An Adaptive Job Allocation Method for Multicomputer Systems

Chung-yen Chang and Prasant Mohapatra
Department of Electrical and Computer Engineering
Iowa State University
Ames, IA 50011

Abstract

The fragmentation problem in multicomputer systems reduces the system utilization and prohibits the systems from performing at their full capacity. In this paper, we propose a generic job allocation method for multicomputer systems based on job size reduction. We reduce the subsystem size requirement adaptively according to the availability of processors. The fragmentation problem is greatly alleviated by this approach. To ensure that the benefit of reducing fragmentation is not outweighed by the penalty of executing jobs on less number of processors, we restrict the number of times the size of a job can be reduced; hence the name restricted size reduction (RSR). Extensive simulations are conducted to validate the RSR method for hypercubes and mesh-based systems with different allocation algorithms. It is observed in both mesh and hypercube that by using the RSR method a simple algorithm can provide better performance than the more sophisticated allocation algorithms. We have also compared RSR method with the limit allocation that is based on a similar idea. Our method outperforms the limit allocation and provides better fairness to different size jobs. The performance gain, fairness, and low complexity makes the RSR method highly attractive.

1 Introduction

The performance of multicomputers depends on the efficient use of the computing resources. Conventional methods to improve the performance of the multicomputers include efficient allocation algorithms and scheduling strategies.

Several job allocation schemes [1]-[8] have been proposed for the directly-connected multicomputers. Because the multicomputers use message-passing techniques, the allocation schemes attempt to locate a contiguous portion of the processors for the execution of a job to minimize the distance of interprocessor communication. The processors allocated to a job retain the same topology as the entire system with the number of processors determined according to the requirement of the individual jobs. In a mesh-connected system, jobs are allocated to submeshes. In a hypercube system, jobs are allocated to subcubes. The allocation algorithm is responsible for detecting the free nodes and recognizing whether the free nodes can form the required topology for the execution of a job. A system may contain sufficient number of processors for the execution of a job while the allocation algorithm fails to find the required topology for task execution. In such cases, the job has to wait until some of the running processes release the nodes they hold and the allocation algorithm can assign these nodes to the waiting job(s). Allocation algorithms with better recognition ability can improve the chance of assigning a job into the system. Although the recognition ability varies dramatically among allocation algorithms, the actual performance does not show significant difference [12, 13]. This is because of the fragmentation problem. Fragmentation occurs when the required number of nodes are available but they do not form the topology required for task execution.

The performance gain observed from efficient allocation schemes is limited mainly by the first-come-first-serve (FCFS) discipline. A job that cannot be allocated to the system will block all the following jobs from entering the system. The scheduling schemes [9, 11, 12] arrange the order of execution so that jobs could be executed out of the order in which they arrived and thus reduce the blocking effect. However, these scheduling schemes do not provide fairness to various tasks. Some jobs may have to wait for a very long time before they get executed while a job arriving later may get executed earlier.

In this paper, we propose a new approach that considers adjustment of job sizes in the pursuit of improving multicomputer system performance. A job is forced to be executed on a smaller subgraph of the system when a subsystem of the required size cannot be found. The size reduction of a job is done by evenly folding the job into a recognizable smaller subgraph by the allocation algorithm. In hypercubes, every size reduction forces a job to be executed on a smaller subcube with one less dimension. In the mesh systems, a size reduction can result in folding the job in one direction with most allocation algorithms or in both directions when the two-dimensional buddy algorithm is used. Folding a job evenly ensures that the workload of the processors are increased evenly and no bottleneck is introduced. A job can be folded further if necessary. Unlike the limit allocation [13] which limits the largest executable job size, to ensure the reduced waiting time is not overwhelmed by the increased execution time caused by job folding, we restrict the number of times a job can be folded. The method is thus called the *restricted size reduction (RSR)* method.

The RSR method is simulated with several allocation schemes. Mesh and hypercube are considered as two example topologies as they constitute the backbones of most of today's multicomputer systems. However, the RSR method is not restricted to these topologies. It is a generic method and can be applied to other topologies using any allocation algorithm with low overhead. The results indicate the effectiveness of this method in improving the multicomputer performance. A simple allocation algorithm with RSR can outperform an algorithm with complete subgraph recognition ability. We also compare the RSR method with the limit allocation. As the result shows, the RSR method in hypercube using the simple buddy allocation has many advantages over the limit allocation considering the average turnaround time and the fairness. We therefore conclude that the RSR method is an efficient way to improve the performance of the multicomputers. Similar conclusions have been also derived independently and reported in [14].

The next section surveys the related work. The detailed discussion and the rationale behind the RSR method is discussed in Section 3. The simulation study of the RSR method and the comparison with the other techniques is shown in Section 4. Section 5 concludes this study.

