Bull. Nov. Comp. Center, Comp. Science, 25 (2006), 69–74
© 2006 NCC Publisher

Load balancing in a heterogeneous computer system by self-organizing Kohonen network

Mikhail S. Tarkov, Yakov S. Bezrukov

Abstract. An effective algorithm is proposed for a balancing computation load in a heterogeneous multicomputer system by the self-organizing neural Kohonen network. The algorithm takes into consideration different performances of the system processors.

1. Introduction

In the general case, a computer cluster is a set of computer units (CU) connected by a network. Each unit has a processor, a memory, and possibly a hard disk. All CUs can have different performances, communication possibilities and operating systems. A structure of a computer system (CS) can be arbitrary and depends upon several factors, such as hardware tools, a goal for which the CS is created, and financial possibilities.

Unlike CS with a shared memory, the computer clusters have difficulties with intercomputer communications because of a relatively low network throughput. To minimize the idle time of all CUs and communications between processes, it is necessary to optimally load all CUs by computations. In the general case, this problem is reduced to optimal mapping of a program graph onto a graph of the computer system [1, 2]. Because of the problem complexity (it is NP-complete), different heuristics are used to seek for the optimal mapping. Currently, the popular ones are methods based on analogies to physics and biology, for example, annealing simulation, genetic algorithms, and neural networks [3]. It is interesting to use a self-organizing Kohonen network [3, 4] for the solution of the mapping problem because of a high ability to parallelize the network functioning.

2. The problem of computation load balancing

A goal of the optimal mapping of parallel program processes (branches) onto processors is to minimize the program execution time. It is equivalent to the idle time minimization for each processor, and simultaneously the minimization of communications between processes placed onto different processors [1, 2].

Let $G_p(V_p, E_p)$ be a graph of a parallel program, where:

- V_p is a set of the program branches, $|V_p| = n$;

- E_p is a set of logical (virtual) channels implementing communications between branches;
- τ_p is a weight of a program branch, the weight τ_p is equal to the number of operations in the branch p;
- v_{pq} is a weight of an edge $(p, q) \in E_p$, the weight v_{pq} being equal to the number of information units transferred between the branches p and q.

Let $G_s(V_s, E_s)$ be a graph of a computer system, where:

- V_s is a set of elementary computers (EC), $|V_s| = m \le n$;
- E_s is a set of connections between ECs;
- ϑ_i is the i-th computer performance;
- δ_{ij} is a time for transferring the information unit between the neighbor computers i and j.

Let $f_m : G_p \to G_s$ be a mapping of the program graph $G_p(V_p, E_p)$ onto the graph $G_s(V_s, E_s)$ of the computer system. A distance (the number of edges of a shortest path) between nodes $i = f_m(p)$ and $j = f_m(q)$ in the graph G_s we denote as d_{ij}. The quality of the mapping f_m is evaluated by the object function

$$\Phi(f_m) = \alpha\Phi_{\mathrm{C}}(f_m) + (1-\alpha)\Phi_{\mathrm{T}}(f_m), \tag{1}$$

where $\Phi_{\mathrm{C}}(f_m)$ evaluates the balance of a computation load, $\Phi_{\mathrm{T}}(f_m)$ evaluate the complete time of intercomputer communications, and $\alpha \in (0, 1)$ denotes a relative weight of $\Phi_{\mathrm{C}}(f_m)$ in the object function.

For the mapping f_m, a complete computation time of the i-th computer is equal to

$$t_i = \frac{1}{\vartheta_i} \sum_{f_m(p)=i} \tau_p.$$

Then

$$\Phi_{\mathrm{C}}(f_m) = \sum_{i=1}^{m} (t_i - t_{\min})^2,$$

where $t_{\min}$ is a running time of the system computers for an ideal load balancing. In [4], the value

$$t_{\min}^{(1)} = \frac{1}{m} \sum_{i=1}^{m} t_i \tag{2}$$

has been proposed as an estimate of $t_{\min}$.

On the other hand, a minimal (ideal) execution time of a parallel program is equal to the value

$$t^{(2)}_{\min} = \frac{\sum_{i=1}^{m} \sum_{f_m(p)=i} \tau_p}{\sum_{i=1}^{m} \vartheta_i} = \frac{\sum_{p=1}^{n} \tau_p}{\sum_{i=1}^{m} \vartheta_i} \tag{3}$$

which can be attained on a computer with the performance equal to the total one for the CS without communication overhead.

Communication overhead is estimated by the function

$$\Phi_{\mathrm{T}}(f_m) = \sum_{p \neq q} v_{pq} \sum_{k=1}^{d_{pq}} \delta_k,$$

where summation is realized for all pairs (p, q) of communicating branches of a parallel program and for all edges in the shortest path between the nodes $i = f_m(p)$ and $j = f_m(q)$ (δ_k is a weight of the k-th edge of the path, $k = 1, \dots, d_{pq}$).

3. The self-organizing neural Kohonen network

A self-organizing Kohonen network consists of one layer of neurons. The number of inputs of each neuron is equal to the dimension L of an input data space. The number of neurons is determined by the number of classes in a set of objects processed by the network. The network is defined by the weight matrix W, where each neuron corresponds to a row W_p and W_{pl} is a weight of the l-th input of the p-th neuron, $p = 1, \dots, n$, $l = 1, \dots, L$.

The network training begins with a random definition of weights in the matrix W and realized by the following iterative algorithm:

1. On the step $t = 1, 2, \dots$ feed the vector x^k, $k = 1, \dots, K$, from the training set to the network input. Calculate the distance $d(x^k, W_i) = \|x^k - W_i\|$ between the input vector x^k and the weight vector W_p, $p = 1, \dots, n$, of each neuron.
2. Choose the neuron-winner W_{win} with the weight vector nearest to the input vector x^k. Modify the weights of the winner and neurons from its neighborhood by the formula $W_p^{t+1} = W_p^t + \eta(x^k - W_p^t)$, where $\eta \in (0, 1)$ is a training coefficient.
3. Go to the following iteration ($t \leftarrow t + 1$).

