Advances in Engineering Research, volume 140
2nd International Conference on Electrical, Control and Automation Engineering (ECAE 2017)

Dynamic Load Balancing Method Based on PBT Tree

Zhijie Lin[1], Chonghui Ren[2,*], Haifeng Huang[3] and Xiaoyong Li[1]
[1]Zhejiang University of Science and Technology, China
[2] The College of Computer and Science of Zhejiang University, China
[3]China Electric Power Research Institute, China
[*]Corresponding author

Abstract- With the development of computer technology and GPU technology, hardware and software technology has been greatly developed, but a single PC machine still cannot meet the real-time computing requirements of high-complexity scenes. In this paper, we propose a dynamic load balancing strategy based on PBT tree for the load balancing problem of Sort-first architecture cluster computing system. The dynamic load balancing strategy is dynamically adjusted in real time according to the change of computing scenarios, Balance caused by the system performance cannot be improved, make full use of system resources and improve the overall performance of the system.

Keywords-cluster; load balancing; sort-first; parallel computing

I. INTRODUCTION

Cluster computing technology, to some extent, solves the real-time computing problem of very large computing scenarios. However, there are still some issues that need to be studied, including cluster computing architecture, task allocation and load balancing, and network transmission. Based on Sort-first framework cluster computing system commonly existed in load balancing, this paper presents a PBT-based dynamic load balancing strategy. The average sort-first strategy of dividing tasks evenly according to the screen space leads to unbalanced load. In this paper, the load balancing strategy can be dynamically adjusted in real time according to the scene changes, and the system resources can be utilized to improve overall system performance.

Sort-first architecture: Sort-frist division tasks for parallel space, each node is responsible for parallel computing in a sub-block, respectively, to solve the sub-modules and then stitching the final calculation results. The main feature of sort-first is to distribute the model in the initial stage of computation. Each node implements its own computation pipeline of computation space. In the case of thinning and high sampling rate of screen block, the communication requirements are very low. Sort-first system load imbalance problem has greatly affected the system performance, how to design a good task and load balancing strategy is the Sort-first system, one of the most important problems to be solved [1].

II. RELATED WORK

A. *Load Balancing Mathematic Definition*

Mathematical description of load balancing:

$$T \equiv \max_{j} \sum_{i \in F^{-1}(j)} \omega_i \tag{1}$$

Where ω_i represents the computation required for computing the parallel elements of pixel i, F(i)→j is the mapping function of the parallel elements to the cluster nodes, $\sum_{i \in F^{-1}(j)} \omega_i$ represents the computation of the j-th node, as can be seen from Equation 1, the total time is determined by the node with the maximum computation Decided.

The goal of the load balancing problem is to find the F-function such that T is the minimum. When all p nodes in the same time to complete the computing task, T to obtain the minimum, this time,

$$T_{\min} = \sum_i \omega_i / p \tag{2}$$

So as to evaluate the quality of the function F that is load balancing degree can be expressed as follows:

$$\epsilon \equiv (T - T_{\min}) / T_{\min} \tag{3}$$

The load balancing algorithm is usually divided into two types according to whether the F mapping function changes at runtime: a static load balancing algorithm and a load balancing algorithm. Static load balancing builds F-mappings only on initialization, once built, no longer changes. Dynamic load balancing dynamically changes the F-map at runtime based on the load balancing status so that each time slice is as balanced as possible.

B. *Static Load Balancing Algorithm*

Static load balancing strategy is to allocate tasks when the system is initialized, in all tasks according to a certain strategy to the image of the various elements assigned to different nodes, each node independently to complete its assigned tasks, once assigned no longer Change, until the end of the program.

The simplest static load balancing strategy divides the entire task into equal-sized contiguous blocks, with each node responsible for one of them, as shown in Figure I:

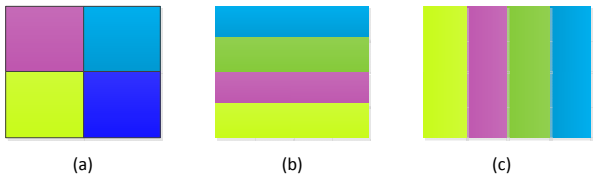

FIGURE I. STATIC LOAD BALANCING CONTINUOUS BLOCK PARTITIONING

This method is straightforward, but it is clear that this range of use is limited and inefficient when the geometric patch is unevenly distributed.

