Efficient Subtorus Processor Allocation in a Multi-Dimensional Torus

Weizhen Mao
Department of Computer Science
College of William and Mary
Williamsburg, VA 23187-8795, USA
wm@cs.wm.edu

Jie Chen and William Watson III
The High Performance Computing Group
Jefferson Lab
Newport News, VA 23606, USA
{chen,watson}@jlab.org

Abstract

Processor allocation in a mesh or torus connected multicomputer system with up to three dimensions is a hard problem that has received some research attention in the past decade. With the recent deployment of multicomputer systems with a torus topology of dimensions higher than three, which are used to solve complex problems arising in scientific computing, it becomes imminent to study the problem of allocating processors of the configuration of a torus in a multi-dimensional torus connected system. In this paper, we first define the concept of a semitorus. We present two partition schemes, the Equal Partition (EP) and the Non-Equal Partition (NEP), that partition a multi-dimensional semitorus into a set of sub-semitori. We then propose two processor allocation algorithms based on these partition schemes. We evaluate our algorithms by incorporating them in commonly used FCFS and backfilling scheduling policies and conducting simulation using workload traces from the Parallel Workloads Archive. Specifically, our simulation experiments compare four algorithm combinations, FCFS/EP, FCFS/NEP, backfilling/EP, and backfilling/NEP, for two existing multi-dimensional torus connected systems. The simulation results show that our algorithms (especially the backfilling/NEP combination) are capable of producing schedules with system utilization and mean job bounded slowdowns comparable to those in a fully connected multicomputer.

1. Introduction

A tightly coupled multicomputer system consists of a collection of processors (nodes) which are connected by communication links (edges) in a topology such as a mesh (without wrap-around edges) and a torus (with wrap-around edges). As a comparison, a flat multicomputer system (flat machine) contains processors connected in a topology of a complete graph, where there is a link between any two processors. This fully connected model is not the focus of discussion in this paper but will be used as a benchmark model in our simulation experiments later on.

A typical job requires a certain number of processors for its execution. In some applications, a job even specifies the configuration of the processors it requires, e.g., a submesh or a subtorus. A job is associated with an arrival time and an (estimated) execution time. As jobs arrive, they are put into a wait queue. Upon certain events such as job completion and job arrival, a job scheduling algorithm chooses a job from the wait queue to execute and a processor allocation algorithm allocates the processors required by the job from the current pool of idle processors. The goal for both algorithms is to maximize the system utilization (or equivalently, to minimize the makespan of the schedule) or to minimize the mean job slowdown (or equivalently, to minimize the jobs' average weighted response time).

The job scheduling algorithm decides which job currently in the wait queue to choose to schedule. To play the fairness game, the job that arrives the earliest is usually chosen. This is the FCFS (First Come First Serve) algorithm. To increase the utilization (or reduce the makespan), the job with the longest execution time is usually chosen. This is the LPT (Longest Processing Time first) algorithm. To reduce the mean job slowdown (or response time), the job with the shortest execution time is usually chosen. This is the SJF (Shortest Job First) algorithm. Among these job scheduling algorithms, the FCFS algorithm is the most widely adopted algorithm in multicomputer scheduling. In addition, other algorithms have been proposed and adopted, such as the backfilling algorithm, which allows a later-arriving job in the wait queue to be chosen to schedule as long as its execution does not delay the earliest possible execution of the earliest-arriving job in the queue. See [8] for a summary on advances in parallel job scheduling research.

In a flat machine, processor allocation is a non-issue since to schedule a job only the required number of processors are needed as any set of processors are fully connected in a flat machine. However, processor allocation becomes

a challenging issue in a system of a mesh or torus topology, especially when the topology is a multi-dimensional torus and the allocated processors are required to be in a torus configuration as well. Research on processor allocation, starting in the early nineties, mostly is on topologies with low dimensions since some of the most popular multicomputers are configured into low-dimensional topologies, such as Symult 2010 (a 2-D mesh), Cray T3D (a 3-D torus), and BlueGene/L [11] (also a 3-D torus). For example, [5] and [12] study algorithms that allocate submeshes in a 2-D mesh, and [4] and [15] present algorithms that allocate submeshes in a 3-D torus. Other work related to processor allocation in 2-D or 3-D topologies includes issues of network contention [14], fault tolerance [2], and non-contiguous processor allocation [13]. In contrast to the active research on processor allocation in low-dimensional topologies, there is little done on high-dimensional topologies. The only reference we are aware of is [1], which studies processor allocation in a multi-dimensional torus with additional communication links added to improve system utilization.

