A Simple Method for Dynamic Scheduling in a Heterogeneous Computing System

Janez Brest and Viljem Žumer

Faculty of Electrical Engineering and Computer Science, University of Maribor, Slovenia

A simple method for the dynamic scheduling on a heterogeneous computing system is proposed in this paper. It was implemented to minimize the parallel program execution time. The proposed method decomposes the program workload into computationally homogeneous subtasks, which may be of the different size, depending on the current load of each machine in a heterogeneous computing system.

Keywords: heterogeneous computing, task scheduling, optimization, heuristic algorithm.

1. Introduction

A computer system of PCs, workstations, minicomputers etc., connected together in a local area network or wide area network represents a large pool of computational power [19].

Research in the field of heterogeneous computing began in the mid 1980s; since then it has grown tremendously [8]. From the scientific community to the federal government, heterogeneous computing has become an important area of research and interest.

Heterogeneous computing includes both parallel and distributed processing [9, 13]. The virtual heterogeneous associative machine concept makes the distributed machines appear as one single machine. By exploiting the different features and capabilities of a heterogeneous environment, higher levels of performance can be attained.

In general, the goal of heterogeneous computing is to assign each subtask to one of the machines in the system so that the total execution time (computation time and inter-machine communication time) of the application program is minimized [23, 12, 7, 11, 16, 18, 24]. Mapping can be specified statically or determined at runtime by load balancing algorithms. In heterogeneous computing the structure of a problem may be known, but the structure of the system can change dynamically.

In a heterogeneous system that consists of PCs or workstations, individual subtasks can be matched to the best-suited processor. During the design/synthesis of a heterogeneous system, the designer must [8]:

- consider all relevant factors, costs, constraints, and objectives during the design;
- decide on the number and type of processors to be included in the system;
- decide on the interconnection between the selected processors;
- determine the effective use of the designed heterogeneous system to perform the given application task;
- map and schedule the subtasks for execution on the processors.

The dynamic scheduling strategies fall under different models, which include the schemes based on predicting the future from the past loads, the task queue model, and the diffusion model [26].

The paper introduces a heuristic algorithm for dynamic scheduling for distributed computing

and provides case studies for two computationally intensive tasks. An improved load balancing strategy in master-slave structures, where the size of task to be assigned to a slave is determined by the last two packets and their execution time, is presented.

The rest of the paper is organized as follows. In Section 2 a brief overview of the heterogeneous computing is given. In Section 3 the task scheduling and the load balancing are described. A simple task scheduling scheme and the algorithm for dynamic scheduling on heterogeneous computing system are presented. The results of the implemented algorithm on two classes of optimization problems are presented in Section 4. In Section 5 the conclusion remarks are given.

2. Heterogeneous Computing

High performance networks of workstations are becoming increasingly popular as a parallel computing platform [14]. Both message passing and software distributed shared memory paradigms have been developed on such distributed platforms. An important performance bottleneck in these systems is network latency, which is poorer than that in high-speed parallel computer interconnection networks.

A heterogeneous computing (HC) system consists of a number of autonomous and independently scheduled heterogeneous computers. A primary objective in many research projects dealing with heterogeneous computing is the minimization of the job completion time [3, 4].

The network layer can provide interconnectivity between computing sites. Communication tools suitable for HC are a message passing interface (MPI) [5, 8], a parallel virtual machine (PVM), portable programs for parallel processors (P4), etc.

Local Area Multicomputer (LAM) [5, 6] is a parallel environment development system of independent computers. It features the MPI programming standard, supported by extensive monitoring and debugging tools.

Figure 1 shows a part of the heterogeneous computing system that has been used in our experiments. There are many different operating systems (Solaris, Linux, etc.) on computers connected in a local-area network. We used LAM/MPI tool as parallel environment.

3. Task Scheduling and Load Balancing

The problem of load partitioning and scheduling in a multiple-processor system has been the area of active and sustained research over the past two decades [8, 25].

