The 2006 International Arab Conference on Information Technology (ACIT'2006)

Parallel Execution of an Irregular Algorithm Depth First Search (DFS) On Heterogeneous Clusters of Workstation

Mohammed. A. M. Ibrahim
Faculty of Engineering &Information Technology
Taiz University
Republic of Yemen, Taiz, P.O. Box: 6038
Sabri1966@yahoo.com

Abstract

This paper presents a status report on our efforts and experiences with heterogeneous clusters of workstation based on dynamic load balancing for parallel tree computation depth-first-search (DFS) project at the High Performance Computing laboratory in Shanghai Jiaotong University. We describe an implementation of one parallel search algorithm DFS, running under the MPI message passing interface and Solaris operating system on heterogeneous workstation clusters. The main goal of this paper is to demonstrate the speed gained by the heterogeneous workstation clusters platform computing to solve large search problem, distribute the tree search space among the processors and show through a parallel simulation application, the maturity of the more recent technologies. We have run the parallelizm and distribution of the search space of the DFS algorithm among the processors successfully found that the critical issue in parallel DFS algorithm is the distribution of the search space among the processors. First experimental results of parallel DFS are given for tests that will serve as a starting point for further development of our project. We are presenting our preliminary progress here, and we expect in the near future to demonstrate a real dynamic load balancing for DFS algorithm running on heterogeneous clusters of workstation computing platform resulting in a good load balance among all the processors.
Keywords: heterogeneous clusters of workstation, parallel tree computation DFS, dynamic load balancing strategy, parallel performance.

1. Introduction

Depth-first search (DFS) is among the most popular techniques for finding a solution in a state space tree (or graph) containing one or more solutions. Many applications in Operations Research, Artificial Intelligence and other areas in Computing Science use DFS as a basic solution method [1,2,3,4]. Because these problems are computationally intensive, the design of efficient parallel algorithms is of prime importance. The main goal of this paper is to describe the implementation of parallel search algorithm (DFS) and demonstrate the speedup gained by the heterogeneous clusters of workstation platform computing. A major advantage of the depth-first search strategy is that it requires very little memory. Since many of the problems solved by DFS are highly computation intensive, there has been a great interest in developing a parallel version of depth-first search. Most of these papers in [5,6,7,8] use DFS implementation on shared memory, which is differ from our purpose and environment in this paper. In the following, we introduce the parallel DFS and present the performance result obtained on heterogeneous clusters of workstation

2. Parallel depth-first search

An investigated on the running parallel program of DFS shows that, the state space spawned by a depth-first search can be partitioned into smaller parts (sub trees) for simultaneous exploration by different processors. Once the sub trees have been distributed among the processors, little communication is necessary for broadcasting improved bound values and solutions for termination detection DFS [10,11,12,13]. In practice, DFS trees tend to be highly irregular, that is, they exhibit varying branching degrees and the DFS is generated dynamically during runtime in a way that is unpredictable in general. Static tree partitioning is insufficient to keep all processors busy, dynamic load balancing method are required to map the workload onto the network, minimizing processor idle times and amount of communication. The main aspects in this paper are the parallelism and distribution of the search space among the processors since the late, is the critical issue in parallel depth-first search algorithm.

3. Implementation approach to parallel (DFS)

DFS has been implemented on heterogeneous clusters of workstation by partitioning the search space into sub trees that are searched in parallel. Each processor searches a disjoint sub tree in depth-first fashion, which can be done asynchronously without any communication. When a processor has finished its work, it tries to get an unsearched sub tree from another processor. When a goal node is found, all of them quit. Efficient workload balancing is important to keep all processors busy. In the near future we are going to implement dynamic load balancing strategy to balance the workload of parallel DFS among the processors. Our idea of dynamic load balancing came from running our parallel program for DFS application on heterogeneous clusters of workstation. We noted that the computation of the DFS algorithm evolves in idealized manner that is, a consistent increase, followed by a consistent decrease into the total workload by DFS. Hence we can divide the computation into distinct phases with each phase having different load balancing requirement and objective view of the workload in tree computation. Execution of tree application typically produces a workload, which evolves from one phase to another, in order to identify the phases in the application. We supposed that DFS application is executed on *P* processors at time *t. S*ince the computation starts with the root task it is possible to identify three main workload phases in the DFS computation. Figure 1 shows the 3 phases of computation workload.

