Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

A self-routing load balancing algorithm in parallel computing: comparison to the central algorithm[#]

Razieh Sadat SADJADY, Kamran ZAMANIFAR

(*Department of Computer Engineering, Islamic Azad University, Najafabad Branch, Isfahan, Iran*)

E-mail: rsadjady@gmail.com; zamanifar@eng.ui.ac.ir

Received June 21, 2010; Revision accepted Nov. 12, 2010; Crosschecked Mar. 31, 2011

Abstract: Load balancing is an important stage of a system using parallel computing where the aim is the balance of workload among all processors of the system. In this paper, we introduce a new load balancing algorithm with new capabilities for parallel systems, among which is the independence of a separate route-finder algorithm between the load receiver and sender nodes. In addition to simulation of the new algorithm, due to similarity in behavior to the proposed algorithm, the central algorithm is simulated. Simulation results show that, the system performance increases with the increase of the degree of neighborhood between the processors. These results also indicate the algorithm's high compatibility with environment changes.

Key words: Parallel computing, Load balancing, Distributed system
doi:10.1631/jzus.C1000211 **Document code:** A **CLC number:** TP393

1 Introduction

With the rapid pace in scientific endeavors and the necessity for high-speed processing, which may even affect the mode of distribution, the need for parallel and distributed systems is soaring. The presence of a number of processors in these kinds of systems shows, from one perspective, the necessity of a uniform distribution of workload among these processors. Studies have shown that in these systems the probability of a processor being idle in the system and other processors having an array of tasks at hand is very high (Chhabra *et al*., 2006).

The issue can, in fact, be thus presented that the use of parallel and distributed systems, due to the speed they add to the processing tasks, is an important factor. But the capital needed to elevate systems to the parallel system type seems logical only when the workload of the system is distributed suitably among the processors. The aim becomes practical in parallel and distributed systems by implementation of a certain type of algorithm, called 'load balancing'.

This paper is an extension of Zamanifar *et al*. (2010). In this paper, the load balancing in parallel calculations is explicated and a new algorithm with new capabilities is presented. The structure of this algorithm is studied based on the categorization presented in Osman and Ammar (2002), and the new algorithm and the central algorithm are simulated. Simulation results show that with the increase of processors' degree of neighborhood, i.e., the ratio of system's total numbers of processors, the system performance increases. These results also indicate the ability of the new algorithm in selecting various paths between the two processors along with condition changes in a system environment.

2 Load balancing

In general, load balancing is used for many benefits, including minimizing execution time and maximizing resource utilization (Chhabra *et al*.,

[#] A preliminary version was presented at the Second International Conference on Communication Software and Networks, Feb. 26–28, 2010, Singapore

2006). Load balancing algorithms are divided into two major groups: static and dynamic. In the former type, based on an estimation of the time needed to complete any given task, tasks are assigned to processors during the compile time, and their relation is determined. There is no decision on this type regarding a shift of task from one processor to another during the execution time. In dynamic load balancing algorithms, however, the load status at any given moment is used to decide task shifts between processors (Wu, 1997; Osman and Ammar, 2002; Grama *et al*., 2003; Chhabra *et al*., 2006).

Random, central, and rendez-vous are among the existing load balancing algorithms which can be seen in Fonlupt *et al*. (1995), Fonlupt *et al*. (1996), Osman and Ammar (2002), and Cruz and Mateus (2003).

Also, load balancing algorithms should be compared with respect to the parameters of quality of which nature and overhead-associated can be mentioned (Chhabra *et al*., 2006). For instance, the dynamic algorithms have higher overhead compared to the static ones.

Processors in parallel and distributed systems in relation to load balancing algorithms can be divided into three groups based on the workload level:

1. Processors that have a large number of tasks to do are referred to as 'heavily loaded processors' or sometimes 'overloaded processors'.

2. Processors that have a small number of tasks to do are referred to as 'lightly loaded processors' and also 'underloaded processors'.

