Transport-Problem-Based Algorithm for Dynamic Load Balancing in Distributed Logic Simulation

Yury V. Ladizhensky, Viatcheslav A. Kourktchi

Donetsk National Technical University
Artema Street, 58a
Donetsk, Ukraine, 83005

Abstract: Advantages and disadvantages of a dynamic load balancing algorithm which minimizes Euclidean norm of data migration are discussed. A new effective algorithm for a dynamic load balancing problem is suggested. The algorithm is based on a transport problem solving and searching for shortest ways in a graph. Experimental results for the algorithm are provided.

1 Introduction

The distributed logic simulation is the logic simulation using a network of computers. Achieving a good load balance between processors of computers is crucial to minimize time spent for a simulation process. The first step of the load balancing is to cut a problem: to distribute the parts of the problem between processors. In some cases it is enough because the balance won't be changed. But in other cases the load balance should be periodically restored. There are many methods and algorithms for static and dynamic load balancing. We refer to review of graph partitioning algorithms [Po96] and a multilevel algorithm [KK95] for static load balancing methods. For dynamic load balancing methods see, for example, [AT95], [AT96], [JSL94], and also reviews [De05].

In [HB95], authors suggest a new criteria of optimization for load balancing – a migration level. The migration level is an amount of data to be transferred between processors to achieve a load balance. Indeed, after the load balancing algorithm has been executed a schedule of data or subproblems migration between the processors is ready. But the data migration needs time, and this time should be also minimized. In this paper, we focus on methods which allow finding a good load balance and minimizing the migration level and suggest a new approach to this problem.

2 Definitions

Let P be a number of processors and $G_P(V_P,E_P)$ be a processor graph, where $V_P=\{1,2,...,P\}$ is the set of vertices representing processors and E_P is the set of edges. An edge $(i, j) \in E_P$ if data can be transferred from a processor i to a processor j. In a

distributed simulation a graph will always connected, and in many cases it will be a clique, because in a computer network any pair of computers can exchange with data.

Let N be a number of subproblems. A subproblem can represent an element of simulated scheme or a cluster of elements that depends of a simulation problem. Let a graph $G_N(V_N, E_N)$ be a problem graph, where $V_N=\{1,2,...,N\}$ is the set of vertices representing subproblems and E_N is the set of edges. An edge $(i, j) \in E_N$ if the subproblem j needs information or events from the subproblem i. We assume all subproblems have equal complexity.

Let $C=\{C_1, C_2, ..., C_P\}$ be a partition of the graph $G_N(V_N, E_N)$. Each $C_i \subset V_N$ is computed on processor i. The subsets C_i and C_j are not intersected, i.e. $C_i \cap C_j = \varnothing$. The problem is to find new partitioning $C' = \{C'_1, C'_2, ..., C'_P\}$ which minimizes concurrently the misbalance, interprocessor communication and migration level. The misbalance can be found using one of the following formulas [AT95], [AT96], [DS98]:

$$M_1(C) = \max_{k=1,P} |l_k| - \min_{m=1,P} |l_m|,$$

$$M_2(C) = \frac{1}{l_{\max}} \sqrt{\frac{\sum_{i=1}^{P} (l_i - l_{avg})}{P-1}},$$

$$M_3(C) = \max_{i=1,P} l_i,$$

where

$$l_i = |C_i|,\ l_{\max} = \max_{i=1,P} l_i,\ l_{avg} = \frac{1}{P} \sum_{i=1}^{P} l_i .$$

The formula $M_1(C)$ presents the difference between the maximum and the minimum load. If $M_1(C)=0$ the misbalance is absent. The formula $M_2(C)$ is the second central moment of the loads of different processors. This statistical magnitude is difficult to evaluate, therefore it is used rarely. And $M_3(C)$ is the maximum value of processor load. The maximum load doesn't tell anything about misbalance directly, but the processor which has the maximum load will work slower than others. Thus, if the maximum load will be as minimal as possible the simulation process is the fastest. Others formulas don't give such advantages. Also the $M_3(C)$ can be easily evaluated, so it is the most suitable.

The migration level can be evaluated as

$$Q(C) = \frac{\sum_{k=1}^{P} \left(|C_k| - |C_k \setminus C'_k| \right) + |C'_k \setminus C_k|}{2} .$$

This function sum up the numbers of subproblems to be transmitted from each processor $|C_k|-|C_k \setminus C'_k|$ and the number of subproblems to be transmitted to each processor $|C'_k \setminus C_k|$.

