International Conference Massively Parallel Processing – Applications and Development, June 21-23, Delft, The Netherlands.

Local Load Balancing for Data-parallel Branch-and-bound

Dominik Henrich

Institute for Real-Time Computer Systems and Robotics*, Department of Computer Science, University of Karlsruhe, Kaiserstrasse 12, D-76128 Karlsruhe, Germany, e-mail: dhenrich@ira.uka.de

Keywords: load balancing, local communication, SIMD architectures, branch-and-bound, operations research

1. INTRODUCTION

Branch-and-bound (B&B) is a well-known and general combinatorial optimisation technique that is used especially for NP-hard problems where no special purpose algorithm is known. Because of the high problem complexity, parallel implementations are required for speeding up the computations. Algorithms with high and stable efficiency, i.e., utilisation of the processing elements (PEs), are desired. This can be achieved by load balancing techniques that intervene if the PE utilisation worsens.

We use a common version of the sequential B&B formulation as a basis for the parallel algorithm from [8] (see Figure 1). The initialisation is hidden in a function call and is discussed in detail in [1]. The main data structure, the OPEN set, stores single nodes of the search tree. Any search strategy (best-first, depth-first, etc.) can be applied. The data-parallel B&B works quite similarly to the serial algorithm, except that each PE has its own OPEN set. All PEs execute the main loop of the B&B synchronously: select a node from its OPEN set, expand it, insert the evaluated successors back into the OPEN set, and update the incumbent if necessary.

After a couple of iterations, some PEs become idle because their OPEN sets are empty or their memory space is exhausted. In both cases, nodes of the local OPEN sets have to be redistributed such that a maximum number of PEs can continue processing. *Load balancing* is the task of equilibrating the load as evenly as possible. The easier task of *load sharing* is to supply each PE at least with some load. For MIMD computers, many approaches have already been studied [4]. Recently, parallel B&B has been investigated for SIMD computers. A global approach is to map busy to idle PEs in a one-to-one-correspondence and to transfer nodes by global communication. This approach has been applied to the IDA* algorithm in [6, 7]. Static or dynamic trigger conditions indicate when load balancing is reasonable (see [3] also).

* This research work was founded by the Deutsche Forschungsgemeinschaft (DFG) with a stipend in the frame of a "Graduiertenkolleg" of the Computer Science Department, University of Karlsruhe. The work was performed at the Institute for Real-Time Computer Systems and Robotics, Prof. Dr.-Ing. U. Rembold and Prof. Dr.-Ing. R. Dillmann.

```
    procedure Branch_and_bound(root :node);
    set   OPEN;            /* set for unsolved nodes */
    node  x, y,            /* x, y nodes of search tree */
          z;               /* z incumbent */
    begin
          if Solution(root) then z := root;
          else z := ø;
          Init_B&B(root);
          while OPEN contains node x with g(x)<f(z) do
                x := Select(OPEN);
                for all successors y of x do begin
                      if Solution(y) and f(y)<f(z) then
                            z := y;
                      if not Leaf(y) and g(y)<f(z) then
                            OPEN := OPEN u {y};
                end;
                if Trigger(OPEN) then
                      Balance(OPEN);
          end;
          if Solution(z) then Return(z);
          else Return(failure);
    end;
```

Figure 1: B&B algorithm in Pascal notation

The general drawback of global approaches to load balancing is that, with an increasing number of PEs, the global communication of nodes will slow down the algorithm. Thus, in the future, only local approaches seem applicable for massive parallel computers. An example for asynchronous (semi-) local load balancing is the Gradient Model in [5]. But with synchronous processing, this model is not suitable for the quick load changes which occur for tree searches.

A fully local balancing approach is nearest-neighbour-averaging (NNA) (see [10] for a survey). In every step, the mean load of the PEs and their direct neighbours is determined. If a PE has more than the mean load, it transfers part of its load to those neighbours with less than the mean load. The transferred load is proportional to the difference of mean and neighbour's load. To our knowledge, this approach was not applied to SIMD machines up to know.

Here, we present and evaluate a local load balancing approach, especially suited for data-parallel B&B algorithms on SIMD computers, and compare it with NNA. Therefore, we introduce a model of equilibrating liquids and apply it to local load balancing in Section 2. To get more insight on the load balancing process, we separated it from the B&B process and discuss simulation results in Section 3. The question of when to do load balancing is answered in Section 4. Experimental results in the scheduling domain show the improvement by the chosen approach in Section 5.