3. Processors named 'idle processors' that have no tasks to do (Fonlupt *et al*., 1995; Chhabra *et al*., 2006).

Dynamic load balancing algorithms are categorized based on a structure presented in Osman and Ammar (2002), a categorization in which algorithms are studied from various perspectives. For example, one of the criteria for algorithm categorization in this structure is 'initialization', in which the way by which algorithms begin to work and whether the algorithm is carried out periodically or is event-driven are focused.

Also, various strategies have been proposed for the categorization of load balancing algorithms. In each of the strategies there are different ranges for the definition of the load balancing algorithm. Examples of these strategies can be seen in Kuchen and Wagener (1990) and Lüling *et al*. (1991).

Another aspect that should be considered is the variety of parallel and distributed operational environments. For example, in a system, the sources may be of the same type and with the same capacity (for homogeneous systems) or of various types and with varying capacities (for heterogeneous systems). The architecture used in the system (multi-data/task) can affect the execution or even the definition of the algorithm. Other examples of this type can be tracked in Chhabra *et al*. (2006). Also, in de Ronde *et al*. (1996), Lai *et al*. (1997), Berger and Browne (1999), Giusti *et al*. (2005), Garcia and Semé (2006), and Borovska and Lazarova (2007), the load balancing problem has been described for various parallel and distributed operational environments.

In the next section, the new algorithm is presented and studied based on the categorization in Osman and Ammar (2002).

3 New load balancing algorithm

To explicate this algorithm, it should be stated that with respect to the variety manifested in the topology used to connect processors, each processor has a definite maximum number of neighbors. For example, in a system in which the processors are ruled by mesh topology, each processor has, at most, four neighbors. We can define a field for each processor, the amounts of which would determine each of its neighbor processors.

For additional explanation, suppose we call the characteristic 'direction'. As an example, in Fig. 1, for each processor there are at most four neighbors; therefore, 'direction' in each processor can acquire four values, i.e., {left, right, up, down} or {1, 2, 3, 4}.

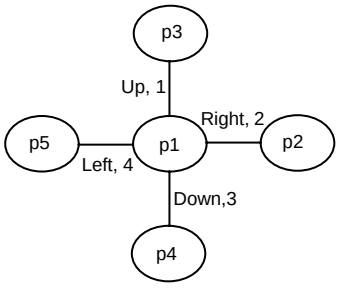

Fig. 1 Specific amounts of direction centered on the p1 processor

The idea of this algorithm stemmed from the perspective that a processor may be neither idle nor overloaded, but has both idle and overloaded processors neighboring it, and can therefore serve to relate them. In other words, the related processor can send the message that is not idle itself but has an idle neighbor processor to its neighboring processors.

The algorithm works as follows. As soon as a processor becomes idle, it sends a message to its neighbors. This message includes the number of the idle processors, the message number, a counter, and a field to determine the validity of the message. The number of the processors is, in fact, its ID. Any processor may become idle many times. To determine the validity of a message (i.e., to assure that it is not a previously expired message), a message number is used. 'Counter' is a characteristic to which one unit is added each time a message is conveyed from one processor to another and it determines the distance of the message from the first idle processor from which it originated.

The neighboring receiver processor saves the message complete with all its related information and with regard to the direction from which it came. Note that the highest number of messages that a processor can store depends on the number of its physical neighbors. In reality, each processor can store only one message coming from any direction, this message being the last message received from that direction.

If a processor reaches the underloaded level, it chooses from the received messages of neighboring processors the one that has the highest priority level (i.e., the closest) and sends the message to its own neighboring processors.

Eventually, an overloaded processor chooses from its received messages the one with the highest priority to send a portion of its load to the processor where the message originated.

The status of the processors is shown by a comparison of the average workload of the whole system and the workload of each processor. A processor without workload is in idle status, while a workload under the average of workload of the whole system is underloaded; otherwise, it is overloaded.

Algorithm 1 gives the pseudo-code of this algorithm, where d is the highest number of neighbors that any processor within a system embodying a specific protocol can have and n is the number of processors.