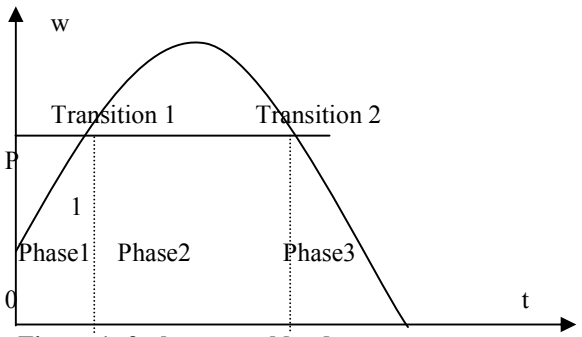

Figure 1: 3 phases workload

t: is execution time. P: processors number. w: workload or tasks

(1) Phase 1: during this phase the machine cannot be fully utilized, since w <*P*.
(2) Phase 2: When w>= *p* there is enough workload for all processors potential to be busy-unbalanced growth of the computation tree meaning: some processors become idle while there is still a large amount of work to be done.

Phase3: Eventually the workload becomes so low that it is not possible to use all processors; as in the phase1 w <*p*.

- **Transitions**

As defined in Figure 1, there are 3 phases workload (tasks). Hence we can identify two-transition points: The first transition marks the phase change from phase 1 to phase 2, while the second transition, between phase 2 and phase3. So both transitions 1 and 2 are detected when the number of tasks reaches the total number of processors; that is when w<= *P* for transition 1 and w<= *P* for transition 2.
The basic parallelization pseudo code of the DFS is given in Figure 2.

```
Var visited Boolean initial false

Status [v] init unsearched (*for each neighbor*/
Start the algorithm
    Visited: =true;
  For number of w in neighbor do
        Begin send [dfs] to w;
        Status [w]: = cal end
        Upon receipt of [dfs] from v:
    if not visited then
      Begin
          Visited: = true;
          Status [v]: = father end;
     if status[v] = unused then

        Begin send [dfs] to v ; status[v] := ret end
    else
     if there is a w with status[w] = unused then
          Begin
               send [dfs] to w ;
               status[w] := cal end
      else
       if there is a w with status[w] = father then
            begin
           Send [dfs] to w end
else stop
```

Figure 2 DFS parallelization pseudo code

4. Application

The 8-puzzle problem consists of 3*3, a typical application consisting of eight squared tiles, located in a squared tray of size 3*3 with one empty square. A tile can be moved into the blank position from a position adjacent to it, thus creating a blank in the tiles' original position. Depending on the configuration of the grid,

up to four moves are possible: up, down, left, and right. The initial and final configurations of the tiles are specified. The objective is to determine a shortest sequence of moves that transforms the initial configurations to the final configuration. We used an 8-puzzle to keep the search space reasonable and the 8-puzle can be naturally formatted as a graph search problem.

5. Hard-and software

As parallelization software, the MPI (Message Passing Interface) the MPI implementation MPICH [15] has been used. This library is a successor to PVM and runs on a large number of different platforms, including Linux PC, standard workstations or even Cray supercomputers. With this library, it is possible to run a program in parallel even on heterogeneous clusters of workstation. The simulation was carried out on heterogeneous clusters of workstation, which consists of the machines listed in the table 1. The performance measurement has been made using the five Solaris machines.

Machine type	***CPU***	***OS***
SUN Ultra Enterprise	SUN UltraSPARC-II(450MHz)	Solaris 2.6
SGI O2	R5000 (180MHz)	Solaris 2.6
2 SUN Ultra30	SUN UltraSPARC-II(248MHz)	Solaris 2.6
SUN Ultra Enterprise	SUN UltraSPARC-II(296MHz)	Solaris 2.6

Table 1 heterogeneous clusters of workstation Machines List

6. Experimental Results

Our program written in c language was running on heterogeneous clusters of workstation table 1 shows the machines characteristics, all they are interconnected by high-speed data links, Solaris operating system has been chosen and using MPI as an application interface layer. The computation starts with single node (root task). The tasks are dynamically generated and consumed through expansion; hence multiple tasks can be executed in parallel processing on heterogeneous clusters of workstation. Our measure of performance relates to the execution time and is measure of how much faster the program runs on the parallel machine than it doses on a serial machine Figure 3 shows the improvement of the execution time on a small number of processors can decrease Significantly as the.

