Available online at www.ijpe-online.com

**International Journal of Performability Engineering**

vol. 14, no. 4, April 2018, pp. 717-728
DOI: 10.23940/ijpe.18.04.p15.717728

# Parallel Visualization of Flow Field based on Streamline Similarity

Bin Tang[a,*], Sipeng Sun[b], Rui Deng[a], and Yi Li[b]

[a]*College of Shipbuilding Engineering, Harbin Engineering University, Harbin, 150001, China*
[b]*College of Computer Science & Technology, Harbin Engineering University, Harbin, 150001, China*

**Abstract**

The streamline visualization method can intuitively reveal the nature and change law of flow field. However, the streamline visualization method easily causes the disorder of visualization effect, which is especially serious in the 3D flow field. With the improvement of calculation accuracy, the time consumed by calculation in the streamline generation process also increases accordingly, which impacts the visualization efficiency. In view of the above problems existing in the streamline visualization method, this paper researches the parallel visualization method of flow field based on streamline similarity. First, this paper judges the streamline shape similarity through the feature points and designs the method for judging 3D streamline similarity by combing the methods for judging the distanced-based similarity between streamlines. This reduces the streamline disorder phenomena in visualization results. Second, this paper utilizes the parallel computing technology to improve the computational efficiency. On the basis of seed point parallel strategy, this paper proposes the parallel task partition method that combines the equal partition of tasks with repartition of redundant tasks. In view of the blocking wait that exists in parallel judgment of similarity, this paper proposes the cyclic check method based on a double-caching line to avoid the blocking wait problem, obviously reducing the flow field visualization time and effectively improving the flow field visualization efficiency.

*Keywords*: flow field visualization; streamline; similarity measurement; parallel computing

(Submitted on December 25, 2017; Revised on February 2, 2018; Accepted on March 10, 2018)

© 2018 Totem Publisher, Inc. All rights reserved.

## 1. Introduction

As one of the important sub-topics of visualization in scientific computing, the flow field visualization is a field that has been researched positively by scholars at home and abroad for many years. The flow field visualization technology utilizes intuitionistic and visual graphic images to describe abstract spatial data [12] to better show the nature and change law of flow field. It is an important method for analyzing scientific computing data. The geometric shape visualization methods [6] include the geometric icon method (dot icon method), streamline method and stream surface method, etc. The streamline visualization method is widely applied because it can intuitively reveal the nature and change law of flow field. In recent years, a lot of research on the streamline placement methods has been made at home and abroad: Turk and Bands propose a kind of image-guided based method in the literature [11]. This method first defines an energy function. The flow field sampling is made by this energy function while the energy function value is the difference value between flow field image and low-pass filtering. The magnitude of difference decides the streamline distribution homogeneity.

Chen et al. propose a kind of local similarity (between streamlines) guided method in the literature [4] to complete the flow field visualization process, and this method defines a measurement method based on Euclidean distance between streamlines and streamline shapes. During the streamline generation process, the streamlines being generated and all streamlines having been generated are compared in similarity to consequently make the streamline distribution relatively even. Yu et al. first select the seed point location according to the significance in the literature [13] and then calculates the Euclidean distance of all points between streamlines. Then, they measure the similarity by calculating the mean distance from one streamline to the closest point of another streamline. Mcloughlin T et al. calculate the three attributes: streamline

* Corresponding author.
*E-mail address*: tangbin@hrbeu.edu.cn

curvature, torsion and sinuosity in the literature [10] and then conduct data normalization to add the normalized attribute values together. They take the sum of these attribute values as the basis for judging the streamline similarity. As for the flow field blocking caused by too much streamline seeding, Lu Daying et al. propose a kind of algorithm based on iterative closet point (ICP) and K-means in the literature [8] to group the streamlines. First, the ICP algorithm is utilized to compare the similarity between streamlines, and then the K-mean clustering algorithm is utilized to group streamlines and control the even distribution of streamlines.

## 2. Research on Streamline Generation Algorithm Based on Similarity

### *2.1. Calculation of Streamline Distance*

Each generated streamline can be expressed as one corresponding point set. Consequently, the judgment on similarity between two streamlines can turn into the judgment on similarity between two-point sets, as shown in Figure 1.

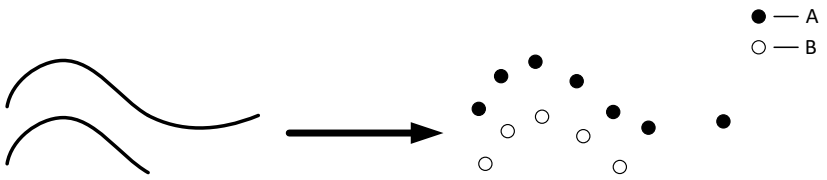


Figure 1. Streamline Point Set

The Hausdorff distance [7] can be utilized to discriminate the similarity degrees between different point sets by the overall shape of objects and differences between locations. So, this paper utilizes this distance to judge the similarity of geometric distance between two streamlines.

