has the following form like (16):

$$K_1 = \begin{pmatrix} k_{11} & & \\ & k_{22} & k_{23} \\ & k_{23} & k_{33} \end{pmatrix} \tag{16}$$

The submatrix K_2 and K_3 can be derived:

$$K_2 = \begin{pmatrix} -k_{11} & & \\ & -k_{22} & -k_{23} \\ & k_{23} & k_{33}/2 \end{pmatrix},$$
$$K_3 = \begin{pmatrix} k_{11} & & \\ & k_{22} & -k_{23} \\ & -k_{23} & k_{33} \end{pmatrix} \tag{17}$$

The three sub-matrices have similar forms, a parallel algorithm for simultaneous calculation of stiffness matrices of r elements is presented.

1) The s-th element is denoted by subscript $[s]$, which can form the following four vectors:

$$\overline{K}_{11} = \left(E_{[1]}A_{[1]}/L_{[1]}, E_{[2]}A_{[2]}/L_{[2]}, \ldots, E_{[r]}A_{[r]}/L_{[r]}\right)^T$$
$$\overline{K}_{22} = \left(12E_{[1]}I_{[1]}/L_{[1]}^3, 12E_{[2]}I_{[2]}/L_{[2]}^3, \ldots, 12E_{[r]}I_{[r]}/L_{[r]}^3\right)^T$$

$$\overline{K}_{23}=\left(6E_{[1]}I_{[1]}/L_{[1]}^2, 6E_{[2]}I_{[2]}/L_{[2]}^2, \ldots, 6E_{[r]}I_{[r]}/L_{[r]}^2\right)^T$$
$$\overline{K}_{33}=\left(4E_{[1]}I_{[1]}/L_{[1]}, 4E_{[2]}I_{[2]}/L_{[2]}, \ldots, 4E_{[r]}I_{[r]}/L_{[t]}\right)^T \quad (18)$$

2) Substitute the algebraic submatrix K_1 into (11), we have (19), as shown at the bottom of this page.

Thus forming the following five vectors of length r

$$\begin{aligned} K_{11} &= \cos^2\alpha\cdot\overline{K}_{11}+\sin^2\alpha\cdot\overline{K}_{22}\\ K_{21} &= \sin\alpha\cos\alpha\cdot\overline{K}_{11}-\sin\alpha cons\alpha\cdot\overline{K}_{22}\\ K_{22} &= \cos^2\alpha\cdot\overline{K}_{22}+\sin^2\alpha\cdot\overline{K}_{11}\\ K_{31} &= -\sin\alpha\cdot\overline{K}_{23}\\ K_{32} &= \cos\alpha\cdot\overline{K}_{23} \end{aligned} \quad (20)$$

3) The element stiffness matrix is obtained in the following form (21)

$$K^*=\begin{bmatrix} K_{11}^* & & & & & \\ K_{21}^* & K_{22}^* & & symmetry & & \\ K_{31}^* & K_{32}^* & \overline{K_{33}}^* & & & \\ -K_{11}^* & -K_{21}^* & -K_{31}^* & K_{11}^* & & \\ -K_{21}^* & -K_{22}^* & -K_{32}^* & K_{21}^* & K_{22}^* & \\ K_{31}^* & K_{32}^* & \overline{K_{33}}^*/2 & -K_{31}^* & -K_{32}^* & \overline{K_{33}}^* \end{bmatrix} \quad (21)$$

K_{ij}^s denotes the position of element stiffness matrix (i, j) of element s. Therefore, the algorithm can obtain the stiffness matrix of r-th cells by calculating five parameters, thereby greatly reducing the amount of calculation.

Derived from the above theory, we can get two kinds of kernel function with different strategies. The first strategy is that each thread calculates the stiffness matrix of a unit, then all input parameters and nine intermediate variables are put into the register, and shared memory only stores 21 floating-point data of each cell matrix. We provide pseudo-code in references [37]. However, since the GPU has a capacity limit on the hardware registers and shared memory, the number of threads in each block cannot allocate a sufficient number of threads, otherwise there will be a risk of failure of kernel function calls.

