Proceedings of the 5th WSEAS International Conference on Telecommunications and Informatics, Istanbul, Turkey, May 27-29, 2006 (pp287-292)

A Dynamic Workload Balancing Technique of a Text Matching Algorithm on a Cluster

ODYSSEAS EFREMIDES
University of Hertfordshire, Hatfield, UK,
in collaboration with IST Studies
Dept. of Computer Science
72 Pireos Str., 183 46, Moschato, Athens
GREECE

GEORGE IVANOV
University of Hertfordshire, Hatfield, UK,
in collaboration with IST Studies
Dept. of Computer Science
72 Pireos Str., 183 46, Moschato, Athens
GREECE

Abstract: A dynamic workload allocation model which utilizes a data pool manager is investigated herein. It aims at heterogeneous multicomputer environments and the implementation is written in C using the MPICH NT 1.2.5 message passing interface for Microsoft Windows based clusters. The algorithm utilized is the data parallel adaptation of the Brute Force exact pattern matching algorithm. Performance evaluation for both homogeneous and heterogeneous system configurations is experimentally investigated.

Key–Words: Parallel pattern matching, Heterogeneous clusters, Workload Allocation, Data pool model

1 Introduction

Applications designed for heterogeneous systems built on a cluster of workstations must implement a mechanism for dynamic workload allocation based on their processing capacity. Nevertheless, for large scale systems consisted of a variety of workstations, predetermining a specific workload amount for each node based on its processing capabilities can become a cumbersome methodology.

An automated *dynamic workload allocation mechanism* that utilizes a *data pool manager* for the parallel - exact string matching problem over a heterogeneous cluster of workstations is investigated herein. The pattern matching algorithm employed is a data parallel adaptation of the well known sequential Brute Force [3] [4] algorithm. The implementation is tested and evaluated in homogeneous and heterogeneous environments with an adjustable number of workstations and is compared to a simple static workload allocation version [12].

2 Data Pool Workload Management

The dynamic workload allocation mechanism follows a different approach in the utilization of available data parallelism compared to the static relaxed - parallel implementation [12]. Instead of partitioning data according to the number of cluster nodes and then assigning a part of roughly the same size to each node that will be processed during the computation phase, data are split into *packets* that form a pool which consists of the entire search file. Worker nodes cannot use this information without being managed by the pool manager. Note, that all segments in the pool have the same size, with the exception of the last one whose size may be smaller than the selected size value.

Information concerning the data packets can only be found at the pool manager whose role is undertaken by the root node. To manage this data, a dynamic packet availability table is built and stored locally, based on the size of the search file and the predefined packet size. Each table position acts as an identifier of a packet. If that specific packet is available for processing the position is appropriately marked, while packets that have been scanned for occurrences by a node are marked as unavailable. Hence, the program *returns* with the entire number of pattern occurrences within the search file when all packets have been processed and therefore the pool is empty. Since this table is only available at the manager, packet size should be such so that it does not incur too much network traffic, due to often requests for identifiers. At the same time it has to be small enough so that processing time for each packet is not excessive at slow nodes.

Partitioning cannot be performed by worker nodes and is a part of the *world initialization process* at the root. This happens because all nodes may have the entire search data stored locally or in a shared medium but not the information on how to use it. This information is only obtained after a message exchange with the pool manager. Each communication concerns only one data packet. Each packet is appropriately overlapped locally by $pattern\_length - 1$ so that no occurrences are lost between packets. Finally, in order to handle packets of any size without any problems due to physical memory restrictions, internal packet splitting during the read - load process is also performed. These packets are also appropriately overlapped, without requiring additional I/O activity, again ensuring that no occurrences are lost.

The entire process, concerning task allocation, is designed so that waiting time is extremely low: the time to process a single request includes a scan in the availability table to locate a non processed packet and then a 4 byte communication of the *id* (*identifier*) of the packet follows. This id will be used by the worker node to locate the specific packet in the search file and will no longer be available for processing.

Nodes that process packets faster than other nodes will request more packets more often than slower ones. As a result the workload is automatically distributed to nodes based on their processing capabilities. Moreover, the execution time is not bounded by the slowest node as workload is appropriately balanced amongst nodes of different capabilities. Finally, the overhead from inefficient utilization of the entire cluster due to nonexistence of an adequate number of packets in order for all nodes to be busy at the end of the scan process is extremely small, because the packets are of relatively small size and require only tenths of seconds of computation time.