Many applications in scientific computing require the use of a multi-dimensional torus. For example, to study the theory of the strong nuclear force, known as quantum chromodynamics (QCD), a 5-D torus connected multicomputer system ($2 \times 2 \times 2 \times 6 \times 8$) was deployed at the Jefferson Lab in 2004, and a multicomputer system containing multiple 6-D tori ($2\times2\times2\times4\times4\times8$) [3] is in construction at various sites such as the Brookhaven National Lab, Columbia University, and University of Edinburgh. The deployment of these high-dimensional systems makes it a pressing issue to study fundamental and new research problems associated with these complex systems, among which are processor allocation and job scheduling. We are thus motivated to study allocating processors in the configuration of a torus when the underlining multicomputer is initially configured as a multi-dimensional torus. We are also interested in the effect of processor allocation on the quality of schedules measured by system utilization and mean job slowdowns.

We organize this paper as follows. In Section 2, we describe our problem model and give an overview of our job scheduling and processor allocation algorithms. In Section 3, we discuss two partition schemes that are essential in our allocation algorithms, specifically, how a multi-dimensional torus with the number of processors required by a job can be carved out from the configuration of idle processors. In Section 4, we give our simulation results that show the effectiveness of our allocation algorithms. Finally, we summarize our results and discuss future research in Section 5.

2. Problem Model and Algorithm Overview

As pointed out earlier, our work on processor allocation is mainly motivated by scientific applications, such as the QCD calculations, which typically require to use a multi-dimensional torus connected multicomputer system. In this section, we describe our problem model followed by an overview of our job scheduling and processor allocation algorithms.

2.1. Problem Model

As our problem falls in the category of resource allocation and job scheduling, we therefore follow the convention to define our problem in three areas: system environment, job characteristics, and performance metrics.

In the system environment, identical processors (nodes) are connected in the configuration of a multi-dimensional torus (with wrap-around edges). Ideally, the size of each dimension (which is defined to be the number of nodes in a dimension) is desired to be a power of two. However, in practice, it is likely that there is only enough fund to purchase a certain number of processors that falls between two consecutive powers of two, as in the 384-node system currently running at the Jefferson Lab. We thus assume, without the loss of generality, that in our multi-dimensional torus the sizes of all-but-one dimensions are powers of two and there is a single dimension of an arbitrary size which can be written as a power of two multiplied by an odd number (including one).

Jobs are submitted by users over time. Each job is characterized by its arrival time, the (estimated) execution time, and the desired number of processors needed for the execution of the job. In some previous research such as [1], users also specify configurations of processors for their submitted jobs. In our problem model, which is inspired by QCD calculations, users typically do not care about the specific configurations of the requested processors although they usually prefer any toroidal configuration in a dimension no less than three. We thus only allow the users to specify the desired number of processors but not a particular configuration when submitting a job. To schedule a job, any subtorus with the requested number of processors will be allocated and assigned to the job. A reasonable effort will be made to allocate such a subtorus with a dimension no less than three, although this is not a strict requirement.

We use the standard performance metrics [7] of utilization and mean job slowdown to evaluation the quality of a schedule. Note that utilization is defined to be the percentage of time when processors are busy in duration of the schedule. If all processors are busy throughout the schedule with no idle processors, then the utilization is 100%. The mean job slowdown is one plus the ratio of the wait time of a job over the job's execution time. When a job is scheduled as soon as it is submitted, its slowdown is optimally 1 since it has a wait time of zero.

2.2. Job Scheduling Algorithms

We now discuss the job scheduling algorithms that will be used in conjunction with our processor allocation algorithms in the simulation experiments. They are the widely adopted FCFS and backfilling algorithms.

In FCFS, the wait queue is ordered by nondecreasing arrival times of the jobs that have arrived. Upon the events such as job completion and job arrival, the first job in the wait queue will be considered to schedule. If the job requests more processors than what can be allocated in a toroidal configuration, the job will stay in the front of the queue to wait for more jobs to be completed.