Another strategy is to calculate the stiffness matrix of a unit by a set of five threads. The four parameters $\overline{K}_{11}$, $\overline{K}_{22}$, $\overline{K}_{23}$ and $\overline{K}_{33}$ are obtained by the elastic modulus E, the cross-section A or the inertia I, and the length l. The intermediate parameters are stored in shared memory, then five advanced parameters $K_{11}, K_{21}, K_{31}, K_{22}$ and K_{32} of each element matrix are formed. Each thread needs two integer spaces in the register to express the group number and offset of the current thread. Since the advanced parameters are multiplexed many times in computation and the global memory transfer delay is up to hundreds of clock cycles, if the strategy of storing the advanced parameters into shared memory is adopted, only 9 floating-point memory space is needed for each thread group. The process of implementing the algorithm in CUDA as shown in Algorithm1:

Algorithm 1 Kernel Function for Element Stiffness Matrix Computation

```
__global__ void BeamElemMatCal(float * E, float* A,
float * I, float * L, float * theta){
    //__shared__ float shared[ 9 * (blockDim.x/5) ];
  extern __shared__ float shared[];
    int tid = blockIdx.x * blockDim.x + threadIdx.x;
            int group = Tid/5;
            int offset = Tid%5;
  if(offset == 0) Calculate K̄11 then store in
SharedMem[group*9];
  if(offset == 1) Calculate K̄22 then store in
SharedMem [group*9+1];
  if(offset == 2) Calculate K̄23 then store in
SharedMem[group*9+2];
  if(offset == 3) Calculate K̄33 then store in
SharedMem[group*9+3];
  __syncthreads();
  if(offset == 0) Calculate K11 then store in
SharedMem[group*9+4] and fill in Element Matric
  if(offset == 1) Calculate K21 then store in
SharedMem[group*9+5] and fill in Element Matric
  if(offset == 2) Calculate K22 then store in
SharedMem[group*9+6] and fill in Element Matric
  if(offset == 3) Calculate K31 then store in
SharedMem[group*9+7] and fill in Element Matric
  if(offset == 4) Calculate K32 then store in
SharedMem[group*9+8] and fill in Element Matric
            __syncthreads();
    }
```

To verify the parallel algorithm efficiency of beam element stiffness matrix based on CUDA, the cantilever beam with Young's modulus of 2.07e5, Poisson's ratio of 0.3, length of 6 and cross-section of 0.1x0.1 square section is selected and divided into finite element models with different sizes. A comparison of the solution times of both versions is presented for the test case shown in Table 3.

As can be seen from the data in Table 3, if the rounding accuracy of floating-point operations in the numerical calculation is taken into account, the results of the two calculations can be considered to be no difference. When the number of elements is small, GPU can not open enough thread bundles to hide the disadvantages of transmission delay and low main

$$t^T K_1 t=\begin{bmatrix} \cos^2\alpha\cdot K_{11}+\sin^2\alpha\cdot K_{22} & symmetry \\ \sin\alpha\cos\alpha\cdot K_{11}-\sin\alpha cons\alpha\cdot K_{22} & \sin^2\alpha\cdot K_{11}+\cos^2\alpha\cdot K_{22} \\ -\sin\alpha\cdot K_{23} & \cos\alpha\cdot K_{23} \end{bmatrix} \quad (19)$$

TABLE 3. Time in millisecond for the stiffness matrix computation for GPU and CPU based methods.

Node	OpenMP	GPU		Residual
		Data Copy	Kernel Fuc	
200	1.827	0.005952	0.2252	-
400	2.562	0.008192	0.3853	0.009%
800	4.799	0.01024	0.7223	0.01%
1600	8.156	0.0174	1.2134	0.011%
3200	14.413	0.0372	2.1617	0.017%
6400	23.579	0.0575	4.1189	0.024%
12800	54.192	0.128	8.7930	0.023%

frequency, so the acceleration ratio is not good. However, with the number of units increasing gradually, the computing efficiency of GPU is obviously higher than that of CPU, and with the number of elements increases, the acceleration ratio is also higher.

IV. PARALLEL STRATEGY

Published studies always focused on the design of algorithms [38], [39]. The blocking mechanism between CPU and GPU may lead to the failure of the device to return control to the host thread before completing the task, resulting in the blocking of the host thread and thus unable to meet the real-time requirements. In this paper, Boost threadpool cross-platform lightweight library is used for underlying programming. Combining with the characteristics of GPU page-locked host memory and asynchronous call, an asynchronous heterogeneous parallel computing architecture based on threadpool is proposed.

The performance optimization of parallel programs can be divided into coarse-grained parallel and fine-grained parallel [40]. The former can be considered as the optimization of the main architecture. By opening a thread pool with several threads, a task structure is constructed in the initialization stage, which contains the identification number calculated by CUDA, the device number used, the stream pointer, the host data pointer, and the device data pointer. When the program is initialized, the computing tasks are distributed to different threads according to the unique ID. The control of CPU is always guaranteed by asynchronous mode until the trigger callback function is completed by GPU operation, the next calculation can be carried out logically. In the latter case, data transmission is carried out at the same time through a GPU with more than 2.x computing capacity so that the execution engine and storage engine can work simultaneously [41].