2.1 Orchestration and Mapping

After the initialization phase, the pool manager waits for incoming requests from any node that wants to be sent the identification of a packet to process. At the same time, worker nodes immediately send a request message to the root. The pool manager processes requests one by one and sends to each node one packet for scanning. More specifically an integer identifier of the segment of the search file is sent and is used by the worker to resolve the data to be scanned. When a worker finishes processing the packet and becomes idle a new request message is sent and so on. When all packets in the pool have been scanned the root sends a terminate message to all workers and a collective communication follows so that results of the computation are sent back to the root.

The overhead from the communication over the network should be relatively small in a 100Mbit/s Ethernet infrastructure. This happens because the request message is only 4 bytes in size and the identifier of the packet sent with each communication to the requesting node is also 4 bytes long. Even if these are encapsulated in a larger packet for communication reasons [5], the overhead should still be trivial.

```
1. Initialize parallel environment
2. Deal with pattern file
     a. Open pattern file
     b. Get pattern file size
     c. Load pattern
3. Deal with search file
     a. Open search file
     b. Get search file size
4. If root node (pool manager)
     a. Initialize the packet availability table based on search file size
        and the predefined packet size
     b. While available packets exist
          i. Wait for work request message from a node
         ii. On receive send an available packet identifier to the source
             of the message
        iii. Mark the packet as no longer available
     c. Send termination message to all worker nodes
5. Else (all worker nodes that access the pool)
     a. While termination message not received
          i. Request a packet from the pool manager
         ii. Wait for pool manager reply
        iii. Resolve the segment to scan from the received message
         iv. Read up to read factor size or end of packet
          v. Sum occurrences = BruteForce()
         vi. If end of packet not reached go to iv; else go to a
6. Reduce by summing occurrences (collective communication)
7. Halt
```

Figure 1: Pool Manager Based Dynamic Workload Allocation Algorithm

2.2 The Algorithm

Figure 1 shows the algorithm of the pool manager based dynamic workload allocation model. Although this implementation depends on communication, its general overhead to performance should be minimal. Hence, the implementation should be efficient enough with a good linear speedup on a heterogeneous or homogeneous cluster.

2.2.1 The Data Secure Adaptation

Two implementations where developed. The first one, called *simple*, implements the entire procedure as described above. The other, described herein, suggests a *simplified* and more *secure* data management approach. Hence, instead of having the dataset scattered amongst a large number of workstations, the search files and patterns are only located in one node, the *data secure pool manager* and no other nodes have direct access to this information.

This mainly differentiates this modified version in the fact that it relies even more on the available interconnection network and its speed. Specifically, it requires the transmission of the pattern that will be used in the search phase. Furthermore, during the computation process each communication between the pool manager and worker nodes involves real data transfer instead of identifiers that help locate the data locally or in a file server. Since the data will only be read by one workstation it also incurs higher I/O overhead.

3 Experimental Results

3.1 The Cluster Configuration

The developed workload allocation model is evaluated using a cluster consisting of maximum 16 workstations with Intel Pentium 4 2.40GHz CPUs and 512MB of RAM and 16 Intel Celeron 2.20GHz CPUs with 256MB of RAM. Microsoft Windows 2000 Professional is the operating system installed on all workstations. Appropriate combinations achieve a heterogeneous or homogeneous computing environment, as desired and denoted where applicable. Note, that the cluster is dedicated [6] during all conducted tests.

Nodes of the cluster communicate using a 100Mbit/s Ethernet interconnection and are actually part of a larger local area network.

3.2 Development Environment

The implementation is written in C, utilizing the Argonne National Laboratory MPICH NT Message Passing Interface (MPI) library version 1.2.5 [8] [9]. Access to the search files is realized through the Win32 application programming interface, in order to support files of huge sizes, based on the provided 64bit addressing.

3.3 The Dataset

The developed workload allocation model is tested using a number of files of sizes ranging from 161MB and up to 3575MB. These text database files contain the Homo sapiens genome and are obtained from the public section of GenBank National Institutes of Health [7], part of the International Nucleotide Sequence Database Collaboration. The utilized pattern sizes range from 3bytes to 60bytes.