2 Related Work

2.1 Allocation Algorithms

In this subsection, we outline some of the popular allocation algorithms proposed for hypercube and mesh-connected systems. Most of the other proposed schemes are conceptually similar to these schemes. However, they vary in terms of implementation complexity.

2.1.1 Allocation in Mesh

Two Dimensional Buddy: The two dimensional buddy (TDB) [1] is a generalization of the one dimensional buddy algorithm [15] for storage allocation. The system is assumed to be a square with the side length equals to a power of two. The size of a requested submesh is rounded up to the nearest square with sides power of two. Every square of processors can form a larger square with its three neighboring buddies. The time complexity for allocating an $(N \times N)$ job in an $(M \times M)$ mesh when there are k jobs in the system is equal to $O(k(\frac{M}{N})^2)$.

Frame Sliding: The frame sliding method [2] is proposed to reduce the fragmentation problem of the TDB allocation. The size of the required submesh of a job is called a frame. A frame filled with available nodes is able to execute the job. The algorithm slides the frame across the system at non-overlapping locations to examine for a free submesh to execute the job. The complexity of this allocation for an $(m \times n)$ job in an $(M \times N)$ system is equal to $O(\frac{MN}{mn})$.

Perfect Recognition: The perfect recognition algorithms for mesh systems are proposed in [3]. They guarantee a required submesh can always be located provided it exists. Two algorithms are proposed, namely the first-fit and the best-fit algorithm. Both algorithm perform equally well at the same time complexity of $O(MN)$ in a $(M \times N)$ system with the space complexity of $O(MN)$.

2.1.2 Allocation in Hypercube

Buddy: Buddy allocation scheme is similar to the memory allocation algorithm[15]. Every subcube has a buddy of the same size. Two adjacent buddies can be combined to form a larger cube for the execution of a larger job. A job is always assigned a subcube for its execution. For an n-cube, the nodes are numbered from 0 to $2^n - 1$. For a job requiring a k cube, the algorithm searches for the least integer m such that all the nodes in the region $[m2^k, (m+1)2^k - 1]$ are available. The time complexity of allocation and deallocation in the above case are $O(2^n)$ and $O(2^k)$, respectively.

Other Algorithms for Hypercube: Many other allocation algorithms have been proposed for the hypercubes to implement perfect subcube recognition ability. Some examples include the multiple gray code [4], maximal subset of subcubes (MSS) [5], free list [6], tree collapsing [7] and PC-graph [8]. These algorithms have perfect subcube recognition abilities at the price of high implementation complexities. Our intension is to show that a simple allocation algorithm such as *buddy* can be modified to provide a good performance. The detailed discussion of these algorithms are beyond the scope of this paper and the readers are referred to [4]–[8] for the details.

2.1.3 Problems of the Allocation Schemes

The allocation algorithms vary in terms of recognition ability and implementation complexity. However, studies [12, 13] show that a better allocation algorithm does not guarantee a significant performance improvement because of the fragmentation problem.

Fragmentation can be classified into internal and external fragmentation. The internal fragmentation is the result of allocating jobs only to subgraphs of certain sizes. In the TDB allocation for the mesh systems, jobs are assigned to the squares with the side lengths equal to the power of two. In hypercubes, jobs are always assigned to a subcube. Jobs that require irregular sizes are assigned more processors than necessary. The extra nodes assigned to a job cannot be used for the execution of any other job and cause the system to be underutilized. There are two types of external fragmentation. The first type of external fragmentation is a result of the imperfect recognition ability of the allocation algorithms. The allocation algorithms only check for the free processors in the subgraphs of the system at certain locations. For example, in the buddy allocations, only nodes that are buddies to one another are considered for the allocation of a job. It may so happen that there are sufficient processors in the shape of a submesh or a subcube for the execution of a job but the allocator fails to locate these processors. This kind of external fragmentation can be solved by using an allocation algorithm that checks all the possible locations for the subgraphs. The other type of external fragmentation arises when the available processors are not physically connected in the shape of a subgraph of the system. None of the allocation algorithms can assign a job to the system for execution in this situation even when the number of free nodes are sufficient.

2.2 Scheduling Approaches

The FCFS discipline used with the conventional allocation schemes augments the fragmentation problem in multicomputer systems. When the allocator fails to assign a job for execution, the job has to wait until more processors become available so that the allocator can allocate it into the system for execution. Because of the FCFS discipline, a waiting process will block all the succeeding jobs from entering the system. The waiting time for a blocking job is thus accumulated in the turnaround time of all the jobs following it. The scheduling approaches improve the performance by reducing the blocking effect. This can be done by allowing the jobs bypass the blocking process or by rearranging the execution order of jobs so that more jobs can be allocated to the system.