A more efficient method Scatter decomposition[2] distributes tasks randomly (pseudo-randomly) to each node in an attempt to load balance each node[3]. The most straightforward version of this approach is to use the mapping function F (i) = (i mod p). This method needs extra space to save the relative position of each sub-task when transferring data. When the task takes up a lot of space, it will greatly increase the network data transmission and reduce the processing speed.

Another popular strategy, tiling strategies[4], divides the overall task into equal-size sub-tasks and then uses random functions to map these sub-tasks randomly to each node, but the sub-task's square size is also determined The problem is that when the square is too small, the strategy degenerates into the above Scatter decomposition algorithm. When the square is too large, it becomes the simplest naive strategies.

C. *Dynamic Load Balancing Algorithm*

The Master-slave method divides the node into a master node and a slave node, wherein the master node is responsible for the dynamic screen space division and the other slave nodes are responsible for calculating the assigned task points. Unlike static load balancing algorithms, Scatter decomposition does not allocate all tasks all at once, but allocates a small block each time. When a node is idle, the master node will continue to allocate new tasks. This method can achieve a better load balancing, but the same problem with the same Scatter decomposition, transmission requires additional data.

The general idea of the Diffustion method [5-7] is to achieve partial load balancing by migrating a portion of the overloaded nodes to their non-overloaded neighbors (neighbors in the image space). This method is characterized by better local load balancing, but some local and local load imbalances may still occur. Task Stealing method [8] In turn, early completion of the work of the node will be from other nodes "steal" part of the task come. Robles algorithm [9] In addition to the heavy load of the task is divided into two, but also the task of combining light load. PBT-based algorithm [10] the general idea is similar to the Robles algorithm, which uses a data structure called PBT (Prediction Binary Tree) to maintain the current task space binary tree. Each leaf node corresponds to an actual task node. The strategy of dividing the next task is based on the PBT tree of the previous task. Because each PBT node records the time spent on the previous task, each update will divide the node with heavy load into two parts, and the load is light the two child nodes merge.

Based on the pre-processing method [11] in the first pass according to the number of tasks through the shader to estimate the amount of each task to estimate, and then based on the estimated value of the screen space bifurcation, that is broken down into with The number of nodes is equal. The method has some limitations, such as ray tracing and other complex algorithms cannot press this estimate.

III. PROPOSED METHOD

A. *Introduction to Prediction Binary Tree*

The PBT (Prediction Binary Tree) is a binary prediction tree that records a division of the current screen space. A leaf node in a binary prediction tree corresponds to a sub-block in the screen. PBT is based on the theoretical premise that there is space-time continuity for the same sub-block between frames, that is, the time required to calculate a sub-block in the next frame is very similar to that of the previous frame, so the time recorded in the previous frame can be used as the pre-valuation.

The root node r of a binary prediction tree represents the complete calculation task of the entire screen space. Each node has two child nodes, representing two sub-blocks divided by the current node, and the child nodes continue to be divided until the leaf node tree Equal to the number of cluster nodes. All leaf nodes, denoted by $L(T)$,Represents the actual division of screen space tasks, each leaf node $\ell \in L(T)$,In addition to saving the relative position and height of the current block, the time required for calculating the sub-blocks corresponding to the node is also recorded, which is recorded as $t(\ell)$ .In order to maintain the continuity and balance of the spatial division, the division is always divided along the longer side.