2. THE LIQUID MODEL

The proposed load balancing method implements a *Liquid Model*. For this model, a flat box is filled with a homogeneous liquid. In the balanced state, the liquid has the same height at any place in the box. If one pours additional liquid into the box at an arbitrary location, the liquid equalises itself such that the height is again the same everywhere. See Figure 2a for illustration.

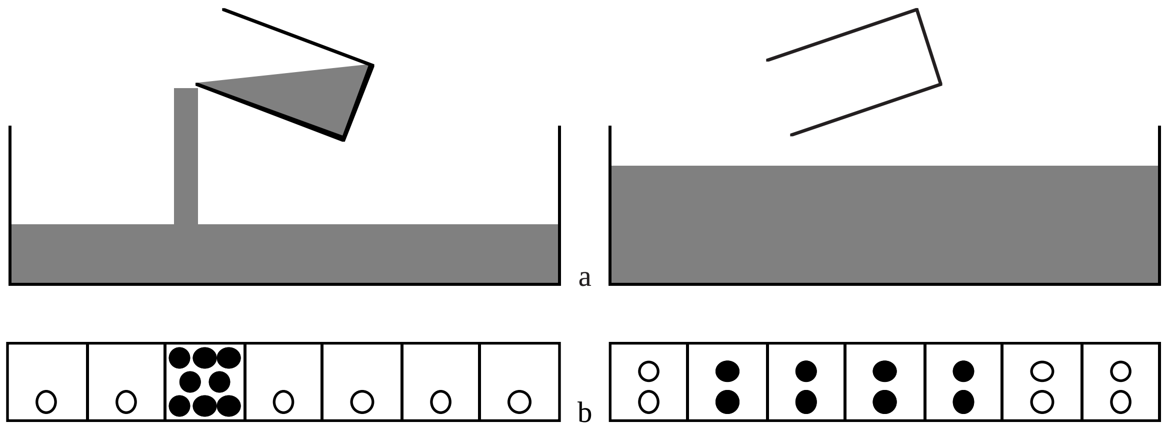

Figure 2: The Liquid Model (a) and its discrete equivalent (b)

This equilibration happens by locally displacing the superfluous liquid to the neighbourhood. By successive displacements, the liquid equalises itself again. This is a local mechanism but with global effect, because none of the additional liquid molecules will "jump" to locations with lacking liquid. The discrete equivalent to the above continuous model is shown in Figure 2b.

In the load balancing process, the load corresponds to the liquid of the Liquid Model. If there is a heavy load at some location and a light load at another location, the load should be transferred from the former to the latter place. Global approaches would detect these locations and transfer the load directly. Instead, with the Liquid Model, the load is transferred implicitly. While there are PEs with light loads, heavily loaded PEs shift nodes to some fo their neighbours and receive nodes from other neighbours. The lightly loaded PEs only receive nodes but do not give any away. By successive shifting, the load is automatically transferred to the PEs with low load.

	PE i shifts a node $\Leftrightarrow$
C0:	$L_i > 0$
C1:	$L_i > 1$
C2:	C1 $\vee$ $[(L_i = 1) \wedge (L_{i-1} > 1)]$
C3:	C1 $\wedge$ $L_i \geq L_{i+1}$
C4:	C2 $\vee$ $L_i \geq L_{i+1}$
C5:	C0 $\wedge$ $L_i \geq L_{i+1}$

Table 1: Different shift conditions depending on the load L_i of PE i

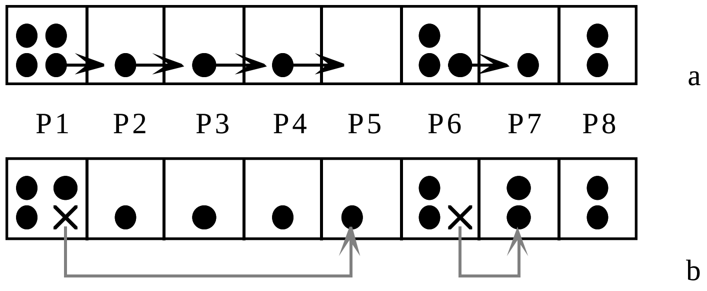

Figure 3: Example of a single load balancing step by the Liquid Model with shift condition C5 in an ring

When load is balanced according to the Liquid Model (LM), the load is transferred by successive shifts. By each shift, one B&B node (respectively elementary work part) is transferred to a direct neighbour. Every PE has to decide whether to shift a node. This is a local decision and can be driven by different conditions. In Table 1, several conditions for any topology with a total order are given. The conditions C0 and C1 use only load information L_i of PE i itself. Conditions C2 through C5 use load information of the preceding or succeeding neighbour. Condition C1 and its extension in C2 guarantee that none of the busy PEs become idle due to shifting nodes.