Number of Processors	Execution Time
1	7.57809
2	2.89275
3	1.12081
4	0.65507
5	0.56545

Table 2 Execution time in second

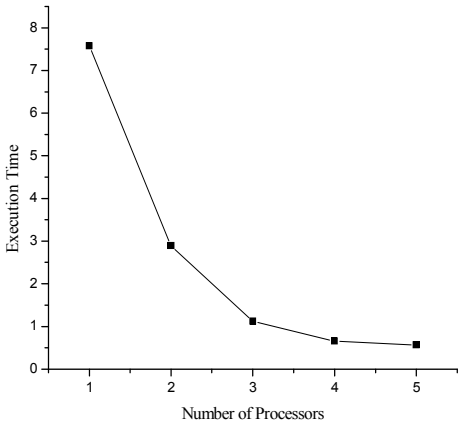

Figure 3 Execution time (sec)

number of reception message increases, the number of messages sent per processor has small impact on the simulation times. In order to make the parallel execution more apparent, we inserted some delay in each search step. We first initialized a job to each processor, and time was measure for all the processors. From Figure 3 we note that as the number of processor increases, the execution time decreases. Up to three processors, the sharp fall in the execution time indicates the advantage we achieve from parallel execution. One important point to be noted from the

curve Figure 3 is that adding further processors beyond three do not improve the execution speed significantly due to the communication overhead. To enhance the performance, we introduced dynamic load balancing in the pervious section, as a promising technique for improvement of performance in such platform computing. Future work will focus on dynamic load balancing on heterogeneous clusters of workstation, but we have no doubt that this kind of architecture is very promising because of its excellent performance/price ratio.

7. Conclusions and future work

We have presented a very efficient parallel algorithm DFS for solving 8-puzzle problem, implemented on a serious alternative (heterogeneous clusters of workstation) to expansive parallel computers using MPI (message passing interface) the method is efficient and allows a good distribution of work to the processors. The over all goal of this work to explore the parallelism for the DFS and demonstrate the speedup gained by heterogeneous clusters of workstation parallel platform computing to solve large search problem. The result in Figure 3 shows the achievement in these regards. Generalizing the use of parallel DFS makes parallel solutions usable for wider range applications that exhibit tree characteristics. Future work will deal with the build of dynamic load balancing direct to the tree computation (DFS) application. We argued that, since the dynamic load balancing integrated within DFS application, there is a potential to optimize for DFS application, the potential way in which dynamic load balancing for DFS may be improved by taking into the account particular characteristics of DFS; specifically when the execution of DFS generate a workload phases detected at the run time when the dynamic load balancing strategy with different objectives is applicable in every phase and termination detection algorism which will works as back ground to the dynamic load balancing strategy also applicable in every phase.

8. References

[1] J.E. Boillat. Load balancing and Poisson equation in a Graph. Concurrency: practice and experience, 2(4): 289-313, December 90.

[2] S. Arvindam, V. Kumar, V. Rao. Efficient parallel algorithms for searching problems: Applications in VLSI CAD.3rd symp.1990,166-169.

[3] A.Reinefeld and V. Schnecke. AIDA* - Asynchronous parallel IDA*. 10th Canadian conf. On art.intel. AI94, (may 1994), Banff, Canada.

[4] R. E. Korf. Depth_ first iterative-deepening: An optimal admissible tree search. Artifical intelligence, 27:97-109, 1985.

[5 Bernard Nadel. Tree search and arc consistency in constraint satisfaction algorithms. In L. N. Kanal and Vipin Kumar, editors, search in artificial intelligence, pages 287-342. Springer-Verlarg, New York, NY, 1988.

[6] V. Kumar, P. s. Gopalakrishnan, and L. N. Kanal. Parallel algorithm for machine intelligence and vision. Springer-Verlge, 1990.

[7] M.C. Wikstrom, J.L. Gustafson, and G.M. Prabhu. A threshold test for dynamic load balancing. In international conference on parallel processing, pages II268-II269, 1991

[8] A.L. Cheung and A.P. Reeves. High performance computing on a cluster of workstation. Proc. of the 1st.int.symposium on September 1992.

[9] M.ciermiak, W. Li, and M.J.Zaki.loop scheduling for heterogeneity. In 4th IEEE Int. symposium on high performance distributed computing, also as URCS-TR 540 CS dept. Univ. of Rochester, August 1995

[10] C.D. Polychronopoulos. Parallel programming and compilers. Kluwer Academic publishers, 1988.

[11] A.S. Grimshaw, J. B. weissman, E.A.West, and E.C. Loyot. Metasystem: an approach combining parallel processing and heterogeneous distributed computing systems. Journal of parallel and distributed computing, 21(3): 257-270,1992

[12] Per H. Andersen and John K. Antonio Implementation and utilization of a heterogeneous Multicomputer cluster for the study of load balancing strategies IEEE 1998.

[13] Anna Brunstrom, Rahul simha. Dynamic vs. static load balancing in pipeline computation technical report the department of computer science college of William and Mary VA 23185.

[14] G. Tel, Topics in Distributed Algorithms, and Cambridge International Series in Parallel Computation: 1, Cambridge University Press, 1991.

[15] MPI home http://www.mcs.anl.gov:80/mpi/.