The original Hausdorff distance is easily affected by outliers, so it can impact the final results of streamline similarity comparison. This paper utilizes the thinking of statistical average to solve the problem. Realization methods:

For two streamlines A and B, their corresponding point sets are respectively $\{a_1, \dots, a_p\}$ and $\{b_1, \dots, b_q\}$. First, define the distance from one point in streamline A to streamline B as:

$$d_{min}(a, B) = \min_{b \in B} ||a - b|| \tag{1}$$

Where parameter $a$ stands for the point in the streamline A point set, parameter $b$, the point in the streamline A point set, and $d_{min}(a, B)$, is the minimum value from the point in the streamline A point set to the point in the streamline B point set.

So, we can define the mean distance from streamline A to streamline B as below:

$$h_m(A, B) = \frac{1}{P} \sum_{a \in A} d_{min}(a, B) \tag{2}$$

Where parameter $P$ stands for the number of points in streamline A.

Similarly, we can define the mean distance from streamline B to streamline A as below:

$$d_{min}(b, A) = \min_{a \in A} ||b - a|| \tag{3}$$

$$h_m(B, A) = \frac{1}{Q} \sum_{b \in B} d_{min}(b, A) \tag{4}$$

Where parameter $Q$ stands for the number of points in streamline B.

So, the distance between streamline A and streamline B can be defined as:

$$H_m(A,B) = \frac{1}{2}\left(h_m(A,B) + h_m(B,A)\right) \tag{5}$$

This paper takes the calculated value $H_m(A,B)$ as the standard for comparison of geometric similarities between streamlines. The larger the $H_m(A,B)$ value is, namely, the distance between two streamlines is relatively far, the lower the geometric similarity between the two streamlines is; the smaller the $H_m(A,B)$ value is, namely, the distance between two streamlines is relatively close, the higher the geometric similarity between the two streamlines is.

### *2.2. Spatial Attribute of Streamline*

The literature [5] points out that the 3D space curve has two very important attributes: curvature and torsion respectively. The two attributes can impact the shape of one space curve. Therefore, this paper introduces curvature and torsion into the measurement of similarity between streamlines.

The curvature stands for the bending degree of a curve, with the calculation method as below:

$$\kappa(p) = \frac{||\dot{V}(p) \times \ddot{V}(p)||}{||\dot{V}(p)||^3} \tag{6}$$

Where $\dot{V}(p)$ and $\ddot{V}(p)$ are respectively, the first and second derivatives of speed at point $p$.

The torsion stands for the twisting degree of the curve, with the calculation method as below:

$$\tau(p) = \frac{det\left[\dot{V}(p)\ ,\ \ddot{V}(p)\ ,\ \dddot{V}(p)\right]}{||\dot{V}(p) \times \ddot{V}(p)||^2} \tag{7}$$

Where $det$ stands for the determinant of matrix.

### *2.3. Similarity Judgment Process*

This section utilizes the total curvature [14] to extract the feature points of each streamline and then divides the streamline into multiple feature segments by feature points. It compares the local shape similarity between feature segments by the chi-square distance [9] according to the total curvature of each point at each feature segment and finally obtains the average value of chi-square distances of each feature segment of the two streamlines to conduct the final comparison of similarities between streamlines.

First, the calculation method for defining the total curvature is as below:

$$\mathrm{T}\kappa(p) = \sqrt{\kappa(p)^2 + \tau(p)^2} \tag{8}$$

Where parameter $\kappa(p)$ stands for the curvature of streamline of point $p$, and parameter $\tau(p)$ stands for the torsion of point $p$.

For all streamlines, the feature points of each streamline should be extracted, and the method and process for extracting the feature points on streamlines are as below:

1) Calculate the maximum total curvature $\mathrm{T}\kappa_{max}$ of the whole curve and record the location of the point;

2) Along the forward direction of streamline, take the point $\mathrm{T}\kappa_{max}$ of as the starting point $\mathrm{T}\kappa(p)$. Then, compare the total curvature of $\mathrm{T}\kappa(p+1)$ with that of $\mathrm{T}\kappa(p)$. If the total curvature of is smaller than that of $\mathrm{T}\kappa(p)$, take $\mathrm{T}\kappa(p+1)$ as the new starting point and then compare the total curvature of $\mathrm{T}\kappa(p+2)$ with that of $\mathrm{T}\kappa(p+1)$. Repeat this process until the total curvature of $\mathrm{T}\kappa(p+i)$ is greater than that of $\mathrm{T}\kappa(p+i-1)$, and take $\mathrm{T}\kappa(p+i-1)$ as one local extremal point;

3) Select $\mathrm{T}\kappa(p+i)$ as the starting point, and compare the total curvature of this point with that of the next point. Repeat and alternate the process until the total curvature of $\mathrm{T}\kappa(p+i+j)$ is smaller than that of $\mathrm{T}\kappa(p+i+j-1)$. Similar to Step 2, take $\mathrm{T}\kappa(p+i+j-1)$ as another local extremal point. Stop until the end point of the streamline appears and record the end point;

4) Along the reverse direction of streamline, take the point of T$\kappa_{max}$ as the starting point T$\kappa_{max}$, and conduct Steps 2 and 3. Stop until the starting point of the streamline appears and record the starting point;

5) For all checked extremal points, any point satisfying the condition where its total curvature is more than twice the minimum value of the total curvature of its two closest points is recorded as the feature point.