Algorithm 1 The new load balancing algorithm

```
If processor p is idle
if last_message_number(p, 2)=0 then
   last_message_number(p, 1):=last_message_number(p, 1)+1;
   last_message_number(p, 2):=1;
   for i:=1 to d do
      send message to direction i of processor p (p, last_
       message_number(p, 1), 1, 1);
   end for
end if

If processor p is underloaded
min:=∞;
walk:=∞;
for i:=1 to d do
   if receive_message(p, i, 4)=1 and receive_message(p, i, 3)
   <walk and last_message_number(receive_message(p, i, 1),
   1)=receive_message(p, i, 2) and last_message_number
   (receive_message(p, i, 1), 2)=1 then
      walk:=receive_message(p, i, 3);
      min:=i;
   end if
end for
if min≠∞ then
   for i:=1 to d do
      if i≠min then
         send message to direction i of processor p (receive_
         message(p, min, 1), receive_message(p, min, 2),
         receive_message(p, min, 3)+1, 1);
      end if
   end for
end if

If processor p is overloaded
if match(p)=0 then
   min:=∞;
   walk:=∞;
   for i:=1 to d do
      if receive_message(p, i, 4)=1 and receive_message(p, i,
      3)<walk and last_message_number(receive_message(p,
      i, 1), 1)=receive_message(p, i, 2) and last_message_
      number(receive_message(p, i, 1), 2)=1 then
         walk:=receive_message(p, i, 3);
         min:=i;
      end if
   end for
   if min≠∞ then
      p1:=p;
      p2:=get _processorid(p, min);
      idle_processor:=receive_message(p, min, 1);
      d1:=min;
      for i:=1 to walk do
         receive_message(p1, d1, 4):=0;
         path(p, idle_processor, i):=d1;
         if i≠walk then
            p1:=p2;
            p2:=−1;
            for j:=1 to d do
               if receive_message(p1, j, 4)=1 and receive_
               message(p1, j, 1)=idle_processor and last_mes-
               sage_number(idle_processor, 1)=receive_mes-
               sage(p1, j, 2) and last_message_num-
```

```
            ber(idle_processor, 2)=1 then
               d1:=j;
               p2:=get _processorid(p1, d1);
            end if
         end for
      end if
      if p2=−1 then
         break;
      end if
   end for
   if p2≠−1 and last_message_number(idle_processor,
   2)=1 then
      last_message_number(idle_processor, 2):=2;
      match(p):=idle_processor;
   end if
 end if
end if
```

The last_message_number array is an $n\times2$ array, in which last_message_number(p, 1) shows the number of the last message sent by processor p when in idle status and last_message_number(p, 2) shows the current status of the message. If last_message_number(p, 2) is zero, the message is no longer valid; if one, the message is valid; and if two, the message is received by an overloaded processor ready to transfer load to the idle processor. After load transfer the quantity of this field will become zero.

Another array used is the receive_message, which is an $n\times d\times4$ array. This array should, in fact, be defined as separate $d\times4$ arrays for each processor, but we have, for simplicity, defined it as noted. In this array, receive_message(p, d1) has four fields, which save the message received from the d1 direction of processor p and, as mentioned, the message includes: (1) the number of the idle processors, (2) the message number, (3) counter, and (4) message validity field. The message validity field falls to zero after the message expires but it is one in normal status.

The path array is an $n\times n\times n$ array. After the execution of the algorithm for any overloaded processor i which finds an idle processor j to receive load, the path of load transfer from processor i to processor j is saved in path(i, j). Note that this type of address provision is different from the typical in that, instead of using the processor id, the direction of transfer along the transfer course is incorporated into this array. In addition, note that this array should also be defined separately for each processor as a local $n\times n$ array, but we have used it in this form for the simplicity of code definition. The get_processorid(p, d1) function returns the number of the processors on the d1 side of processor p.

After an overloaded processor chooses an idle processor, the workload of the overloaded processor is divided between the two processors according to their processing capacities. If the system is a multi-task system, the workload is measured in terms of the number of tasks; if a multi-data system, the data amount is measured. The workload is then divided between the two processors.

In relation to the algorithm execution time, it should be said that each processor in any status (idle, overloaded, underloaded) can carry out its duty independently toward the load balancing algorithm, but it is a better option if, in the load transfer stage, the execution is harmonious and simultaneous.

