Research on Dynamic Load Balancing Algorithms for Parallel Transportation Simulations

Dongliang Zhang[1,2], Changjun Jiang[1,2], and Shu Li[1,2]

[1] Department of Computer Science and Technology, Tongji University, Shanghai, China, 201804
[2] The Key Laboratory of "Embedded System and Service Computing", Ministry of Education, China, Shanghai 201804

Abstract. To the issue of dynamic load balancing in parallel transportation simulations, we describe two algorithms for different types of task partitions, parallel lines partition and grid partition. In the algorithms, load balance is obtained by iteratively moving the boundary lines according to the relative balance of adjacent sub-domains. Assuming real traffic distribution as the experimental work load, we test the performance of the algorithms. And the result we observe confirms the value of the methods. Based on the discussion of the communication overheads under different types of partitions, due to the relative small amount of boundary lines, grid partition can decrease the communication overheads and is a more adaptive partition model.

Keywords: load balance, parallel computing, transportation simulation.

1 Introduction

As an important component of Intelligent Transportation Systems (ITS), the large-scale transportation simulation system is widely used in researching and designing transportation control systems. However, a large scale transportation simulation needs a large amount of calculation. The development of distributed and parallel computing technology provides sufficient calculating resource for traffic simulation. In recent years, the research on parallel traffic simulation has gained much progress [1, 2, 3].

In parallel computing, load balancing is an effective way to minimize the processing time and maximize the utilization of calculation resource [5, 6]. In a distributed and parallel transportation simulation system, the work loads of processors are often unbalanced, and so the efficiency of calculation is quite low. The main task of a transportation simulation is to simulate the behavior of vehicles on the road network. And in a view of large scale, the simulation of transportation is much similar to that of molecular dynamics [4]. We can take vehicles as molecules with special moving rules in a two dimensional square. The typical approach to parallelizing these computations is to decompose the spatial region into sub-domains, and associate each sub-domain with a processor. The computational work for a given processor at each time step depends on the number of vehicles in the corresponding spatial domain. Due to the ununiformity of the traffic load in the real word, the simulation tasks differ between

M. Xu et al. (Eds.): APPT 2007, LNCS 4847, pp. 560–568, 2007.

sub-domains. In the simulation, when a vehicle is running across the boundary of two sub-domains, the source processor needs to pass it to the destination processor. So the transfer of vehicles changes the distribution of simulation work loads during the simulation procedure. To balance the loads between processors, the boundary lines need to be relocated.

In this paper, we partition the traffic map using parallel lines and quadrilateral grids separately, and for these two kinds of partition two corresponding algorithms are proposed. In the algorithm for parallel lines partition, our approach is to adaptively repartition the space by moving the vertical lines. The algorithm is introduced for determine how to move the given vertical line depends on the relative loads of sub-domains which have that line in common. And in the algorithm for quadrilateral grids partition, our approach is to repartition the space by moving the vertex in the grid. And the algorithm is focused on how to move a given vertex depends on the relative loads of sub-domains which have that vertex in common. These lines or vertex movement, as well as the transfer of any vehicle from one processor to another, can be carried out in parallel. Furthermore, we discuss the communication overheads induced by the algorithms, and the adaptability of each algorithm.

The rest of the paper is organized as follow: in section 2 we describe the algorithms for parallel lines partition and quadrilateral grids partition separately. In section 3 we present testing results of the two algorithms. Finally, section 4 provides the performance of the algorithms and some conclusion comments.

2 Algorithms

2.1 Load Balancing Algorithm Using Parallel Lines Partition

Using parallel vertical lines to partition the simulation region, just like using longitude to divide the map, has an obvious advantage in minimizing the message channel between processors. Using this type of partition, every processor has utmost two adjacent processors to communicate. And the decrease of message channels may cut down the communication time greatly at some occasions. Figure 1 illustrates the basic model of parallel lines partition.