The streamline can be divided into multiple feature segments according to feature points extracted in the above steps, and this paper utilizes the chi-square distance to measure the differences between feature segments of streamlines. The measurement method of chi-square distance is as shown in Formula (9):

$$\chi^2(A,B) = \sum_n \frac{(P_n^A - P_n^B)^2}{P_n^A + P_n^B} \tag{9}$$

Where parameter $A$ and parameter $B$, respectively, stand for feature segments requiring to be compared, parameter $P_n^A$ and parameter $P_n^B$ respectively stand for the total curvature of point $n$ on feature segments $A$ and $B$, and parameter $n$ stands for the number of points on the feature segments.

Suppose one streamline has $n$ feature points, then the $n$ feature points can divide the streamline into $n-1$ feature streamline segments, and the calculation method for the final mean chi-square distance of two streamlines is as shown in Formula (10):

$$\chi_m^2(A,B) = \frac{1}{n-1}\sum_{i=1}^{n-1} \chi^2(A_i, B_i) \tag{10}$$

Where $\chi_m^2(A,B)$ stands for the finally obtained mean chi-square distance, parameter $A_i$ and parameter $B_i$ respectively stand for feature segment $i$ on streamline A and streamline B.

In the similarity comparison, it should be noticed that numbers of feature points extracted on two streamlines may be different. So, in the measurement of feature segments of two streamlines, we should select the streamline with less feature segments as the standard. Meanwhile, the numbers of points within the feature segments may be also different, and we should select the feature segment with fewer points as the standard, make equal partition of feature segments with more points according to those with fewer points, and then compare.

Two streamlines are different not only in shapes, but also in distance. So, this paper combines the measurement of streamline features and that of distance between streamlines as designed in the above section, and the designed judgment process is mainly divided into two steps:

1) First, calculate the $\chi^2$ value of each feature segment. Then, judge the overall shape similarity of streamline by calculating $\chi_m^2(A,B)$ and reserve the streamline in case of dissimilarity. Or, conduct 2;

Utilize Hausdorff distance to judge the similarity of geometric distance between streamlines. Abandon the streamline in case of similarity or add the streamline to the streamline set.

## 3. Parallelization Design of Streamline Generation Algorithm

In view of the problem of slow implementation of streamline generation algorithm based on similarity, this section analyzes the parallelization strategy for streamline generation and proposes the parallelization method for streamline generation.

### *3.1. Analysis of Parallel Strategy*

This paper utilizes the fourth-order Runge-Kutta method for integral of the streamline generation, and this method features high calculation accuracy and large calculation amount, with the calculation time accounting for 90% above. During the streamline generation process, all streamlines are mutually independent causing the streamline generations to have a high parallelism. This paper utilizes the seed point parallel strategy [1] as shown in Figure 2.


Figure 2. Seed Point Paralleling

### *3.2. Parallel Task Partition*

This paper utilizes the equidistant even seeding. In the 3D mesh data, seed points are arranged at a fixed distance. The parallel algorithm of domain decomposition is utilized in task partition because the calculations of all streamlines are mutually independent and unrelated to each other. First, utilize the domain decomposition method to decompose the definition domain into subdomains that are mutually independent. The partition methods of 3D grid mainly include three types [2] as shown in Figure 3:

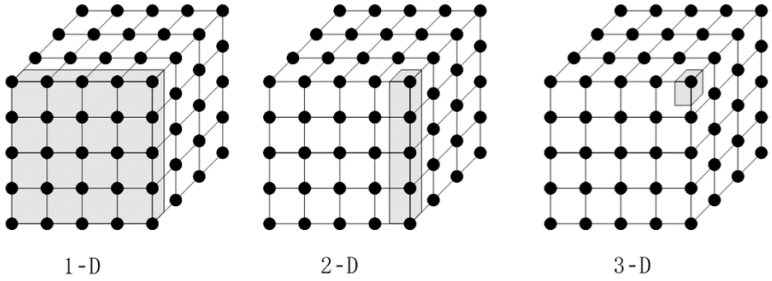


Figure 3. 3D Grid Partition Methods

Because of the huge grid amount of 3D flow field data as well as the limited number of processors, there may be several orders of magnitude in the difference between the two items. The 2-D and 3-D partition granularities in Figure 3 are relatively small, which may cause longer consumption in the task partition and impact the efficiency. So, the partition method as shown in 1-D in Figure 3 is utilized in the task partition. In the task partition, different tasks are allocated to different processors according to the processing capacities of processors to avoid leaving some processors unused. Under the multi-core environment, processing capacities of all processors are the same, so the partition method that is equal as much as possible may be utilized only to divide tasks to ensure the loads of all processors are relatively balanced. The partition methods in this paper are introduced specifically below.