Considering that the process of solving the elastic vibration equation in flight simulation can be decoupled into three independent computations of transverse, normal and roll directions, and there are no intermediate variables coupled with each other, the main structure of the program is designed as shown in Fig. 4. The main thread maintains a FIFO (first-in-first-out) queue to complete task assignment and track the

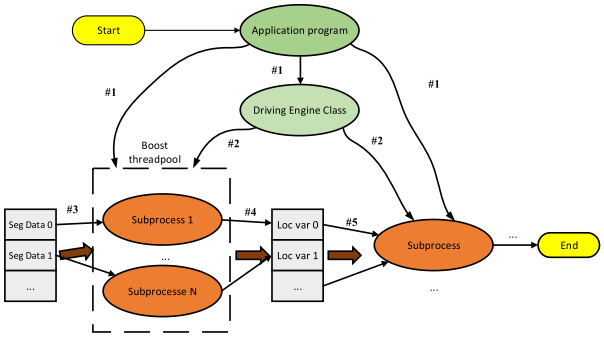

FIGURE 4. Multi-thread asynchronous schema schematic.

progress and status of slave threads in the thread pool, while slave threads in the thread pool are responsible for completing the actual GPU function calls and generating local solution results.

The calculation steps are shown in the sequence number. First, the program starts to generate the main thread and the slave thread pool. Then after the main thread allocates the task, the data operation is completed by the thread from the thread pool, and the local variable is saved and the final result is obtained. In the fifth step, the start-up condition of the reduction step is that the message bus class has acquired all the computed messages and allows the CPU to perform other operations while waiting.

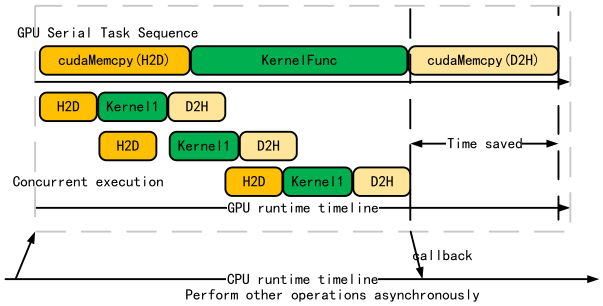

FIGURE 5. Flow chart of blocking and asynchronous operation.

Fig. 5 illustrates the timing comparison of GPU operations with streams. When GPU runs synchronously, once the amount of data transmitted is large, it may lead to the vacancy of GPU devices. If data segmentation is carried out in the initialization phase, the original data in a certain direction is divided into several small blocks, and data transmission can be hidden in the form of streams to improve the computational efficiency. Since Kepler architecture, Hyper Q technology enables multiple CPU cores or threads to start tasks concurrently on a GPU, which improves the utilization of GPU in the form of multiple work queues and avoids pseudo-dependency. The pseudocode is as follows:

The first step creates a stream object according to the pointer in the task structure, and obtains various hardware parameters of the computing device. The second to the eighth steps complete the process of memory copy, kernel function

Algorithm 2 The Multi-Stream GPU Kernels

Begin

```
CreateStream
While i<workloadNum
  cudaHostAlloc && cudamalloc
  cudaMemcpyAsync(H2D)
  Interpolate<<<grid, block, sharedSize,
  stream>>>(...)
  cudaMemcpyAsync(D2H)
  cudaStreamAddCallback(stream, callbackfunc,
  data, flag), i++
End while
```

End

calculation, function callback and capture data according to the given stream object. The third template parameter of step 5 is the size of the shared memory with low access delay. The shared memory space is dynamically allocated in the kernel function through the extern *__shared__* keyword to save precious first-level cache space on each Streaming multiprocessor(SM) in order to prevent kernel function call failed. In view of the fact that the largest three-dimensional aerodynamic coefficient data used in this paper does not exceed 1200 points, the linearized data can be put into shared memory to accelerate the reading procedure.

