Implementation and Utilization of a Heterogeneous Multicomputer Cluster for the Study of Load Balancing Strategies

Per H. Andersen and John K. Antonio
Department of Computer Science
Texas Tech University
Lubbock, TX 79409-3104
{p.andersen, antonio}@ttu.edu

Abstract

The focus of this paper is the implementation and utilization of an inexpensive heterogeneous multicomputer cluster for the study of load balancing techniques. The basic conclusion of the paper is that excellent performance is possible provided that the proper choices among various parameters and implementation options of the load balancing schemes are employed.

1: Introduction

The heterogeneous multicomputer cluster (HMC) hardware consists of seven Intel-based workstations: two Pentium 100s, each with 32Mbytes; a 486/DX100 with 20Mbytes; a 486/DX66 with 8 Mbytes; a 486/DX40 with 8 Mbytes; and two 386/DX40s, each with 8 Mbytes. Each system is equipped with a 3Com 509 network interface card. The interconnection network is a 24-port 3Com Superstack II 1000 10BT ethernet switch.

Linux was chosen as the operating system for the HMC and LAM/MPI (Local Area Multicomputer /Message Passing Interface) as the parallel application development package. LAM/MPI was developed and is distributed freely by the Ohio Supercomputer Center (OSC) at Ohio State University. It follows the MPI 1.1 Standard as proposed by the MPI Forum (and includes some additional enhanced features).

The implemented load balancing strategies are dynamic because they divide and allocate computational work based on the actual loading of the machines during execution. Each of these load balancing schemes is defined based on many different static parameters and implementation options, including: polling schemes; thresholds that define whether a machine is idle or busy; counters that affect how often a machine responds to or makes a request for work; techniques for initializing the workload; techniques for terminating the search; and work splitting mechanisms. The basic conclusion of the paper is that excellent performance is possible provided that the proper choices among these parameters and implementation options are employed. Sample timing results are included, which are based on the application of load balancing techniques to parallel search algorithms for solving the traveling salesman problem.

2: Parallel search techniques

There are three processing phases associated with implementing a parallel search algorithm that are considered: the startup phase; the working phase; and the shut-down phase [1].

Within the *startup phase*, four possible initialization methods are described in [1]: root initialization, enumerative initialization, selective initialization, and direct initialization. For this study, root initialization and a variation on direct initialization are investigated.

During the *working phase*, as a workstation completes it's work, it has to either request more work from busy workstations (idle initiated) or respond to requests to take work from busy workstations (busy initiated) [3]. Work is performed by expanding nodes of a search tree. Work is divided between workstations by a queue-sharing method [4].

The technique used to terminate the search during the *shutdown phase* depends on the type of search algorithm that is implemented. For further details, refer to [4].

3: Load balancing schemes

Dynamic load balancing is activated during the working phase of a parallel search. Different tasks (i.e., partitions of the search space) can be re-partitioned and distributed among workstations during program execution. One of two mechanisms are considered for initiating the re-partitioning and distribution of work: an idle initiated work request or a busy initiated work request.

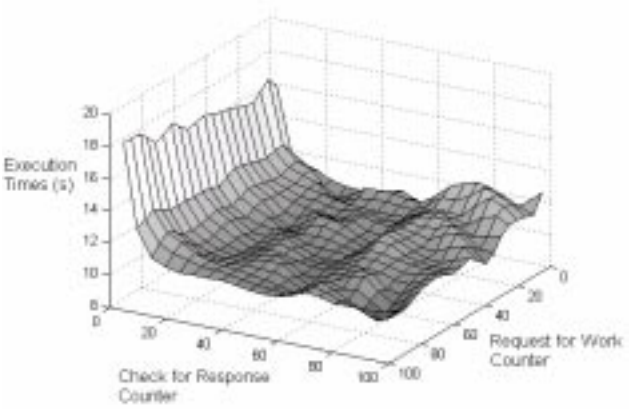

Fig. 1: Counter value study for the idle initiated scheme.

For idle initiated, idle workstations start making requests to receive work when the size of their work queue drops to below an "idle" threshold. The value of this idle threshold as well as user-selected counters (defined in the next section) must be defined.

For busy initiated, busy workstations start making requests to give away work when the size of their work queue exceeds a "busy" threshold. Similar to the idle initiated case, the value of the busy threshold as well as a user-selected counters must be defined.

To implement the idle initiated approach, a target machine must be selected by the workstation requesting work. Similarly, to implement the busy initiated approach, a target machine must be selected by the workstation attempting to off-load work. Three possible techniques for selecting a target machine are considered: asynchronous round robin (ARR); random polling (RP); and global round robin (GRR) [2].

4: Data collection and empirical studies

Numerous experiments were conducted with many possible combinations of parameter values and implementation options. Structured variations of parameter values and/or implementation options were made while the remaining parameters and options were kept fixed. For some cases considered, there was not measurable performance variations observed by varying the parameters of interest. In other cases, the variation in performance was similar for several different cases.

Fig. 1 is an illustrative example of the types of timing effects recorded. The figure shows execution time versus the (user-selected) counter values – higher counter values

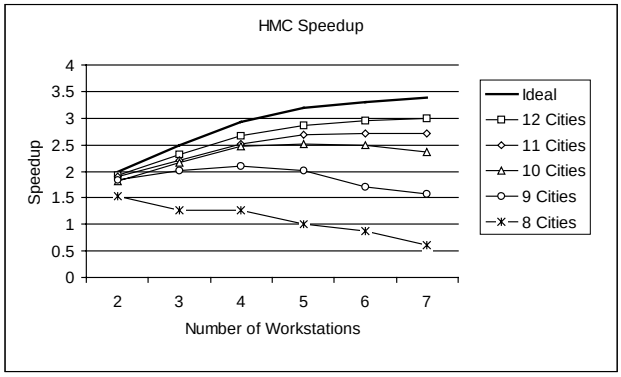

Fig. 2: Typical speedup curves.

correspond to more local work being performed between successive checks for a response (or request) for work. From the figure, note that poor performance results if the machines check for responses too often, relative to the amount of time spent performing local computation. Fig. 2 shows typical speedup curves (relative to the fastest machine) obtained for various problem sizes. The ideal speedup curve is not linear because the relative speeds of the seven machines in the HMC were not uniform. The machines are numbered in descending order of speed.

5: Conclusions

Based on the data collected and for the problem sizes and search algorithms considered, it was found that idle initiated load balancing is generally preferred over busy initiated. It is concluded that good performance is possible on the HMC provided that the threshold and counter parameters of the load balancing schemes are properly selected. Performance can be further tuned and optimized by proper selection of other parameters and implementation options; further details are in [4].

References

[1] D. Henrich, "Initialization of Parallel Branch-and-Bound Algorithms," *2nd Int'l Workshop on Parallel Proc. for Artificial Intelligence*, Chambery, France, Aug. 29 1993.

[2] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing*, Benjamin/Cummings Publishing Co., Redwood City, CA, 1994.

[3] N. G. Shivaratri, P. Krueger, and M. Singhal, "Load Distributing for Locally Distributed Systems," *IEEE Computer*, pp. 33-44, Dec. 1992.

[4] P. Andersen, *Implementation and Utilization of a Heterogeneous Multicomputer Cluster for the Study of Load Balancing Strategies*, M.S. Thesis, Dept. of Computer Science, Texas Tech Univ., Lubbock, TX, Aug. 1997.