### *3.3. Task Equal Partition Method*

Before task partition, the seed point grid dimension should be set first. For one 3D flow field, its dimension is $i * j * k$, and we select 10% of the minimum value of dimension as the seed point spacing. If $i < j < k$, we select 10% of as the standard to define the size of seed point grid dimension $l$、$m$、$n$, as shown in Formula (11).

$$\begin{cases} l = i \,/\, (i * 10\%) \\ m = j \,/\, (i * 10\%) \\ n = k \,/\, (i * 10\%) \end{cases} \tag{11}$$

Then, we select the maximum value of dimension of the seed point as the partition standard. If $n$ is the maximum value, we should make a one-dimensional partition according to the xy plane and divide the seed point into $n$ planes, each of which includes $l * m$ seed points as shown in Figure 4.

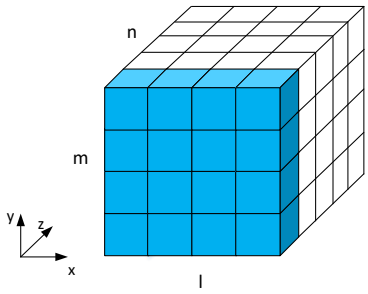

Figure 4. Seed Point Grid Dimension

We divide the seed point grid into $n$ data blocks. Suppose the total working thread number is $p_t$, and we number all threads $p_{id} \in [0,\ p_t - 1]$. To make the data block sizes the same as much as possible [3], we divide the partition methods into the following types:

If $n < p_t$, it shows that the task number is smaller than the thread number. At this time, we allocate $n$ tasks to the threads numbered $0 \sim n$ and the seed points allocated to each thread are $l * m$. This kind of task partition is relatively simple and rarely occurs, so we don't analyze it here in details. In actual situations, $n$ far exceeds $p_t$. If $n$ can be divided exactly by $p_t$, namely $(r = n \oplus p_t) = 0$, the technology partition method is relatively simple. We just allocate $n\ /\ p_t$ tasks to each thread. The seeds allocated to each thread are:

$$\text{SeedPoints}(p_{id}) = l * m * \frac{n}{p_t} \qquad p_{id} \in [0,\ p_t - 1] \tag{12}$$

For the convenience of seed point arrangement, we should also determine the coordinate of each seed point and first define the starting location for allocating tasks to each thread. Suppose$a = n\ /\ p_t$, the starting location for allocating tasks to each thread is:

$$\begin{aligned} \text{start}(p_{id}) &= p_{id} * a \\ \text{end}(p_{id}) &= (p_{id} + 1) * (a + 1) - 1 \end{aligned} \qquad p_{id} \in [0,\ p_t - 1] \tag{13}$$

The coordinates of all seed points corresponding to each thread can be defined as:

$$\text{SeedLocation}(p_{id}) = \begin{cases} x & x \in [0,\ l - 1] \\ y & y \in [0,\ m - 1] \\ z & z \in [\text{start}(p_{id}),\ \text{end}(p_{id})] \end{cases} \tag{14}$$

The above situation is that $n$ tasks can be allocated to $p_t$ threads equally. However, under different situations, the partition can't be of equal partition each time because the task number $n$ and thread number $p_t$change dynamically. In processing the situation, we can intuitively think that $n\ /\ p_t$ tasks are allocated to the previous $p_t - 1$ threads, and $n - (p_t - 1) * n\ /\ p_t$ tasks are allocated to the last thread. Then, the seed points allocated to each thread are:

$$\begin{aligned} \text{SeedPoints}(p_{id}) &= l * m * \frac{n}{p_t} \qquad p_{id} \in [0,\ p_t - 2] \\ \text{SeedPoints}(p_t - 1) &= l * m * \left[n - (p_t - 1) * \frac{n}{p_t}\right] \end{aligned} \tag{15}$$

This method can make the task allocation equal as much as possible. For example, suppose $n = 10$ and $p_t = 3$, then we can work out that 2 threads only execute one more task than other threads, and the task allocation among all threads are relatively balanced. But, this partition method may also cause too many tasks allocated to $p_t - 1$ threads, and the load of one single thread is too much. Suppose $n = 10$and $p_t = 4$, then the load of 3 threads in the allocation scheme is higher than that of other threads.

### *3.4. Repartition Method of Redundant Tasks*

To address the increased load of a single thread caused by too many tasks allocated to single threads in the previous section, this paper designs a kind of task partition method according to the changes of task number $n$ and thread number $p_t$. First,

calculate the average tasks allocated to each thread, and for the remaining task number $r = n \oplus p_t$, if $n$ can be divided exactly by $p_t$, execute the partition methods described in section 3.3 or the remaining task number $r$ is surely smaller than the thread number $p_t$; allocate the remaining $r$ tasks to the previous $r$ threads, and the previous threads $r$ are allocated with one more task to the last $p_t - r$ threads, namely, make task allocation by the following methods:

$$\text{SeedPoints}(p_{id}) = \begin{cases} l * m * \left(\frac{n}{p_t} + 1\right) & p_{id} \in [0,\ r-1] \\ l * m * \frac{n}{p_t} & p_{id} \in [r,\ p_t - 1] \end{cases} \tag{16}$$