We now wish to call the readers' attention to the following two points:

1. A threshold can be considered for the message counter. Another reason for using the message counter is presented here. This threshold can be between one and the maximum distance between the two processors, so that any amount surpassing the threshold would automatically render the message invalid. In addition, having a threshold enables us to control the overhead resulting from continual repetition of message sending and to change the threshold level based on the circumstances. Through adjustment of the threshold level, we can switch between various communication policies (local, uniform, global) even during execution; that is, we can allow workload transfer for any processor within a certain circumference of its locality.

2. Considering that this is a dynamic load balancing algorithm, we will analyze it based on the categorization presented in Osman and Ammar (2002). In terms of initiation, the algorithm can be activated by both periodic and event-driven initiation. As previously mentioned, the load-transfer stage would be better controlled periodically. In relation to 'load balancer location', the algorithm operates in a distributed and asynchronous manner, meaning that all processors are included in the algorithm execution, but there is no need for simultaneous operation. Concerning information exchange, processor decision can be based on global information, but this depends on the threshold level. Communication topology is uniform, meaning that the processor neighbors do not change at any given moment during execution, and also it is worth mentioning that the workload exchanges can be global and this, again, depends on the threshold level.

4 Simulation of the new algorithm

Simulation was carried out for two topologies: 2D mesh with a wraparound link (2D torus) and ring with 64, 16, and 4 processors. In addition, to study the system efficiency in homogeneous and heterogeneous environments, all simulations were conducted for environments in which processors were equal in terms of processing capacity. And, for environments in which processors were within the processing capacity range of [1, 3], it was supposed that a processor with a processing capacity equal to three has a processing speed three times that of a processor whose processing capacity equals one.

Another issue at hand was the architecture of the parallel system. We considered two architectures: single program multiple data (SPMD) and multiple instruction stream, multiple data stream (MIMD). We chose for the SPMD architecture an [80, 240] byte range with uniform random distribution as the processor data range from which we chose data quantities, and for the MIMD architecture a [6, 202] KB range with normal random distribution (μ=44 KB, σ=400). Also, we chose the task computation time for the MIMD architecture from a range of [64, 768] ms with an exponential distribution (λ=0.006) for each unit of data. These numerical ranges and their distributions were chosen on the guidance from the simulations presented in Lüling *et al*. (1991) and Legrand *et al*. (2004).

When executing the load balancing algorithm in simulation, it is initially necessary to determine the duty of each processor in relation to the algorithm based on its momentary load in the system. In this situation, after each processing operation, any processor in whatever status, carries out its duty to the load balancing algorithm. Then, at specific intervals after repeated processing steps, the migration of workload is allowed between the processors. The simulation of the load migration stage from the load balancing algorithm occurs simultaneously in all of the processors. During workload migration, the time consumed for this transition must be determined; consequently, it must be borne in mind that there may be a common link in a number of workload migration paths between processors. An equal measure of link bandwidth was assigned to each workload migration path.

Finally, after execution of the new load balancing algorithm, efficiency parameters of the algorithm need to be determined. Among these parameters are speedup (serial/parallel runtime) and workload migration percentage. This percentage is computed by dividing the overall amount of workload migration by the overall amount of system workload production.

For each of the resulting functional areas, the experiment was repeated a number of times for confirmation. Eventually, the simulation outputs for eight environments with varying features involving 64, 16, and 4 processors were obtained (Fig. 2).