In order to ensure high-performance universality in the numerical algorithm with arbitrary complexity, the first two parameters in step 5 should be considered to have an impact on the utilization of the hardware. A SM has hardware limitations in terms of threads, threads and registers. The specific parameters vary with the GPU's computing power. Under the conditions given by the calculation mode and algorithm, the number of registers used in each thread is determined, and the size of the shared memory needed to be allocated depends directly on the size of the aerodynamic coefficient data table. Therefore, the second parameter of kernel function, namely the number of threads in each thread block, is the most influential factor for the theoretical efficiency of SM. According to CUDA occupation [42], the function of SM utilization associated with shared memory and warps in blocks is established, so the theoretical efficiency index table can be established according to the functional relationship in the program initialization phase. The theoretical efficiency index chart of each SM with 25 registers for a single thread is shown in Fig. 6.

To illustrate the advantages of asynchronous architecture over synchronous architecture, the GPU-based synchronization framework is selected as the benchmark. The asynchronous architecture represented by streams is added to compare with synchronous architecture. By comparing the time-consuming difference between them, it shows that asynchronous architecture is more advantageous in concurrent execution. At the same time, it compares the efficiency differences between executing multiple concurrent tasks, as shown

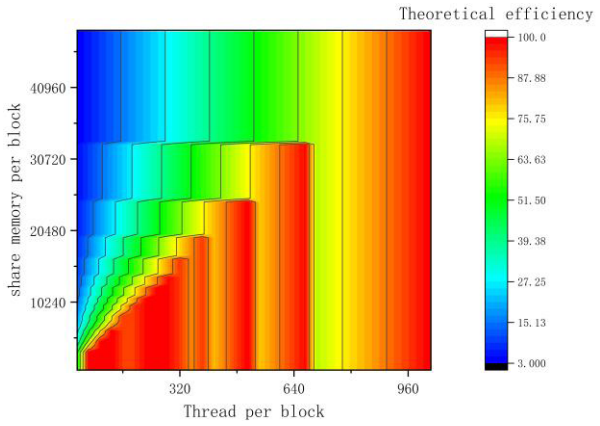

FIGURE 6. Schematic diagram of occupancy rate under different parameters.

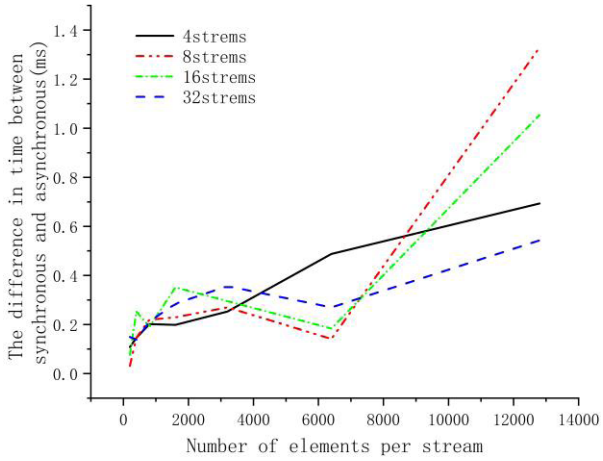

FIGURE 7. The time-consuming difference of calculating stiffness matrix of beam element between synchronous and asynchronous.

in Fig 7. The four polylines represent the time saved by asynchronous operation with the change of the number of units in the flow under different numbers of flows, in which the value is positive, indicating that the computation time of the asynchronous architecture is less than that of the synchronous architecture. As can be seen from the figure, the time-consuming difference generally decreases first and then increases, the data transfer latency of asynchronous computation can be better hidden, mainly because data transmission takes a larger proportion of the total time when the number of cells is small. As the number of cells increases gradually, the time-consuming proportion of kernel function calculation in total time increases, and the function of data transmission hiding decreases. With the increase of the number of cells, the time-consuming difference between synchronous and asynchronous architectures increases linearly with the increase of data transmission. Meanwhile, by comparing the number of streams, the performance improvement is lower when the number of streams is 32, mainly because the frequent scheduling and polling instructions of hardware devices affect the efficiency of asynchronous operation.

V. PERFORMANCE ANALYSIS

A. RUNNING TIME

A parallel solver is built on multi-GPU platform to simulate the flight process of slender elastic vehicle, and the parallel performance and extensibility of single GPU and multi-GPU is compared. Since the research of acceleration algorithm and asynchronous architecture is the focus of this paper, the aerodynamic coefficient data of a certain type of rocket is selected. Because the data is sensitive, it is not convenient to give real values, but for the sake of generality, a random variable is added to each dimension of aerodynamic coefficients. The hardware of the platform is Intel E3 1230 V5 with 3.4G Hz and two GeForce GTX1060 3G graphics cards. The programming environment is MSVC14.0 compiler under VS2017 and NVCC compiler under CUDA 9.0. CudaEvent_t is chosen as the timestamp for timing, which accuracy is 0.5 μs. Considering the data coupling relationship between the elastic module and other modules, 20ms is selected as the real-time criterion.