The previous threads are allocated with one more task to the last $p_t - r$ threads before task partition. The above calculation formula can be changed into:

$$\text{SeedPoints}(p_{id}) = \begin{cases} l * m * \left\lceil \frac{n}{p_t} \right\rceil & p_{id} \in [0,\ r-1] \\ l * m * \left\lfloor \frac{n}{p_t} \right\rfloor & p_{id} \in [r,\ p_t - 1] \end{cases} \tag{17}$$

Suppose $a = n\ /\ p_t$, the starting location for allocating tasks to each thread is:

$$\begin{aligned} \text{start}(p_{id}) &\begin{cases} p_{id} * (a+1) & p_{id} \in [0,\ r-1] \\ \text{r} + \text{a} * p_{id} & p_{id} \in [r,\ p_t - 1] \end{cases} \\ \text{end}(p_{id}) &= \begin{cases} (p_{id}+1) * (a+1) - 1 & p_{id} \in [0,\ r-1] \\ \text{r} + \text{a} * (p_{id}+1) - 1 & p_{id} \in [r,\ p_t - 1] \end{cases} \end{aligned} \tag{18}$$

The coordinate of each seed point corresponding to each thread can be calculated by Formula (14), and we should only change the calculation methods of $\text{start}(p_{id})\ and\ \text{end}(p_{id})$ into the calculation method in this section. The combination of methods in section 3.3 and section 3.4 can map the tasks to each thread equally as much as possible and accurately position the seed points allocated to each thread. We can traverse within the definition domains successively.

### *3.5. Parallel Judgment of Similarity*

During the streamline generation process, the streamline generation algorithm based on similarity in this paper should compare the streamlines being generated with all streamlines having been generated in terms of similarity, so as to further reduce the streamlines and make even distribution of streamlines. For the similarity comparison of streamlines, all streamlines generated can be put into the buffer area. To ensure the correctness of streamline similarity comparison results, the streamline sets generated should be set as a critical area, and when the similarity comparison is being made for one thread, it can't be made for other threads. But, the utilization of this method may form blocking wait. When the number of utilized threads increases, the blocking wait time can also increase correspondingly, which will greatly impact the parallel efficiency.

For the situation of blocking wait, we can establish the private streamline set for each thread within each thread and compare it with the streamline set in the respective thread in the similarity comparison. In this way, each thread operates its own streamline set without blocking wait. But, the similarity comparison requires comparison with all streamlines, and this method can only compare with streamlines generated by the current threads, which can impact the final results. Therefore, after each thread generates streamlines, dissimilar streamlines should be sent to the streamline sets of other threads to ensure the streamline sets maintained by all threads are the same. While inserting streamline sets of other threads, if the streamline similarity comparison is being made for the inserted threads, the blocking wait is caused again.

For this situation, this paper proposes a kind of cyclic check method based on double-caching line to reduce blocking wait, and its design model is shown in Figure 5.

Take two threads (thread 0 and thread 1) for example. Two caching lines are set for each thread to store streamlines that are respectively in streamline set 0 and streamline set 1. The specific implementation steps are as follows:

1) Thread 0 generates one streamline and then accesses streamline set 0 for streamline similarity comparison. If streamline set 0 is under the streamline insertion operation or empty, access streamline set 1 for comparison of streamline similarity;

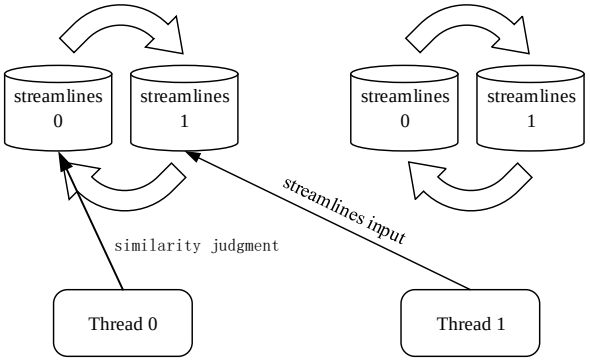


Figure 5. Cyclic Check Based on Double-caching Line

2) When the streamline should be inserted into the streamline set of thread 0 after judgment of streamline similarity in thread 1, insert the streamline to the streamline set 1 of thread 0 in priority. If streamline set 1 is under the streamline similarity comparison, access streamline set 0 and insert streamline to streamline set 0;

3) Access streamline set 0 to judge the similarity for thread 0, and if there is a similar streamline, abandon the streamline. Meanwhile, stop the judgment process, and execute step 1; or, record the streamline number after streamline set traversal as setSize0 and execute step 4;

4) Continue to access streamline set 1 to judge similarity for thread 0. The judgment is the same as that in step 3. If there is a similar streamline, abandon the streamline. Meanwhile, stop the judgment process, and execute 1; or, record the streamline number in streamline set 1 as setSize1 and execute step 5;

