Performance Evaluation of Three Dynamic Load Balancing Algorithms on SPMD Model

Najib A. Kofahi* and Quazi Abdur Rahman**

Received on Aug. 22, 2004
Accepted for publication on July 18, 2005

Abstract

In this research paper, we focus on the performance of different task migration and load balancing algorithms on SPMD model based on their controlling parameters. A network of workstations has been chosen and PVM libraries have been used for implementation. Matrix multiplication has been selected as the application. Three algorithms have been investigated namely, fixed granularity, variable granularity (guided self scheduling) and one global centralized task migration algorithm. Program execution time, communication overhead and processing time have been considered as the performance measure. From the experimental results, it is found that the granularity is the controlling factor in the fixed granularity algorithm, the ratio that decreases the task size in guided self scheduling plays a vital role and that if the parameters are chosen carefully, variable task granularity and the task migration algorithms perform better than the fixed granularity.

Keywords: Fixed granularity, Global centralized task migration, Load Balancing, PVM, SPMD Model, Variable granularity.

1. Introduction

Two important observations are becoming the trend of the new era: First, the development of new large-scale, complex and computation intensive applications and software packages including 3D and 4D graphics, and the second, the increasing demand

© 2005 by Yarmouk University, Irbid, Jordan.

* Department of Computer Sciences, Faculty of Information Technology and Computer Sciences, Yarmouk University, Irbid, Jordan

** Department of Information and Computer Science, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia.
to manipulate large volumes of data. These two observations require an increased interest in cluster computing [1] and in general in high performance computing (HPC) [2,3].

Efficient and economically affordable high performance could be achieved from several factors mainly, the increase in clock rate, in the giga range, and by incorporating innovations and features present in supercomputers and mainframes into present day microprocessors.

New developments in network technologies and operating systems made it also possible and easy to connect a collection of homogeneous or heterogeneous systems [4].

Load balancing aims at improving the performance of parallel computations by distributing the workloads of processors automatically during the execution of parallel programs [5-9]. There are very well known algorithms to do this process allocation. The algorithms are mainly divided into two categories: static and dynamic. Considering the application and process sizes, heterogeneity of the distributed environment and other factors, in most of the cases dynamic algorithms show much better performance [8,10-14].

There is much research going on in this direction [11-12,15-18]. We tried in this research work to reach some conclusion comparing global centralized load balancing algorithms with the load sharing algorithms. There are some controlling parameters of these algorithms. We mainly focus on those parameters so that setting these different parameters suitably can increase the performance of these algorithms. Our contribution is in studying the characteristics of these parameters and then implementing and comparing algorithms of different strategies [15,16].

The rest of the paper is organized as follows: Section 2 presents the load balancing schemes in general, which is then followed by Section 3, the experimental setup and tools used in the imperial study we are conducting. Next, Section 4 gives an overview of the SPMD (Single Program Multiple Data) model, and Section 5 introduces the actual algorithms used in our study. Section 6, Results and Analysis, explains the results obtained for each of the three algorithms under investigation. Finally, we present our conclusions and future possibilities in Section 7.

2. Load Balancing

Efficient load balancing can provide major performance benefits [19-22]. In general, there are two major categories of load-balancing algorithms: static (i.e., at compile time), or performed dynamically (i.e., during execution). A performance comparison between dynamic and static load balancing algorithms can be found in [12,23-24].

2.1. Static Load Balancing

In this approach, the tasks are allocated to the workers by the master depending on the performance of the workers at the beginning of execution. Using optimum task distribution (OTD), static load balancing algorithms may provide better performance. The advantages of this technique include simplicity of the implementation and low
communication overhead between the master and slave, as there is no need to monitor the workstations for performance statistics, which may boost the performance. But on the other hand, it cannot adjust to the runtime performance of the machine and non-homogeneous nature of the application, and hence, static load balancing algorithms are not well suited to environments where loads may vary significantly with the time of the day.

2.2. Dynamic Load Balancing

In this strategy [12,20], algorithms adapt to changes in the execution environment. As a result, dynamic load balancing algorithms can provide a significant improvement in performance over static algorithms. However, this comes at the additional cost of collecting and maintaining load information, so it is important to keep these overheads within reasonable limits.

There are many dynamic load balancing algorithms proposed. They can be categorized into two main types: load sharing and task migration. There are four basic steps that nearly all algorithms have in common:

a. Monitoring workstation performance (load monitoring)
b. Exchanging information between workstations (synchronization)
c. Calculating new distributions and making workers as needed to carry out the movement decision (rebalancing criteria)
d. Actual data movement (job migration)

2.3. Load Sharing Algorithms

Load sharing algorithms are mainly based on master slave concept. The master first creates as many workers as needed to carry out the intended tasks. Then it assigns tasks to the workers based on some criteria. The algorithms differ from each other based on the issue of choosing these criteria.