In [9], a scheduling method for the mesh systems combining a priority technique with a reservation scheme is proposed. The reservation scheme allows a job to reserve processors for its execution and let other jobs bypass it to enter the system if they can. The priority technique is used to reduce the chance of fragmentation. The *lazy* [11] scheduling scheme proposed for hypercubes temporarily delays the allocation of a job if any other job of the same dimension is running. The delayed job is then executed on the existing subcube rather then acquiring a new subcube. The fragmentation of the system and the blocking problem with the FCFS scheme are both reduced. Another scheduling scheme for the hypercubes called *scan* [12] groups jobs of the same size together for the allocation and thereby reduces fragmentation.

There are several problems associated with these scheduling approaches. First, the complexities of the scheduling approaches are high. In addition to the underlying allocation algorithm used, the scheduler imposes additional overhead for determining the order of the execution. In the reservation approach, a node can be idle waiting for the availability of other nodes that are reserved together for the execution of the same job. This causes the system to be underutilized and limits the performance. The performance gain of the scan policy is dependent on the workload environment. Second, all the scheduling approaches have one common problem which is the violation of fairness as the jobs are not executed in the exact order as they are submitted to the system.

2.3 Other Approaches

A rather unconventional approach taken to improve the performance of the multicomputer system is by adjusting the size of the jobs. Changing the job size to avoid fragmentation has been studied in [10] and [13]. In [10], the authors proposed two allocation policies for the mesh-connected system, namely the *equi-partition* and *folding* allocations. Both policies assumes the initial submesh requirements are all equal to the size of the system. Jobs are also migrated between nodes in the system. The job size assumption is not practical and the overhead for migrating jobs are not ignorable. The two methods in [10] are hence less attractive for the actual implementation.

The *limit* allocation proposed in [13] is an efficient processor management strategy for the hypercube systems. Three limit allocation algorithms are considered, namely the *limit-k, greedy,* and *average.* The basic idea of limit allocation is to reduce fragmentation by limiting the maximum job size in the system. Job that requires a subcube larger than the limit will have to be folded to the limited size. This causes serious underutilization of the system under low load and results in poor performance. The results in [13] and our simulation both prove this point. Another major problem with the limit allocation is its unfair treatment to jobs of different sizes. A folded job is executed on less processors than it initially requested. This increases the execution time of the job. Comparing with the execution time of the unfolded jobs, the folded jobs are obviously treated unfairly. With the limit allocation, larger jobs are folded more often than the smaller ones. In the limit-k allocation, jobs smaller than the limit size k never get folded while jobs larger than the limit always get folded. The turnaround time of the smaller jobs is thus likely to be shorter than the turnaround time of the larger jobs, which makes the scheme unfair.

3 Restricted Size Reduction (RSR) Scheme

We propose a generic job allocation method to improve the performance of the multicomputers by reducing the fragmentation problem. The proposed method reduces the size of a job for execution when fragmentation prevents it from execution. The number of times that size reduction can be applied to a job is restricted to minimize the side-effect of the increased execution time caused by the size reduction. The allocation method is thus called the *restricted size reduction (RSR)* method. Executing a job on a smaller subsystem adaptively benefits the performance in two ways. First, the fragmented nodes are utilized and more jobs can be accommodated by the system simultaneously. Second, the waiting delay is reduced because jobs can be allocated earlier. The reduction of waiting delay is especially important for system with higher load because the blocking problem is more serious for systems with high load.

The detailed description of the algorithm is provided in Section 3.2. Here we list the important properties of the RSR scheme.

1. It is a generic processor management concept. The RSR scheme is not limited to a single architecture or a particular allocation algorithm.

2. It is fair because jobs are serviced with the FCFS discipline. No job is treated unfairly on the basis its size request.

3. It is adaptive. A job is only folded when the fragmentation prevents it from execution. This property guarantees that the system maintains a reasonable utilization at all ranges of workload.

4. It is flexible. The system administrator can determine the optimal restriction on the size reduction according to the individual system's need and workload.

3.1 Suitability Study

In a multicomputer system, jobs come in different sizes according to their inherent parallelism. The available parallelism of a job prohibit changing the job size randomly. However, we argue that it is safe to scale down a job to some extent. In a multicomputer, a job is divided into subtasks running on different processors. The number of subtasks that can be run concurrently on different processors is limited by the degree of parallelism. It is always possible to reduce the degree of parallelism in a job. The nCUBE's software environment [16] explicitly supports the execution of a job on different size cubes. For applications that require at least some certain number of processors to execute, it is still possible to fold the program and run them on the smaller subsystem in a context-switching fashion.