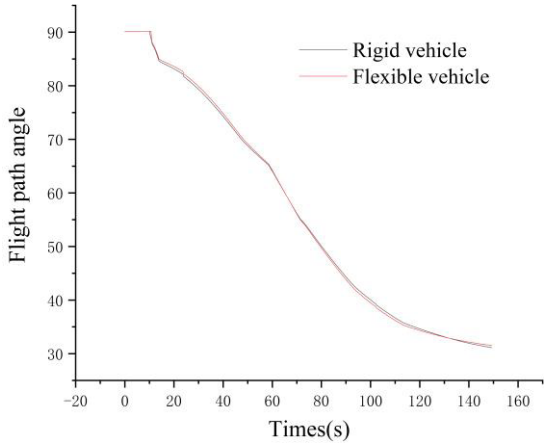

FIGURE 8. Comparison of flight path angle between rigid and elastic vehicle.

B. ANALYSIS OF COMPUTATIONAL ACCURACY AND EFFICIENCY

The correctness and rapidity of the program are important indicators to measure the performance of the program. A time-domain simulation example is selected from the moment of take-off to the first stage separation, the angle of flight path variation of the slender body aircraft is shown in Fig. 8. It can be seen that the change of flight path angle between rigid and elastic model is generally consistent. Because of the robustness of the control system, the aircraft can still satisfy the stability of flight attitude and ensure the tracking of the standard trajectory under certain deviation conditions. Therefore, the additional quantity caused by elastic deformation to the guidance and control computer has little effect on the trajectory. As shown in Fig. 9, by comparing the calculation results of CPU and dual-precision GPU operation, it can be considered that the operation accuracy of CPU and GPU is no different.

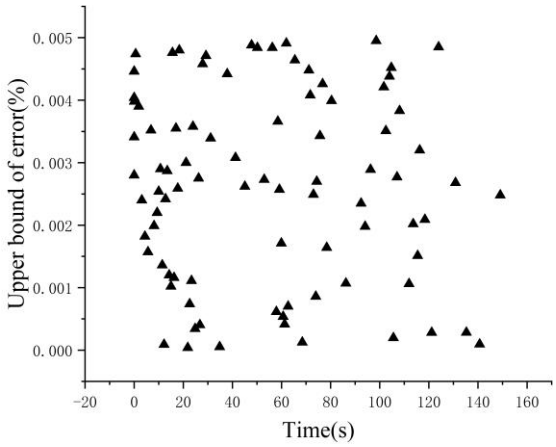

FIGURE 9. The flow chart of blocking and asynchronous operation.

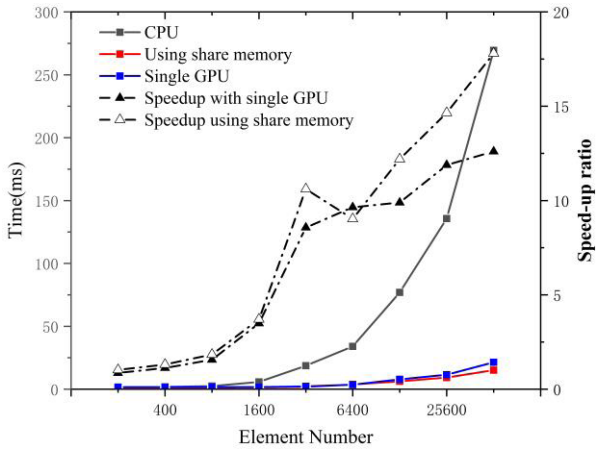

FIGURE 10. Comparison of the calculate results between CPU and GPU with different scales.

As shown in Fig. 10, the acceleration effect is not obvious when the number of nodes is small in single GPU case, because the computation needs to transmit the calculated data through low-speed PCI-E bus. If the number of threads started is not enough to hide the time overhead caused by data transmission and the kernel function, that is, the additional communication and function startup time brought by heterogeneous architecture cannot offset the performance improvement brought by GPU operation. Although the conventional Geforce card is chosen in this paper, it has no ECC memory and has fewer dual-precision computing units in each stream multiprocessor, the dual-precision computing using CUDA technology can still ensure a better acceleration ratio when the number of cells is larger, and the real-time simulation calculation can be guaranteed when the number of nodes is about 25,000 after optimization. As the number of nodes increases, the acceleration ratio reaches about 13 and the slope decreases steadily. While the number of aerodynamic coefficient data is small enough to be put into shared memory, the acceleration effect is not obvious when the sites are reduced. As the number of nodes increases, the calculation

is changed from I/O-intensive to computational-intensive, and the acceleration effect is gradually obvious. Unlike the conventional acceleration ratio of reading data from global memory, there is a downward trend in the number of nodes around 3000. The reason may be that multiple threads in the thread block access a shared memory address at the same time, resulting in memory access conflicts and thus affecting the operation efficiency.