5) Check whether the size of streamline set 0 changes, namely, whether the size of streamline set 0 is equal to setSize0. If the size of streamline set 0 is equal to setSize0, it means that no new streamlines are inserted to streamline set 0 in similarity comparison. At this time, add the streamlines being judged to the thread streamline set, stop the check process, and execute step 1; if the size of streamline set 0 is not equal to setSize0, it means that new streamlines are inserted to streamline set 0 in similarity comparison. At this time, continue to make similarity comparison from setSize0−1 of streamline set 0. If there are any similar streamlines, abandon the streamline. Meanwhile stop the check and execute 1; or, update the size of setSize0 and execute step 6;

6) Conduct judgment in step 5 for streamline set 1 and change judgment of setSize0 into judgment of setSize1; namely, judge whether the size of streamline set 1 is equal to setSize1. If the size of streamline set 1 is equal to setSize1, add the streamlines being judged to the thread streamline set and execute step 1; if the size of streamline set 1 is not equal to setSize1, continue to make similarity comparison from setSize1−1 of streamline set 1. If there are any similar streamlines, abandon the streamline. Meanwhile, stop the check and execute 1 or update the size of setSize1 and execute step 5;

7) Cyclically execute step 5 and step 6 until the sizes of streamline set 0 and streamline set 1 don't change. Namely, when the size of streamline set 0 is equal to setSize0 and the size of streamline set 1 is equal to setSize1, stop the cyclic check and send the streamline to other threads;

8) Cycle the above process until all thread seed point sets are empty;

The blocking wait time existing in the similarity comparison can be reduced by the above design method during the parallel streamline generation process.

## 4. Experimental Verification and Result Analysis

### *4.1. Introduction to Experimental Environment*

Under the multi-core environment, this paper conducts hybrid programming based on Qt and VTK, achieves the serial algorithm and parallel algorithm generated by streamlines based on similarity, and completes the corresponding comparison tests. The hardware environment utilized by the experiments in this paper is as shown in Table 1.

Table 1. Experiment Hardware Environment

| Device name | System parameter |
|---|---|
| Lenovo | CPU：Intel(R) Core(TM) i5-6300HQ CPU @ 2.30GHz |
| | RAM：8GB 2133MHz DDR4 |
| | Hard Disk：500GB |

### *4.2. Experimental Verification and Analysis*

The data utilized in this paper are simulation experiment data of wind fields with different flow field structures and scales, as shown in Table 2:

Table 2. Experiment Data

| Experimental data | Data size | Format | Data type |
|---|---|---|---|
| Data 1 | 32*32*32 | binary | *.dat |
| Data 2 | 48*48*48 | binary | *.dat |
| Data 3 | 64*64*64 | binary | *.dat |
| Data 4 | 80*80*80 | binary | *.dat |
| Data 5 | 96*96*96 | binary | *.dat |
| Data 6 | 128*128*128 | binary | *.dat |

First, test the numbers of streamlines generated by not utilizing and utilizing similarity. The numbers of streamlines generated by utilizing are all 1,000, and the numbers of streamlines generated by all data after the utilization of similarity are as shown in Table 3:

Table 3. Streamline Number

| Experimental data | Data 1 | Data 2 | Data 3 | Data 4 | Data 5 | Data 6 |
|---|---|---|---|---|---|---|
| Streamline the number | 98 | 95 | 99 | 96 | 95 | 102 |

As seen from the data in Table 3, the number of generated streamlines is reduced after similarity judgment as well as for the test of visualization effect of streamline generation algorithm by not utilizing and utilizing the similarity. The visualization effect comparison is conducted by utilizing data 6, with the results as shown in Figure 6:

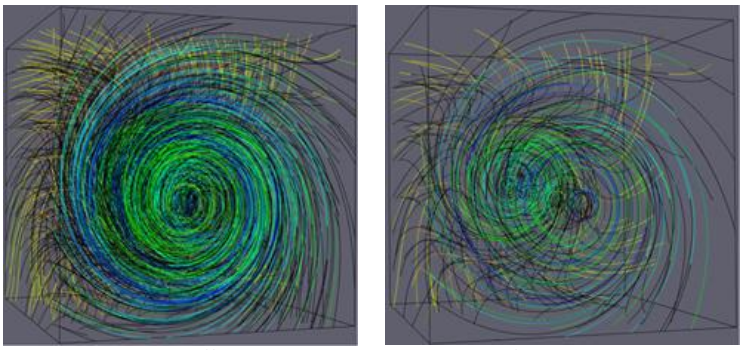

Figure 6. Comparison between Streamline Generation Effect by Not Utilizing Similarity (Left) and Streamline Generation Effect by Utilizing Similarity (Right)

As can be seen from Figure 6, the generated streamline number greatly reduces when comparing the streamline generation algorithm by utilizing the similarity (right) with that by not utilizing the similarity (left). The generated streamlines are reduced and meanwhile, the shorter streamlines are abandoned; the disordered shielding phenomena of visualization effect are reduced and the detailed information of flow field can be shown effectively without missing important information of flow field. This shows the effectiveness of streamline generation algorithm based on similarity.