The factor ½ is needed because each transmitted subproblem is considered twice in the numerator. Hence, the $Q(C)$ is the number of transmitted subproblems.

In order to facilitate the algorithm development we assume the interprocessor communications are not important for the task and do not consider it. But in [HB95] it was noted that the reasoning with such assumption should take into account future necessity of considering interprocessor communication. To do that, authors propose to allow data migration between two processors only if there are an edge $(i, j) \in E_P$ in processor graph and such subproblems $k \in C_i$, $l \in C_j$ that $(k,l) \in E_N$.

Let's consider a graph $G(C,E_C)$, where C is a problem graph partitioning and $(C_i,C_i) \in E_C$ if and only if $(i, j) \in E_P$ and the following condition is satisfied: $\exists k \in C_i \exists l \in C_i ((k,l) \in E_N)$. If there is such C_i in C that $l_i - l_{avg} \geq 1$, one or several subproblems $k \in C_i$ should be transferred to other part of the problem graph. The target part can be selected only from the set $\{C_j | (C_i,C_j) \in E_C\}$. If there is no such C_i that $l_i - l_{avg} \geq 1$ the load balance is achieved.

We use $l_i - l_{avg} \geq 1$ rule to check the necessity of data migration because if $l_i \leq l_{avg}$ data should be transferred to the C_i, but not from the C_i. And if $l_i - l_{avg} < 1$ only a part of subproblem should be transferred to achieve the load balance, but the subproblems cannot be divided. The rule $l_i > l_{avg}$ cannot be used because in cases with non integer l_{avg} a processors with the surplus load always exist, so the load balancing will be endless.

3 Minimizing Euclidean norm of data migration

The algorithm which minimizes Euclidean norm of data migration is described in [HB95]. The authors assume that any subproblem can be infinitely divided and consider a load as real number. Thus the amount of data to be transferred from the part C_i is defined as $l_i - l_{avg}$.

3.1 Algorithm description

The amount of data to be transferred from the part C_i to the part C_j is denoted as δ_{ij} ($\delta_{ij} = -\delta_{ji}$). To make the balance optimal the amounts δ_{ij} should satisfy the following linear system of equations:

$$\sum_{(C_i,C_j) \in E_C} \delta_{ij} = l_i - l_{avg} \quad i = \overline{1,P}.$$

The number of independent equations is less than P, therefore the system of equations has an unlimited number of solutions. A solution with minimum data migration should be selected, that is we should

Minimize $Q_E \sum_{\substack{C_i \in C, C_j \in C \\ i \neq j}} \delta_{ij}^2$,

subject to $\sum_{(C_i, C_j) \in E_C} \delta_{ij} = l_i - l_{avg}$.

This problem can be replaced by another system of equations $L\lambda=b$, where λ is a variable vector of size P, L is a matrix of size $P \times P$, b is a vector of size P:

$$(L)_{ij} = \begin{cases} -1, & if\ i \neq j\ and\ (C_i, C_j) \in E_C \\ \deg(C_i), & if\ i = j \\ 0, & otherwise \end{cases}$$

$$(b)_i = l_i - l_{avg}$$

A degree of the vertex C_i if the graph ($G(C,E_C)$ is the $\deg(C_i)=|\{C_k|(C_i,C_i) \in E_C\}|$.

After the value of the λ vector was found, the data migration amount can be evaluated as $\delta_{ij} = \lambda_i - \lambda_j$.

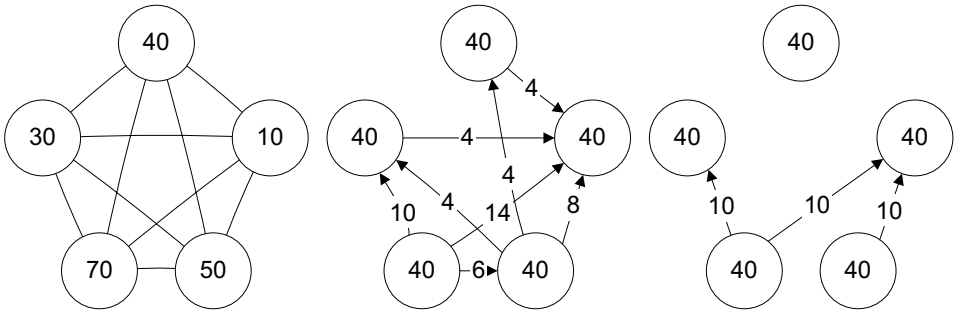
The load balancing algorithm is [HB95]:

(1) Find the average load and the vector b.
(2) Solve linear system of equations $L\lambda=b$ for λ.
(3) The load to be transferred from a processor i to a processor j is $\lambda_i - \lambda_j$.

