An Integrated Processor Management Scheme for the Mesh-Connected Multicomputer Systems *

Chung-yen Chang and Prasant Mohapatra
Department of Electrical and Computer Engineering
Iowa State University, Ames, Iowa 50010
E-mail: prasant@iastate.edu

Abstract

The performance of a multicomputer system depends on the processor management strategy. Processor management deals with processor allocation and job scheduling. Most of the processor allocation and job scheduling schemes proposed in the literature incur high implementation complexity and are therefore impractical to be integrated. In this paper, we propose an integrated processor management scheme that includes a bypass-queue *scheduling policy and a* fixed-orientation *allocation algorithm. Both policies have very low complexities and are hence suitable to be integrated. Both policies improve the system performance considerably when applied in isolation. The integrated scheme provides even better performance.*

1 Introduction

Several processor allocation algorithms have been proposed for the mesh-connected systems in the literature [1]–[3]. These algorithms allocate a job to a set of contiguous processing nodes to minimize the distance of communication paths and to avoid the interprocess interference. Contiguous allocation leads to several fragmented groups of processors that cannot be used for the new tasks. Fragmentation is the main factor that limits the performance of multicomputer systems. Non-contiguous allocation algorithms [4] have been proposed to eliminate the fragmentation problem. Because of the unpredictable communication latency caused by non-contiguous allocations, they may not be suitable for a variety of application environments.

Job scheduling is concerned with the sequencing of jobs for allocation. The high complexities of the several allocation algorithms restrict the use of any complicated scheduling policy. The ordinary first-come-first-serve (FCFS) discipline severely restricts the performance of a multicomputer system. In the FCFS scheduling, a waiting job will block all the following jobs from being serviced even if there are idle processors in the system. Scheduling schemes proposed in [5, 6] avoid the idling of processors by rearranging the order of jobs to be executed. These scheduling schemes have shown promising performance improvement. However, they introduce significant overhead to the already complex allocation process.

Most of the allocation and scheduling schemes that demonstrate higher performance incur high time and implementation complexity. The integration of such allocation and scheduling schemes is thus impractical. In this paper, we propose an integrated processor management policy which considers both allocation and scheduling issues. An efficient job scheduling policy based on a *bypass-queue* (BQ) is proposed. A *fixed-orientation* (FO) allocation scheme is also proposed to be used in combination with the bypass-queue policy. Both schemes have low complexity and are therefore suitable for integration.

Extensive simulations are conducted to evaluate the proposed processor management schemes. The BQ scheduling and FO allocation are both shown to provide good performance for multicomputer systems when applied alone. It is further shown that the integrated processor management scheme performs better than using a single approach.

The rest of this paper is organized as follows. The detailed discussions about the proposed processor management schemes are in Section 2. The simulation results and the comparison with other policies are presented in Section 3. The last section concludes this study.

2 The Proposed Schemes

To combine the advantages of both processor allocation and job scheduling, an integrated processor management policy is proposed. The integrated policy consists of a job scheduling policy based on the *bypass-queue* (BQ) technique, and a *fixed-orientation* (FO) allocation algorithm.

2.1 Bypass-Queue (BQ) Scheduling

It is observed that the fragmentation problem is worsen by the blockade situation incurred while using the FCFS discipline. Many processors can be left idle even when there are jobs waiting for executions. By executing the jobs in a carefully arranged order, the blocking effect of the FCFS queue can be diminished as reported in [5, 6]. However, these schemes require

*This research was supported in part by the National Science Foundation through the grants CCR-9634547, MIP-9628801, and CDA-9617375.

multiple queues and have high implementation complexity.

A bypass queue is a variation of the FCFS queue without the blocking problem. Jobs in a bypass-queue are checked for allocation in the order of their arrival as is done in the FCFS queue. A job is allowed to bypass the unallocated jobs if it can be allocated. The process continues until all the jobs in the queue have been checked or an executing job departs from the system. In case of a departure, the entries in the queue are checked for allocation starting from the head of the queue. To ensure that every job can obtain the service after a reasonable waiting time, a threshold time is set for the system. The threshold time is the maximum time a job allows other jobs to bypass it before getting allocated. If any of the jobs waits longer than the threshold time, the bypassing is disabled and the jobs are served in a strict FCFS manner. Because of the threshold time, a job only has to wait for the release of the occupied nodes by other jobs if it has waited longer than the threshold time and is at the head of the queue. The jobs bypassing the blocked ones help better utilize the system.

