Available at: www.sabauni.net
Journal home page: www.sjitn.sabauni.net

Designing Algorithm For Resources Management Between Processes In Cluster Networks

Dr. Al-Khulaidi Abdualmajed Ahmed*

Saba University, Sana'a Republic of Yemen

Article info

Article history :
Accepted : 2013

Keywords:
Clustering
Task scheduler
Resource manager

ABSTRACT

The paper is concerned with the problem of optimizing the distribution of tasks executed in the cluster. To solve this problem, algorithm has been developed to perform management tasks. Verification of the proposed algorithms has been implemented for solving the transportation problem on a cluster network. The data obtained allow us to judge the effectiveness of implemented algorithms.

* Corresponding author: Dr. Abdualmajed A. Al-Khulaidi.
E-mail address: a.alkhulidi@sabauni.net

Introduction

In recent years, both abroad and our country have been working actively in association with the use of parallel processing technology. With the use of parallel processing technology, it may significantly increase the performance speed. Because supercomputers are expensive, only large companies and institutions can have such computing power [1,2,3]. The solution to this problem is the clustering of complete processing units together, therefore it is possible to create a system that make computing power comparable to a supercomputer and cost much less.

Algorithm Of Resource Management Between Processes

The job is an abstract entity consisting of a set of commands and options. The task presented to the user in the form of a script contains the resource requirements, job attributes and a set of commands to be executed. Once a script task is created, it can be used as many times as necessary and possible modification of it. The Job must be first put in the scheduler queue, then out of the queue. It will be passed to a node for execution. The job can be conventional (regular) and interactive. The usual task is queued and then we wait for its execution, the result will be recorded in a user-specified location. Interactive definition differs in the input and output streams that are redirected accordingly on the screen and keyboard respectively, Tasks of teams are entered from the keyboard itself. Each job may require running a lot of different resources such as processors, memory, time (and the usual CPU). You may also need storage space. With the help of a resource manager, we can set a limit for each resource. If the limit is not set for any resource, it is assumed to be infinity algorithm:

1. Algorithm is determined by the type of resources used by this task and the number and priority of the task.
2. Assignments are distributed by stages and arranged in order of priority.
3. Management Service are then sent to an executable script. Binding to an existing reservation is made by ASL, the reservation of resources at this point should already be done. Resources can be of the following species: the number of processors, memory, required software, virtual memory, time and so on…

Resource management in computing systems deals with the allocation of available system resources to the various tasks ready to be executed. This is a process that significantly affects the overall performance of the system. Typically, resource allocation algorithms take as input a list of tasks or processes that are ready to be executed at some particular time as provided by a system scheduler. The scheduler considers a task flow graph in order to resolve task dependencies. Traditionally, resource allocation in multiprocessor systems concentrates on the allocation of software tasks to each of the processor nodes (usually individual processors with local and memory), such that the overall performance of the system is maximized. This is a well-known problem with a large amount of research contributions towards efficient utilization of the massive hardware parallelism available in such systems using various exact and heuristic approaches. In a case of many core systems, important on-chip constraints such as limited buffer capacity for on-chip communication, on-chip network congestion, power density, and limited must be used.

In an effort to integrate the emerging challenges, I/O bandwidth necessitate the evolution of existing algorithms or even the development of new algorithms, In such dense systems, efficient workload distribution could benefit from real-time system information knowledge such as the status/utilization of each core and, additionally, the status of the interconnection network.

Interconnect-associated delays are an important factor in efficient decision-making and the problem is further more complicated while controlling Memory is related to traffic, such as cache misses and synchronization data is taken into consideration. Whilst dealing with a similar problem, research looks at general-purpose systems (CMPs) from a slightly different viewpoint , when compared to application-specific systems (MPSoCs). General purpose systems face runtime uncertainties where cache misses, data hazards, and unpredictable interconnect modeling can potentially alter the expected execution time of one task significantly. On the other hand, application-specific many core systems (typically heterogeneous MPSoCs) usually deal with predictable schedules and execution times. Consequently, MPSoC-related research mostly focuses on finding optimal, static, design time allocation, where the mapping of tasks to the cores can either take place as part of the compilation, or mapped prior to execution on the processor cores. However, as the number of cores increases, MPSoCs are expected to contain groups of cores, where all cores in a group are of the same type. Any core inside a group can potentially execute some task, shifting the task allocation process towards finding the best core that can execute one type of task among the cores in a group. As a result, allocation in future massively parallel CMPs and MPSoCs is expected to face similar issues and should not necessarily be treated independently.