Figure II shows an example of a binary prediction tree for eight nodes:

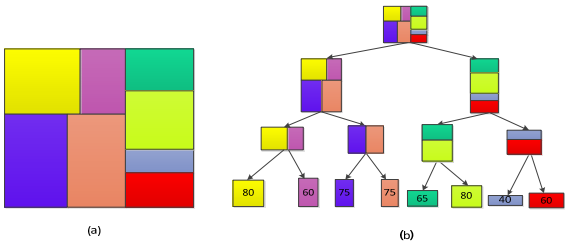

FIGURE II. BINARY PREDICTION TREE FOR EIGHT NODE

B. *Initialize Prediction Binary Tree*

In order to get as close to the load balancing state as soon as possible, an approximate estimate of the time required to calculate each pixel in the screen space during initialization is made, and then the screen space is divided based on these estimates. Of course, this estimation is rough, and the characteristics of the ray tracing algorithm also determine that it cannot be accurately estimated, because a ray emitted from the eye toward a pixel of the screen recursively with a few

slices of the scene to pay, the number of recursion is not sure in advance. On the other hand, because the proposed algorithm is based on feedback dynamic load balancing, it does not achieve accurate task partitioning at initialization, but slowly approaches the state of dynamic equilibrium according to dynamic adjustment, and the estimation at initialization reduces the arrival the steps and times needed for this status. The specific initialization process is as follows:

First of all, the model is projected to a screen space of size W * H according to the camera setting. For each pixel in the screen space, the pixel is marked as 0 and 1 according to whether or not a geometric patch falls on the pixel, recorded as $C(i, j)(0 \le i < H, 0 \le j < W)$ .For the upper left corner position is (x_0, y_0) ,the current block D of size w * h (assuming w> h).

$$E_D = \sum_D C(i, j) \tag{4}$$

Represents the estimate of the total time required to compute the current block D.

Divide D into W by pixel, $D_{(i,j)}$ represents D in the i-th column to the j-th sub-block.The goal of division is to find k, divide D into two part: $D_{(0,k)}$ (top left corner (x_0, y_0) ,size k*h),and $D_{(k+1,w)}$ (top left corner $x_0 + k + 1, y_0)$ size (w-k)*h),let k satisfy at the same time:

$$E_{D_{(0,k)}} \ge \tfrac{1}{2} E_D \tag{5}$$

and

$$E_{D_{(0,k-1)}} < \tfrac{1}{2} E_D \tag{6}$$

You can get k as follows:

k increases from 0 to w, and when k satisfies Equation 5 and Equation 6, it exits the loop, where k is the desired value.However, this method requires a priori value. For this reason, the first division needs to traverse all the pixels in D to obtain. There is a large amount of redundancy calculation in this method, and k can be found by traversing with a dynamic programming method once.

Let f(i) be the value of k corresponding to $D_{(0,i)}$, f(0) is initialized to 0, starting from the 0th column, updating $E_{D(0,f(i-1))}$ and $E_{D(0,i)}$ for each additional column, and judging whether the moving dividing line needs to be moved. If $E_{D(0,f(i-1))} \ge \tfrac{1}{2} E_{D(0,i)}$, then the value of f remains unchanged; otherwise The split line to the right one column, that is, f value plus 1. Mathematics is defined as follows:

$$f(i) = \begin{cases} 0, & i = 0 \\ f(i-1), & E_{D(0,f(i-1))} \ge \tfrac{1}{2} E_{D(0,i)} \\ f(i-1)+1, & E_{D(0,f(i-1))} < \tfrac{1}{2} E_{D(0,i)} \end{cases} \tag{7}$$