2.2 Fixed-Orientation (FO) Allocation

The feasibility of non-contiguous allocation algorithms depends on the trade-offs between communication latency and queuing delay. In this paper, we consider only the contiguous allocation scheme. Among the contiguous algorithms, the schemes proposed in [2, 3] perform better because they allocate jobs in an alternative orientation when the requested submeshes cannot be found in one orientation. However, checking alternative orientations increases the complexity of the algorithm.

We propose a fixed-orientation algorithm for the mesh systems. Instead of checking alternative orientations to allocate a job, we allocate all rectangular submesh requests in the same orientation. The orientation for allocation is chosen according to the orientation of the system. If a job requests a submesh with different orientation than the chosen orientation, it is rotated before allocation.

The FO allocation has two advantages over the other algorithms. First, the allocation time is smaller compared to algorithms which check for alternative orientations. Because all jobs are allocated in a fixed orientation, our algorithm only needs to check for the available submesh in one orientation. Second, the fixed-orientation allocation has less fragmentation. Consider the example shown in Figure 1 with jobs arriving in the labeled order. If the allocator only checks the submesh in the requested orientation, job 4 ends up being blocked as shown. By forcing all the jobs to be allocated in the same orientation, all 4 jobs can be accommodated. There is still fragmentation due to the dynamic departure of the jobs. However, the simulation results in Section 3 indicate that the few fragmentation associated with fixed-orientation algorithm has insignificant impact on its performance.

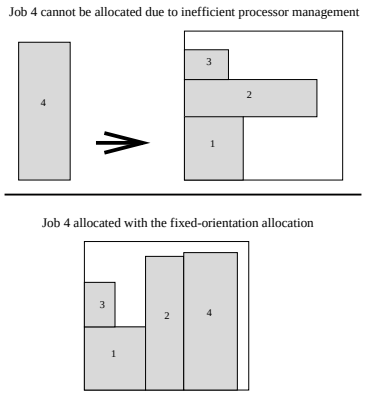


Figure 1: Reducing Virtual Fragmentation with the Fixed-Orientation Allocation.

2.3 Implementation of the Integrated Scheme

To take advantage of both processor allocation and job scheduling, the proposed BQ scheduling and FO allocation are used in combination as an integrated processor management scheme. A flag *_rotated* is associated with every job in the system. It is set upon the arrival of a job by comparing the orientation of the request and the orientation used for allocation. If this flag is set, address translation of the processors as discussed in [2] is required upon the allocation of this job. A linked list is used to implement the bypass-queue. All the waiting jobs are appended to the end of the list for allocation. The information contained in the linked list include the size, submission time, and the flag *_rotated* of the waiting jobs. Variable *th_time* is the threshold time set for the bypass-queue scheduling while *_submission* represents the submission time of the job at the head of the queue.

The integrated scheme consists of two main processes, *job arrival* and *job departure*, and is listed in Figure 2. The complexity for the FO allocation is equal to the first-fit and best-fit algorithm. It is more efficient than the algorithms [2, 3] that check alternative orientations for allocation. The manipulation of the bypass-queue does not introduce any significant overhead to the FCFS system. It does increase the number of allocation attempts because of the bypassing. Therefore, it is even more important to use an allocation algorithm with a low complexity such as the FO algorithm. The value of threshold time affects the performance of the integrated scheme. A system administrator can determine the threshold time based on individual system's need.

3 Performance Evaluation

Extensive simulations are conducted to evaluate the proposed schemes using a (32×32) mesh. Job arrivals are assumed as a Poisson process. Service time for a job is assumed to be exponentially distributed. The

Job Arrival:

Step 1 Check the orientation of the incoming job. Change the orientation and set _rotated *if necessary.*

Step 2 If queue is not empty, append the incoming job to the tail of the queue. Goto step 4.

Step 3 If queue is empty, check for allocation of the incoming job. If job is successfully allocated, assign processors to it and perform the required address translation when _rotated *is set. Otherwise, put the job at the head of the queue.*

Step 4 Wait for next arrival or departure.