An example for one LM step in a ring is given in Figure 3. Thereby, the PEs shift nodes to their neighbours on their right iff condition C5 holds true. In (a), the initial situation is shown and the shifts are indicated by arrows. In (b), the result with the "virtually" transferred nodes is shown. Two effects of the LM can be seen from this example. In the left PE group, PE 1 through 5, a global transfer by local shifts is performed. Additionally, in the right PE group, PE 6 through 8, the load is balanced.

Using LM in combination with a B&B algorithm, the transferred nodes are inserted into the OPEN list after each shift. If the best nodes are always selected for shifting, a global best-first B&B strategy can then be approximated after several shifts in chordal rings [9]. Though we presented the LM for rings, other PE topologies can be handled as well. In multi-dimensional topologies, one shift is performed in each dimension. For example, in a 2-dimensional torus, one shift is made horizontally, the other vertically.

There are two main differences between LM and NNA. First, with NNA, none of these global effects are possible by shifting nodes locally. This is because if the load of two neighbours is already balanced, then no nodes will pass this pair in one step. Therefore, this balanced pair form a burden for (virtual) node transfer. Second, the NNA achieves load sharing only as a side effect of balancing the load. This contrasts with LM, where load is shared among the PEs in the first place, and after that load balancing takes place. We investigate the second effect in greater detail in the following section.

3. SIMULATION OF THE LOAD BALANCING PROCESS

The B&B algorithm with dynamic load balancing can be viewed as two interlaced, adversary processes. The B&B disturbs the load distribution by generating or pruning nodes of the search tree in an unpredictable way. On the other hand, the load balancing process tries to re-equilibrate the load by transferring nodes. Investigating solely the balancing process apart from the B&B algorithm makes the effects more clearly recognisable.

We compare NNA and LM with different shifting conditions in ring simulations of size p. As the worst case scenario, one PE holds the total load $L_{tot} = c * p$, where $c > 0$ is an arbitrary integer, and the remaining PEs are idle. Thus, in the perfectly balanced system state, every PE holds c load elements. Each balancing method is executed synchronously until the balanced state is reached. For NNA, a synchronous variation of the method in [10] is used. If a non-integer amount of load should be shifted then the amount is rounded upwards. This insures that the perfectly balanced system state is actually reached.

In Figure 4, the simulation results with different numbers of PEs are shown. As time measure, the maximum number of node transfers for each balancing step (with LM that is one shift) is summed. This measure is reasonable if the start-up time for communication is short compared to the node transfer time itself. The results show almost linear increase in time with p for all balancing methods. For NNA, there are polynomial terms involved but with only small coefficients. Additionally, simulation results not shown here indicate a linear increase in time with the size L_{tot} of initial load. For load balancing, NNA is roughly four times slower than LM. For load sharing, NNA needs about 23 times as long as LM.

In LM, nodes are always shifted, but the load difference to the successor is negative (C3 through C5). Thus, this mechanism gives priority to load sharing. In the worst case, only