In backfilling, however, in the case that the first job in the wait queue fails to be scheduled due to the lack of enough processors, other jobs in the wait queue can be considered and scheduled as long as the execution of these jobs do not delay the first job in the queue. Here we say that the first job in the queue is delayed if it starts after its earliest possible start time, which is the time when there is just as many idle processors as requested by the job in a toroidal topology, which can be allocated to the job for its execution. Note that the backfilling we consider in this paper is indeed the aggressive backfilling as opposed to the conservative backfilling, where all jobs (not just the first job) in the wait queue are not allowed to be delayed.

2.3. Processor Allocation Algorithms

Allocating subtori in a multi-dimensional torus is a challenging problem that has not received much research attention. Here, we present an overview of our processor allocation algorithm design, which is also the main contribution of this work.

The first step is preprocessing, done only once and before any scheduling starts. Recall that the multicomputer system we assume in our problem model is a torus with all-but-one dimension sizes to be powers of two. That is, let the torus of dimension d be $2^{n_1} \times 2^{n_2} \times \cdots \times 2^{n_{d-1}} \times 2^{n_d} \cdot p$, where $p \geq 1$ is odd. When $p = 1$, all dimension sizes are powers of two. (Note that when we describe a torus in this paper, we do not care about the orientation of dimensions, e.g., $a \times b$ is the same as $b \times a$.) The preprocessing step partitions the original torus into a set of semitori with all dimension sizes to be powers of two, where a *semitorus* is similar to a torus except that some of the wrap-around edges may be missing. Note that any torus or mesh is a semitorus. For example, if the original torus is $2 \times 2 \times 2 \times 6 \times 8$, then the fourth dimension of size 6 can be written as $6 = 2 \cdot 3 = 2 \cdot (2 + 1)$, where the odd factor is expressed as a sum of powers of two based on the odd factor's binary representation. Therefore, this torus can be partitioned into two semitori, $2 \times 2 \times 2 \times 4 \times 8$ (with one wrap-around edge missing in the fourth dimension) and $2 \times 2 \times 2 \times 2 \times 8$. We name the set of obtained semitori as the initial available set.

When a job chosen by a job scheduling algorithm (FCFS or backfilling) is being considered, its requested number of processors will first be converted to the next nearest power of two since this condition usually makes processor allocation less tedious and users usually do not mind receiving more processors than they need. Assume the input contains (1) the number of processors $m = 2^k$ requested by the job being considered currently and (2) the set of currently available semitori, A, which is initialized by the preprocessing step and maintained dynamically throughout the schedule. Our allocation algorithms then follow the following three steps.

- Semitorus identification: Identify a semitorus $S \in A$ that has at least as many nodes as m and remove S from A. If no such semitorus exists, the job cannot be scheduled now. If such a semitorus S exists, let S be $2^{k_1} \times \cdots \times 2^{k_h}$. If $\sum_{i=1}^{h} k_i = k$, assign S to R and go to the torus conversion step. If $\sum_{i=1}^{h} k_i > k$, go to the semitorus partition step.
- Semitorus partition: (1) Partition scheme: Partition S into semitori, one of which has m nodes and (2) Assign the semitorus with m nodes to R and add the remaining semitori in the partition to A.
- Conversion to torus: Combine (collapse) all dimensions with missing wrap-around edges in R to get a torus T that will be allocated to the job.

Two partition schemes are used in the partition step, resulting in two processor allocation algorithms: allocating with equal partition and allocating with non-equal partition. We will discuss these schemes in details in the next section.

In the conversion step, a semitorus R with the desired number of nodes m is to be reconfigured into a torus with the same number of nodes by reducing the number of dimensions. One approach is to take all the dimensions in R with missing wrap-around edges and replace them with one dimension. For example, a semitorus $2^2 \times 2^3 \times 2 \times 2^2$ with the first two dimensions missing wrap-around edges can be converted to a torus $2^5 \times 2 \times 2^2$, where the $2^2 \times 2^3$ mesh formed in the first and second dimensions in the semitorus is made to be a ring of 2^5 nodes. This approach has the advantage to produce a torus with more dimensions. Another approach is to pair a dimension that misses a wrap-around edge with a dimension with no edge missing and replace them with one dimension. For example, the same semitorus $2^2 \times 2^3 \times 2 \times 2^2$ can be converted to a torus $2^4 \times 2^4$, where the first and fourth dimensions are paired and the second and third dimensions are paired. This approach has the advantage to produce a more cube-like torus. See Figure 1 for an example of the torus conversion step, in which a 2-D

semitorus with missing wrap-around edges in both dimensions is converted to a 1-D torus (ring).