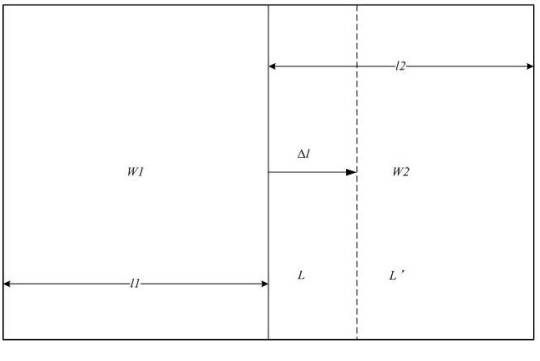

Fig. 1. This figure shows the basic model of parallel lines partition

We assume the total work load is *W*. And we use an arbitrary vertical line *L* to divide the work load into *W1* and *W2*, let $\overline{W}$ and $W_{\max}$ be the average work load and maximize work load among the processors separately. Then we define $I_l = 1 - \overline{W} / W_{\max}$ be the measure of imbalance of the workloads, and let I_{ideal} be the ideal imbalance. Obviously, if the workloads are evenly distributed, I_l would be quite small. Then we define another component:

$$\delta = \left| \frac{W_1 - W_2}{W} \right|$$

Here δ determines which side has the heavier load between two sub-domains. Then define:

$$\Delta l = (1 - \varepsilon)\delta \times l2 \qquad (0 < \varepsilon < 1)$$

Here Δl is the shift of *L* and ε is a parameter which controls how aggressively we seek to the balance.

We need to make several remarks:

1. When moving the line *L*, the work loads change monotonously and we can surely find an optimal position for balance.
2. When using more than one line, we can firstly adjust the lines which have an odd index, and then adjust the even lines.

The main steps of the algorithm are:

Step I. Calculate I_l according to the method mentioned above, if $I_l < I_{ideal}$, stop the algorithm, otherwise turn to S*tep II*.

Step II. For each odd line, calculate Δl.

Step III. Adjust *L* by Δl, and recalculate the work loads of the two sides.

Step IV. For each even line, calculate Δl.

Step V. Adjust *L* by Δl, and recalculate the work loads of each part, then turn to *Step I.*

2.2 Load Balancing Algorithm for Quadrilateral Grids Partition

Using quadrilateral grids or triangle grids to partition the simulation region is a widely accepted method. In this paper, we choose quadrilateral grids to divide the traffic map, and we describe the algorithm based on the basic partition model shown in figure 2.

As is illustrated in fig.2, Let *P* be an arbitrary grid point, and *E1*, *E4, E2, E3* are the neighbor grid points. Let the work loads of the four grid area which shares *P* be *W1*, *W2*, *W3*, *W4*, separately, and let $\overline{W}$, $W_{\max}$ be the average and maximize workload separately. Then the definition of imbalance is quit similar to that of prior algorithm: $I_l = 1 - \overline{W} / W_{\max}$. And let the ideal value of I_l be I_{ideal}.

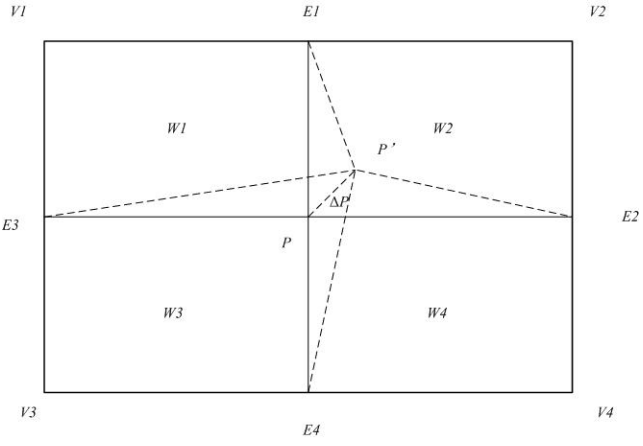

Fig. 2. The basic model of quadrilateral grids partition

Obviously, if we adjust P to P', $P'V1$, $P'V2$, $P'V3$, $P'V4$ will repartition the region, and so the work load of each sub-domain will be modified.