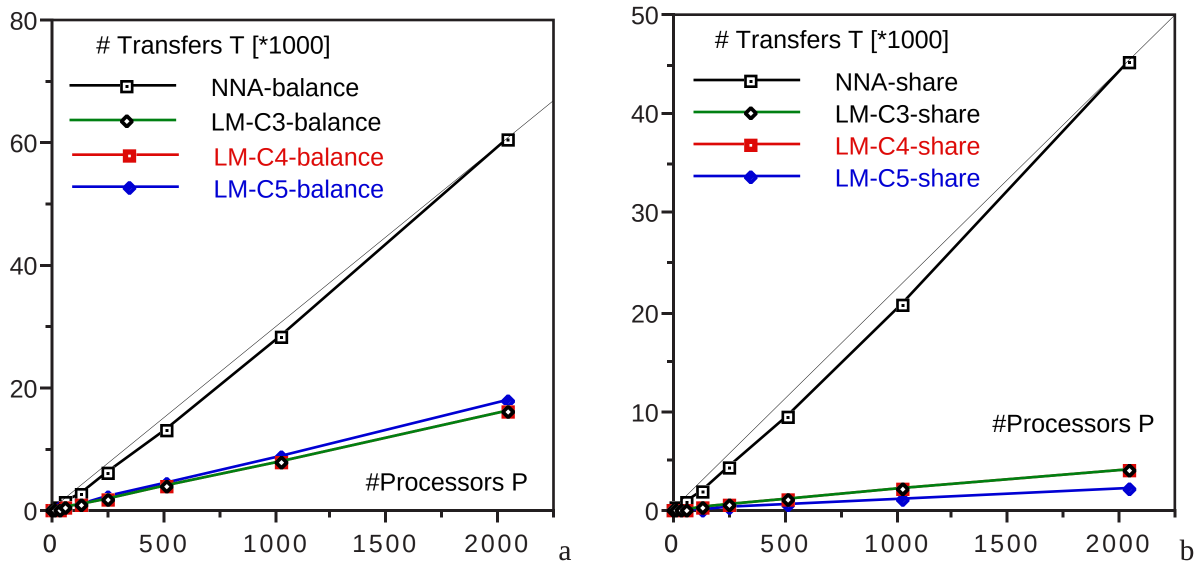

Figure 4: Simulation results of load balancing (a) and sharing (b) by NNA and LM for synchronous processing in the worst case ($c = 5$)

when all PEs are supplied with a node, is the load balanced. The time consumption of LM for load sharing is $T = p - 1$. This explains the huge time difference between LM and NNA with respect to load sharing. In fact, LM does local load sharing in rings with very short time, because an ring with p PEs has diameter of $p\ /2$.

According to the simulation results, the time for load balancing in the worst case is at most polynomial. This is not contrary to the analysis in [11], where the local load balancing according to the α-splitting method needs exponential time in the worst case. An important difference to our method is that a receiver-initiated method is used, thus, load is balanced only when the receiving PE is idle. Another difference lies in the measure of time. There, one transfer (or time unit) includes the communication of multiple nodes. This is reasonable only with long set-up times for communication. Here, we count each shift of a single node as one time unit. This is reasonable with modest set-up times, which holds true for most SIMD machines. Thus, local load balancing is not necessarily a bad choice on SIMD machines.

4. LOCAL TRIGGER MECHANISM

In the last section, we have regarded the load balancing process itself. Together with the B&B process, these two processes are interlaced, i.e. the processes alternate. The question is now, which process should proceed. With SIMD computers, this question becomes even more important because when one process is triggered all PEs are forced to execute this process. In the B&B algorithm, after each iteration, a trigger mechanism tells whether load balancing is necessary.

In former research, two types of mechanisms for triggering synchronous load balancing have been used (see [3, 6, 7]). First, the static trigger S^x indicates load balancing if the number of active PEs A decreases a fixed fraction x of all available PEs p. Thus, $A \leq x * P$. Second, the dynamic trigger D^P or D^K adapt themselves and indicate load balancing if the gain by balanced load is greater than the loss necessary for executing balancing. All of these