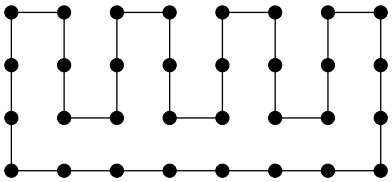

Figure 1: Conversion from a semitorus to a torus (a mesh to a ring)

Comparing to processor allocation, deallocation is relatively easier, which is done every time a job finishes execution. In the allocation algorithms, we need to keep a record of the chosen semitorus S in the identification step and its children (these children are siblings to each other) which are obtained in the partition step. When a job return its torus to the available set A, the deallocation algorithm checks to see if all its siblings are in A. If so all siblings together with the returned torus are replaced by their parent S, which is put back to A.

3. Partition Schemes

In the semitorus partition step, a semitorus S of $2^{k_1} \times \cdots \times 2^{k_h}$ is to be partitioned into a set of semitori, all with powers of two sizes in all dimensions and one with exactly $m = 2^k$ nodes, where m is the number of processors requested by the job being scheduled.

In our Equal Partition (EP) scheme, S is partitioned into $p = 2^{\sum_{i=1}^{h} k_i}/m$ semitori of identical topology (regardless of the orientation of dimensions). There are many ways to obtain an equal partition, however, we design the Equal Partition scheme with a goal to obtain high-dimensional semitori in the partition as much as possible since a job may prefer to receive a higher-dimensional torus and furthermore a higher-dimensional semitorus can easily be collapsed to a lower-dimensional one if necessary. Specifically, in the Equal Partition, if $k \leq h$, each semitorus in the partition is $2 \times \cdots \times 2$ with k dimensions. If $k > h$, each semitorus in the partition has h dimensions with all dimensions sizes to be as close as possible. For example, if $m = 2^3$ and S is $2 \times 2^2 \times 2^2 \times 2^3$, then each semitorus in the partition is $2 \times 2 \times 2$. If $m = 2^6$ and S is $2 \times 2^2 \times 2^2 \times 2^3$, then each semitorus in the partition is $2 \times 2 \times 2^2 \times 2^2$. See Figure 2(a) for an illustration of the Equal Partition scheme.

The Equal Partition scheme has an easy concept and is easy to implement. However, it has the tendency to create fragmentation. Each time, the Equal Partition creates a total of p semitori, only one of which is going to be allocated with the other $p - 1$ to be put in the available set for future use. In the unlikely extreme case when a job requires only $m = 1 = 2^0$ processor, the entire semitorus S will be disconnected into a set of single nodes, all of which may be not usable at all. In an optimistic case, however, when all jobs tend to request the same number of processors, the Equal Partition scheme may be the way to go since all identical semitori resulting from the Equal Partition may be allocated.

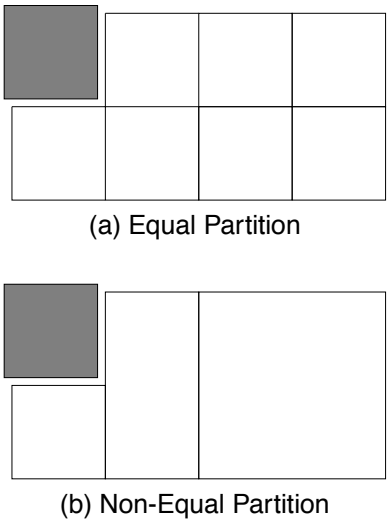

Figure 2: Two Partition Schemes for Processor Allocation

In our Non-Equal Partition (NEP) scheme, S is partitioned into semitori of various sizes to reduce fragmentation. The Non-Equal Partition fully utilizes the power-of-two property of the dimension sizes in the semitorus and creates a much smaller partition of $1 + \log p$ semitori. The implementation is more complex than that of the Equal Partition, so we will only use an example to explain how it creates a partition. Let S be $2 \times 2^2 \times 2^2 \times 2^3$ and $m = 2^4$. The Non-Equal Partition scheme creates a partition containing 5 semitori (since $p = 2^8/2^4 - 2^4$ and $1 + \log p = 5$):

1. $2 \times 2^2 \times 2$
2. $2 \times 2^2 \times 2$
3. $2 \times 2^2 \times 2^2$
4. $2 \times 2^2 \times 2^2 \times 2$
5. $2 \times 2^2 \times 2^2 \times 2^2$