The most critical aspect of a task-scheduling algorithm is the strategy used to allocate problems to slaves. Generally, the chosen strategy will represent a compromise between the conflicting requirements for independent operation (to

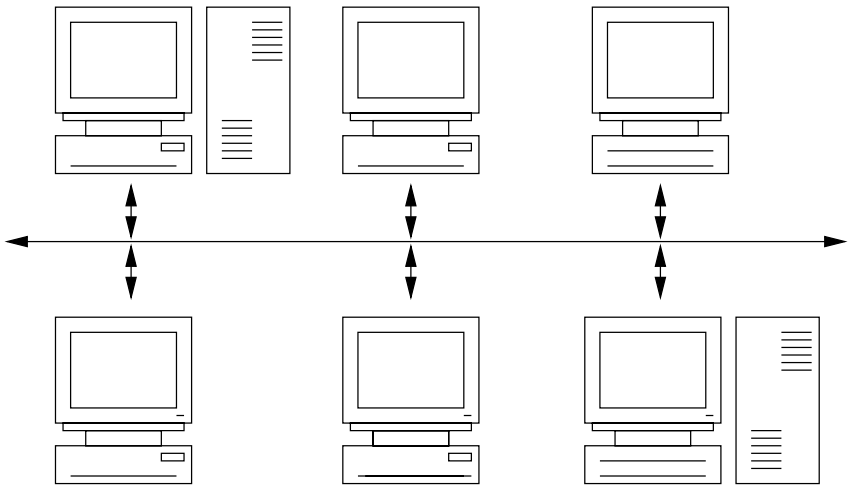

Fig. 1. An example of a heterogeneous computing system, which consists of PCs and workstations.

reduce communication costs) and global knowledge of computation state (to improve load balance).

3.1. Master/Slave

Figure 2 illustrates a simple task scheduling scheme that is nevertheless effective in moderating a number of processors [9].

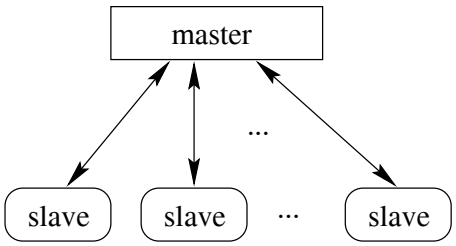

Fig. 2. A master/slave scheme.

In the so-called embarrassingly parallel problem, computation consists of a number of tasks that can be executed more or less independently, without communication [9]. These problems are usually easy to adapt for parallel execution. The same computation must be performed using a range of different input parameters. The parameter values are read from an input file, and after parallel program execution, the different computation results are written to an output file.

3.2. Arbitrarily Divisible Jobs

The problem of heterogeneous load balancing and task scheduling is examined for two practical workload paradigms [10]: indivisible-task jobs and arbitrarily divisible jobs.

This paper deals with load balancing and task scheduling in the context of arbitrarily divisible jobs. The arbitrarily divisible load model can be used in an application where the load consists of a large number of small elements with identical processing requirements. Examples can be found in the application for image and signal processing, and also in iterative algorithms [14].

If the execution time per task is constant and each processor has the same computational power, then it is a good idea to decompose the available problems into equal-sized sets and allocate one such set to each processor. In other situations, each slave task repeatedly requests parameter values from the input task, computes using these values, and sends results to the output task. The execution time can vary. The input and the output task cannot expect to receive messages from various slaves in any particular order. This nondeterminism affects only the allocation of problems to slaves and the ordering of results in the input file, but not the actual results computed.

3.3. Our Method

We have modeled the computing performance of each computer with a single parameter: its response time needed for the execution of the task.

At time $t+2$ the new task size $s_p(t+2)$ for processor p is a function of two previous values of response times $\tau_p(t+1)$, $\tau_p(t)$ and task sizes $s_p(t+1)$, $s_p(t)$:

$$s_p(t+2) = f(s_p(t+1),\ \tau_p(t+1),\ s_p(t),\ \tau_p(t)),\quad t = 0,\ 1,\ 2,\ \ldots \tag{1}$$

This single parameter includes more aspects of heterogeneity of each computer during the given operating conditions. The task size (*chunk size*) should be smaller than $\frac{number\ of\ total\ size}{number\ of\ processors}$.

3.4. Algorithm

The algorithm for dynamic scheduling on a heterogeneous computing system is presented in this section.

There is one node that represents *master* or *manager*, all other nodes are *slaves* or *workers* (see Fig. 2). The master does not compute, but it

- collects global status information of the system,
- performs the dynamic scheduling algorithm that also distributes tasks and/or parameter values,
- collects the results.