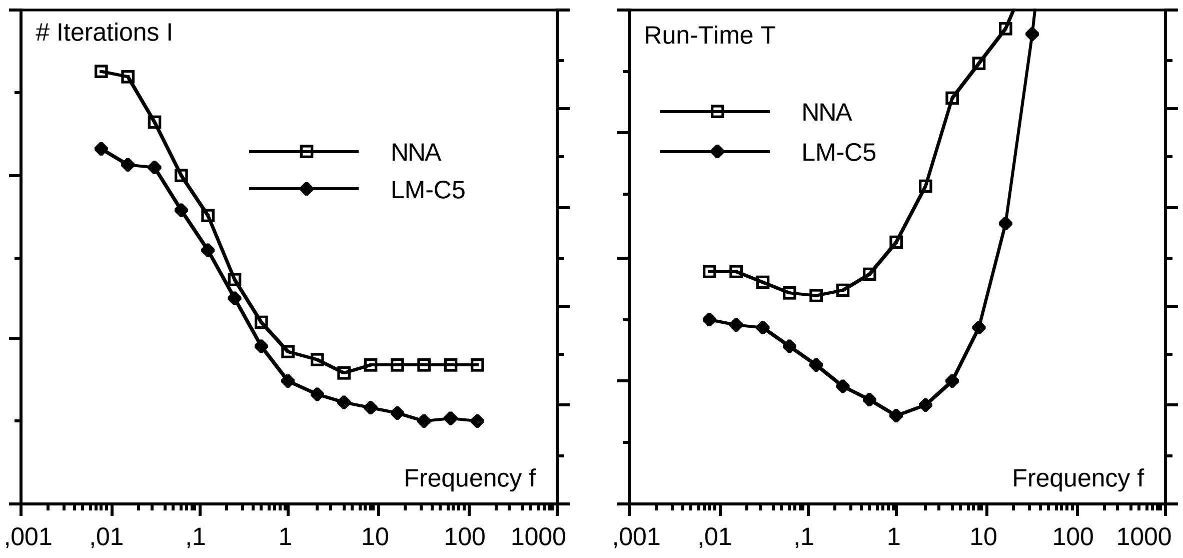

Figure 5: Dependency of iteration number *I* and run-time *T* on frequency *f*

triggers have the drawback of using global information, since in both cases, the number of active PEs is computed.

Here, we use a periodic trigger P^f. The load is balanced with frequency *f*, i.e. every $1/f$-th B&B iteration. The trigger makes no use of any global information, therefore it does not adapt itself to the system state. In SIMD computers, the single instruction stream indicates when load balancing is performed. Thus, which trigger should be used is dependent on the performance of the global information retrieval mechanism (reduction).

Using the periodic trigger, one has to determine the frequency of load balancing. For our problem domain (see Section 5) we have chosen a couple of problem instances and solved them with different trigger frequencies. The mean iteration number and the mean run-time are shown in Figure 5 (unscaled). As expected, the number of iterations decreases with increasing frequency and approaches the optimum. The overall run-time has a minimum. For NNA and LM the frequencies f_{NNA} = 0,125 and f_{LM} = 1 shift per iteration are optimal. Certainly, the optimal frequencies rely on the fraction of communication and calculation time.

5. EXPERIMENTAL RESULTS

To show the efficiency of the LM, a scheduling application domain is used. We solve problems of the following NP-hard scheduling problem. Given: a fixed number of elementary jobs with different processing times that operate on identical machines. The single job is not preemptable and there are no precedence relations between the jobs. The optimisation problem is to find the sequence of jobs on each machine that minimises the total processing time (makespan) of the system.

Before the main loop of the data-parallel B&B is entered, the Selective Initialisation from [1] is executed efficiently supplying the PEs with nodes. The search itself uses a depth-first strategy. After every B&B-iteration, one of the above load balancing methods is invoked, as

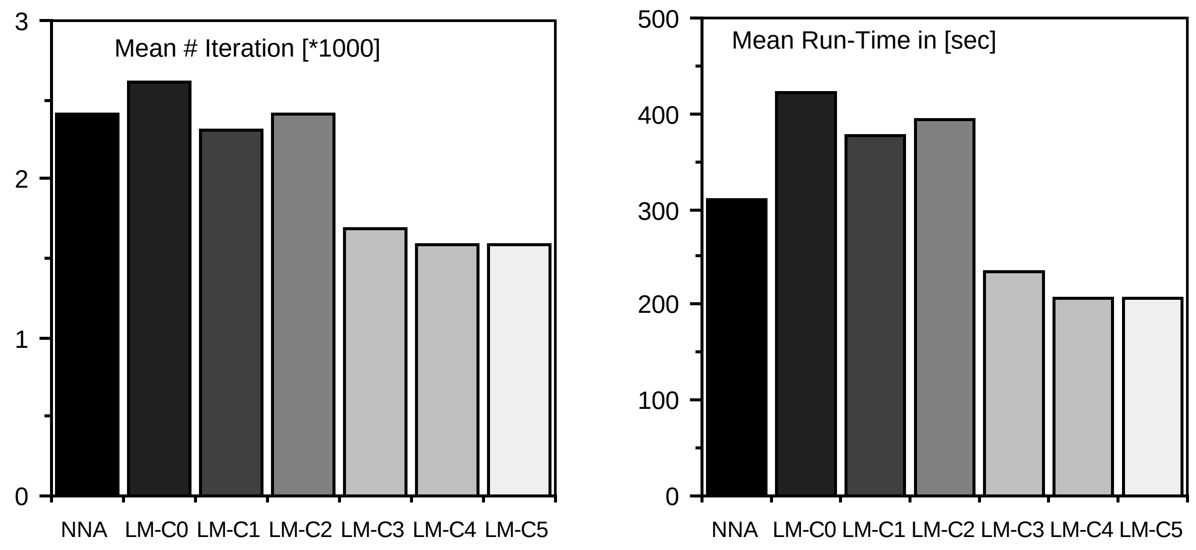

Figure 6: Mean number of iterations *I* and mean run-time *T* of B&B with different load balancing methods.

indicated in Figure 1. To ensure a fair comparison, the number of nodes searched must remain constant with different load balancing methods. Therefore, all solutions are searched and a predefined pruning threshold (first heuristic incumbent) is used.