As an example, in Fig. 2a the result of simulation for an environment with SPMD architecture and 2D homogeneous torus topology involving 64 processors is shown. Here, the amount of workload transfer and

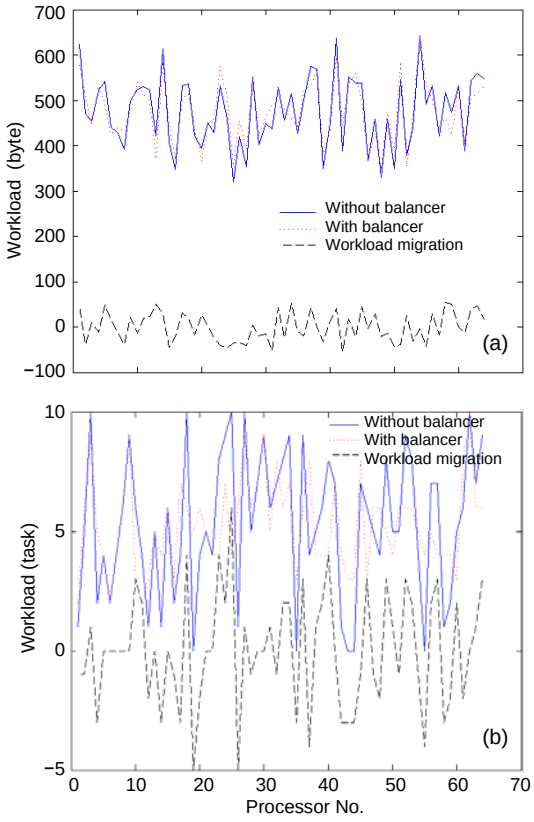

Fig. 2 Results of the new algorithm applied to homogeneous torus topology with single program multiple data (SPMD) architecture (a) and heterogeneous ring topology with multiple instruction stream, multiple data stream (MIMD) architecture (b) involving 64 processors

A negative measure for workload migration for some processors means that these processors have received workload; a positive measure in other processors means that they have transferred a part of their workload

the system workload in the absence and presence of the load balancing algorithm are shown. A negative measure for workload migration for some processors means that these processors have received workload; a positive measure in other processors means that they have transferred a part of their workload. Also, the workload pictured for systems with SPMD architecture is based on the data created for each processor covering the entire simulation time and for systems with MIMD architecture. The tasks created cover the entire simulation time for each processor. As can be seen in the graph of the workload, with the load balancing algorithm, for all environments, the workload of overload processors has decreased and transferred to idle processors. What can be seen as the result of simulation is the processor status for the entire duration of simulation time; therefore, these figures do not show the processor status at any particular moment.

Table 1 shows the characteristics of each experimental environment, the speedup average of the executed experiments, and the workload migration percentage. Note that the number of processors is the same and follows a common pattern, i.e., 64, 16, and 4 processors, but the total processing capacity in various environments differs. In homogeneous environments the total processing capacity equals the number of processors, and in heterogeneous environments, it is twice the number of processors. Therefore, in homogenous environments the maximum speedup equals the number of processors; in heterogeneous environments, it is double that number.

5 Simulation of the central algorithm

One of the existing load balancing algorithms was simulated for comparison. Studies of existing algorithms show that the central algorithm is simulated using a method similar to that for the simulation of the new load balancing algorithm. The reason for this decision lay in the similar behavior of these two algorithms, despite their different structures. Among the significant similarities was the fact that both algorithms allow workload migration only between overloaded and idle processors.

The most significant structural difference between the central algorithm and the proposed load balancing algorithm is their decision method. In the central algorithm, the load balancing decisions are made by one processor and transmitted to all other processors, whereas in the new algorithm all the processors contribute to load balancing decisions.

As mentioned, nearly all of the stages of simulation are similar to those of the novel load balancing algorithm. There are, however, some changes in the load balancing stage of the simulation for initiation of the central load balancing algorithm structure. In addition, a very significant difference in the simulation of these two algorithms relates to path-finding.

In the simulation of the new load balancing algorithm, a path-finder algorithm is not needed, and the algorithm is itself able to find the path between idle and overloaded processors. In the simulation of the central load balancing algorithm, however, there

Table 1 The characteristics of each experimental environment in the simulation with 64, 16, and 4 processors of the new algorithm