3.2 Algorithm discussion

The algorithm from [HB95] was originally designed for the parallel finite element solution of PDE's based on unstructured meshes. The big advantages of this algorithm are simplicity of implementation and possibility of the parallel implementation on the same parallel machine which is used for simulation. The authors suppose to use a parallel machine with sparse interprocessor connections. This means that a degree of each vertex in a processor graph is small.

In the distributed logic simulation any two processors can usually exchange data using approximately equal time. Hence the processor graph has an edge for each pair of vertices. We have studied properties of the algorithm in this case. Let's consider the full graph with 5 vertex-processors (see fig.1a) and the processors load is l=(40, 10, 50, 70, 30).



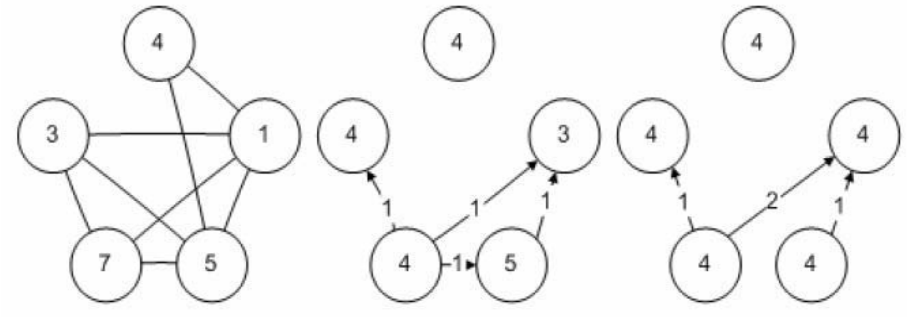
a) b) c)

Fig. 1 Example of the algorithm work on the full graph: a) the initial graph with the processors load; b) data migration and new processors load obtained using the algorithm; c) an alternative solution

Using the algorithm we have found λ=(–2.811, –6.811, 1.189, 7.189, –2.811). The data migration in this case is shown on fig. 1b. We see that the load is balanced. The number of transferred subproblems is 4+4+4+4+6+8+10+14=54, or Q_E = 920. But other possible solution exists (see the fig. 1c) which gives the same load balance but transfers only 10+10+10=30 subproblems (Q_E = 600). Therefore, this algorithm doesn't give the best possible solution for the distributed systems.

The logic simulation has other difference from the finite elements method. The number of subproblems can exceed the number of processors not much. Let's consider one more example to illustrate the work of the algorithm in that case. On the fig. 2a the processor graph with the small processors load is presented. The figures 2b and 2c show the subproblems migration schedule obtained with the algorithm and manually. We can see that algorithm's balance is not optimal, and it needs to transfer 4 subproblems (Q_E = 8). Alternative solution gives the optimal balance and transfers 4 subproblems also, but Q_E = 12. Since the optimal load balance achieving is more important, alternative solution which does it and transfers the same number of subproblems is better.

Reasoning from these disadvantages of the algorithm [HB95] for the load balancing, we need new algorithm avoided these drawbacks.



a) b) c)

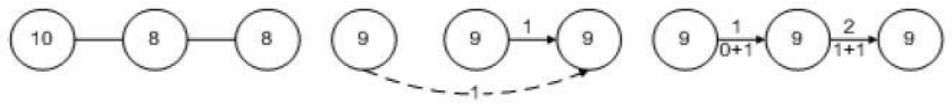
Fig. 2 Example of the algorithm work with small load: a) the initial graph with the processors load; b) data migration and new processors load obtained using the algorithm; c) an alternative solution

4 Transport-problem-based algorithm

The main problems of the load balancing are determination of a number of subproblems to be transferred from each processor and a target processor for each transferred subproblem.