Job Departure

Step 1 If queue is empty, goto step 4, else set _submission *to the submission time of the first job in the queue. Choose the first job in queue as the candidate for allocation.*

Step 2 If candidate is allocable, assign nodes to the candidate with proper address translation (if required) and remove it from the queue. If the last job in the queue has been checked, goto step 4.

*Step 3 If (*current_time − _submission $<$ th_time*), set the next job in queue as candidate. Set* _submission *to the submission time of the first job in the queue and goto step 2.*

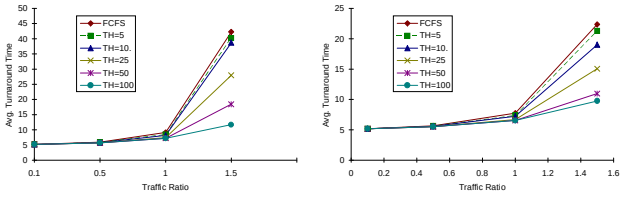
Step 4 Wait for next arrival or departure.

Figure 2: The Proposed Scheme.

arrival rate is calculated from the *traffic ratio* which is defined as $\frac{arrival\ rate}{service\ rate}$. A high traffic ratio represents a system under high load. The mean service time of a job is assumed to be 5 time units. We assume two different distributions for the submesh requests. The uniform distribution assumes the side-lengths of a job range from 1 to 32 with equal probabilities. We also assumed a truncated normal distribution for the side-lengths of a job in which the mean is set to 16.5 and the variance is 6.6. Any value outside the interval of [1,32] is truncated. The performance measures are derived by averaging the results obtained through the completion of 10,000 jobs.

3.1 Bypass-Queue Scheduling

Figure 3 shows the effect of using the bypass-queue with different threshold time using the first-fit and adaptive-scan allocation algorithms. The average turnaround time of jobs is efficiently reduced by the BQ scheduling for both allocation schemes. The operational range of the system is also increased. With the FCFS queue, the system gets saturated quickly for traffic above 1.5. With the bypass-queue, the system has a higher saturating point. Only the results for uniform job size distribution is shown. The normal job size distribution exhibits similar behavior.

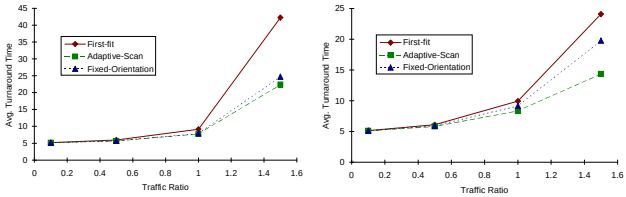


(a) First-Fit (b) Adaptive-Scan

Figure 3: Effect of the Bypass-Queue Scheduling.

3.2 Fixed-Orientation Allocation

The FO scheme is compared with the first-fit and adaptive-scan allocations. Figure 4 illustrate the comparisons with the uniform job size distribution and the normal job size distribution, respectively. The FO scheme outperforms the first-fit algorithm as expected because of the reduced fragmentation. It provides shorter turnaround time than the first-fit algorithm for all traffic ratios. The average turnaround time is reduced by as much as 42% from the first-fit algorithm at traffic ratio of 1.5 in Figure 4(a).



(a) Uniform Distribution (b) Normal Distribution

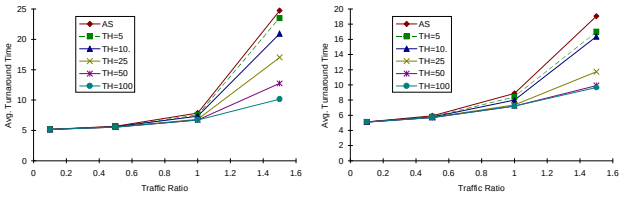
Figure 4: Effect of Different Allocation Algorithms.

The adaptive-scan does have less fragmentation than the FO scheme because of the alternative orientations of jobs it utilizes. However, the nearly identical performance of the FO allocation indicates that this difference is insignificant. By properly arranging the allocation of jobs, the FO scheme avoids fragmenting the system in most cases. The adaptive-scan gains its slightly better performance at the price of a higher computational complexity. Under low traffic, the FO and the adaptive-scan performs equally well with negligible difference. At higher traffic ratio, the adaptive-scan starts to perform slightly better than the fixed-orientation scheme. This is because in a heavily loaded system, jobs arrive and depart more frequently and more serious fragmentation is observed. The adaptive-scan solves the fragmentation by checking alternative orientation for submesh allocation and therefore takes more time to perform. In our simulations, the adaptive-scan takes 20% to 30% more time to complete the allocation process. The fixed-orientation allocation is feasible to be used in an integrated processor management policy because of its low computation demand.