No.	Topology	Homo/Hetero	Architecture	Speedup			Load migration percentage (%)		
				64	16	4	64	16	4
1	2DT$\sqrt{n}\times\sqrt{n}$	Homo	SPMD	50.0774	12.1242	3.3313	4.2694	2.1514	1.3359
2	2DT$\sqrt{n}\times\sqrt{n}$	Homo	MIMD	37.9567	11.3794	3.1264	11.4985	11.7232	14.5010
3	2DT$\sqrt{n}\times\sqrt{n}$	Hetero	SPMD	56.1752	13.5603	4.6803	5.2911	3.6572	0.0543
4	2DT$\sqrt{n}\times\sqrt{n}$	Hetero	MIMD	59.0516	13.5706	4.1582	11.6284	15.3935	13.7175
5	Ring n	Homo	SPMD	49.1170	12.5816	3.3318	4.2726	1.7016	1.3359
6	Ring n	Homo	MIMD	35.2917	9.1345	3.1264	17.7797	12.4231	14.5010
7	Ring n	Hetero	SPMD	54.0600	15.9126	4.8683	5.6055	3.0144	1.4098
8	Ring n	Hetero	MIMD	56.2839	17.0976	4.6966	13.6626	15.3693	14.4105

n is the number of processors, n=64, 16, or 4. 2DT$\sqrt{n}\times\sqrt{n}$: two-dimensional torus $\sqrt{n}\times\sqrt{n}$; ring n: ring with n processors. SPMD: single program multiple data; MIMD: multiple instruction stream, multiple data. In homogeneous environments the total processing capacity equals the number of processors, and in heterogeneous environments, it is twice the number of processors

is a need for such an algorithm. This issue becomes more significant when for each new topology used to create a new functional environment, an independent path-finder algorithm has to be programmed to determine the best path between the processors. Obviously, the new load balancing algorithm can be far more easily implemented.

Fig. 3 shows instances of the simulation results of the central algorithm in two functional environments involving 64 processors. Table 2 shows the characteristics of each experimental environment, the average speedup in the experiments carried out for each particular environment, and the workload migration percentage for the central algorithm based on the number of processors involved.

6 Analysis of simulation results

The new load balancing algorithm has the following novel capabilities:

1. This algorithm is able to find a path between the workload receiver and sender processors during the workload migration stage. In other words, this algorithm does not need a distinct path-finder algorithm. The implementation of this algorithm is thus simple in any environment with any topology. This algorithm has a high degree of flexibility in conforming to environmental changes without need for a code change; in contrast, in the case of the central algorithm a change of code is compulsory.

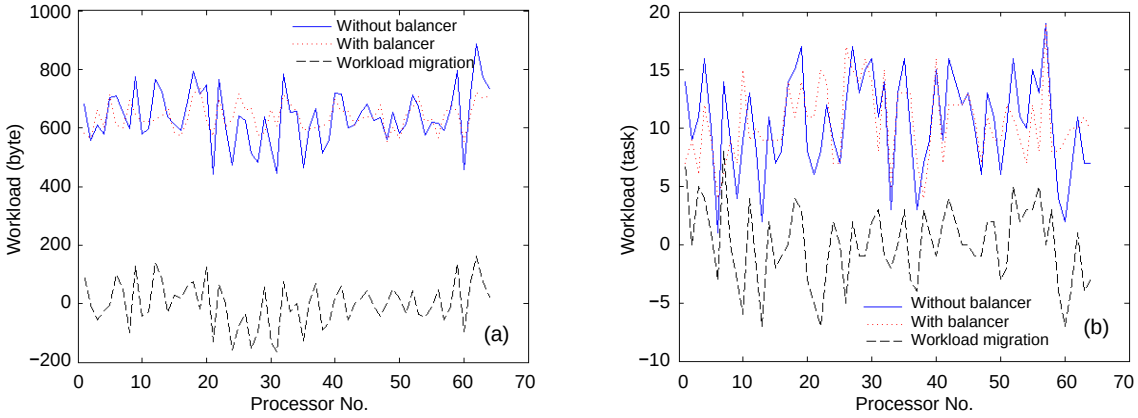

Fig. 3 Results of the central algorithm applied to homogeneous torus topology with single program multiple data (SPMD) architecture (a) and heterogeneous ring topology with multiple instruction stream, multiple data stream (MIMD) architecture (b) involving 64 processors