For the experiments, we used the MasPar SIMD machine MP-1 with 16384 PEs arranged in a two-dimensional torus. Altogether, 20 problem instances with 10^6 up to 10^8 expanded nodes were solved [2]. In Figure 6, the mean number of B&B iterations and the mean run-time are plotted for different local load balancing methods. In the periodic trigger, the minimal frequencies f_{NNA} and f_{LM} from Section 4 are used.

The different shift conditions mentioned in Table 1 show very different behaviour in the experimental results of Figure 6. All three conditions performing only the load sharing task (C0-C2) are far off. In contrast, the LM using any shift condition including load balancing (C3-C5) are very efficient. All of them outperform the NNA method.

6. CONCLUSION

The realisation of the Liquid Model leads to a scalable and efficient dynamic load balancing technique. This is ensured by the strong locality of the algorithm in combination with the ability for global balancing effects. As it has been shown for the ring and the 2-dimensional torus, it is expected to be suitable and efficient for various other topologies.

Besides the simplicity of the presented algorithm, the main advantage lies in the combination of load sharing and load balancing. The most simple version following both the sharing and the balancing task (C5) yields the best overall run time. The Liquid Model gives high priority to the sharing task before balancing is performed. Especially for Branch-and-bound, this prioritisation proves to be efficient. But this property is useful in far more applications than only Branch-and-bound algorithms.

The Liquid Model has outperformed a former local load balancing approach, Nearest-Neighbour-Averaging (NNA). None of the above two properties are reached by the NNA algorithm. It does not have such global effects when executing only a single local operation, and NNA performs load sharing only as a consequence of balancing.

The Liquid Model is independent of the trigger used to initiate load balancing. The introduced periodic trigger does not use any global information and is therefore non-adaptive. The choice of the best trigger mechanism relies on the fraction of the communication and computation speed.

Future research should investigate the applicability of the LM in asynchronous processing. On common MIMD machines, the fraction of set-up time to communication time is greater than on SIMD machines. The periodic trigger has to be adapted and this will certainly change the overall run-time behaviour. Additionally, the Liquid Model should be tested in other problem domains of the parallel Branch-and-bound.

REFERENCES

1. Henrich D., 1993, "Initialization of parallel branch-and-bound algorithms", Workshop on Parallel Processing in Artificial Intelligence (PPAI-93), Aug. 28., Chambery, France.
2. Henrich D., 1994, "Paralleles Branch-and-bound und Scheduling", Universität Karlsruhe, Fakultät für Informatik, Interner Bericht 1994,11. - *in preparation*
3. Karypis G., Kumar V., 1992, "Unstructured tree search on SIMD parallel computers", Technical Report, Department of Computer Science, University of Minnesota, Minneapolis, April 1992.
4. Kumar V., Ananth G. Y., Rao V. N., 1991, "Scalable Load balancing techniques for parallel computers", Technical Report 91-55, Department of Computer Science, University of Minnesota.
5. Lin F. C. H., Keller R. M., 1987, "The gradient model load balancing method", IEEE Transactions on Software Engineering, vol 13, no 1, pp 32-38.
6. Mahanti A., Daniels C. J., 1993, "A SIMD approach to parallel heuristic search", Artificial Intelligence, vol 60, no 2, pp 243-282.
7. Powley C., Ferguson C., Korf R. E., 1993, "Depth-first heuristic search on a SIMD machine", Artificial Intelligence, vol 60, no 2, pp 199-242.
8. Roucairol C., 1988, "Parallel branch-and-bound algorithms - An overwiew", Proc. of the Int. Workshop on Parallel and distributed Algorithms, Gers, France, pp. 153-163.
9. Wah B. W., Ma Y. W. E., 1984, "MANIP - A Multicomputer Architecture for Solving Combinatorial Extremum-Search Problems", IEEE Transactions on Computers, vol C-33, no 5, pp 377-390.
10. Willebeek-LeMair M. H., Reeves A., 1993, "Strategies for dynamic load balancing on highly parallel computers", IEEE Trans. on Parallel and Distributed Systems, vol 4, no 9.
11. Kumar V., Rao N., "Paralllel depth first search - Part II - Analysis", Int. Journal of Parallel Programming, vol 16, no 6, 1987.