The tradeoff for running a job on a smaller subsystem is the increase of the execution time. However, when a job is folded to half of the size it requested, it is unlikely for the job's execution time to exceed twice of its execution time when its request is granted without folding. Additionally, the communication overhead for less processors is expected to be reduced for several reasons. First, the communication path is shorter in a smaller subsystem. Second, the smaller number of processors in the smaller subsystem causes less interference between messages transmitted by different processors. Third, the frequency of interprocessor communication could be reduced because each processor is now running for a larger share of information. In case of multiple process of the same task running on the same processor by context-switching, the communication overhead could be even less because some inter-processor communication might become intra-processor communication. Pessimistically, we assume a linear increase on execution time when a job is folded as is assumed in [13].

3.2 The RSR Algorithm

The RSR scheme is implemented along with an allocation algorithm. A single queue is used to hold jobs waiting for allocation. Jobs are serviced with the FCFS discipline to preserve the fairness. A job that gets to the head of the job queue is examined for allocation. If the underlying allocation algorithm finds a suitable subsystem of free processors for the execution of the job, the job is allocated for execution. If a subsystem of the required size cannot be located, the allocator reduces the size of the job to the next smaller allocable subsystem and examine the availability of free nodes for the job's execution. This process repeats until either the job is allocated or the number of times of size reduction of the job reaches a preset restriction. The allocation attempt is stopped when the job cannot be allocated after all allowable size reductions have been considered. When a job departs after execution, the allocation process is repeated.

The reason for the restriction on the job size reduction is to ensure the performance gain of reducing fragmentation is not outweighed by the loss of reduced parallelism in the application. The RSR allocation is flexible for implementation. The system administrator can determine the maximum number of folding that a job can endure based on the system status and the workload parameters to get the optimal performance. Discussion on the selection of the number of folding is presented in Section 4.2. The restriction on size reduction also distinguishes the RSR allocation from the limit allocation. The limit allocation limits the maximum subcube that a job can use. Only the jobs larger than the limit are considered for folding. The RSR allocation limits the maximum number of folding that can be applied to a job. A job is only folded when fragmentation prevents it from execution. Jobs of all sizes have the possibility of being folded. This makes the RSR method fairer than the limit allocation. Additionally, as discussed in Section 2.3, the limit allocation has the problem of underutilization because it folds jobs regardless of the system load. Some jobs may be folded while other processors are idling. The RSR allocation is an adaptive allocation and it only folds a job when necessary. A job is assigned to as many resources it can get under the size reduction restriction. The underutilization problem is avoided and the performance is expected to improve.

To describe the RSR algorithm for a particular architecture, two things have to be considered, the underlying allocation algorithm and the restriction of the size reduction. Any existing allocation algorithm can be used with the RSR scheme. Our results show that a simple algorithm with RSR technique can outperform more robust allocation algorithms. Therefore, it is advantageous to choose an algorithm with low time complexity. The size reduction is done depending on the underlying allocation algorithm. For instance in the mesh systems with TDB algorithm, reducing the size of a job once results in a smaller square which requires 1/4 of the processors it requested before the reduction. For all allocation algorithms in the hypercubes, a size reduction folds a job into a smaller cube with half the number of the processors. The restriction of the number of size reductions can be applied to a job is a parameter of system load and performance. An RSR allocation with the maximum number of size reductions allowed to a job set to t is called *RSR-t allocation.* The size of a job is guaranteed to be reduced less than t times to ensure that the performance gain. The sketch of the RSR-t allocation in the hypercube system is shown below.

The RSR-t Allocation for Hypercube

1. *Let k be the size of the subcube requested by the job to be allocated. Set the minimum allowable size $s = MIN\{k - t, 0\}$.*

2. *Check the availability of the k-cube using the embedded allocation algorithm. If found, allocate the job and goto step 4.*

3. *Set k to $k - 1$. If $k \geq s$ goto step 2, else goto step 5.*

4. *If the job queue is not empty, goto step 1 to allocate the first job in the queue.*

5. *End*

Judicious selection of the value of t is essential to exploit the advantages offered by the RSR allocation scheme. Some pointers toward the selection of the value of t are discussed in Section 4.2.

4 Performance Analysis

The advantages of the RSR allocation are justified through three steps. First, the impact of the RSR allocation on different allocation algorithms is shown in Section 4.2. Second, the performances of different allocation algorithms combined with the RSR concept are compared in Section 4.3 to show how a simple algorithm using the RSR method can outperform an algorithm with perfect subgraph recognition ability. Last, the performance of the RSR allocation is compared with the limit allocation in Section 4.4.