We define parameter $\delta 1$:

$$\delta 1 = \frac{(W1+W2)-(W3+W4)}{W1+W2+W3+W4}$$

Here $\delta 1$ presents which pair has the heavier load, the up pair or the down pair. Then we let *pE1* denotes the vector from *P* to *E1* and *pE4* be the vector from *P* to *E4*, then a vector $offset1$:

$$offset1 = \begin{cases} |\delta 1|\, pE1 & \delta 1 > 0 \\ |\delta 1|\, pE4 & \delta 1 < 0 \end{cases}$$

In the same way , using *W1+W3* and *W2+W4* , we define $\delta 2$, vector *pE2*, *pE3*, and $offset2$. Then the offset of point *P* is defined as follow:

$$\Delta P = (1-\varepsilon)\frac{offset1+offset2}{2}$$

Note that we move the point *P* by vector *pE1, pE2*, *pE3* and *pE4*, and the four sub-domains will remain convex.

Obviously, only by moving *P* will not necessarily reach the balance status. For example, if $W1 \approx W4 \gg W2 \approx W3$, we must adjust the locations of the points *E1*, *E2*, *E3*, *E4*. The movement of these points is quite similar to that of *P*. We take *E1* for example, define:

$$\delta_{E1} = \frac{W1-W3}{W1+W3}$$

And let *E1V1* be the vector from *E1* to *V1* and *E1V2* be the vector from *E1* to *V2*, and then the vector $offsetE1$ and the offset $\Delta E1$ are defined as follow:

$$offsetE1 = \begin{cases} |\delta 1|\, pE1 & \delta 1 > 0 \\ |\delta 1|\, pE4 & \delta 1 < 0 \end{cases}$$

$$\Delta E1 = (1-\varepsilon) offsetE1$$

The adjustment of *E2, E3* and *E4* is similar to that of *E1*, and we do not make any further discussion.

Based on the definition above, for a partition of $M \times N$ grid, we number all the points on the edges and inside of the region as $P_{ij}(0 \le i \le M, 0 \le j \le N)$ by its location. And all the work loads are numbered as $Wij(1 \le i \le M, 1 \le j \le N)$. For all the points, we apply a red-black coloring scheme to adjacent points to adjusting their location. We choose this type of scheme because in this strategy multi-processors can work synchronously without any conflict.

The algorithm is described as follow:

Step I. Calculate the work loads of all the sub-domains, and the imbalance degree I_l. If $I_l < I_{ideal}$, stop the algorithm, else turn to *Step II*.

Step II. For all the inside points, P_{ij} $(1 \le i \le M-1,\ 1 \le j \le N-1)$, choose the ones in odd rows and odd columns and the ones in even rows and even columns. Adjust these points themselves and the adjacent points.

Step III. For all the inside points, choose the ones in odd rows and even columns and the ones in even rows and odd columns. Adjust these points themselves and the adjacent points. And then turn to *Step I*.

3 Experimental Result and Analysis

We take the real traffic map of Shanghai China as the experimental task region. And we present the real distribution in a 1024×1024 bit map, in which each colored pixel stands for a number of vehicles. From white to red the color of each pixel stands for the density of vehicles from low to high. The balancing result based on the parallel lines partition is illustrated in figure 3.

Figure 4 illustrates the detail of the balancing procedure and the distribution of work loads between processors.

Figure 5 and figure 6 illustrate the balancing result based on a 4×4 quadrilateral grid partition.

From the experimental result we can see that using parallel lines to partition the domain, the balancing algorithm can gain a rather good result. But this type of partition may lead to big communication overheads between some processors. So in order to compare the algorithm, we design another model to estimate the communication overheads between processors.