Table 2 The characteristics of each experimental environment in the simulation with 64, 16, and 4 processors of the central algorithm

No.	Topology	Homo/Hetero	Architecture	Speedup			Load migration percentage (%)		
				64	16	4	64	16	4
1	2DT$\sqrt{n}\times\sqrt{n}$	Homo	SPMD	59.1857	14.0895	3.3406	11.3081	6.5364	1.8642
2	2DT$\sqrt{n}\times\sqrt{n}$	Homo	MIMD	40.6816	11.2786	3.3575	22.3060	11.6868	7.7310
3	2DT$\sqrt{n}\times\sqrt{n}$	Hetero	SPMD	105.6657	23.8489	4.9546	25.2634	16.0546	6.7685
4	2DT$\sqrt{n}\times\sqrt{n}$	Hetero	MIMD	64.9125	15.3083	4.3222	36.0637	23.0547	5.7877
5	Ring n	Homo	SPMD	56.8565	14.3498	3.3400	11.7333	6.9373	1.8642
6	Ring n	Homo	MIMD	41.6678	8.9531	3.3575	19.6533	15.1396	7.7310
7	Ring n	Hetero	SPMD	96.0291	23.4675	4.9546	28.0615	16.1382	6.7685
8	Ring n	Hetero	MIMD	56.2703	22.5466	4.0052	30.4965	27.3537	8.0787

n is the number of processors, n=64, 16, or 4. 2DT$\sqrt{n}\times\sqrt{n}$: two-dimensional torus $\sqrt{n}\times\sqrt{n}$; ring n: ring with n processors. SPMD: single program multiple data; MIMD: multiple instruction stream, multiple data. In homogeneous environments the total processing capacity equals the number of processors, and in heterogeneous environments, it is twice the number of processors

2. With the implementation of a path-finder algorithm, traditional load balancing algorithms always set forth the same path in the process of path finding between two processors, regardless of traffic of path, whereas the path proffered with the implementation of the new algorithm is, apart from being short, not fixed, which enables the proposed algorithm to establish different paths between the two processors involved in load transfer according to the environmental conditions.

3. This algorithm tries, as far as possible, to solve the load balancing problem of an idle processor locally using the least possible amount of communication cost. In addition, decision-making is carried out in a distributive manner.

Also, a study of algorithms simulation outputs provided the following results:

1. In most cases the central algorithm speedup is higher than that of the novel algorithm, but the degree of load migration is much higher; therefore, the communication and transfer cost for the central algorithm is very high. As a result, it may be concluded that the new algorithm requires a longer period of time for load balancing but with a significantly lower degree of load migration. Thus, the new algorithm has a lower communication cost.

2. Implementation of the new algorithm is much simpler than that of the central algorithm.

3. With a decrease in the number of processors in simulation, the performance of the new algorithm improved and, in some cases, the new algorithm functioned even better than the central algorithm. This can be explained in terms of the proportion of the number of each processor's neighbors to the total number of processors. Actually, we reduced the number of processors without changing the number of processor neighbors and thus, in the new algorithm, the speed with which an overloaded processor was sought for an idle one increased. Therefore, it can be stated that with an increase in the number of processor neighbors, the performance of the new algorithm can be improved (Table 1). In the torus topology, each processor has four neighbors and in the ring topology, two, as can be seen in similar experiments executed with only a change in topology (for example, compare experiments 1 and 5 or 2 and 6 for 64 processors in Table 1). The performance of the novel algorithm in the torus topology involving a higher number of neighbors is better. This point occurs less commonly for 16 processors in Table 1 due to a decrease in the number of processors, and thus a decrease in the neighborhood effect.

7 Conclusions

This paper has studied load balancing in parallel systems and presented a new load balancing algorithm with new capabilities, of which the most significant is its independence of a separate route-finding algorithm functioning as path-finder between the sender and receiver workload nodes. Simulation results show that the new algorithm has a high compatibility for implementation in different operational environments. The results also indicate the influence of degree neighborhoods of the processors on system performance.