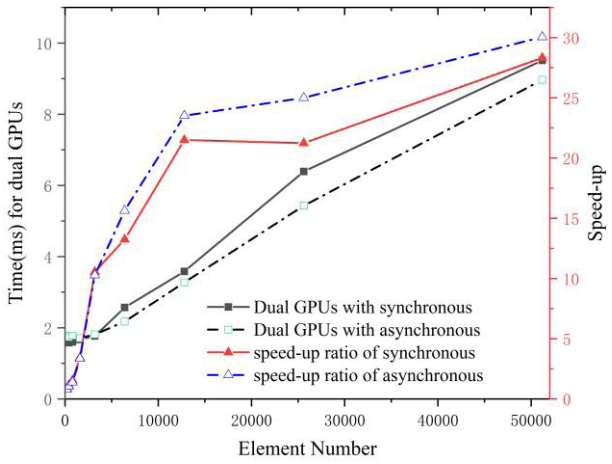

FIGURE 11. Time consuming comparison of multi GPU synchronous/asynchronous architecture with different scales.

Compared with synchronous design and multi-GPU, asynchronous design has little difference or even less efficiency when the data size is fewer, as shown in Fig. 11. The reason is that data partitioning and additional graphics resources are required in a multi-GPU environment. asynchronous operation needs more thread synchronization operation, which causes the acceleration ratio to be less desirable. As the number of nodes increases, the acceleration ratio of multi-GPU is faster and the acceleration effect is more obvious, even up to 50,000 sites, the acceleration ratio does not appear to be gentle. The reason is that the data splitting time accounts for a smaller percentage of the total duration. The proportion of transmission of the parallel part is gradually reduced compared with that of the computing operation, which leads to sufficient delay hiding, making the stream multi-processor more efficient and also proves that the scheme is suitable for solving large-scale numerical operations from the side.

VI. CONCLUSION

In this work, we demonstrate that rapid algorithm for solving the elastic module of slender vehicle. In the multi-GPUs environment, the FEM model of about 1200 elements with 40 order can be guaranteed to achieve real-time simulation. An index table of aerodynamic coefficients based on octree is proposed for GPU architecture. The dynamic parallel allocation of thread blocks is implemented and the index step of bilinear interpolation is optimized. A parallel algorithm for solving element stiffness matrix is designed, and the asynchronous heterogeneous architecture based on multi-GPUs and thread pool further improves the efficiency of the calculator. The results demonstrate the rapidity and scalability of the parallel architecture. As GPU based systems continue to scale, it will be interesting to completing real-time simulation of most complex models in the computing cluster environment.

REFERENCES

[1] P. Hongjun, H. Xianlin, and L. Yang, "Hypersonic vehicle aeroelasticity modeling and analysis based on the induced angle of attack," in *Proc. 32nd Chin. Control Conf.*, Xi'an, China, Jul. 2013, pp. 8798–8802.

[2] Y. He and L. Liu, "Effect of the connection position to the vibration characteristics for beams," in *Proc. Int. Conf. Mech. Elect. Technol.*, Sep. 2010, pp. 449–452.

[3] X. Lu, B. Wang, L. Zha, and Z. Xu, "Can MPI benefit Hadoop and mapreduce applications?" in *Proc. 40th Int. Conf. Parallel Process. Workshops*, Sep. 2011, pp. 371–379.

[4] S. Bai, E. Khosravi, and S.-J. Park, "An MPI-enabled mapreduce framework for molecular dynamics simulation applications," in *Proc. IEEE Int. Conf. Bioinf. Biomed.*, Dec. 2013, pp. 1–3.

[5] T.-C. Dao and S. Chiba, "HPC-reuse: Efficient process creation for running MPI and Hadoop mapreduce on supercomputers," in *Proc. 16th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGrid)*, May 2016, pp. 342–345.

[6] H. Asaadi, D. Khaldi, and B. Chapman, "A comparative survey of the HPC and big data paradigms: Analysis and experiments," in *Proc. IEEE Int. Conf. Cluster Comput. (CLUSTER)*, Sep. 2016, pp. 423–432.