There are three main types of load sharing algorithms, namely: fixed granularity, variable granularity and adaptive granularity. In fixed granularity algorithms, task granularity is fixed statically. In variable granularity algorithms, task granularity decreases as the computation progresses. While both these types of algorithms take the system state into account implicitly, task granularity itself is not dependent on load variations in the system. In the adaptive task granularity algorithms the task granularity varies depending on the current system load [10].

2.4. Task Migration

In this approach, the master assigns tasks to the workers and tasks are migrated between workers from overloaded workers to the under-loaded workers. Depending upon the initiation of migration, they are categorized into two types, sender-initiated and receiver-initiated strategies. In sender-initiated policies, congested nodes attempt to move work to lightly loaded nodes. In receiver-initiated policies, lightly loaded nodes look for heavily loaded nodes from which work may be received [15].
Task migration algorithms may be divided depending on where the load balancing decision will be made and what information will be used to determine the load balancing decision [17]. If the load balancing decision is made based upon performances of all the nodes, then it is global. The decision also can be made on the basis of groups of nodes' performance. This type is referred to as local. A master makes the decision in the case of centralized load balancing and individual workers make the decision in decentralized policy.

In our experimentation, we shall focus on the global centralized sender-initiated load balancing algorithm proposed by Lee, Cai, and Heng [17].

3. Experimental Setup and Tools

To evaluate the performance of the selected algorithms, we used a network of workstations running SUN SOLARIS operating system. We used PVM (Parallel Virtual Machine) libraries to create the master processor and the workers and for message passing among different processes. PVM spawns tasks or processes to different processors. Message passing is easy and flexible in PVM. We used C language for programming.

_gettimeofday()_ library function of UNIX has been used to calculate the time duration of the program. All through the experiment, we used 500x500 matrices of type double. The entries of the matrices are initialized by their row numbers. We have 8 processors to create 10 nodes. In our experiment, the master also works as a worker. The master/slave model is shown in Figure 1.

![Master/slave model](image)

**Figure 1.** Master/slave model
4. The SPMD Model

We used the SPMD model as it is very much suitable to study the performance of load balancing algorithms. In this model, each worker runs the same program but on different set of data. These types of problems are easy to implement and finely tuned for experimental purposes. SPMD model mainly relies on arrays of processors, distributed data, some global data, message passing between processors and loose synchronization.

Figure 2 illustrates the SPMD model for message passing between two processors. Since each processor runs the same executable program, message passing must be masked by IF statements.

```
   IF (I_AM_P1) THEN
     SEND(P2, hello)
     RECV(P2, acknowledge)
   ELSE
     RECV(P1, hello)
     SEND(P1, acknowledge)
   END IF
```

**Processor 1**

```
   IF (I_AM_P1) THEN
     SEND(P2, hello)
     RECV(P2, acknowledge)
   ELSE
     RECV(P1, hello)
     SEND(P1, acknowledge)
   END IF
```

**Processor 2**

Figure 2. An example of message passing between two processors in a SPMD system.

The very popular problem for this model, namely that of matrix multiplication, is used in the evaluation study. To make use of parallelism in the study of this problem, the first matrix is broadcasted to all slaves. Then the number of rows of the first matrix is distributed to the slaves or among the workers based on the algorithm. For the task migration strategies, the rows of the first matrix are considered as the migrating tasks.

5. The Three Load Balancing Algorithms under Consideration

The three load balancing algorithms chosen for our empirical study are as follows.

5.1. Load Sharing Fixed Granularity Algorithm

In this approach, after the first matrix has been broadcasted to all slaves, each slave is assigned fixed size of tasks say C columns of the second matrix. As soon as a worker finishes the required multiplication operation, it reports the results back to the master. The master then collects the result, writes it into the appropriate place of the result matrix and then assigns a new task to that worker. That is, the master sends C more columns of the second matrix to the worker. This continues until all columns of the second matrix have been sent. It should be noted that the assigned task size is fixed all through the distribution phase.
5.2. Load Sharing Variable Granularity Algorithms

The description of three algorithms that use variable task granularity is found in [10,15]. These algorithms have been proposed for multiprocessor systems for loop processing. The difference between fixed and variable task granularity algorithms is that the task size decreases with time. For example, the first task may have a size of ten columns, whereas the last one may have only one column. This can improve performance in two ways:

1. These algorithms attempt to minimize the time the master has to wait to receive the last results as task sizes at the tail end are much smaller.

2. It is well known that sending several short messages results in a performance that is worse than when a small number of large messages is sent. In the case of matrix multiplication, depending on the parameters used, variable task size algorithms can potentially decrease the number of messages sent while increasing their average size.