4.1 Main idea of the algorithm

We have some processors with the load $l_i - l_{avg} \geq 1$, each of these processors has to give away $\lfloor l_i - l_{avg} \rfloor$ subproblems. Also we have some processors with the load $l_i < l_{avg}$, each of these processors may receive $\lfloor l_{avg} - l_i \rfloor$ subproblems. If we define a distance between any pair of processors we can solve this problem as the transport problem using the method of potentials. If some subproblem will be transferred from one processor to another through the third one, it will be transferred twice (see fig. 3). Hence, the subproblem should always be transferred using the shortest path in the graph. The shortest paths between all pairs of vertices in the graph can be found using the Floyd algorithm. We use the method of potentials [ME78] and the Floyd algorithm [Fl62] as subtasks of the new algorithm.



a) b) c)

Fig. 3 Data migration route: a) the processor graph; b) the optimal but prohibited migration schedule (the dotted line is the prohibited way of migration); c) the migration schedule using the intermediate processor

The main idea of the algorithm is to find the shortest paths in the graph between all pairs of processors and form the input data for the transport problem, solve it using the method of potentials. Then the results of the transport problem should be interpreted in terms of the load balancing.

4.2 Algorithm implementation

The transport-problem-based algorithm for dynamic load balancing is provided on the fig. 4. This algorithm returns a matrix Δ of the $P \times P$ size, $\Delta[i,j]$ are the number of subproblems to be transferred from a processor i to a processor j. Initial values of $\Delta[i,j]$ is assumed to be zero.

On the first step of this algorithm an average load is evaluated. The sets O and I of processors which have to pass and receive the subproblems appropriately are calculated then. The Floyd algorithm is used on the fourth step. The results of the Floyd algorithm are the array D of the ways' lengths and the array W which contains information about ways.

Input data for the transport problem are formed on the steps 6-8: the array T of the ways' length between processors which pass their subproblems and processors which receive them. Then the potential method is used on the step 9 to solve the transport problem. It uses the array T of the ways' lengths between processors, the set $O - \lfloor l_{avg} \rfloor$ of amounts of subproblems to be given away from processors, the set $\lfloor l_{avg} \rfloor - I$ of amounts of subproblems to be received by processors. The result of the potential method is an array TS of size $|O| \times |I|$, each element $TS[k,m]$ is an amount of subproblems to be transferred from an element $O[k]$ to an element $I[m]$.

1. $l_{avg} = \frac{1}{P} \sum_{i=1}^{P} C_i$
2. $O \leftarrow \{C_i \mid |C_i| - l_{avg} \geq 1\}$
3. $I \leftarrow \{C_i \mid |C_i| < l_{avg}\}$
4. $\langle D, W \rangle \leftarrow Floyd(G(C, E_N))$
5. $T \leftarrow CreateMatrix(|O| \times |I|)$
6. *foreach* $k \in O$
7. *foreach* $m \in I$
8. $T[k,m] \leftarrow D[k,m]$

endfor

endfor

9. $TS \leftarrow SolveTransport(T, O - \lfloor l_{avg} \rfloor, \lfloor l_{avg} \rfloor - I)$
10. *foreach* $k \in O$
11. *foreach* $m \in I$
12. $c := vs$
13. *repeat*
14. $\Delta[c, W[c,m]] \leftarrow \Delta[c, W[c,m]] + TS[k,m]$
15. $c \leftarrow W[c,m]$;
16. *until* $c = m$;

endfor

endfor

Fig. 4 Transport-based algorithm for dynamic load balancing

After that the results of the method of potentials must be interpreted. For each "give away - receive" pair of processors (steps 10-11) the shortest past should be restored (steps 12-16). The variable c for path restoring is initialized on the steps 12 and refers to the current processor of path. The variable c is the current vertex (processor) of the path. Then the amount of subproblems to be transferred between processors k and m is added to each edge of the path in the graph as it is shown on the fig. 3. In the algorithm implementation this amount is $TS[k,m]$ and it is added to the element $\Delta[c, W[c,m]]$ of the return matrix, where $W[c,m]$ is the next processor in the path. Operations are performed

on the steps 13-16 until the value variable c which represents current processor will equal target processor m, i.e. the path is over.