Bidding Algorithms

This work uses the concept of bidding to decide how to allocate processes to the various cores of the system. Bidding algorithms have been widely used to solve several optimization problems as part of auction-based algorithms. Typically, for a specific number of items, there are n possible “bidders”, with each “bidder” placing a bid in an attempt to “buy” a number of items. Usually, the highest bidder claims one or more items, and bidding continues until there are no more items or no more bidders. Such algorithms can also be used in more complex scenarios, under various constraints. Bidding-based algorithms traditionally offer load balancing across distributed systems and networks, optimizing performance and utilization. In the proposed algorithms, cores actively decide based on their workload and utilization of their related communication resources (on-chip network) whether, and to what extent, to bid for (request) additional process (task) assignment. Each core (or group of cores), computes its bid independently and sends it, through the on-chip network, to the system level on-chip allocation engine. Transferring this decision to the cores is done with minimal overhead; it also improves flexibility and scalability of the system. Subsequently, the allocation engine decides where to assign the list of pending processes/tasks dispatched by the system scheduler, based on the placed bids.

In this work, we present two different simple bidding-based algorithms for performing on-chip resource allocation in a many core system, where processes are the “auctioned” items, and a processor core (or a group of cores) places its “bid” based on its status (or the status of each core in the group). since we are targeting hardware implementation of the algorithm where speed and simplicity are critical factors, We are aiming for a simple and fast bidding based solution, rather than an optimal one, Still, the obtained experimental results demonstrate significant performance improvement and highly balanced utilization among the various cores. The presented algorithms improve the system’s performance when compared to a standard (static) allocation mechanism such as round robin implemented in hardware. We use round robin as a hardware reference algorithm in evaluating our optimization algorithms, due to its simplicity to be implemented

(very low hardware overhead) as well as the lack of other existing comparable solely hardware-based solutions for the specific problem under consideration.

The first algorithm, Necessary Resorting (NRS), is simpler in terms of operations performed and, therefore, faster. At some time instance T, it starts by sorting the list of clusters C and the list of processes P . Then, it binds the highest bidder (cluster at the top of C) with the largest process (the one at the top of P), the second highest bidder with the second largest process, and so on. This scenario tends to distribute the various processes among the available clusters in such a way that clusters with smaller, already allocated, workload and/or smaller network traffic are allocated to larger processes. Hence, workload balancing is inherently achieved without being explicitly targeted.

The second algorithm, Dynamic Resorting (DRS), follows a similar rationale as the NRS algorithm with the exception that a cluster's bid is re-calculated every time a process is bound to the cluster and the list of available clusters in C is resorted in order to reflect the allocation. Hence the allocation is more dynamic in DRS than in NRS. In contrast to NRS that binds at least one process to each available cluster before considering binding additional processes to a cluster, the DRS algorithm can bind several processes to a cluster, and possibly none to others, based on the dynamically recomputed bids. Bidding, in this context, offers several inherent benefits. Bid computation is distributed inside the cores/clusters, eliminating unnecessary traffic. Also, if the clusters cannot respond due to network congestion or them being busy, their bid value is assumed to be zero and, hence, these clusters are excluded from the allocation during the busy intervals. The bidding process is scalable, since an increase in the bidders can easily be integrated by increasing the lists of bids and tasks as well as using more than one allocation units (each managing groups of clusters/cores). As the network size grows, network delay, a more important factor in large networks, is a linear component of the bid. Similarly, core simplicity and core clustering allow for hierarchical multilevel allocation engines, which can take into consideration more detailed intra-cluster conditions.