Different ways are used by researchers to decrease the task size. One approach is the guided self scheduling algorithm [7]. In this algorithm, the task size is a function of the remaining number of columns. The task size is typically set to \( \frac{1}{C} \) of the remaining columns. It is clear that the larger the value of \( C \), the smaller will be the task size. Figure 3 shows the non-increasing task size for consecutive workers for initial task size of 10. The size of the whole task is 100 [15].

![Decreasing Task Size](image)

**Figure 3.** Decreasing task size in guided self scheduling algorithm
5.3. Global Centralized Load Balancing Algorithm

Lee, Cai, and Heng in [17] have proposed a dynamic load balancing algorithm which is in nature global and centralized. In this algorithm, the load balancing decision is made in the master and the load is distributed according to the performance of all the nodes. Lee, Cai, and Heng in [17] have also proposed an SPMD model which uses lock step. This model has normally three different steps, e.g., calculation phase, data distribution phase and synchronization phase. In the first phase, all the workers will do computation and there will be no communication. In the data distribution phase, each task will distribute the relevant data to other tasks that need it for next lock step. This phase confirms that all the nodes have reached a synchronization point. This is to avoid the problem of nodes working with wrong data.

The two important issues now are: what will be the rebalancing criteria after every lock step; and the distribution of data among the nodes. Lee, Cai, and Heng [17] have also introduced the following parameters to explain the algorithm.

\( T_{compute_i} \) = The interval between the time the first task on workstation \( i \) starts executing and the time at which the last task on the same workstation completes the computation and waits for the synchronization.

\( T_{task_i} \) = The average computation time for task \( i \) on the given workstation.

\[
T_{compute_i} = \frac{T_{compute_i}}{n_i}
\]

Where \( n_i \) is number of tasks on workstation \( i \).

\( T_{high} \) = The maximum \( T_{compute} \) over all workstations.

\( T_{low} \) = The minimum of \( T_{compute} \) over all workstations.

For load balancing, tasks from workstations with longer \( T_{compute} \) should be moved to the workstation with shorter \( T_{compute} \).
Rebalancing Criteria: $T_{task_k} \ast n_k > \min \left\{ T_{task_i} \ast (n_i + 1) \right\} (1 \leq i \leq P, i \neq k)$

Where,

$k$ is the index of the task with highest $T_{compute}$

$P$ is the number of workstations (or processors), and

$n$ is the current no of tasks assigned to a workstation.

The master checks the rebalancing criteria after each lock step. If it is true, then some tasks will be moved from the slow workstation to a fast one.

6. Results and Analysis

Following the experiment setup and using the tools described in section 3, we developed and executed the programs and collected results for different parameters. Execution time and the communication overhead are the measures of performance in our experiments. We calculated the actual processing time by subtracting the communication overhead from the total execution time. That is:

Actual execution time = Total execution time – Communication overhead.

6.1. Load Situation

Since the machines used to carryout our experiments were not loaded enough, we had to create some background jobs to load the workstations. To do this, we wrote a little program, which consumes CPU cycles to keep the processors busy. This makes the processors imbalanced in load and introduces some variance in the load situation on different machines to create a more realistic working environment. As both master and worker are running on one machine, we did not create more loads on this particular machine. We created different background load on other machines only.

6.2. Fixed Granularity Algorithm with Different Task Size

Task size is the most important factor in fixed granularity algorithm. The performance of the application generally depends on the task size. In our computation model, as stated in section 4, the number of rows of the first matrix is the task size. Figure 4 shows the execution time, communication overhead and processing time for different task sizes.
Figure 4: Performance variation of fixed granularity algorithm with varying task size

It is obvious from Figure 4 that the execution time, the communication overhead and the processing time decreases as the task size increases. This really makes sense in a distributed computing environment, as larger task size needs less communication. If the task size is too big and one of the machines becomes heavily loaded, this may create substantial decrease in the performance of the system.

6.3. Variable Granularity Algorithm with Different Values of C

Choosing the value of C to be equal to the number of slaves can lead to diminished load sharing, as large tasks will take much more time on slow machines than on faster ones. Therefore, the experiment should be conducted for different values of C as shown in Figure 5. In this figure, we implemented Guided Self Scheduling algorithm with different task sizes. The decrease in task sizes is indicated as a function of the ratio 1/C as explained earlier. Figure 5 shows the results of our experiment.

Figure 5. Guided Self Scheduling algorithm with different constants
Again, this result supports the fact that increasing the task size improves the performance. From Figure 5, it is clear that when the value of the constant C increases, the task size decreases, and hence, it takes more execution time, communication overhead and processing time.

6.4 Global Centralized Load Balancing Algorithm with Different Lock Steps

The algorithm discussed in section 5.3, has two controlling parameters, namely: the number of lock steps and the number of tasks moving from one node to another after every lock step. We fixed the second parameter to 4, i.e. after every lock step, 4 rows will be removed from the slower node and will be assigned to a faster node. Then we collected the result with different lock step. Figure 6 shows the situation.