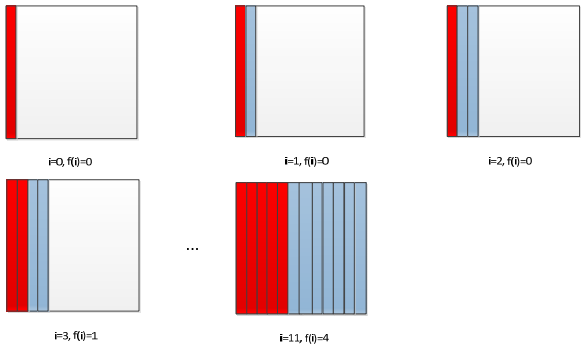

FIGURE III. EXAMPLE OF A FIRST SUBDIVISION OF A SUB-BLOCK OF WIDTH 12

After the completion of the first division, the two sub-blocks $D_{(0,k)}$ and $D_{(k+1,w)}$ are recursively recursively divided, and the sum of the estimated values $E_{D_{(0,k)}}$ and $E_{D_{(k+1,w)}} = E_D - E_{D_{(0,k)}}$ of the two sub-blocks has been obtained after the previous division. Therefore, at the When we ask for the k value of $D_{(0,k)}$ and $D_{(k+1,w)}$, we only need to add the evaluation sum of the currently traversed columns from left to right in one cycle, and then end the loop when traversing to the sum of the total is greater than the sum of undivided estimates k is the location of the required split line. When the number of subdivisions is equal to the number of nodes, the recursion ends. When the width is much larger than the width, the rows are divided according to the rows, just replace the rows and columns in the above algorithm.

Specific steps are as follows:

1) initialize k = 0;

2) Calculate the valuation of the k-th column $e_k = \sum C(k, j)$;

3) $E_{D_i} = E_{D_i} + e_k$

4) If $E_{D_k} \ge \tfrac{1}{2} E_D$,Then k is the desired split point, which is divided into two sub-blocks $D_{(0,k)}$ and $D_{(k+1,w)}$;otherwise k = k + 1, turn 2) to continue;

5) Repeat step 1 to 4 of $D_{(0,k)}$ and $D_{(k+1,w)}$ util the number of sub-blocks is equal to the number of cluster nodes;

During initialization, the result of each partition is recorded in the binary prediction tree. When the initialization is complete, a complete binary prediction tree is generated.

C. *Update Prediction Binary Tree*

While the PBT is initializing, the actual computing tasks are also assigned to the corresponding compute cluster nodes in a consistent manner as stored in the PBT. After each frame is calculated, the cluster management node records the time spent by each node in completing its computing tasks, for each leaf node $\ell \in L(T)$, Update its $t(\ell)$ value to t.

When the next frame begins to be calculated, the previous frame $t(\ell)$ can be used as the basis for task division. The division process is similar to the division strategy at initialization, except that $C(i, j)$ needs to be redefined. Here the value of $C(i, j)$ is no longer 0 or 1, but according to its i, j to PBT to find the $t(\ell)$ value of the sub-block where it is divided by the sub-block size w * h obtained value, namely:

$$C(i, j) = t(\ell(i, j)) / (w_\ell * h_\ell) \tag{8}$$

Where $\ell(i, j)$ represents a sub-block corresponding to a leaf node in a binary prediction tree (as shown in Figure 3.3), where a pixel p (i, j) in the screen space can be obtained by binary search of a binary prediction tree. With $C(i, j)$, we can regenerate a new binary prediction tree according to the partitioning algorithm after the first partition at initialization.

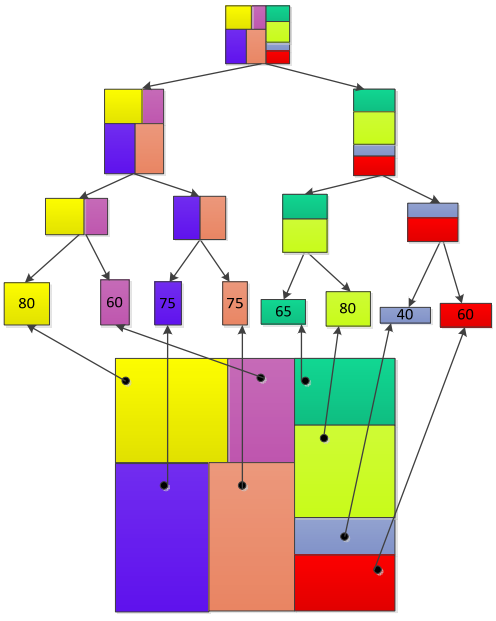

FIGURE IV. CALCULATED BY PBT

IV. EXPERIMENT

In order to test the impact of dynamic load balancing strategy based on PBT tree on the overall performance of the system, this paper tests the system performance without using load balancing and using load balancing, the test parameters are as follows:

Test environment: 8 computing nodes;

Test model: Li Buddha model (640,000 pieces);

Performance: Calculate the time spent in a frame (ms);

Figure 4.1 shows the comparison of the time spent on each frame with and without load balancing policies in 20 consecutive frames. The results show that using PBT-based dynamic load balancing strategy, the overall average performance of the system is improved about 51%, and more stable.