4.1 Simulation Environment

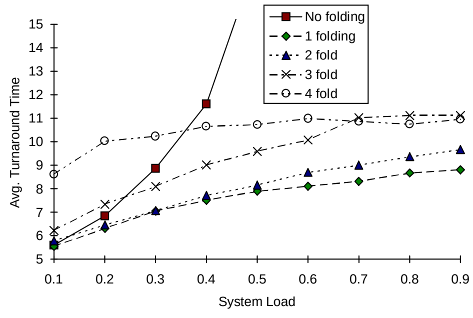
The simulations for the mesh systems are done for a 32×32 system. For the two-dimensional buddy algorithm, the jobs are assumed to be squares with the side-length following uniform distribution between 1 and 32. For other allocation algorithms, the job sizes are assumed to be uniformly and uncorrelatedly distributed in both the x and y dimensions with the values between 1 and 32. We also simulated the effect of the RSR allocation for an eight dimensional hypercube. Job size is assumed either uniformly or normally distributed between 0 and 7 dimensions. The probability for any request between a 0 and a 7-cube is equals to 1/8 for the uniform distribution. Jobs with normally distributed dimensions are also simulated for the hypercubes. The normal distribution of the job size is obtained by discretizing the probability of a normal distribution between -2.5σ and $+2.5\sigma$ of its mean. The probability obtained is normalized to one to include the probability outside this region. The resulting probabilities for different job sizes in the 8-cube system are $(p0 = p7 = 0.025,\ p1 = p6 = 0.076,\ p2 = p5 = 0.162,\ p3 = p4 = 0.237)$.

In all the simulations, jobs are assumed to arrive in a Poisson process. The distribution for the execution time of a job is assumed to be exponential. For hypercube, we also simulated the truncated normal service. Both distribution have the same mean of 5 time units. For the truncated normal service time, the standard deviation is assumed to be 2 time units. We simulate the modeled system with job arrival rates calculated for different system loads. The arrival rate λ is calculated as $\frac{\textit{system size}}{\textit{mean job size} \times \textit{mean service time}} \times \textit{system load}$. The system load is approximately equal to the system utilization before a system gets saturated.

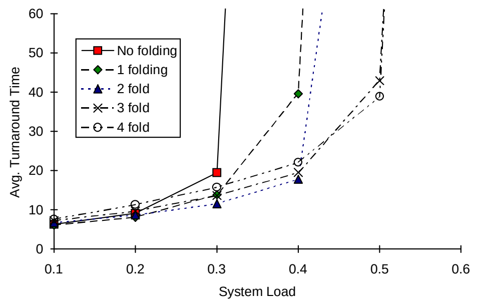
All allocation schemes are implemented as event-driven simulations. The size reduction is done by folding the job into a smaller subcube for the hypercube system. For the mesh using the TDB allocation, every size reduction results in a smaller square. For other allocation algorithms in the mesh, every size reduction results in folding the job in the direction which it has longer side length. This is done to ensure the resulting jobs are in a regular shape and to avoid future fragmentation of the system. We simulated all allocation algorithms until the completion of 50,500 jobs. The first 500 jobs are ignored in the measurement to avoid premature results. Because the execution time is increased after a job is folded, the increased execution time needs to be measured. Therefore, the average turnaround time which includes the execution time and the queuing delay of a job is measured for fair comparison.

4.2 The Impact of RSR Scheme

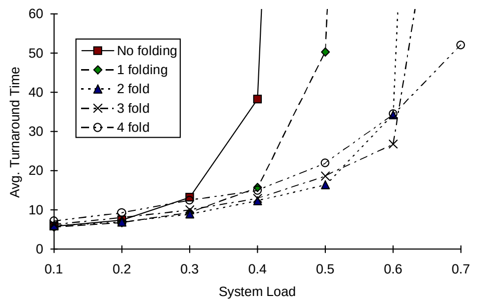
For the mesh systems, the allocation algorithms simulated are the TDB, the frame-sliding, and one of the perfect recognition algorithm – the first-fit algorithm. These results are illustrated in Figure 1.



(a) The TDB algorithm



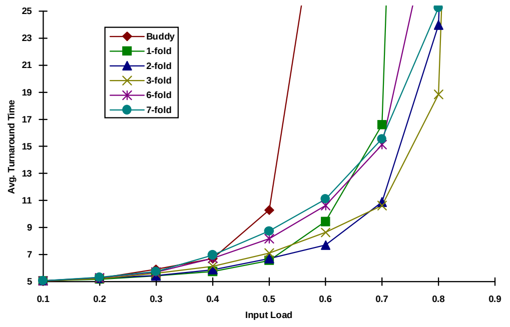
(b) The frame-sliding algorithm



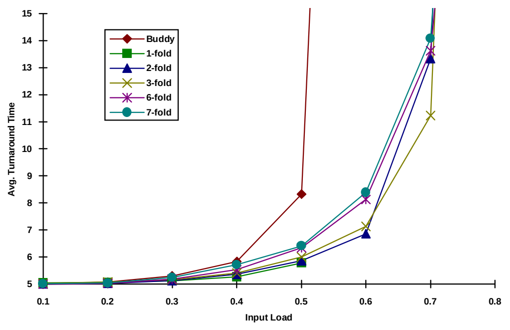
(c) The first-fit algorithm

Figure 1: The average turnaround time vs. system load for different allocation algorithms in a 32×32 mesh.