To verify the effectiveness of parallel algorithm of streamline generation in this paper, we calculate the time of flow field data with different scales under serial and parallel environments with the specific operation time shown in Table 4.

Table 4. Execution Time of Serial and Parallel Algorithms (Unit: Second (s))

| Experimental data | Thread 1 | Thread 2 | Thread 3 | Thread 4 |
|---|---|---|---|---|
| Data 1 | 21.205 | 12.236 | 8.454 | 6.876 |
| Data 2 | 37.848 | 21.237 | 14.876 | 12.251 |
| Data 3 | 45.787 | 26.659 | 18.174 | 15.564 |
| Data 4 | 50.211 | 31.284 | 20.033 | 18.095 |
| Data 5 | 56.725 | 36.911 | 23.255 | 21.377 |
| Data 6 | 65.993 | 44.817 | 27.097 | 25.217 |

For the data obtained by the above experiments and tests, a longitudinal comparison can be made as shown in Figure 7. It can be easily found that: under the parallel environment, the visual calculation time increases with the increase of data scale under the condition of changeless thread number.

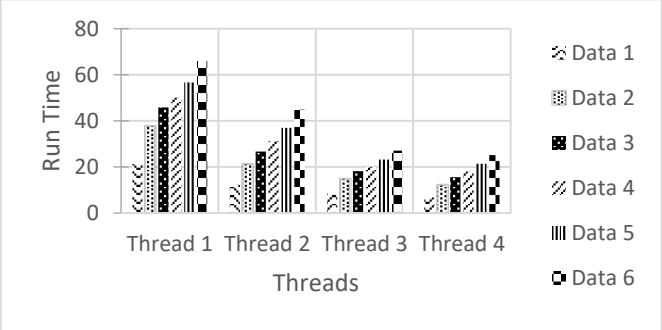


Figure 7. Longitudinal Comparison

Correspondingly, a horizontal comparison can be made, as shown in Figure 8, and it can be found that: under the parallel environment, the calculation time of visualization process decreases correspondingly with the increase of thread number under the precondition of changeless data scale. Furthermore, the acceleration effect becomes more obvious with the increase of thread number.

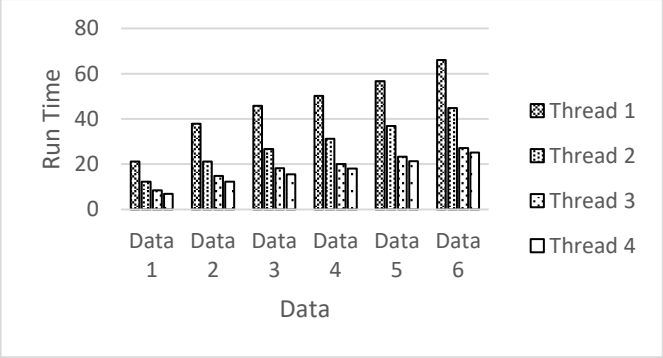


Figure 8. Horizontal Comparison

But, due to the impact of flow field structure, the increase of parallel efficiency of data with different scales may vary. As can be seen from the comparison between Figure 8 and Table 4, the increase of computational efficiencies of data 1 and data 2 is higher, and comparatively, that of data 5 and data 6 is slower, while the acceleration effect is still obvious. The speed-up ratio and efficiency of program are analyzed below, as shown in Figure 9 and Figure 10:

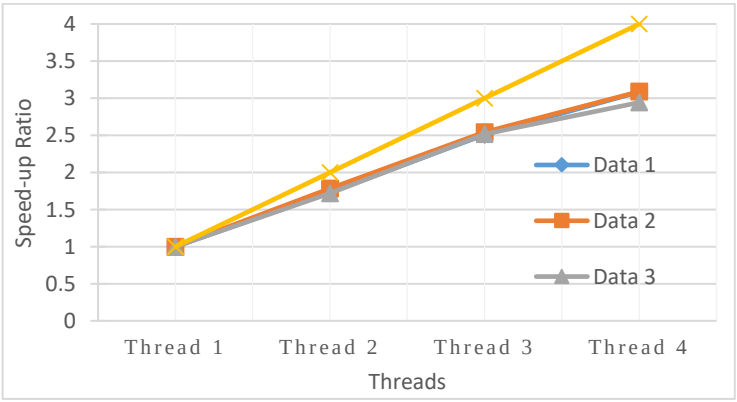


Figure 9. Relation between Speed-up Ratio and Thread Number

As can be seen from Figure 9, the parallel speed-up ratio fails to reach the ideal speed-up ratio due to extra overhead existing between threads in the program running. However, the speed-up ratio increases with the continuous increase of thread number, which shows that the computational efficiency also increases when the thread number increases. But, the

rising tendency of speed-up ratio begins to flatten with the increase of thread number. The reason is that the overhead of thread creation, thread destruction and synchronous communication between threads increase when the thread number increases, making its increase flatten. The acceleration effect is still obvious although the extra overhead exists.