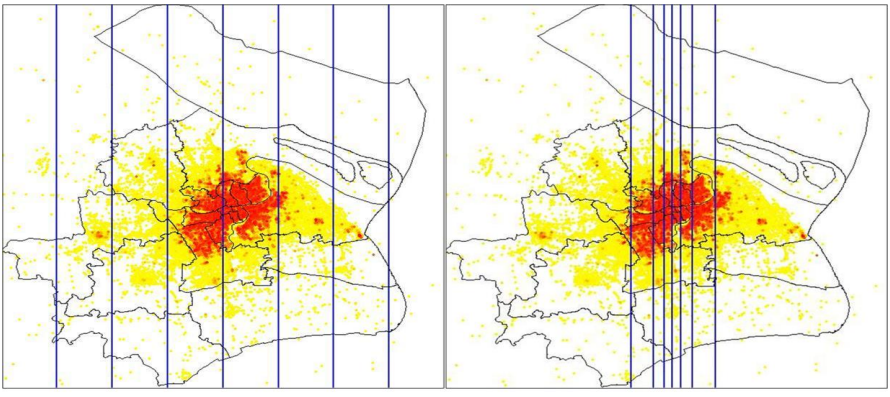

Fig. 3. This figure shows the status before balancing (a) and 100 algorithm cycles later (b) based on parallel lines partition

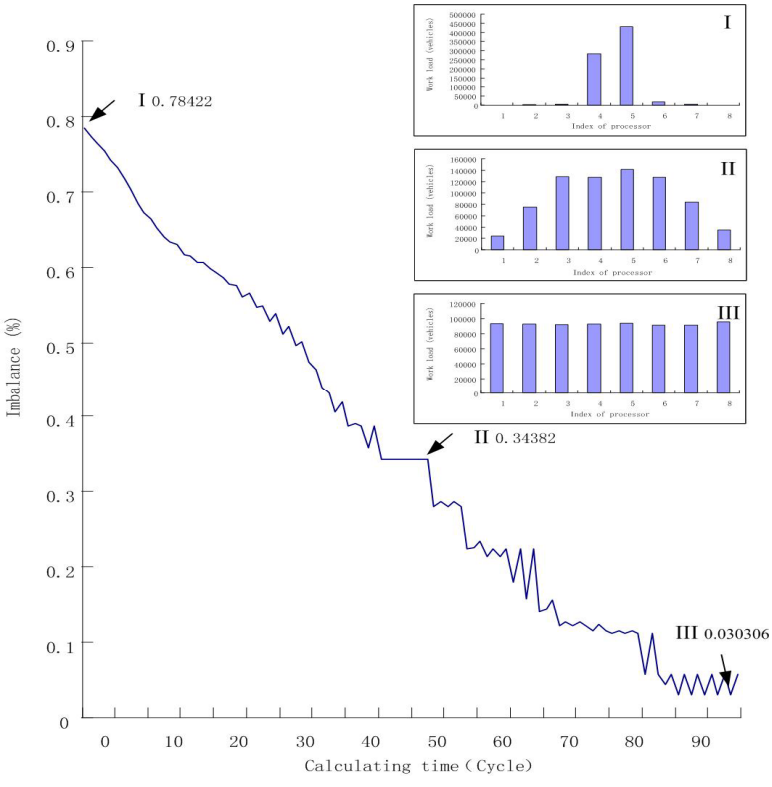

Fig. 4. This figure shows the relationship between the imbalance and calculating cycles ($\varepsilon = 0.9$) in 100 cycles based on parallel lines partition. Inset in this figure are three histograms showing the initial, midpoint, and final load distributions.

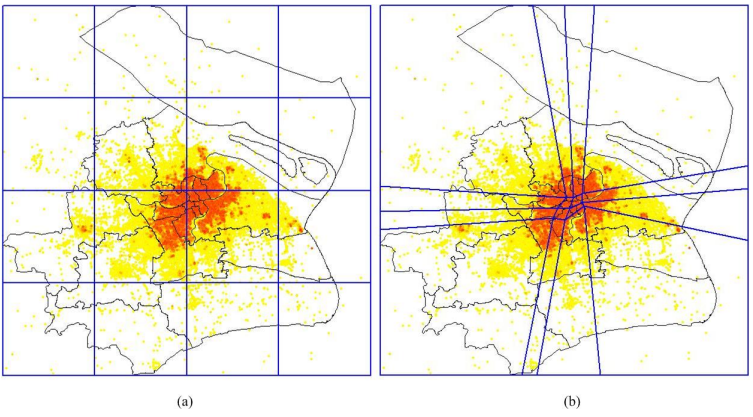

Fig. 5. This figure shows the status before balancing (a) and 32 algorithm cycles later (b) based on a 4×4 grid partition