Figure 1.(a) shows the average turnaround time of jobs using the TDB allocation algorithms and RSR scheme. The RSR-1 scheme provides a tremendous improvement on the average turnaround time. Further increase on the number of allowable size reductions does not provide performance gain. Because TDB algorithm allocates jobs to square subcubes of certain sizes, job size reduction dramatically increases the probability of allocating a job. RSR also causes jobs to be assigned in similar sizes and avoids fragmentation. Each job size reduction in TDB increases the execution time of a job by four times. This increase of execution time quickly overshadows the benefit of reduced queuing delay with larger number of job size reductions. Figures 1.(b) and (c) are the results for the frame sliding and the perfect recognition algorithms, respectively. The behavior of the three algorithms are similar. Under low system load, the average turnaround time of the jobs are very close for RSR schemes allowing different number of size reductions. The RSR allocations allowing smaller number of size reductions have slightly shorter turnaround time than the RSR schemes allowing larger number of size reductions. However, the RSR schemes allowing larger number of size reductions provide a wider operational range.



(a) Uniform job size



(b) Normal job size

Figure 2: The average turnaround time vs. system load for different allocation algorithms in an 8-cube using the buddy allocation with exponentially distributed service time.

The RSR scheme is also simulated with the buddy allocation algorithm for the hypercube systems. The results are shown in Figure 2. The operational range of the system is improved with the assistance of the RSR scheme. The turnaround time of the RSR schemes allowing less numbers of size reductions have shorter turnaround time than those allowing larger numbers of size reductions. This is exactly the same observation we have made for the mesh system. As discussed earlier, the performance gain is mainly caused by the efficient reduction of the fragmentation problem. This is because of the blocking effect associated with the FCFS discipline. Using the RSR, a job is executed as early as the size reduction restriction allows. Therefore, it is less likely to block other jobs. The fragmented processors are also utilized to execute the folded jobs. The turnaround time of the RSR schemes are hence shorter.

Another observation made from these results is the tradeoff between the larger operational range and the lower average turnaround time. When the system load is high, a system needs to accommodate as many jobs as possible in order to avoid saturation. Allowing more size reduction makes this possible. However, the allocations allowing smaller number of size reductions provides shorter turnaround time under low to medium load. This is because the allocations allowing more size reductions tend to reduce the size of a job more often. As the blocking effect is not serious under these loads, the execution time is the dominant factor of the average turnaround time. Therefore, allowing less number of size reduction avoids the unnecessary job size reduction and provide a better performance under such loads. On the other hand, the system performance is improved rapidly with a small number of size reductions. The performance improvement of allowing more size reduction is not significant. Moreover, when the system is running beyond a reasonable load, most jobs will be executed with severe size reduction if the restriction on size reduction is high. For example, in the mesh system using frame-sliding allocation, the operational range is increased to around 50% utilization with three size reductions allowed. Beyond this operational range, the average turnaround time of the job becomes approximately 10 times of the expected execution time if the job is not folded. This phenomena is more obvious in the hypercubes where just one size reduction improves the operational range to more than 80% utilization. Therefore, it is sufficient to improve the system performance with just a few times of size reduction.

4.3 Comparison Between Different Allocations

To illustrate the effectiveness of the RSR scheme in improving the multicomputer performance, we compare different allocation schemes for mesh in this section. The algorithms compared is the frame-sliding algorithm and the first-fit algorithm. The TDB algorithm is not compared because of its special assumption that job sizes have to be square and cannot be categorized with the other two algorithms which take any rectangular jobs. Our intension is to show that an allocation algorithm with an inferior subsystem recognition ability can outperform a better algorithm with the help of the RSR scheme. The first-fit algorithm [3] has the ability to detect any available submesh provided it exists. It provides equally-well performance as the other perfect recognition algorithm (the best-fit algorithm).

As observed from Figure 3, the frame-sliding algorithm performs closely to the first-fit algorithm after allowing one size reduction. By allowing more size reduction, the frame-sliding algorithm outperforms the first-fit algorithm with a wider operational range. The turnaround time of the job under very low system load is slightly worse for the RSR scheme using frame-sliding algorithm because of the job size reduction. However in most of the working region, the RSR scheme with the frame-sliding algorithm does provide comparative (if not better) performance. These results indicate that an algorithm with low submesh recognition ability can provide a wider operational range and lower turnaround time with the help of the RSR scheme.