Application Of The Proposed Algorithm In A Cluster Package Mpi / Mpich And Checking It More Efficiently For Distribution To The Transportation Problem

The practical significance of the work lies in the fact that the algorithm of job control was used to expand the library mpich (for free licenses)
that is in the creation of the programming model cluster, which includes the implementation of the proposed algorithm. Using the library has been developed, parallel program is found to support program in the transportation problem based on the method of Vogel (penalty) for the cluster network.
Execution was carried out on 4 machines with dual-core Intel Core 2 Duo E6700 and 2 GB of RAM are given in Table 1. All times are in seconds.
Comparative system performance while running the example shown in Figure 1.
The abscissa in parentheses indicates the number of processes. The graph shows that there is a slight performance boost while running in an environment MPI / MPICH_NEW. However, when the number of processes from 20 to 40 increase in performance ,it becomes more significant (1.13 times faster when calculating the matrix of size 1000x1000).

Conclusions

Thus, the algorithm was tested in a distributed job to control problem on a cluster system MPI / MPICH and MPI / MPICH _NEW. Tests showed that the algorithm developed an efficient solution for the selected class of distribution problems.

Table 1.Results of computational experiments on the algorithm proposed management jobs in the cluster model of MPI / MPICH_NEW

Dimension of task	Number of processes	Run-time parallel algorithm for finding the support program (seconds)		Acceleration
		MPI/MPICH	MPI/MPICH_NEW	
50x50	20	0,005	0,005	–
70x70	20	0,009	0,009	–
100x100	20	0,020	0,019	1,05
150x150	20	0,031	0,029	1,06
500x500	20	1,020	0,935	1,07
500x500	40	1,005	0,920	1,09
1000x1000	40	2,501	2,209	1,13

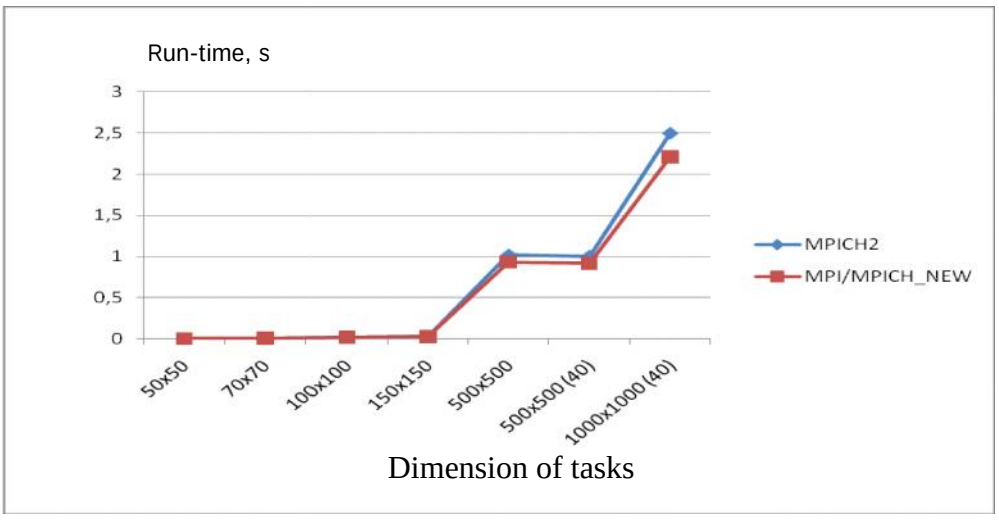

Fig. 1. Graph of execution time in MPI environment before and after application of the algorithm developed

References

[1]. AL-Khulaidi A. A. , Analysis of existing packages in the cluster networks, N.N. Sadovou, Journal Vestnik Don State Technical University, Russia 2010.T.10.

[2]. Foster I., Kesselman C., Tuecke S. “The Anatomy of the Grid: Enabling Scalable Virtual Organizations”. International Journal of High Performance Computing Applications, London.

[3]. Foster I., Kesselman C., J. Nick, Tuecke S. “The Physiology of the cluster: An Open Grid Services Architecture for Distributed Systems Integration”.
http://www.globus.org/research/papers/ogsa.pdf