4.3 Algorithm discussion

The new algorithm allows avoiding cycles in data transferring. This is a big advantage in compare to the [HB95] algorithm. It should avoid cyclic or indirect (see fig. 1.b) data migration. Moreover, if a vertex has a surplus load it will only give the load away, and if a vertex has lack of load it will only receive the load. With one exception – if there is no edge between vertices, the data will migrate using the shortest path. We assume that it will significantly decrease the migration level which is critical for distributed systems. Also new algorithm will take away all surplus loads from each processor, so it should create such migration schedules that will always minimizes the maximum load in the best way. This is important for logic simulation, because it allow speed up simulation process. For the example problems from figs. 1 and 2 the new algorithm gives an optimal solution (like on the figs. 1c and 2c). The new algorithm has a disadvantage. While [HB95] algorithm solves the linear system of equations which can be done in parallel, new algorithm solve the transport problem and cannot be so effectively parallelized.

5 Experimental results

The algorithm from [HB95] and the transport-problem-based algorithm were implemented on Delphi language. Experiments on random graphs were performed. Problem graphs were generated using the following scheme:

(1) Create the P parts of the graph; each part contains a random number of vertices between C_{min} and C_{max}. The overall number of vertices is $|V_N|$.
(2) Create the $|V_N|-1$ edge to create a connected problem graph. First edge is added between two randomly selected vertices. Each next edge is added between a randomly selected vertex from already connected vertices and other vertex from not yet connected vertices.
(3) Create all other edges at random until a needed graph density is reached.
(4) A processor graph is assumed to be a full connected graph with P vertices.

Input graph			Minimum load			Maximum load			Migration	
G	$\|V_N\|$	$\|E_N\|$	$C_{\min}$	$C_{\min}^{TB}$	$C_{\min}^{HB}$	$C_{\max}$	$C_{\max}^{TB}$	$C_{\max}^{HB}$	Q^{TB}	Q^{HB}
1	997	4970	96	97	99	104	100	101	10	9
2	1004	5040	96	96	99	104	101	102	8	8
3	1001	5010	95	97	99	104	101	101	9	12
4	999	4989	95	99	99	104	100	101	15	16
5	1002	5020	95	96	99	104	101	101	6	8
6	998	4980	96	99	98	103	100	101	11	7
7	990	4900	96	99	98	104	99	101	12	12

8	989	4890	95	98	98	103	99	100	11	8
9	1000	4999	96	100	99	104	100	101	12	9
10	1000	4999	95	100	99	104	100	101	11	8

Table 1 Results for problem graphs with around 1000 vertices and 5000 edges on 10 processors

The experimental results are presented in Tables 1-3. Here G is the sequence number of graph in the table, C_{min} is the minimum load, C_{max} is the maximum load, $|V_N|$ is the number of vertices in the problem graph, $|E_N|$ is the number of edges in the problem graph, $C^{TB}{}_{min}$ is the minimum load if the transport-problem-based algorithm was used, $C^{TB}{}_{max}$ is the maximum load if the transport-problem-based algorithm was used, Q^{TB} is the number of subproblems was transferred by the transport-problem-based algorithm, $C^{HB}{}_{min}$ is the minimum load if the [HB95] algorithm was used, $C^{HB}{}_{max}$ is the maximum load if the [HB95] algorithm was used, Q^{HB} is the number of subproblems transferred by the [HB95] algorithm, Q_{avg} is the average data migration norm for each algorithm.

Input graph			Minimum load			Maximum load			Migration	
G	$\|V_N\|$	$\|E_N\|$	$C_{\min}$	$C^{TB}_{\min}$	$C^{HB}_{\min}$	$C_{\max}$	$C^{TB}_{\max}$	$C^{HB}_{\max}$	Q^{TB}	Q^{HB}
1	10001	11002	425	482	498	570	501	502	386	510
2	10264	11588	431	498	511	570	514	515	353	485
3	10104	11230	427	490	503	574	506	507	414	558
4	10155	11343	445	503	506	565	508	509	273	385
5	9668	10281	429	476	481	573	484	486	319	427
6	10126	11278	428	493	504	566	507	508	383	514
7	9777	10514	428	486	487	558	489	492	417	503
8	10124	11274	430	491	504	572	507	509	370	504
9	10131	11290	444	498	504	569	507	509	359	457
10	10059	11130	427	502	501	566	503	505	391	517

Table 2 Results for problem graphs with around 10000 vertices and 11000 edges on 20 processors