3.3 The Integrated Policy

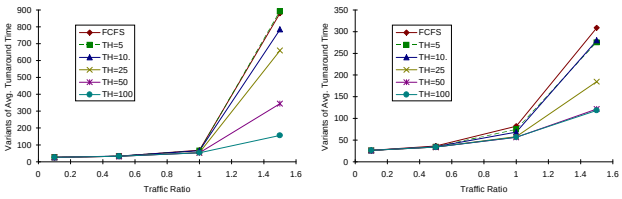
The integrated processor management scheme is simulated and the results are in Figure 5. The lines labeled *AS* is the average turnaround time obtained for the adaptive-scan algorithm using FCFS queue. It is included for comparison because the adaptive-scan scheme has the best submesh recognition ability. The integrated scheme delivers better performance than the adaptive-scan with a small threshold time. Increasing the threshold time further reduces the average turnaround time of jobs. Both the BQ scheduling and the FO allocation contribute to this performance improvement. Because of the less fragmentation of the FO allocation, more jobs can bypass other jobs in the queue and get executed.

Larger jobs have the tendency to be passed by other jobs with the BQ scheduling. Therefore, we compare the variance of the average turnaround time in Figure 6. Contrary to our expectation, the variance does not increase when the threshold time increases. The variance is actually reduced when larger threshold is used. This is attributed to the fact that a larger threshold time reduces the turnaround time so efficiently that most of the jobs can be served within a short period of time and thus results in a small variance. A low variance for the turnaround time is a good property for the system because most of the jobs can be expected to finish within a certain range of the average.



(a) Uniform Distribution (b) Normal Distribution

Figure 5: The Integrated Processor Management Policy.



(a) Uniform Distribution (b) Normal Distribution

Figure 6: Variance of the Avg. Turnaround Time.

4 Concluding Remarks

Processor allocation and job scheduling are both efficient ways of improving the performance of multicomputer systems. It is therefore desirable to integrate processor allocation and job scheduling to take advantage of both the approaches. The high complexities associated with existing processor allocation algorithms and job scheduling strategies makes the integration of the two approaches impractical. In this paper, we propose a bypass-queue scheduling policy and a fixed-orientation allocation algorithm. Both schemes have very low computational complexity and are therefore suitable for integration.

Simulation results indicate that the bypass-queue scheduling and the fixed-orientation allocation both provide good system performance when applied in isolation. Integrating these two schemes improves the system performance further. The ease of implementation and the performance make the proposed integrated processor management scheme highly attractive.

References

[1] Y. H. Zhu, *"Efficient Processor Allocation Strategies for Mesh-Connected Parallel Computers,"* Journal of Parallel and Distributed Computing, 16, pp. 328-337, 1992.

[2] J. Ding and L. N. Bhuyan, *"An Adaptive Submesh Allocation Strategy for Two-Dimensional Mesh Connected Systems,"* Proc. of Int. Conf on Parallel Processing, Vol. II, pp. 193-200, Aug. 1993.

[3] D. D. Sharma and D. K. Pradhan, *"A Fast and Efficient Strategy for Submesh Allocation in Mesh-Connected Parallel Computers,"* Proc. of the 5th IEEE Symp. on Parallel and Distributed Processing, pp. 682-693, Dec. 1993.

[4] W. Liu, V. Lo, K. Windisch, and B. Nitzberg, *Non-continuous Processor Allocation Algorithms for Distributed Memory Multicomputers,"* Proc. of the 1994 Int. Conf. on Supercomputing, pp. 227-236, 1994.

[5] D. Das Sharma and D. K. Pradhan, *"Job Scheduling in Mesh Multicomputers,"* Proc. Int. Conf. on Parallel Processing, vol. II, pp. 251-258, 1994.

[6] D. Min and M. W. Mutka, *"Efficient Job Scheduling in a Mesh Multicomputer Without Discrimination Against Large Jobs,"* Proc. of the IEEE Symposium on Parallel and Distributed Processing pp. 52-59, October, 1995.