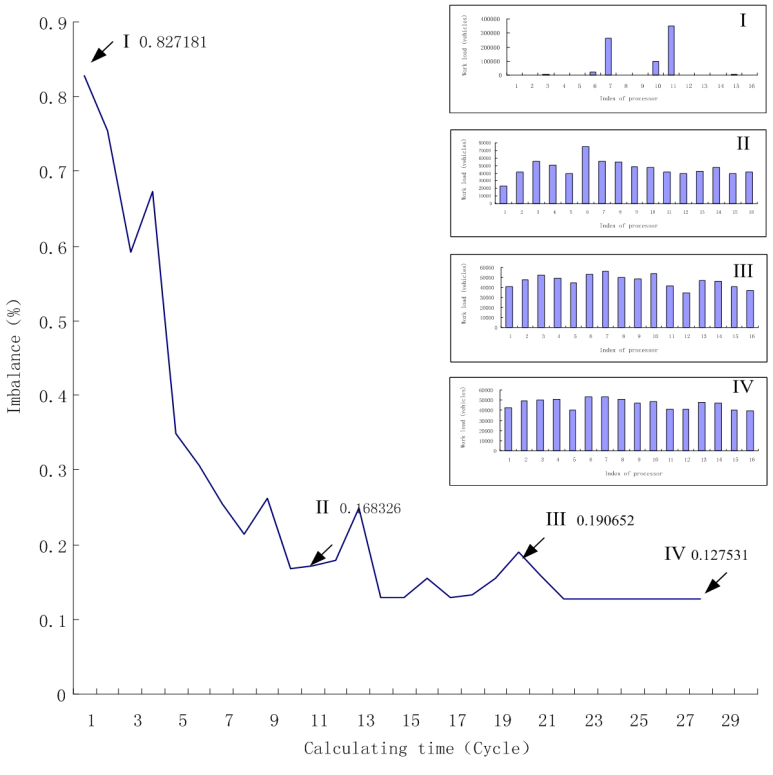

Fig. 6. This figure shows the relationship between the imbalance and calculating cycles ($\varepsilon = 0.9$) in 100 cycles based on a 4×4 quadrilateral grid partition. Inset in this figure are four histograms showing the initial, midpoints, and final load distributions.

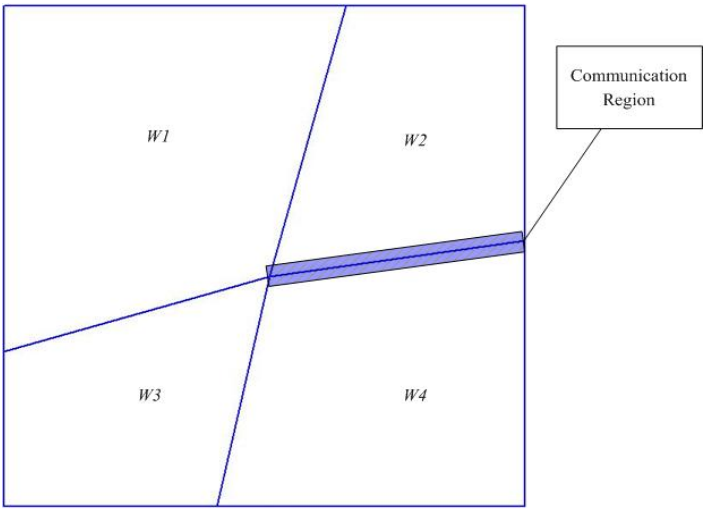

Fig. 7. This figure shows the basic communication overhead estimating model

In the parallel simulation of transportation, vehicles may run across the boundary lines and need to be transferred from one processor to another. So the communication overheads can be estimated by accounting the number of vehicles near the boundary lines. As is shown in figure 7, we name the adjacent area of a partition line a communication region, and by accounting the vehicles inside the area we estimate the approximate communication overhead for each processor. In the experiment we set the width of the communication region to two pixels in the bit map. Figure 8 illustrates the result on the communication overheads estimation.