3.4 Results Analysis

As found experimentally, pattern size does not noticeably affect performance and only extreme variations in pattern size are expected to have a minimum impact [10]. Results from the use of only one pattern are shown for that reason.

As the search file size increases, the benefits from parallelization in this implementation become more noticeable, because the higher computation requirements overlap the communication time. Thus, there is a noticeable drop in the performance for the two smallest files as the system scales. As depicted in

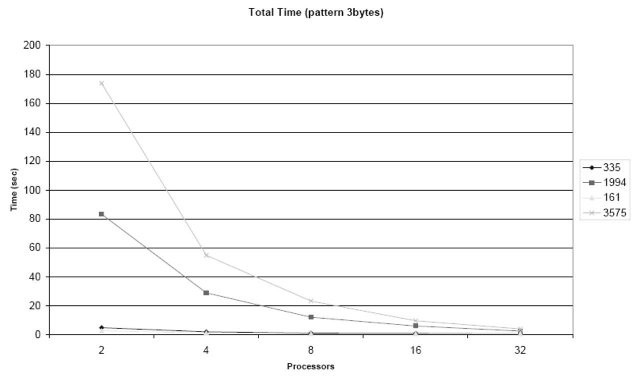

Figure 2: Total execution time for different search files with a pattern of 3 bytes

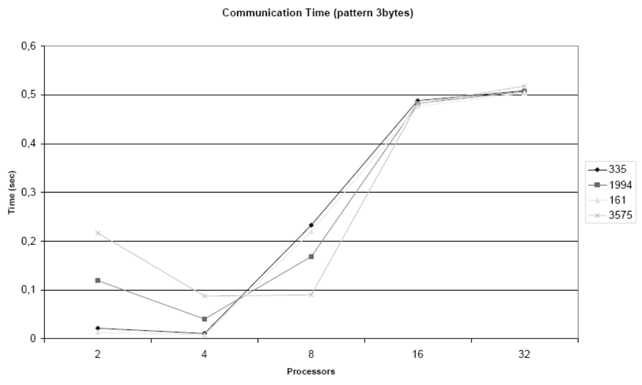

Figure 3: Communication time for all search file scans using the 3 bytes pattern

Figure 2 when all nodes are utilized the total execution time for all search files is roughly identical. This is because communication overhead increases as the number of nodes increase and is almost equivalent for all search files as seen in Figure 3. Hence, the overhead of communication overlaps the gains from parallelization in this case, being clearly visible for files that require less computation, compared to a sequential implementation.

To evaluate and compare the performance of the parallel dynamic workload allocation implementation with the static one [12], supplementary tests are presented below. The main characteristic of the static implementation is that it (statically) assigns the same amount of work to all nodes, including the root, and ignores workstation performance characteristics. The static and dynamic workload allocation versions are tested in a heterogeneous environment utilizing four, eight and sixteen processors. In each case, only half of the workstations are of the same type. Since the static implementation is relaxed, the root node has no implementation specific management duties and hence all nodes take part in the computation process. Con-

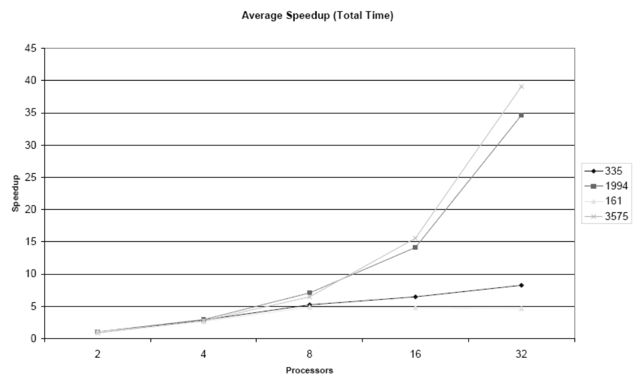

Figure 4: Speedup based on the average of results using all patterns

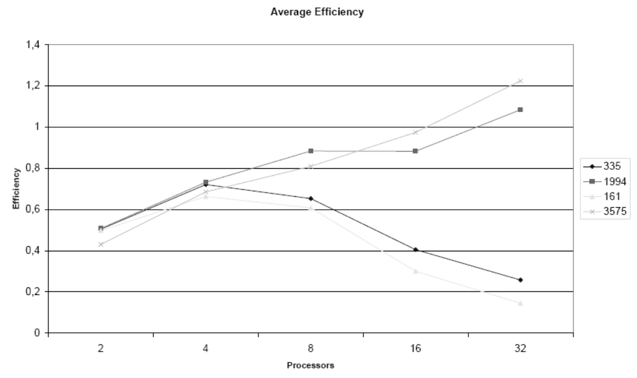

Figure 5: Parallelization efficiency based on the average of results using all patterns