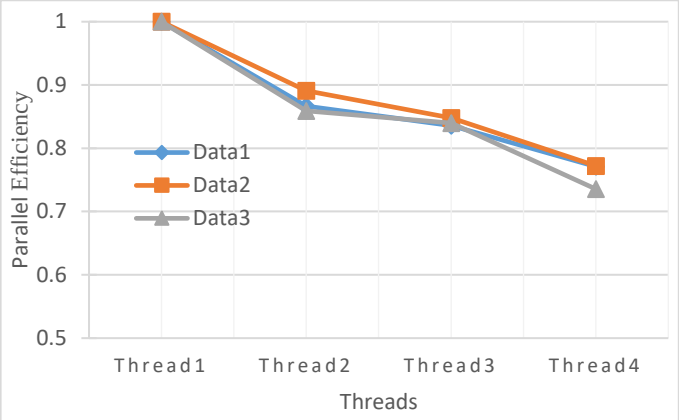


Figure10. Relation between Parallel Efficiency and Thread Number

Generally speaking, the parallel algorithm fails when the parallel efficiency is lower than 0.5. As can be seen from Figure 10, the extra overhead between threads increase correspondingly with the increase of thread number, resulting in gradual reduction of parallel efficiency. But, the parallel efficiency is still 0.75 or so at thread 4, which shows the effectiveness of parallel algorithm in this paper. The parallel efficiency can also express the extensibility of parallel algorithm, which further shows the parallel algorithm in this paper has a better extensibility.

The analysis of the above experiment results shows that the streamline generation process has good parallelism, and the computation time can be reduced under the parallel environment. The parallel method in this paper obtains a good speed-up ratio and parallel efficiency and can solve the problem of low computational efficiency during the visualization process.

## 5. Conclusions

The analysis of the above experiment results shows that the streamline generation process has good parallelism, and the computation time can be reduced under the parallel environment. The parallel method in this paper obtains a good speed-up ratio and parallel efficiency and can solve the problem of low computational efficiency during the visualization process.

## Acknowledgements

This research is supported by the National Natural Science Foundation of China (Grant No. 51679053; 51209048); Foundation of the Chinese People's Liberation Army (Grant No. 6140241010116CB01005).

## References

1. D. Camp, C. Garth, H. Childs, et al. “Streamline integration using MPI-hybrid parallelism on a large multicore architecture[J] ”, IEEE Transactions on Visualization & Computer Graphics, vol.17, no 11, pp. 02-13, 2011
2. G. L. Chen. Parallel Computing [M]. China Higher Education Press, pp.163-164, 2011
3. G. L. Chen. Parallel Computing Practice [M]. China Higher Education Press, pp.66-68, 2004
4. Y. Chen, J. Cohen, J. Krolik. “Similarity-Guided Streamline Placement with Error Evaluation”, IEEE Transactions on Visualization & Computer Graphics. vol. 13, no 6, pp.1448-1455, 2007
5. Z. Guo. “Conclusions on the Vertical Curvature and Torsion of the Corresponding Tangent of Two Curves”, Journal of Hetao College. pp.85-88, 2017
6. C. Li, L. D. “WU, B. Zhao. Visualization of 2D Dynamic Vector Field Based on Topology Analysis”, Journal of System Simulation. vol.28, no.10, pp.2385-2393. October 2016
7. S. Lu, Y. J. Lei, W. W. Kong, Y. Lei. “Image registration method based on key point feature and improved Hausdorff distance”, Systems Engineering and Electronics. vol.33, no.7, 1663-1667,July 2011
8. D. Y. Lu, D. M. Zhu, Z. Q. Wang. “Streamline Selection Algorithm for Three-Dimensional Flow Fields”, Journal of Computer-Aided Design & Computer Graphics. vol.25, no.5, pp.666-673 ,May 2013
9. Z. Y. Ma, S. j. Jia, Y. X. Zhang, “Chi-square Distribution Based Similarity Join Query Algorithm on High-dimensional data”, Journal of Computer Application.vol.36, no 7, pp.1993-1997, July 2016

10. T. Mcloughlin, M. W. Jones, R. S. Laramee, et al. “Similarity Measures for Enhancing Interactive Streamline Seeding”, IEEE Transactions on Visualization & Computer Graphics. vol.19, no 8, pp.1342-1353, 2013
11. G. Turk, D. Banks. “Image-guided streamline placement. Conference on Computer Graphics and Interactive Techniques”, ACM, pp.453-460, 1996
12. H. H. Wang. “Research on Key Techniques of Feature-Based Texture Visualization for Complex Flow Field”, Doctoral dissertation of China national defense science and technology university. pp.1-3, January 2015
13. H. Yu, C. Wang, C. K. Shene, et al. “Hierarchical Streamline Bundles”, IEEE Transactions on Visualization & Computer Graphics. vol. 18, no 8, pp.1353-67, 2012
14. Y. J. Zhu, L. S. Zhou, J. Wang. “Contour Extraction and Feature Point Detection for 3-D Fragment Reassembly”, Transactions of Nanjing University of Aeronautics and Astronautics. vol. 22 , no 1, pp.23-29, Mar 2005