[7] W. Kwedlo and P. J. Czochanski, "A hybrid MPI/OpenMP parallelization of K-means algorithms accelerated using the triangle inequality," *IEEE Access*, vol. 7, pp. 42280–42297, 2019. doi: 10.1109/ACCESS.2019.2907885.

[8] Y. Peng and F. Wang, "Cloud computing model based on MPI and OpenMP," in *Proc. 2nd Int. Conf. Comput. Eng. Technol.*, Apr. 2010, pp. V7–V85.

[9] D. Ediger and D. A. Bader, "Designing hybrid architectures for massive-scale graph analysis," in *Proc. IEEE Int. Symp. Parallel Distrib. Process., Workshops Phd Forum*, May 2013, pp. 2262–2265.

[10] C. Gómez, F. Martínez, A. Armejach, M. Moretó, F. Mantovani, and M. Casas, "Design space exploration of next-generation HPC machines," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2019, pp. 54–65.

[11] J. Sun, N. Guan, X. Wang, C. Jin, and Y. Chi, "Real-time scheduling and analysis of synchronous OpenMP task systems with tied tasks," in *Proc. 56th ACM/IEEE Design Automat. Conf. (DAC)*, Jun. 2019, pp. 1–6.

[12] G. Barlas, *Multicore and GPU Programming: An Integrated Approach*. Burlington, MA, USA: Morgan Kaufmann, 2014, pp. 11–37.

[13] C. Raphael, *Designing Scientific Applications on GPUs*. London, U.K.: Chapman & Hall, 2013, pp. 13–21.

[14] A. Bartezzaghi, M. Cremonesi, N. Parolini, and U. Perego, "An explicit dynamics GPU structural solver for thin shell finite elements," *Comput. Struct.*, vol. 154, pp. 29–40, Jul. 2015.

[15] D. Fleischmann, S. Weber, and M. M. Lone, "Fast computational aeroelastic analysis of helicopter rotor blades," in *Proc. AIAA Aerosp. Sci. Meeting*, Kissimmee, FL, USA, 2018, p. 1044.

[16] B. Baghapour, A. J. Mccall, and C. J. Roy, "Multilevel parallelism for CFD codes on heterogeneous platforms," in *Proc. 46th AIAA Fluid Dyn. Conf.*, Washington, DC, USA, 2016, p. 3329.

[17] Q. Dinh and Y. Marechal, "Toward real-time finite-element simulation on GPU," *IEEE Trans. Magn.*, vol. 52, no. 3, Mar. 2016, Art. no. 7207304. doi: 10.1109/TMAG.2015.2477602.

[18] A. Dziekonski, P. Sypek, A. Lamecki, and M. Mrozowski, "Communication and load balancing optimization for finite element electromagnetic simulations using multi-GPU workstation," *IEEE Trans. Microw. Theory Techn.*, vol. 65, no. 8, pp. 2661–2671, Aug. 2017.

[19] S. Cook, *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*. Amsterdam, The Netherlands: Elsevier, 2012.

[20] M. U. Ashraf, F. A. Eassa, A. A. Albeshri, and A. Algarni, "Performance and power efficient massive parallel computational model for HPC heterogeneous exascale systems," *IEEE Access*, vol. 6, pp. 23095–23107, 2018. doi: 10.1109/ACCESS.2018.2823299.