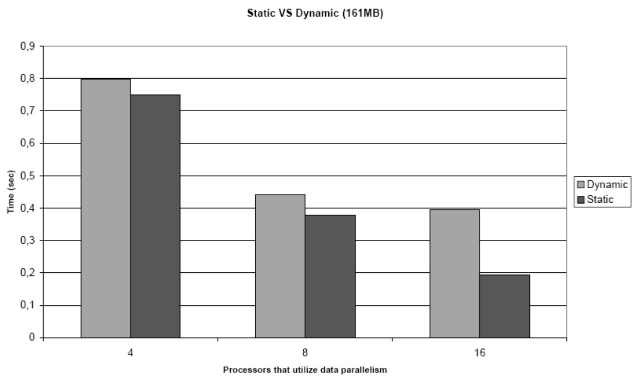

Figure 6: Static VS Dynamic Performance Comparison (Search File: 161MB)

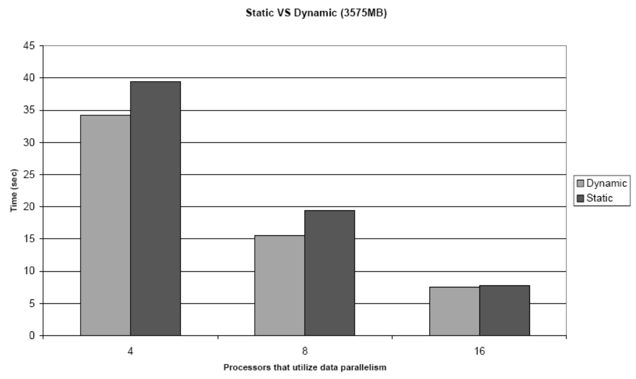

Figure 7: Static VS Dynamic Performance Comparison (Search File: 3575MB)

sequently, for comparability reasons, one additional node is added in each of the dynamic workload allocation implementations to assume the role of the pool manager, for the same level of available data parallelism to be exploited. Results from the comparison using the smallest and largest search files are presented in Figures 6 and 7.

The outcomes of the comparison indicate a satisfactory performance of the dynamic implementation for the largest search file. This further supports the conclusion that the benefit from dynamic allocation is mostly visible for files that require more computation, in which case the percentage of the communication overhead in the total execution time is small. Still, as the utilized multicomputer scales and computation time is significantly reduced, the communication overhead becomes noticeable. Additionally, for very small files any advantages of this implementation are hidden by the communication time. Note though that the performance difference for the two versions is between 0.1 and 0.2 seconds for the small file, while the benefit from the dynamic adaptation for the large file is in the range of 1 to 5 seconds.

In contrast to the satisfactory performance of the simple pool manager, the secure data pool manager showed extremely poor results, as expected, mostly due to the slow 100Mbit/s Ethernet interconnection of the networked workstations that was available, which cannot be compared to local I/O read times, in conjunction with the relatively fast computation times that characterize the pattern matching problem on the assembled parallel system. Figures 8, 9, and 10 show the total execution time as well as the computation and communication times, respectively. Note that the implementation is not compared with the static workload allocation version, since it would have no meaning under the available test environment.