Through calculating the amount of vehicles in the communication areas, we estimate the communication overhead between each pair of adjacent processors. And as

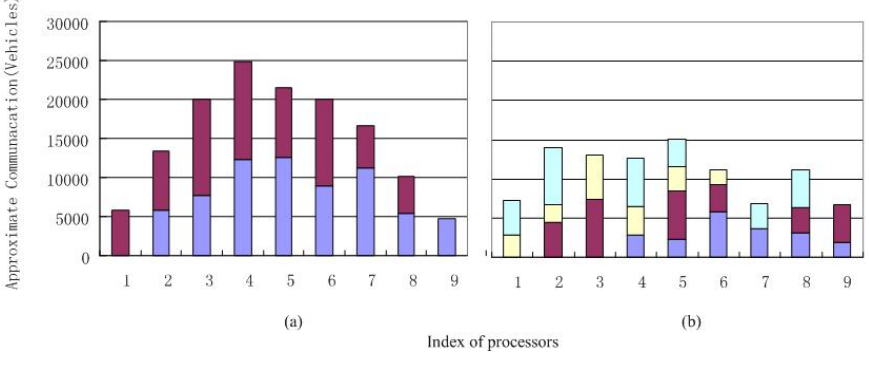

Fig. 8. This figure shows the communication overhead of 8 parallel lines partition (a) and 3×3 grid partition (b), different colors represent the communication overheads with different processors.

illustrated in figure 8, we use different colors to represent the communication with different processors. In figure 8(a), we can see the communication of each processor is composed of utmost two parts, but is much larger than that in figure8(b).

In figure 8(b), the communication overhead of processor 5 is composed of four parts, because it is in charge of the central sub-domain in the 3×3 grid. However the over communication head is not very large. This is because the four boundary lines of its sub-domain are quite short after balancing.

4 Conclusion

On the problem of dynamic load balancing in parallel transportation simulation, we propose an algorithm for parallel lines partition and an algorithm for quadrilateral grids partition. Based on the real traffic density distribution, we make several experiments to test the performance of each algorithm. The result shows that the algorithm using parallel lines partition can obtain a relative low imbalance in 100 algorithm cycles, and the algorithm using quadrilateral grids partition can obtain a relative high imbalance. But this does not necessarily mean that the former is more advisable and in the later experiment of comparing the communication overheads of an 8 parallel lines model and a 3×3 grid model. We find that the communication overhead of the former is much bigger than that of the later and is not adaptable in real application unless on a quit fast network.

References

1. O'Cearbhaill, E.A., O'Mahony, M.: Parallel implementation of a transportation network model. Journal. Parallel Distributed Computing 65, 1–14 (2005)
2. Nagel, K., Rickert, M.: Parallel implementation of the TRANSSIMS micro-simulation. Parallel Computing 27, 1611–1639 (2001)
3. Klefstad, R., Zhang, Y., Lai, M., Jayakrishnan, R., Lavanya, R.: A Distributed, Scalable, and Synchronized Framework for Large-Scale Microscopic Traffic Simulation. In: Proceedings of the 8th International IEEE Conference on Intelligent Transportation Systems, pp. 813–818 (2005)
4. Deng, Y., Peierlsy, R.F., Riveraz, C.: An Adaptive Load Balancing Method for Parallel Molecular Dynamics Simulations. Journal of Computational Physics 161, 250–263 (2000)
5. Rus, P., Tok, B., Mole, N.: Parallel computing with load balancing on heterogeneous distributed systems. Advances in Engineering Software 34, 185–201 (2003)
6. Genaud, S., Giersch, A., Vivien, F.: Load-balancing scatter operations for grid computing. Parallel Computing 30, 923–946 (2004)