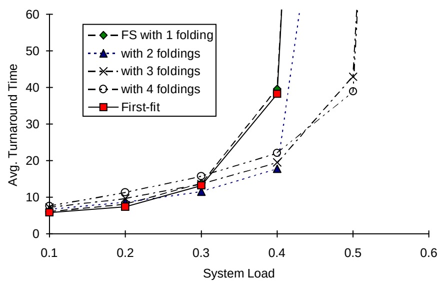


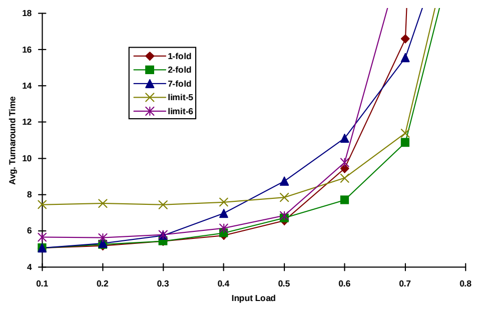
Figure 3: Comparing the RSR schemes using frame-sliding algorithm with the first-fit algorithm.

4.4 Comparing with the Limit Allocation

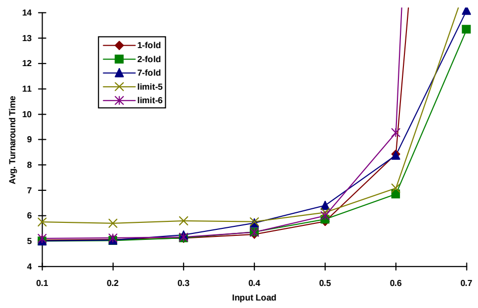
We compare the RSR scheme with the limit allocation [13] proposed ealier for the hypercube system. The *limit-k* allocation reduces all jobs larger than a k-cube to k-cube. The *RSR-t* allocation allows the size of a job to be reduced at most t times. Therefore, we feel it is fair to compare the allocation from different family with the same maximum number of folding. In the 8-cube system we simulated, limit-0 and RSR-7, limit-5 and RSR-2, limit-6 and RSR-1 are comparable schemes.

Figure 4 shows the comparison between the two schemes. The RSR methods perform better than the corresponding limit schemes. All the limit-k allocations have relatively poor performance at low to medium load. This is because of the underutilization problem discussed in Section 2.3. Because larger jobs are forced to run in a smaller cube in limit allocation regardless of the system load, the larger jobs always have longer execution time. In the extreme case such as limit-0, a job initially requesting for a 7-cube would spend 128 times of the execution time when it is granted a 7-cube. For the RSR allocations, a job is folded only when necessary. Under a low input load, a job almost never gets its size reduced. When the load increases, more jobs get folded. Since we restrict the number of times size reduction can be applied to a job, increase on the execution time is limited.

Along with better performance, the RSR scheme also provides fairness toward job of different size. Figure 5 shows the average turnaround time for jobs of different sizes. The extreme comparisons between the RSR-7 and the limit-0 allocation are illustrated in part (a). The RSR-1 and RSR-2 are compared with the limit-6 and limit-5, correspondingly in part (b). The turnaround time is plotted in logarithmic scale because of the big difference on the response time for the limit-k allocations. Limit-6 and RSR-1 have close response time for all job sizes. This is because the longer queuing delay jobs experienced in such load outweighs the execution time. RSR-2 and RSR-7 started to treat jobs unfairly at such a load. Jobs requiring larger subcubes have higher turnaround time than smaller jobs. Since larger jobs are more likely to be affected by the fragmentation, their sizes are more likely to be reduced. With a higher size reduction restriction, larger jobs could be folded more than once. The reduction of the job sizes increases the execution time of the larger jobs. The unfair treatment of limit-0 and limit-5 is visible because the queuing delay does not outweigh the turnaround time for these two allocations.



(a) Uniform job size distribution



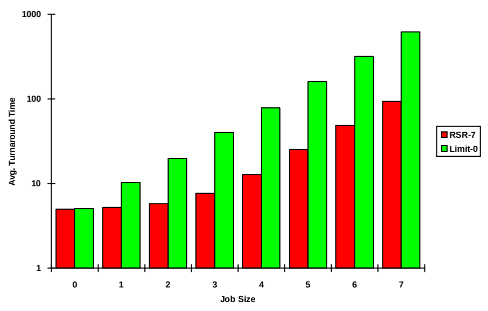
(b) Normal job size distribution

Figure 4: Comparing between the limit allocations and the RSR schemes with the exponential service time distribution in an 8-cube system.