As the network training proceeds, a neighborhood radius for the winner and the training step are reduced. As a training result, the neuron weight vectors are shifted to the cluster centers of the training set [3, 4].

4. Load balancing by the Kohonen network

The Kohonen network neurons correspond to the parallel program branches. The Kohonen network is imbedded into the topological space, where computers are interpreted as regions. The neighboring regions correspond to the adjacent nodes of the CS graph, the processes being interpreted as points in the space. A point neighborhood is introduced in accordance with the graph G_p of the parallel program. A process is in the neighborhood with radius r of another process, if the distance between them in the graph G_p is less than r. As a result of training, processes, described by the neuron vectors, are mapped onto the regions corresponding to the system computers.

Let us consider the computation load balancing by the Kohonen network in a heterogeneous CS. The neurons are in correspondence with the branches of the parallel program and are described by the vectors W_s, $s \in V_p$, in the topological space.

The sequential mapping algorithm:

1. Let initially the branches of the parallel program be arbitrarily mapped onto the computers of the system. For this distribution compute the initial value of the object function Φ_t for this distribution.
2. Choose a computer randomly and calculate new values of the object function for mapping each branch with its neighborhood onto this computer. If there are new values of Φ_i, $i = 1, \ldots, n$, less than Φ_t, then choose a minimal value $\Phi_t = \min_i \Phi_i$ and save the respective mapping.
3. Repeat Step 2, while there is a possibility to diminish the object function.

A parallel version of the mapping algorithm is as follows:

Partition a set of computers into subsets, and find a new minimal value of the object function for each subset independently. Then summarize the results for all subsets.

5. Experiments

The experiments have been executed for the optimal mapping of typical program graphs onto typical computer system graphs. The following mappings were investigated:

- a line and a mesh onto a line;
- a ring and a torus onto a ring; and
- a line, a ring, a mesh, and a torus onto a complete graph.

As the mapping criterion, the object function

$$\Phi = \sum_{i=1}^{m} (t_i - t_{\min})^2$$

has been used, where t_i is a real processor load, and $t_{\min}$ is the processor load for the balanced mapping of processes onto processors (here we set

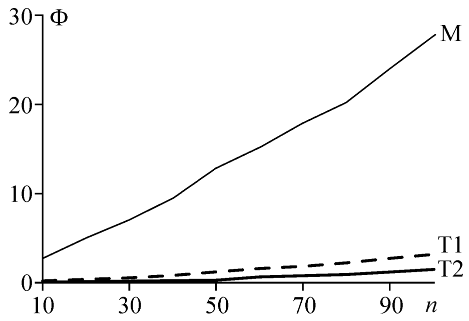

Figure 1. Comparison of mapping versions with (T1 and T2) and without (M) accounting for heterogeneity of a computer system

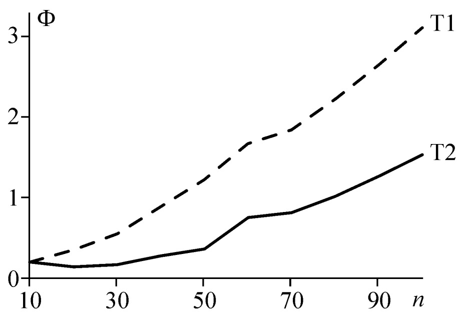

Figure 2. Comparison of mapping versions with T1 and T2 estimates for a minimal processing time

$\alpha = 1$ for expression (1)). The values $t^{(1)}_{\min}$ from (2) and the proposed here $t^{(2)}_{\min}$ from (3) are used as $t_{\min}$.

The performance analysis of modern computers shows that their relative performances can be approximately estimated as random integers uniformly distributed on the segment $[1, 40]$. The relative weights of the program branches in the experiments are distributed on the segment $[1, 20]$.

Figures 1, 2 show examples of mean values of the function Φ in the experiments for different versions of mapping a ring onto a complete graph with $m = 10$ nodes. The curves T1 are calculated for the Wyler's estimate (2) and the curves T2 are calculated for the estimate (3). Figure 1 shows that a consideration of the CS heterogeneity allows one to essentially diminish the non-uniformity of a computation load (approximately, by 10 times!). Also, the experiments show that estimate (3) provides a better mapping than estimate (2) by K. Wyler [4] (see Figure 2) and the mapping performance is slightly dependent upon the choice of a program graph and a CS graph.

6. Conclusion

A new algorithm for mapping a parallel program graph onto a graph of a heterogeneous computer system is proposed. The algorithm is based on the self-organizing neural Kohonen network and can be easily parallelized. The experiments show that the developed algorithm implements an effective mapping of typical program structures (a line, a ring, a mesh, and a torus) onto different structures (a line, a ring, a mesh, a torus, and a complete graph) of the heterogeneous computer system. A parallel version of the mapping algorithm can be used for the dynamic load balancing in heterogeneous computer systems.

References

[1] Bokhari S.H. One the mapping problem // IEEE Trans. Comp. — 1981. — Vol. C-30, No. 3. — P. 207–214.

[2] Tarkov M.S. Mapping parallel program structures onto structures of distributed computer systems // Optoelectronics, Instrumentation and Data Processing. — 2003. — Vol. 39, No. 3. — P. 72–83.

[3] Osowski S. Neural Networks for Information Processing. — Moscow: Finansi i Statistika, 2002 (In Russian).

[4] Wyler K. Self-organizing mapping in a multiprocessor system // IAM-93-001. — 1993.