[21] T. Agarwal and M. Becchi, "Design of a hybrid MPI-CUDA benchmark suite for CPU-GPU clusters," in *Proc. 23rd Int. Conf. Parallel Archit. Compilation Techn. (PACT)*, Aug. 2014, pp. 505–506.
[22] J. Zhu, J. Li, E. Hardesty, H. Jiang, and K.-C. Li, "GPU-in-Hadoop: Enabling mapreduce across distributed heterogeneous platforms," in *Proc. IEEE/ACIS 13th Int. Conf. Comput. Inf. Sci. (ICIS)*, Jun. 2014, pp. 321–326.
[23] M. Reza, A. Sinha, R. Nag, and P. Mohanty, "CUDA-enabled Hadoop cluster for sparse matrix vector multiplication," in *Proc. IEEE 2nd Int. Conf. Recent Trends Inf. Syst. (ReTIS)*, Jul. 2015, pp. 169–172.
[24] N. Khare, A. Khare, and F. Khan, "HCudaBLAST: An implementation of BLAST on Hadoop and Cuda," *J. Big Data*, vol. 4, no. 1, 2017, Art. no. 41. doi: 10.1186/s40537-017-0102-7.
[25] V. Klôh, D. Yokoyama, A. Yokoyama, G. Silva, M. Ferro, and B. Schulze, "Performance and energy efficiency evaluation for HPC applications in heterogeneous architectures," in *Proc. Symp. High Comput. Syst. (WSCAD)*, Oct. 2018, pp. 162–169.
[26] J. Anderson, *Fundamentals of Aerodynamics*. New York, NY, USA: McGraw-Hill, 1984.
[27] T. Cheng, "Accelerating universal Kriging interpolation algorithm using CUDA-enabled GPU," *Comput. Geosci.*, vol. 54, pp. 178–183, Apr. 2013. doi: 10.1016/j.cageo.2012.11.013.
[28] P. K. Mohanty, M. Reza, P. Kumar, and P. Kumar, "Implementation of cubic spline interpolation on parallel skeleton using pipeline model on CPU-GPU cluster," in *Proc. IEEE 6th Int. Conf. Adv. Comput. (IACC)*, Feb. 2016, pp. 747–751.
[29] Y. Sa, "Improved bilinear interpolation method for image fast processing," in *Proc. 7th Int. Conf. Intell. Comput. Technol. Automat.*, Oct. 2014, pp. 308–311.
[30] R. Jambunathan and D. A. Levin, "Grid-free octree approach for modeling heat transfer to complex geometries," *J. Thermophys. Heat Transf.*, vol. 30, no. 2, pp. 379–393, 2016. doi: 10.2514/1.T4653.
[31] J. Cheng, M. Grossman, and T. Mckercher, *Professional CUDA C Programming*. Fishers, IN, USA: Crosspoint Boulevard Indianapolis, 2014.
[32] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Reading, MA, USA: Addison-Wesley, 2011, pp. 38–46.
[33] V. Kindratenko, *Numerical Computations With GPUs*. Urbana, IL, USA: National Center for Supercomputing Applications, 2014.
[34] K. Gupta and J. Meek, *Finite Element Multidisciplinary Analysis*, 2nd ed. Brisbane, QLD, Australia: AIAA Education, 2015, pp. 231–250.
[35] N. Bell and M. Garland, "Implementing sparse matrix-vector multiplication on throughput-oriented processors," in *Proc. IEEE Conf. High Perform. Comput. Netw., Storage Anal.*, Nov. 2009, pp. 1–11.
[36] N. Bell and M. Garland, "Efficient sparse matrix-vector multiplication on CUDA," Nvidia, Santa Clara, CA, USA, Tech. Rep., 2008.
[37] 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.*, Oct. 2017, pp. 1–4.
[38] R. Rakvic, R. Broussard, and H. Ngo, "Energy efficient iris recognition with graphics processing units," *IEEE Access*, vol. 4, pp. 2831–2839, 2016. doi: 10.1109/ACCESS.2016.2571747.
[39] H. Hong, L. Zheng, and S. Pan, "Computation of gray level co-occurrence matrix based on CUDA and optimization for medical computer vision application," *IEEE Access*, vol. 6, pp. 67762–67770, 2018. doi: 10.1109/ACCESS.2018.2877697.
[40] I. E. Venetis, N. Nikoloutsakos, E. Gallopoulos, and J. A. Ekaterinaris, "Towards the implementation of wind turbine simulations on many-core systems," in *Proc. 53rd Conf., AIAA Aerosp. Sci. Meeting*, Kissimmee, FL, USA, 2015, pp. 1–12.
[41] M. J. Saikia and R. Kanhirodan, "High performance single and multi-GPU acceleration for diffuse optical tomography," in *Proc. Int. Conf. Contemp. Comput. Inform. (ICI)*, Nov. 2014, pp. 1320–1323.
[42] *CUDA Occupancy Calculator*, NVIDIA, Santa Clara, CA, USA, 2016.

BINXING HU received the B.S. and M.S. degrees from the Kunming University of Science and Technology, Kunming, China, in 2010 and 2013, respectively. He is currently pursuing the Ph.D. degree in flight vehicle design with the School of Astronautics, Northwestern Polytechnical University. His current research interests include reentry guidance, and parallel programming and applications.

LI XINGGUO received the B.S., M.S., and Ph.D. degrees from Northwestern Polytechnical University, Xi'an, China, in 1987, 1990, and 1997, respectively. He is currently a Professor with the School of Astronautics, Northwestern Polytechnical University. His current research interests include flight dynamics, reentry guidance and control, failure diagnosis, and trajectory replanning.