Further work can include the simulation of other load balancing algorithms and a close study of the effect of an increase in the degree of neighborhood on the probable improvement of the performance of the new algorithm. Note that an increase in the degree of neighborhood can occur physically and non-physically: in physical mode, the objective is an increase in the number of links between each processor and other processors, and thus an increase in the degree of neighborhood; in non-physical mode, the degree of neighborhood is increased due to an increase in the number of messages stored by each processor.

References

Berger, E., Browne, J., 1999. Scalable Load Distribution and Load Balancing for Dynamic Parallel Programs. Proc. Int. Workshop on Cluster-Based Computing, p.1-5.

Borovska, P., Lazarova, M., 2007. Token-Based Adaptive Load Balancing for Dynamically Parallel Computations on Multicomputer Platforms. Int. Conf. on Computer Systems and Technologies, p.31-36. [doi:10.1145/1330598.1330611]

Chhabra, A., Singh, G., Waraich, E., Sidhu, B., Kumar, G., 2006. Qualitative Parametric Comparison of Load Balancing Algorithms in Parallel and Distributed Computing Environment. Proc. World Academy of Science, Engineering and Technology, p.39-42.

Cruz, F.R.B., Mateus, G.R., 2003. Parallel algorithms for a multi-level network optimization problem. *Parall. Algor. Appl.*, **18**(3):121-137.

de Ronde, J.F., Schoneveld, A., Sloot, P.M.A., Floras, N., Reeve, J., 1996. Load balancing by redundant decomposition and mapping. *LNCS*, **1067**:555-561. [doi:10.1007/3-540-61142-8_596]

Fonlupt, C., Marquet, P., Dekeyser, J., 1995. Analysis of Synchronous Dynamic Load Balancing Algorithms. *In*: Parallel Computing: State-of-the-Art Perspective Advances in Parallel Computing, p.1-8.

Fonlupt, C., Marquet, P., Dekeyser, J., 1996. Data-parallel load balancing strategies. *Parall. Comput.*, **24**(11):1665-1684. [doi:10.1016/S0167-8191(98)00049-0]

Garcia, T., Semé, D., 2006. A Load Balancing Technique for Some Coarse-Grained Multicomputer Algorithms. 21st Int. Conf. on Computers and Their Applications, p.301-306.

Giusti, A.D., Naiouf, M., Giusti, L.D., Chichizola, F., 2005. Dynamic load balancing in parallel processing on non-homogeneous clusters. *J. Comput. Sci. Technol.*, **5**(4): 272-278.

Grama, A., Gupta, A., Karypis, G., Kumar, V., 2003. Introduction to Parallel Computing (2nd Ed.). Addison Wesley, USA.

Kuchen, H., Wagener, A., 1990. Comparison of Dynamic Load Balancing Strategies. Technical Report, RWTH Aachen, Department of Computer Science, Aachener Informatik-Berichte (AIB).

Lai, A., Shieh, C., Ueng, J., Kok, Y., Kung, L., 1997. Load Balancing in Software Distributed Shared Memory Systems. IEEE Int. Performance, Computing, and Communications Conf., p.152-158.

Legrand, A., Renard, H., Robert, Y., Vivien, F., 2004. Mapping and load-balancing iterative computations. *IEEE Trans. Parall. Distr. Syst.*, **15**(6):546-558. [doi:10.1109/TPDS.2004.10]

Lüling, R., Monien, B., Ramme, F., 1991. A Study on Dynamic Load Balancing Algorithms. Proc. 3rd IEEE SPDP, p.686-689.

Osman, A., Ammar, H., 2002. Dynamic load balancing strategies for parallel computers. *Sci. Ann. J. Cuza Univ.*, **11**:110-120.

Wu, M., 1997. On runtime parallel scheduling for processor load balancing. *IEEE Trans. Parall. Distr. Syst.*, **8**(2): 173-186. [doi:10.1109/71.577261]

Zamanifar, K., Nematbakhsh, N., Sadjady, R.S., 2010. A New Load Balancing Algorithm in Parallel Computing. 2nd Int. Conf. on Communication Software and Networks, p.449-453. [doi:10.1109/ICCSN.2010.27]