Input graph			Minimum load			Maximum load			Migration	
G	$\|V_N\|$	$\|E_N\|$	$C_{\min}$	$C^{TB}_{\min}$	$C^{HB}_{\min}$	$C_{\max}$	$C^{TB}_{\max}$	$C^{HB}_{\max}$	Q^{TB}	Q^{HB}
1	9986	49860	432	489	497	574	500	501	419	536
2	9990	49900	425	490	498	550	500	500	324	441
3	9803	48049	425	474	489	572	491	492	419	553
4	9479	44925	431	473	471	565	474	476	308	424
5	10067	50672	430	491	502	572	504	505	347	472
6	10138	51389	427	505	506	574	507	509	382	509
7	10182	51836	431	492	507	573	510	510	444	561
8	10368	53747	429	509	517	571	519	520	396	514
9	9865	48659	429	479	491	571	494	497	300	436
10	9964	49640	429	483	497	569	499	501	369	478

Table 3 Results for problem graphs with around 10000 vertices and 50000 edges on 20 processors

In all cases (see tables 1-3) a maximum load after using the transport-problem-based algorithm is less or equals a maximum load $M_3(C)=C_{max}$ after using [HB95] algorithm,

that is $C^{TB}_{max} \le C^{HB}_{max}$. Thus a logic simulation will be performed faster if the transport-problem-based algorithm is used. But the index $M_1(C)=C_{max}-C_{min}$ is worse for the transport-problem-based algorithm.

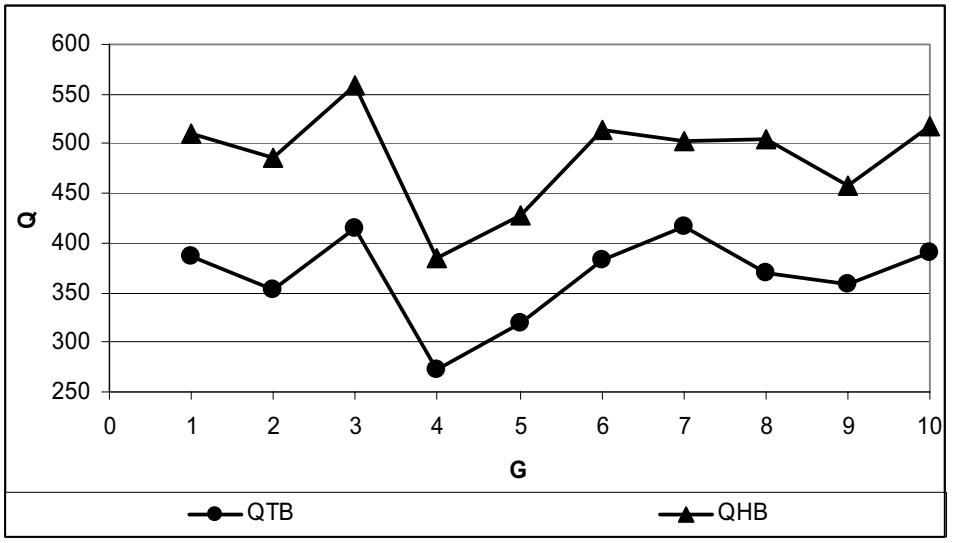


Fig. 5 The migration level for the transport-problem-based algorithm (QTB) and known algorithm (QHB) for each graph with 10000 vertices and 10000 edges on 20 processors (see table 2)