Additionally, the master reads data from the input file, distributes the data, makes comparisons between temporary solutions to find the best one, etc.

Slaves do not have information about the global status of the heterogeneous system such as system load, program execution progress. Each slave has to do only two jobs: to make a computation in order to find the solution, i.e. task execution, and make time measurements, i.e. local information. Local information is sent from slaves to the master that has an overview of all activities in the heterogeneous computing system. The master uses the function (1) to calculate the new task size. The master can store all values $s(t)$ and $\tau(t)$, but because of the time locality and space assumption for storing those values, we used only the last two values. Given a parameter $\epsilon > 0$, the function f is defined as follows.

Let

$$k = \frac{\frac{s(t+1)}{\tau(t+1)}}{\frac{s(t)}{\tau(t)}} \qquad (2)$$

If

1. $k < 1-\epsilon$: the new task size can be increased by factor 2 (the load execution time will be better if the new task size is greater than the current task size),
2. $k > 1+\epsilon$: the new task size can be decreased by factor 2,
3. $1 - \epsilon \leq k \leq 1 + \epsilon$: task size does not need to be changed.

Parameter ϵ should not be too small or too large. In the former case, the function for calculating

MASTER and **SLAVE** algorithms in pseudo-code:

```
Module MASTER
begin
    /* performs task scheduling and load balancing */
    Read data from input files
    Use simple heuristic method to compute an initial solution
    Send input data to slaves
    Send to slaves:
        the initial solution
        size of subtask
    while (program application not finished) do begin
        Receive temporary solution from any slave
        if this solution is currently the best one then begin
            Receive all data from the slave with the best solution
            Store this temporary best solution and other data
        end
        Receive measurements from the slave
        Use current and previous measurements to compute new size of the next subtask
        Send the new subtask to the slave
    end
    Print the best solution, results,...
end master
```

```
Module SLAVE /* worker */
begin
    /* Slave computes and sends measured data to the master */
    Receive input data from master
    while (something to do) do begin
        Receive the initial or temporary solution and a size of subtask
        Compute to find solution
        Send the local best solution and time measurements to the master
    end
end slave
```

the new task size is too sensitive to all small changes in the system load. In the latter case, the dynamic scheduling strategy is rigid. In both cases lower levels of performance were observed. Of course, there are lower and upper limits of minimum and maximum task size. We have obtained the highest performance for $0.05 \leq \epsilon \leq 0.1$. We have decided to use factor 2 because of the following reasons: increasing (dividing) the task size is a simple operation, the increase of the sequence 1, 2, 4, 8,... is rather high. The disadvantage of using factor 2 is that the sequence contains some values only.

4. Results

We have evaluated proposed dynamic scheduling strategy on two classes of optimization problems, where heuristic search algorithms were used. Both of the optimization problems belong to NP-hard problems, and therefore we used dynamic scheduling on computers connected in a local area network to solve the problem faster and/or better.

The first one was the problem of continuous speech recognition. We have implemented an algorithm for bigram word clustering [17, 22] on a heterogeneous computing system. The corpus consisted of approximately 750.000 words and the vocabulary size was 20.000. We used master and slave algorithms, described in previous section, and 10 PCs and workstations (one slave per computer) connected into a local area network, and LAM/MPI tool.

The total execution time of parallel program is the sum of communication time and computation time. Utilization can be defined as the ratio between the computation time and the total execution time. Table 1 shows utilizations of slaves, which were about 0.9 (the total execution time of parallel program was $700.437s$). The parallel version of the algorithm for bigram word clustering found a better final solution (approximately 10%) than the sequential version, if the execution times of both algorithms were equal. Note that obtained 10% better final solution in the bigram word clustering is very good.

Slave	Computation time $[s]$	Utilization $[\%]$
1	646.719	92.33
2	626.861	89.50
3	632.048	90.24
4	627.857	89.64
5	662.287	94.55
6	612.457	87.44
7	659.365	94.14
8	659.221	94.12
9	660.629	94.32

Table 1. Utilization.