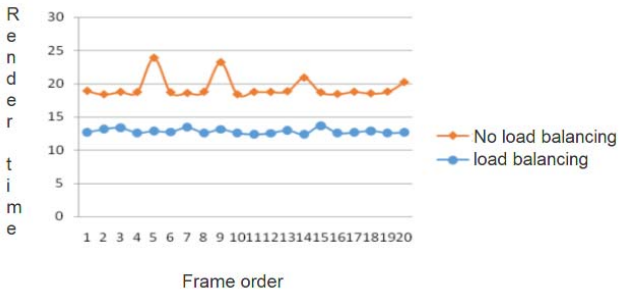

FIGURE V.1 LOAD BALANCING PERFORMANCE COMPARISON

V. CONCLUSION

According to the analysis of the unbalanced load in the Sort-first architecture, the load balancing problem has become one of the most important problems in the Sort-first architecture. Sort-first architecture combines PBT-based dynamic load balancing strategy with estimation and feedback method to achieve a good load balancing effect. Aiming at the problem of load balancing in Sort-first architecture cluster computing system, a dynamic load balancing strategy based on PBT tree is proposed. The average sort-first strategy of dividing tasks evenly according to the screen space leads to unbalanced load. Our load balancing strategy can be adjusted dynamically in real time according to the scene changes, so as to fully utilize the system resources and improve the overall system performance. Through the comprehensive application of the above technologies, real-time rendering speed and realism of cluster computing have been greatly improved.

VI. ACKNOWLEDGEMENT

The article was supported by the project of Intelligent control platform architecture and real-time transparent access technology (No. 2017YFB0902601) ",Supported by Science and Technology Plan Project of Zhejiang Province (2017C33176) and Technology, Zhejiang Provincial Natural Science Foundation (ly16f02008).

REFERENCES

[1] Mueller C. The Sort-First Rendering Architecture for High-Performance Graphics: Proceedings of the 1995 symposium on Interactive 3D graphics, 1995[C]. ACM Press.

[2] Salmon J. A mathematical analysis of the scattered decomposition: Proceedings of the third conference on Hypercube concurrent computers and applications: Architecture, software, computer systems, and general issues-Volume 1, 1988[C]. ACM.

[3] Eilemann S, Pajarola R. Direct send compositing for parallel sort-last rendering: ACM SIGGRAPH ASIA 2008 courses, 2008[C]. ACM.

[4] Wheat S R. A fine-grained data migration approach to application load balancing on MP mind machines[J]. 1992.

[5] Corradi A, Leonardi L, Zambonelli F. Diffusive load-balancing policies for dynamic applications[J]. Concurrency, IEEE, 1999,7(1):22-31.

[6] Heirich A, Taylor S, CALIFORNIA I O T P. A parabolic load balancing method: Proceedings of the 24th International

[7] Conference on Parallel Processing, Milwaukee, 1995[C].1995.

[8] Heirich A. A scalable diffusion algorithm for dynamic mapping and load balancing on networks of arbitrary topology[J]. International Journal of Foundations of Computer Science, 1997,8(3):329-346.

[9] Blumofe R D, Leiserson C E. Scheduling multithreaded computations by work stealing[J]. Journal of the ACM (JACM), 1999,46(5):720-748.

[10] DOUGLAS R A. Load Balanced Parallel Scanline Z Buffer Algorithm for the iPSC Hypercube: Proceedings of Pixim, 1988[C].

[11] Cosenza B, Cordasco G, De Chiara R, et al. Load Balancing in Mesh-like Computations using Prediction Binary Trees: 2008 International Symposium on Parallel and Distributed Computing, 2008[C]. IEEE.

[12] Whitman S. Dynamic load balancing for parallel polygon rendering[J]. Computer Graphics and Applications, IEEE, 2002, 14(4):41-48.