Although, computation times decrease indicating good speedup values, the total execution time is severely limited by communication (Figure 10) and thus results are extremely poor. Actually, the time required to run the application is roughly equal to the time required to copy the search file from one node to another, based on a transfer rate of 10MB/s achieved at best in an 100Mbit/s Ethernet. The only noticeable improvement in the execution time appears in the

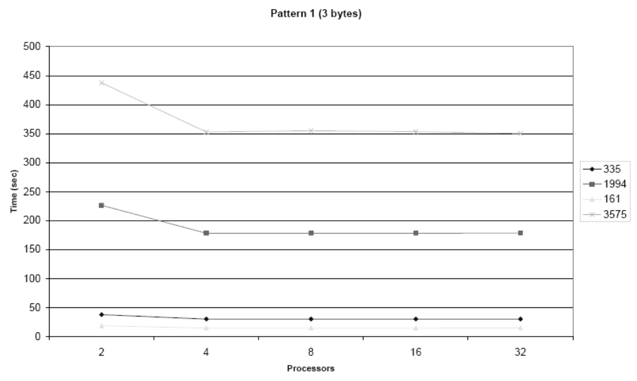

Figure 8: Execution time for different search files with a pattern of 3 bytes

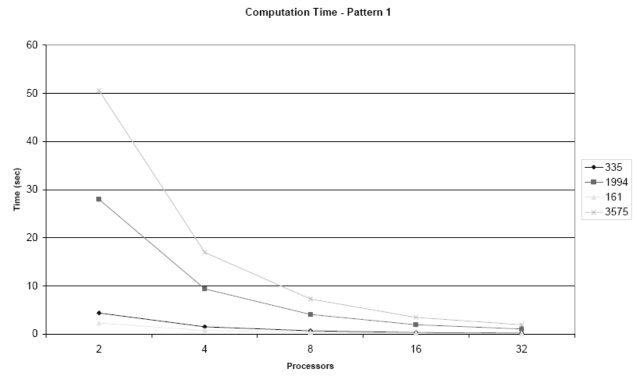

Figure 9: Computation time for different search files with a pattern of 3 bytes

transition from 2 to 4 nodes (Figure 8). This is due to the fact that when using two nodes, there is only one worker and the application does not at all take advantage of data parallelism. However, when the nodes become four, three of them are workers and thus parallelization is exploited. This also validates the threefold improvement of the computation time in the transition from two to four nodes, compared to twofold improvements that appear as processor numbers double and support the theory in the case of a relaxed algorithm (Figure 9). In all other cases the benefits from parallelization are hidden by communication time.

Communication impact on system performance is evident in the speedup obtained separately from the average execution and computation times using all patterns. Speedup results, based on computation time (Figure 12), show excellent implementation performance that supports the theory: speedup increases almost linearly along with the number of processors. It should be noted that, the root does not take part in the computation process, so the number of processors actually scanning the search file is one less than the number shown. In contrast, the speedup acquired

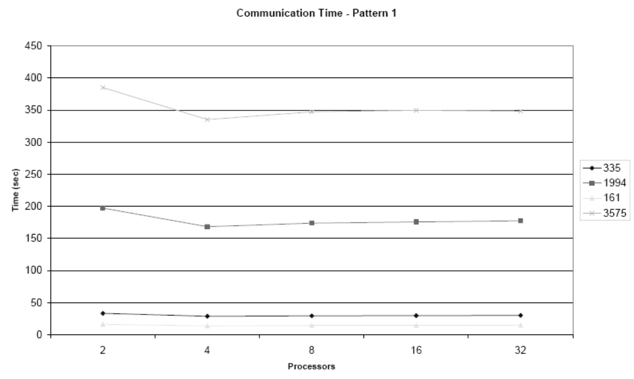

Figure 10: Transmission time for various search file sizes with a pattern of 3 bytes

form the total execution time (Figure 11), which includes communication time, is extremely poor to such an extent which leads to the conclusion that the use of more than four nodes is pointless, when solving the parallel pattern matching problem over a slow network interconnection. This simply happens because the network is unable to sustain a fast enough data rate to each processor, making obvious the communication overhead in the total execution time.

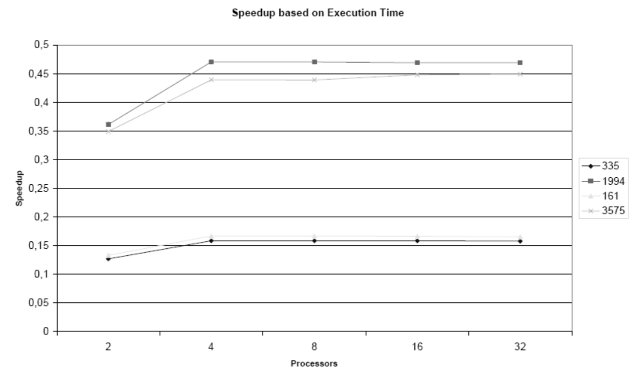

Figure 11: Speedup based on average execution times of all patterns for the four search files