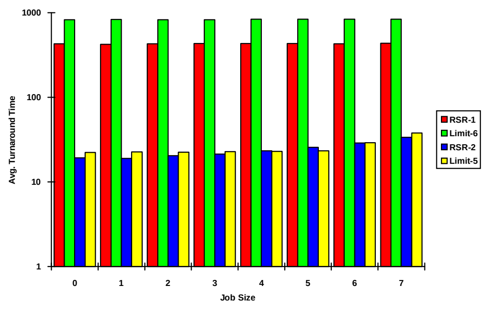
The greedy limit is a variant of our proposed RSR allocation scheme when the restriction of size reduction allows all jobs to be executed on a single node when necessary. However, our results in Section 4.2 points out that allowing too much size reduction results in unnecessary folding of the jobs under low to medium load. The performance of the system under such load is sacrificed with the excessive size reduction. Therefore, it is not desirable to use the greedy limit allocation. The RSR schemes also provide much fairer treatment to jobs of different sizes. Therefore, we conclude that the RSR scheme is better than the limit allocations from the comparisons.

5 Concluding Remarks

A new allocation strategy called restricted size reduction (RSR) is proposed to improve the multicomputer performance. The RSR allocation is a generic



(a)



(b)

Figure 5: Fairness comparison between the limit allocation and the RSR schemes at a high system load (0.7) for an 8-cube system.

approach that can be used with any allocation algorithm in any architectures. The suitability of adjusting the job size is studied. The possible impact on the execution time of an individual process is justified from the speedup laws. The complexity of the RSR strategy is also analyzed. The complexity is believed to be no higher than the underlying allocation algorithm used.

Extensive simulations are performed to validate the advantages of the RSR allocation in both mesh and hypercube systems. The impact of folding a job is first studied for different allocation algorithms. Different allocation algorithms applying RSR are then compared for the mesh system. The RSR allocation is also compared with the limit allocation which is believed to be the most efficient processor management method for the hypercubes. It is concluded that the RSR allocation improves the performance of every allocation algorithm by efficiently reducing the fragmentation. Thus, RSR allocation makes it possible for a system to have a superior performance without using a sophisticated allocation algorithm that has high time complexity.

References

[1] K. Li and K. H. Cheng, *"A Two-Dimensional Buddy System for Dynamic Resource Allocation in a Partitionable Mesh Connected System,"* Journal of Parallel and Distributed Computing, 12, pp. 79-83, 1991.

[2] P. J. Chuang and N. F. Tzeng, *"An Efficient Submesh Allocation Strategy for Mesh Computer Systems,"* Int. Conf. on Distributed Computing Systems, pp. 256-263, May 1991.

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

[4] M. S. Chen and K. G. Shin, *"Processor Allocation in an N-Cube Multiprocessor Using Gray Codes,"* IEEE Trans. on Computers, vol. C-36, No. 12, Dec. 1987.

[5] S. Dutt and J. P. Hayes, *"Subcube Allocation in Hypercube Computers,"* IEEE Trans. on Computers, vol. 40, No. 3, Mar. 1991.

[6] J. Kim, C. R. Das, and W. Lin, *"A Top-Down Processor Allocation Scheme for Hypercube Computers,"* IEEE Trans. on Parallel and Distributed Systems, pp. 20-30, Jan. 1991.

[7] P. J. Chuang and N. F. Tzeng, *"Dynamic Processor Allocationin Hypercube Computers,"* Int. Symp. on Computer Architecture, pp. 40-49, May 1990.

[8] H. Wang and Q. Yang, *"Prime Cube Graph Approach for Processor Allocation in Hypercube Multiprocessors,"* Int. Conf. on Parallel Processing, pp. 25-32, Aug. 1991.

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

[10] C. McCann and J. Zahorjan, *"Processor Allocation Policies for Message-Passing Parallel Computers,"* Proc. ACM SIGMETRICS Conf. pp. 19-32, May, 1994.

[11] P. Mohapatra, C. Yu, and C. R. Das, *"A Lazy Scheduling Scheme for Hypercube Computers,"* Journal of Parallel and Distributed Computing, 27, pp. 26-37, May, 1995.

[12] P. Krueger, T. H. Lai, and V. A. Radiya, *"Processor Allocation vs. Job Scheduling on Hypercube Computers,"* Int. Conf. on Distributed Computing Systems, pp. 394-401, 1991.

[13] C. Yu and C. R. Das, *"Limit Allocation: An Efficient Processor Management Scheme for Hypercubes,"* Int. Conf. on Parallel Processing, vol. II, pp. 143-150, 1994.

[14] B. S. Yoo, C. R. Das, and C. Yu, "*Processor Management Techniques for Mesh-Connected Multiprocessors,*" Int. Conf. on Parallel Processing, vol. II, pp. 105 - 112, 1995.

[15] K. C. Knowlton, *"A Fast Storage Allocator,"* Communications of ACM, vol. 8, pp. 623-625, Oct. 1965.

[16] *nCUBE2 Programmer's Guide,* nCUBE, 1992.