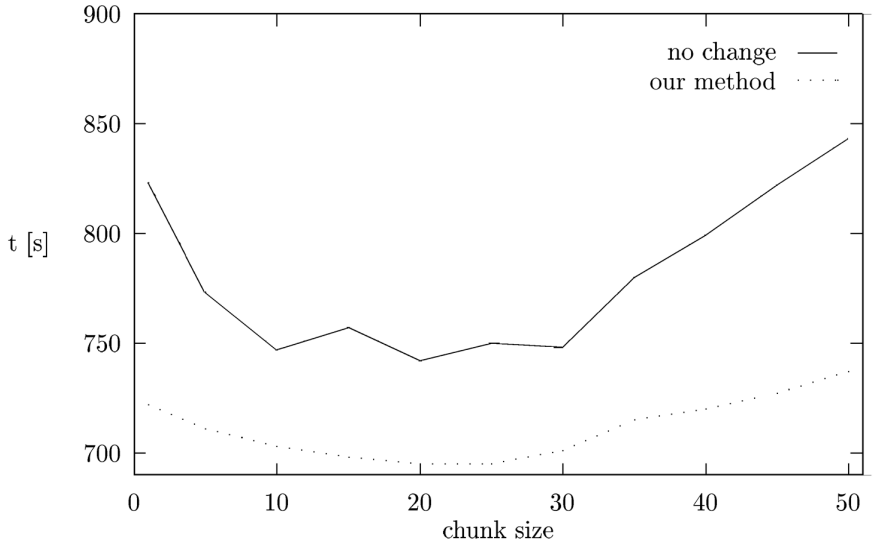

Fig. 3. The execution time vs. chunk size, when the size of program application was 1000 units.

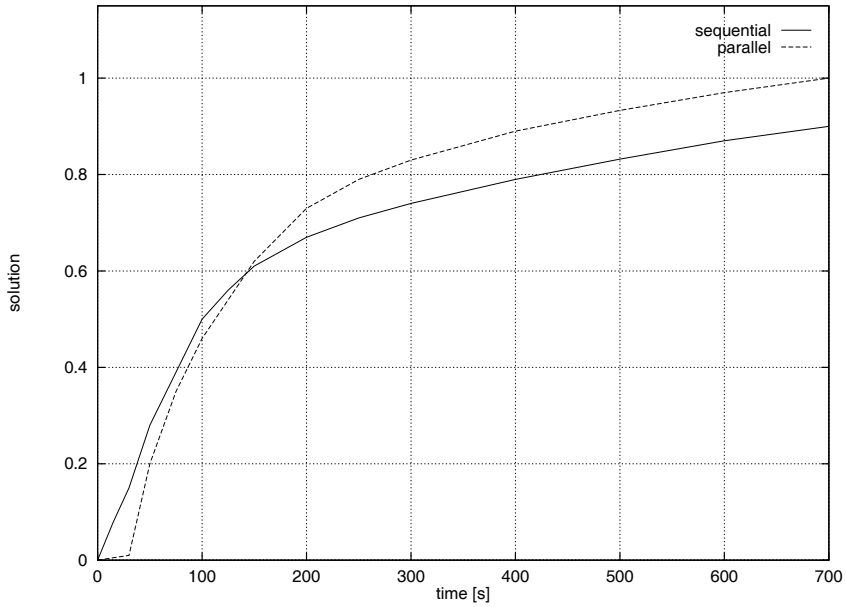

Fig. 4. Solutions obtained by parallel and sequential algorithms vs. time.

Figure 4 shows normalized solutions obtained by parallel and sequential algorithm for bigram word clustering, respectively. Time τ indicates the point, where the solution obtained by the parallel version of the program is equal to the solution obtained by the sequential version. At the beginning of the program execution, parallel version needs some time for an initialization of the computing system and data transfer. Therefore, before the time τ, which is not known a priori, the solution obtained by the sequential version is better than the solution obtained by the parallel version. In our experiment we wanted to determine the value of the time τ, when both solutions were close to each other. The value of the time τ is obtained by the averages of 20 independent runs of both parallel and sequential program versions: $130\ s < \tau < 160\ s$.

If computers are heterogeneous, they have processors of different power, so it is difficult to measure the *speedup* of the parallel computation, since it is not clear which computer to use as a reference of the sequential computation.

Figure 3 shows the comparison of our method (initially, the ϵ is set to the chunk size of the task) with scheduling algorithm when there is no change of the chunk size. It can be noticed that the simple method gives better results in terms of the execution time (up to 10%) compared to the scheduling algorithm with no change in chunk size. The disadvantage of dynamic scheduling with no change in chunk size is that the execution time is increased when the chunk size is too small or when the chunk size is too big. The obtained results show that our dynamic scheduling algorithms perform better when the initial chunk size is small.