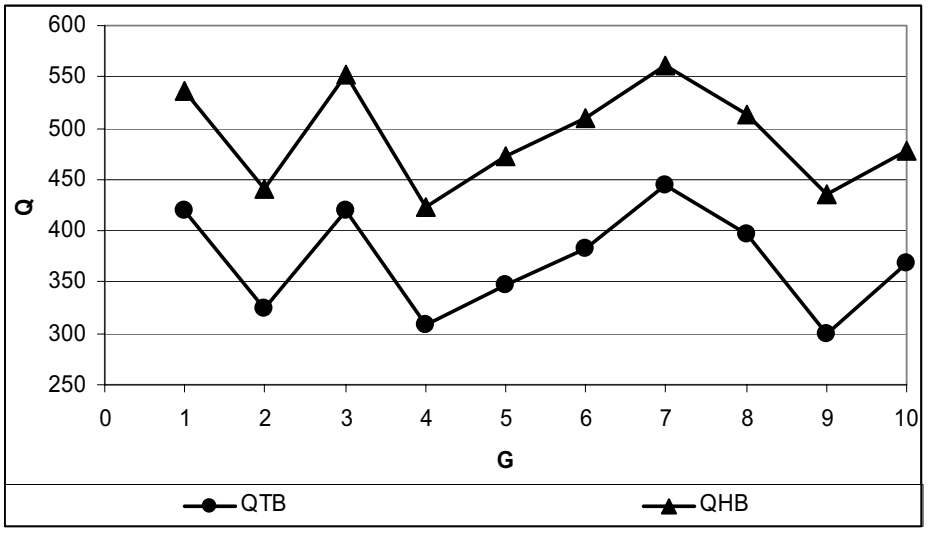


Fig. 6 The migration level for the transport-problem-based algorithm (QTB) and known algorithm (QHB) for each graph with 10000 vertices and 50000 edges on 20 processors (see table 3)

For small graphs (see table 1), a number of transferred subproblems by the transport-problem-based algorithm can be bigger but if C^{TB}_{max} is strictly less than C^{HB}_{max}. For

bigger graphs (see tables 2 and 3) the number of transferred subproblems by the transport-problem-based algorithm is always less, it decreases a number of transferred subproblems on 24.7% at average against the [HB95] algorithm. On the figs. 5 and 6 there are charts which show the difference between the migration levels of transport-problem-based and [HB95] algorithms in dependency of graph. The 24.7% gain from the new algorithm is almost constant.

6 Conclusions

Analysis of the load balancing algorithm which minimizes Euclidean norm of data migration reveals some drawbacks of the algorithm using for the distributed logic simulation. This algorithm gives not good solutions if a processor graph has high density or a load of each processor is low. The new load balancing algorithm is developed. It uses Floyd algorithm to find shortest ways in a problem graph and then solves a transport problem using the potential method to minimize data migration between processors and the lengths of ways for data transferring between processors. Computational experiments show that the new algorithm decreases a number of transferred subproblems on 24.7% at average against the existed algorithm. Also new algorithm improves a load balance for the big problems. We plan improve the transport-problem-based algorithm by considering the interprocessor communication as the third criteria of optimization in future.

Bibliography

[AT95] H. Avril, C. Tropper, Clustered time warp and logic simulation//9th Workshop on Parallel and Distributed Simulation (PADS'95), 1995, pages 112-119.

[AT96] H. Avril, C. Tropper, The Dynamic Load Balancing of Clustered Time Warp for Logic Simulations//Workshop on Parallel and Distributed Simulation, 1996, pages 20-27.

[De05] K. Devine and al., A new challenges in dynamic load balancing.//Applied Numerical Mathematics, v 52, 2005, pp 133-152.

[DS98] Deelman E., Szymanski B., Dynamic load balancing in parallel discrete event simulation for spatially explicit problems.//PADS'98, IEEE CS Press, 1998, pages 46-53.

[Fl62] Floyd, Robert W. Algorithm 97: Shortest Path. Communications of the ACM, volume 5 (issue 6), 1962, p 345.

[HB95] Y.F. Hu, R.J. Blake, An optimal dynamic load balancing algorithm. Preprint DL-P-95-011, Daresbury Laboratory, Warrington, WA44AD, UK. (To be published in Concurrency: Practice & Experience), 1995

[JSL94] M-R. Jiang, S-P. Shieh, C-L. Liu, Dynamic load balancing in parallel simulation using time warp mechanism.// ICPADS, 1994, pp. 222-229.

[KK95] G. Karypis, V. Kumar, Analysis of multilevel graph partitioning. Tech. Report 95-035, Computer Science Department, University of Minnesota, 1995.

[ME78] Handbook of Operation Research: Foundation and Fundamentals.// Edited by J.J. Moder, S.E. Elmaghraby. – Litton Education Publishing, 1978

[Pi00] G-B Ping and al., Load balancing for conservative simulation on shared memory multiprocessor systems.//14th Workshop on Parallel and Distributed simulation, May 28-31, 2000.

[Po96] A. Pothen, Graph partitioning algorithms with applications to scientific computing.//Parallel Numerical Algorithms, Kluwer Academic Press, 1996.