![Global Centralized Algorithm](image)

**Figure 6: Global Centralized Load Balancing Algorithm with Different Lock steps**

Increasing number of lock steps actually consumes more execution time, communication overhead and processing time. If the machines are too much heterogeneous, then less number of lock steps may be a problem.

6.5. Comparison among Three Algorithms

To reach a conclusion about the performances of different algorithms, we need really to exhaustively work on all possible controlling parameters in a perfect heterogeneous environment with huge load imbalance. But as the implementation phase of the algorithms needs a substantial amount of effort and we since we do not have a perfect heterogeneous environment with much load imbalance, we had to compare the algorithms based on some assumptions. The main question that arises when we compare the algorithms is that what should be the values of different parameters discussed in previous sections? In the literature, it is found that this type of evaluation always chooses suitable parameters for different algorithms.

In this study, we used the following parameters:
i) For fixed granularity, the task size is 5. This is chosen mainly because as there is no load rebalancing, so it may be inefficient to choose a large number of tasks. Even in some cases, only task of size one is chosen.

ii) For variable granularity, we chose 10 as the ratio as indicated in Figure 3. If the ratio is too big, then after two or three times, the task size will be all the same always. On the other hand, if the ratio is too small, the risk of assigning a very big task to one node is always there.

iii) For the algorithm proposed by Lee in 1995 [17], we choose the lock step as 4. Many more lock steps may create problems by imposing the overhead every time.

Considering the above assumptions, the result obtained for the three algorithms is as shown in Figure 7.

![Comparison Among Algorithms](image)

**Figure 7:** Comparison among algorithms

The graph shows that variable granularity performs much better than fixed granularity and the algorithm proposed by [17] performs a little bit better than variable granularity. We need to remember that setting the parameters differently may change the scenario.

7. **Conclusion and Future Work**

From our experiments and analysis, it has been shown that the performance of fixed granularity algorithms increases with increasing task size. It is also observed that the performance of Guided-Self Scheduling algorithm increases with decreasing task size ratio. The performance of the algorithm proposed by [17] decreases with increasing number of lock steps.

This gives us an indication that if the task is distributed in small number, then the number of communications increases. This will increase the communication overhead and hence the total processing time will also increase. This happened in this experiment because the machines were not exactly heterogeneous and the load is nearly balanced. The following conclusions can be drawn from our experiment.
i) The performance of the three algorithms depends much on the parameters discussed earlier. However, if the parameters are carefully chosen, guided self scheduling algorithm and the algorithm proposed by Lee, Cai, and Heng [17] may provide better performance than fixed granularity algorithm.

ii) For the three algorithms investigated, communication overhead is the real problem. So, developing any dynamic load balancing algorithm should take care of keeping the communication overhead to minimum.

The conclusions above may be richer if the experiment is done on more heterogeneous and load imbalanced environment. CPU utilization for each worker may be considered as another good measure of performance. In addition, more experimentation on the parameters in future may explore more aspects of these algorithms and this type of evaluation is always suitable for choosing among different alternatives.

نتقييم أداء ثلاثة خوارزميات ديناميكية لتوزيع الأحمال بقصد موازنتها في أنظمة الحواسيب الموزعة باستخدام نموذج البرنامج الواحد وعدد البيانات (SPMD) وتعدد البيانات.

نجيب عبدالكريم الكوفجي و كازي عبدالرحمن

ملخص

يناقش هذا البحث أداء ثلاثة خوارزميات ديناميكية لتوزيع الأحمال بقصد موازنتها في أنظمة الحواسيب الموزعة باستخدام نموذج البرنامج الواحد وعدد البيانات (SPMD). لقد تم اختيار شبكة من محطات العمل الحاسوبية بالإضافة إلى الكتبة البرمجية المضافة (أي مكتبة الأجهزة الحاسوبية التخليلية المتوازية) لاختبار أداء هذه الخوارزميات. كذلك تم اختيار تطبيق عمليات ضرب مصفوفتين fixed granularity, variable granularity (guided self scheduling) و one global centralized task migration algorithm.

لقد تم اختيار أداء الخوارزميات الثلاثة من ثلاثة جوانب في: مدة تنفيذ البرنامج، والوقت اللازم لانتقال المعلومات عبر محطات المعالجة، والوقت المبذول في المعالجة.

المصطلحات

fixed granularity هو عامل التحكم في خوارزمية granularity.

guiding self granularity كذلك ذلك التجربة التي تم إجرائها أن نسبة تنافص حجم العمل في خوارزمية variable task granularity تلعب دوراً هاماً في أداء الخوارزمية. كذلك فإن أداء خوارزمية scheduling variable task في حال اختيار بعض العناصر في خوارزمية fixed granularity بعناية.
References