The second class of optimization problem used to evaluate presented dynamic scheduling method was asymmetric traveling salesman problem (ATSP) [15, 20, 1]. In this experiment we used a homogeneous computing system, which consists of 41 computers (one processor per computer) connected using ATM network. Results of the execution time and speedup for two ATSP instances with 358 and 443 nodes, respectively, are presented in Table 2. Other problem instances are smaller than *rbg*443 (currently the largest problem in TSPLIB library with 443 nodes) and therefore speedup values were lower than those obtained by *rbg*443.

Problem	$T_1\ [s]$	$T_{41}\ [s]$	Speedup $\frac{T_1}{T_{41}}$
rbg358	1359.21	48.08	28.87
rbg443	3591.50	112.80	31.84

Table 2. Speedup.

Some results obtained in our experiments are shown in Table 3. The first column indicates

Problem	Optimum	min	%	avg	%
rbg358	1163	1166	99.74	1170.85	99.33
rbg443	2720	2720	100.00	2720.70	99.97

Table 3. Results for the asymmetric TSP instances.

the ATSP instances and the number of cities. The shortest tours are known for all of them and are shown in the second column. The optimal solution values can be found in TSPLIB library (`http://softlib.rice.edu/softlib/tsplib/`) [20, 21]. For each of the problem instances, 30 independent runs were performed. In the last two columns the best tour solutions and the averages of the runs are shown. The results (tour lengths) of other problem instances are similar to the results obtained by sequential algorithms (see [1], [2], page 247). We have done many experiments on heterogeneous system consisting of different processors, operation systems (HP-UX, Linux, Solaris, etc.) and the obtained results of the tour lengths were similar to the results presented in Table 3. Recall that the time needed by parallel algorithms was smaller (see Table 2.)

The proposed method is simple, but the results are comparable to other approaches described in the literature.

5. Conclusion

This paper presents dynamic scheduling on a heterogeneous computing system. Our method decomposes the program workload into computationally homogeneous subtasks, which may be of different size, depending on the current load of each machine in the heterogeneous computing system. Experimental results of two practical applications are presented to evaluate the described dynamic scheduling.

In future research, we intend to apply the power of the proposed method to a wider range of practical problems.

References

[1] J. BREST, J. ŽEROVNIK, An approximation algorithm for the asymmetric traveling salesman problem, *Ricerca Operativa*, **28**, pages 59–67, 1999.

[2] J. BREST, J. ŽEROVNIK, An approximation algorithm for the asymmetric traveling salesman problem, in N. Callaos, M. Bennamoun, and J. Aguilar, editors, *World Multiconference on System, Cybernetics and Informatics*, volume 3, pages 244–249, July 1997.

[3] J. BREST, V. ŽUMER AND M. OJSTERŠEK, Dynamic scheduling on a network heterogeneous computer system, *Lecture Notes in Computer Science*, **1557**, pages 584–585, 1999.

[4] J. BREST, V. ŽUMER, M. OJSTERŠEK, Dynamic Scheduling on a PC Cluster, in J. Carrol and H. Haddad, editors, *ACM Symposium on Applied Computing, SAC'99*, pages 496–500, 1999.

[5] G. D. BURNS, R. B. DAOUD, J. R. VAIGL, LAM: An Open Cluster Environment for MPI, in *Supercomputing Symposium '94*, Toronto, Canada, June 1994.

[6] G. D. BURNS, The Local Area Multicomputer, in *Proceedings of the Fourth Conference on Hypercube Concurrent Computers and Applications*, ACM Press, March, 1989.

[7] M. CERMELE, M. COLAJANNI, G. NECCI, Dynamic load balancing of distributed spmd computations with explicit message-passing, *Proceedings: Sixth Heterogeneous Computing Workshop (HCW'97)*, pages 2–16, April 1997.

[8] M. M. ESHAGIAN, editor, *Heterogeneous Computing*, Artech House, Inc., Norwood, MA 02062, ISBN 0-89006-552-7, 1996.

[9] I. FOSTER, *Designing and Building Parallel Programs*, Addison-Wesley, ISBN 0-201-57594-9, 1995.