To verify that the semitori given above do form a partition of S, we merge the first two semitori of $2 \times 2^2 \times 2$ in the partition to obtain $2 \times 2^2 \times 2^2$. This semitorus, which can also be expressed as $2 \times 2^2 \times 2^2 \times 2^0$, is then merged with the third semitorus to become $2 \times 2^2 \times 2^2 \times 2$. Merging this new semitorus with the fourth semitorus $2 \times 2^2 \times 2^2 \times 2$ in the partition, we get $2 \times 2^2 \times 2^2 \times 2^2$, which is then merged with the

fifth semitorus to get $2 \times 2^2 \times 2^2 \times 2^3$. We see that this semitorus is indeed S. See Figure 2(b) for an illustration of the Non-Equal Partition scheme.

4. Simulation Experiments

We use a simulation-based approach to evaluate the effectiveness of the proposed partition schemes coupled with different scheduling policies. An event-driven simulator was developed to process actual job logs from various supercomputing centers. We simulated a batch system in which arriving jobs are placed in a queue in the order of arrival. The processor allocation algorithms are invoked along with corresponding job scheduling algorithms upon every job arrival and job completion. We experimented with both FCFS and backfilling scheduling policies. The results of simulations for all four combinations of the aforementioned partition schemes and scheduling policies were then studied to determine the impact of their respective methods. In this section we first describe briefly two multi-dimensional torus multicomputers: one is called QCDOC with 1024 node units installed at the Brookhaven National Lab (BNL); the other is a Jefferson Lab (JLAB) 384-node LQCD Linux cluster constructed as a 5-D torus using Gigabit Ethernet. We then discuss our simulation environment, followed by an overview of the workload characteristics for two job logs we consider in the simulation. Finally we present the simulation results.

4.1. The QCDOC Machine and the JLAB LQCD Cluster

The QCDOC machine under construction utilizes a 6-D torus and computing nodes fabricated with IBM system-on-chip technology. It contains multiple crates, each of which is a $2 \times 2 \times 2 \times 4 \times 4 \times 8$ torus. The JLAB LQCD cluster in operation, on the other hand, utilizes commodity Linux based computing nodes connected via Gigabit Ethernet to form a 5-D torus of $2 \times 2 \times 2 \times 6 \times 8$. We use the above two tori to perform simulation of our processor allocation and job scheduling algorithms.

4.2. The Simulation Environment

The simulation environment models the above two tori. The 6-D torus has all dimension sizes being values of power of two. In contrary, the 5-D torus has all-but-one dimension sizes being values of power of two. The event-driven simulator receives as input a job log, the type of process allocation schemes (EP or NEP) and the type of job schedulers (FCFS or backfilling) to simulate. There are two primary events in the simulator: (1) an *arrival event* occurs when a job is first submitted for execution and placed in the wait queue; and (2) a *completion event* occurs upon the completion of a job, at which point the processors running the job is deallocated back to the system. The processor allocation and job scheduling algorithms are invoked at the occurrence of these events.

A job log contains information on the arrival times, execution times, and sizes (numbers of processors required) of all jobs. Given a torus of N nodes, a job log of J jobs, and for each job j, the arrival time t_j^a, execution time t_j^e, and size s_j, the simulation calculates the start time t_j^s and finish time t_j^f of each job. The following parameters are then calculated for each job:

1. Wait time $t_j^w = t_j^s - t_j^a$,
2. Response time $t_j^r = t_j^f - t_j^a$, and
3. Bounded slowdown $t_j^{bs} = \frac{max(t_j^r, \Delta)}{max(t_j^e, \Delta)}$, where Δ is 10 seconds to take care the cases when jobs have very short execution time which may distort the slowdown [9].

The system load can be characterized as

$$L = \lambda \sum_j \frac{s_j t_j^e}{JN},$$

where λ is the inter-arrival rate of jobs. The mean job bounded slowdown is defined as

$$S = \frac{1}{J} \sum_j t_j^{bs}.$$

In addition global system statistics are obtained. Especially, the system utilization is determined, which is defined as

$$U = \frac{1}{TN} \sum_j s_j t_j^e,$$

where the simulation time span is $T = max_{\forall_j}(t_j^f) - min_{\forall_j}(t_j^a)$.