4 Conclusion Remarks

A dynamic workload allocation model, based on a pool manager implementation was investigated, herein. The implementation exhibits an excellent performance for larger files that require more processing time. For such files the communication time with the pool manager is negligible to the total execution time. Hence, there appears a significant performance increase in a heterogeneous system configuration, compared to the static workload allocation approach [12]. On the other hand, small files present a slightly worst performance mainly due to the over-

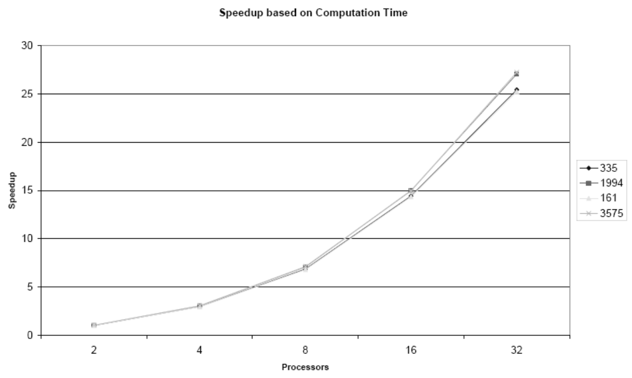

Figure 12: Speedup based on average computation times of all patterns for the four search files

head imposed by the pool manager and the intercommunication infrastructure. Furthermore, as the cluster scales the computation time becomes smaller while the communication overhead increases, leading to a further performance drop. It is expected that for larger cluster configurations utilizing 128 or even 256 workstations, the pool manager can become a significant performance bottleneck. In such configurations the role of the pool manager may have to be assigned to more than one node, following an appropriate load balancing approach [11].

The data secure pool manager adaptation was motivated by both security and ease of data management reasons. The approach suggests that all data should be stored in only one workstation, the secure root node. Consequently, performance can be severely limited by local I/O read speed and network transmission speed, which was the case in the test environment. This resulted in extremely low performance compared to the simple pool manager and the static workload allocation version, to a point where parallelization gains are completely lost due to communication. Still, it is expected that given a faster network interconnection, system performance should be much better. Furthermore, other data parallel problems with a more complex computation phase could also perform better. Then the benefits from having the dataset local in only one node in conjunction with parallelization should be apparent. Both of the above research questions are left open for future work.

References:

[1] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, Third Edition, 2003

[2] V. Donaldson , F. Berman and R. Paturi, Program Speedup in a Heterogeneous Computing Network, *Journal of Parallel and Distributed Computing*, vol. 21, no. 3, 1994, pp. 316–322

[3] Aho Alfred, Corasick Margaret. (1975). Efficient String Matching: An Aid to Bibliographic Search. Association of Computing Machinery Inc.

[4] C. Christian and L. Thierry, *Handbook of Exact String Matching Algorithms*, Kings College Publications, 2004

[5] A. Tanenbaum, (2002), *Computer Networks*, Prentice Hall Publishing, Fourth Edition, 2002

[6] G. Pfister (1998), *In Search of Clusters*, Second Edition, Prentice Hall, 1998

[7] GenBank: National Institutes of Health genetic sequence database
URL: http://www.ncbi.nih.gov/Genbank

[8] D. Ashton, W. Gropp and E. Lusk, *Installation and User's Guide to MPICH, a Portable Implementation of MPI Version 1.2.5.* URL: http://www-unix.mcs.anl.gov/mpi/mpich/

[9] The Message Passing Interface Forum, MPI: A Message Passing Interface Standard
URL: http://www.mpi-forum.org/

[10] P. Michailidis and K. Margaritis, Parallel Text Searching Application on a Heterogeneous Cluster of Workstations, *Proceedings of the 2001 International Conference on Parallel Processing Workshops IEEE (ICPPW'01)*, 2001^a, pp. 1530–2016

[11] R. Buyya, *High Performance Cluster Computing: Architectures and Systems*, Prentice Hall PTR, 1999

[12] G. Ivanov, A Study of Parallel Programming Techniques on a Cluster of Workstations, *IST Studies, University of Hertfordshire*