[10] E. HADDAN, Load Balancing and Scheduling in Network Heterogeneous Computing, in M. M. Eshagian, editor, *Heterogeneous Computing*, pages 224–276, Norwood, MA 02062, ISBN 0-89006-552-7, 1996. Artech House, Inc.

[11] D. HENSGEN, editor, *Proceedings: Sixth Heterogeneous Computing Workshop (HCW'97)*, IEEE Computer Society Press, Los Alamitos, CA, April 1997.

[12] H. HLAVACS, C. W. UEBRHUBER, Simulating load balancing on heterogenous workstation clusters, *Lecture Notes in Computer Science*, **1557**, pages 533–540, 1999.

[13] K. HWANG, Z. XU, *Advanced Computer Architecture: Technology, Architecture, Programming*, McGraw-Hill, New York, 1998.

[14] P. H. JOSEPH, S. VAJAPEYAM, Program-Level Control of Network Delay for Parallel Asynchronous Iterative Applications. In *3rd International Conference on High Performance Computing*, pages 88–93, Trivandrum, India, December 1996. IEEE Computer Society Press.

[15] E. L. LAWLER, ET AL., editors. *The Traveling Salesman Problem*, John Wiley & Sons Ltd., 1985.

[16] W. M. LIN, W. XIE, Minimizing Communication Conflicts with Load-Skewing Task Assignment Techniques on Network of Workstations. *INFORMATICA: An International Journal of Computing and Informatics*, **23**, pages 57–66, 1999.

[17] S. MARTIN, J. LIERMANN, H. NEY, Algorithms for bigram and trigram word clustering. *EUROSPEECH'95. 4th European Conference on Speech Communication and Technology*, pages 203–213, September 1995.

[18] B. MONIEN, ET AL., Efficient use of parallel and distributed systems: From theory to practice. *Lecture Notes in Computer Science*, **1000**, pages 62–77, 1995.

[19] B. RAYKUMAR, editor, *High Performance Cluster Computing: Architectures and Systems*. Prentice Hall - PTR, NJ, USA, 1999.

[20] G. REINELT, TSPLIB — A traveling salesman problem library, Technical Report 250, Universität Augsburg, Institut für Mathematik, Augsburg, 1990.

[21] G. REINELT, TSPLIB — A traveling salesman problem library, *ORSA Journal on Computing*, **3** (No. 4), pages 376–384, 1991.

[22] M. SEPESY MAUČEC, Statistical language modeling based on automatic classification of words. In *Advances in speech technology*, pages 173–180, 1998.

[23] M. TAN, et al., Minimizing the Application Execution time Through Scheduling of Subtasks and Communication Traffic in a Heterogeneous Computing System. *IEEE Transaction on Parallel and Distributed Systems*, **8** (No. 8) 173–186, August, 1997.

[24] B. WILKINSON, M. ALLEN, *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*, Prentice-Hall, Englewood Cliffs, NJ 07632, USA, 1998.

[25] M. Y. WU, On Runtime Parallel Scheduling for Processor Load Balancing, *IEEE Transaction on Parallel and distributed systems* **8** (No. 2), pages 173–186, February, 1997.

[26] M. J. ZAKI, S. PARTASARATHY, W. LI, Customized dynamic load balancing, in B. Raykumar, editor, *High Performance Cluster Computing: Architectures and Systems*, Prentice Hall – PTR, NJ, USA, 1999.

Received: January, 2000
Revised: July, 2001
Accepted: December, 2001

Contact address:

Janez Brest, Viljem Žumer
University of Maribor
Faculty of Electrical Engineering and Computer Science
Smetanova 17, 2000 Maribor, Slovenia
Phone: ++386-62-220-7452
Fax: ++386-62-211-178
e-mail: janez.brest@uni-mb.si

PROF. DR. VILJEM ŽUMER is head of Institute of Computer Science on Faculty of Electrical Engineering and Computer Science, University of Maribor, Slovenia. Prof. Žumer is a member of the IEEE Computer Society.

JANEZ BREST is a PhD candidate in the Department of Computer Science, at the University of Maribor. He received his BSc degree in computer science from the University of Maribor in 1995, and MSc degree in 1998. He is a member of the IEEE Computer Society, and the ACM.