4.3. Job Logs

We obtained two job logs from the *Parallel Workloads Archive* [10] [6]. The first log is from the NASA Ames's 128-node iPSC/860 collected during 1993 and contains jobs of sizes of powers of two only. The other log is from the San Diego Supercomputer Center's (SDSC) 128-node IBM RS/6000 SP collected from 1998 to 2000 and consists of some jobs of sizes that are not powers of two. In addition this log has larger variation in job sizes compared to the NASA log. We scaled all job sizes in the log files by a factor of either 8 or 2 (to fit our platforms of 1024 (QCDOC) and 384 (JLAB) nodes) and rounded up any size that is not a power of two to a nearest value of power of two. In addition

we generated logs of varying workloads by multiplying the execution time of each job by a coefficient c, varying c from 0.2 to 2.0 in increments of 0.05. Simulations were then performed for all combinations of processor allocation and job scheduling algorithms on each of the logs. In order to better understand the impact of our algorithms, the simulation using the same scheduling algorithms on the same job logs was performed for a hypothetical fully connected (*flat*) machine with the same number of nodes as the two tori provide. The bounded slowdown as a function of system utilization for both logs were then plotted.

4.4. Simulation Results

Figures 3 to 6 present plots of mean job bounded slowdown versus system utilization for each of the combinations of processor allocation and job scheduling algorithms, each of the two job logs, and each of the two torus connected systems, along with the results calculated for the flat machines. All four figures have similar shapes and the most significant performance improvement is obtained through the combination of the Non-Equal Partition and backfilling. We will examine results from each log for each torus separately.

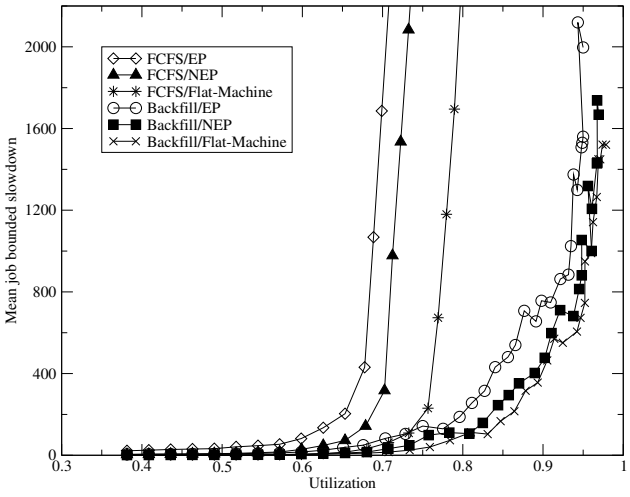

Figure 3: NASA iPSC/860 and QCDOC 2x2x2x4x4x8 torus

Figure 3 presents the results for the NASA log with the QCDOC machine. All four processor allocation and job scheduling combinations provide similar mean job bounded slowdown for utilization up to 50%. The FCFS scheduling strategy saturates at rather low utilization values. The Equal Partition allocation scheme produces the lowest saturation point at 65%, while the Non-Equal Partition allocation scheme delivers a higher saturation point at 70%. In contrast, the results from a flat machine with 1024 nodes has the highest saturation point at 77%. This trend is directly related to the reduction of the fragmentation effect from the Equal Partition allocation to the Non-Equal Partition allocation and to the flat machine that has no fragmentation effect. The backfilling strategy coupled with either processor allocation scheme, however, provides much higher saturation points of system utilization. Especially, the Non-Equal Partition scheme provides a high saturation point above 90% and low mean job slowdown values, which are similar to what the flat machine can provide.

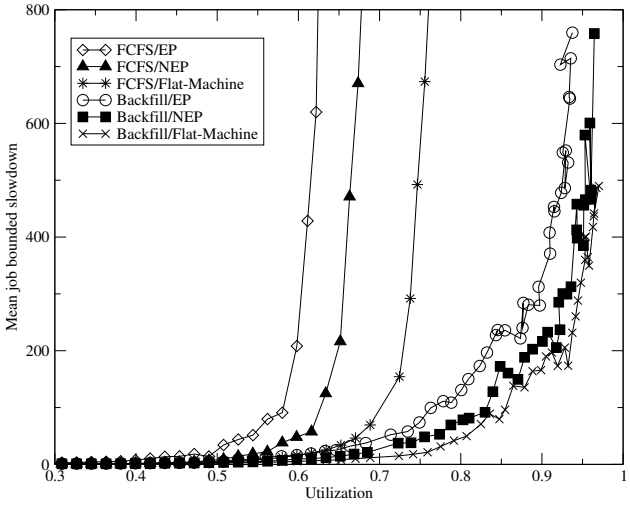

Figure 4: NASA iPSC/860 and JLAB 2x2x2x6x8 torus

Figure 4 shows the results for the NASA log with the JLAB LQCD cluster. According to our processor allocation algorithms, the initial available semitori set consists of two semitori of $2 \times 2 \times 2 \times 2 \times 8$ and $2 \times 2 \times 2 \times 4 \times 8$, resulting from the preprocessing partition of the original torus of $2 \times 2 \times 2 \times 6 \times 8$, which has one of the dimension sizes being not a power of two. Indeed the backfilling combined with the Non-Equal Partition allocation delivers very low job slowdown values until system utilization reaches 80%. Similar to the previous configuration, it provides the best performance that is on par with the performance of the 384-node flat machine using the backfilling scheduling algorithm.

Figure 5 presents the results for the SDSC log with the QCDOC machine. The FCFS combined with the Equal Partition scheme saturates at mere 45%. The FCFS combined with the Non-Equal Partition scheme reaches saturation at 55%. These values are smaller than those produced from the NASA log because the SDSC log has more varied job sizes leading to more fragmentation effect. Nonetheless, backfilling coupled with the Non-Equal Partition scheme once again proves to be the best combination delivering high system utilization with low job slowdown.

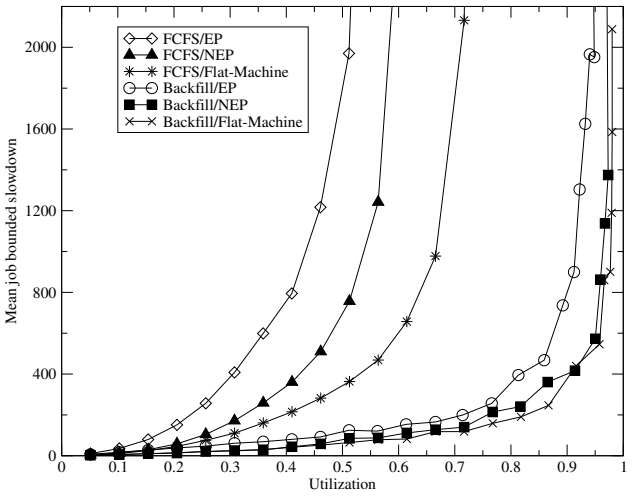

Figure 5: SDSC RS/6000 SP and QCDOC 2x2x2x4x4x8 torus

Finally, Figure 6 shows the results for the SDSC log with the JLAB cluster. Backfilling combined with the Non-Equal Partition scheme has high utilization saturation point near 95%. The FCFS scheduling saturates at relatively low 45% and 55% utilization for the Equal and Non-Equal Partitions, respectively, as expected.

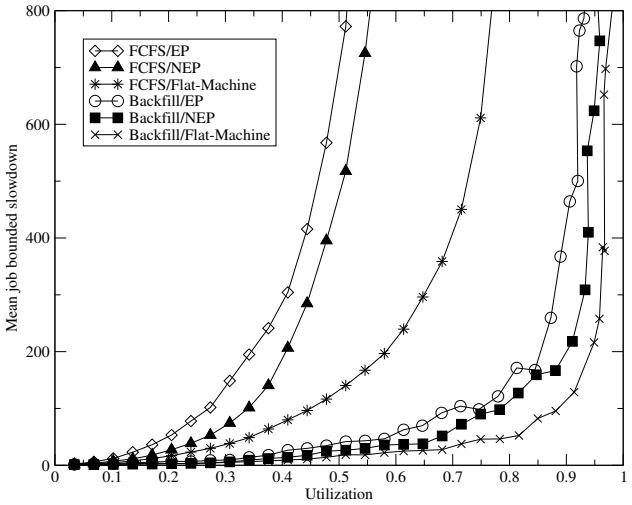

Figure 6: SDSC RS/6000 SP and JLAB 2x2x2x6x8 torus

To further illustrate the effectiveness of the combination of the Non-Equal Partition and backfilling, Figure 7 plots the system utilization against system load for the SDSC log with the QCDOC machine. The system utilization for FCFS with the Equal Partition saturates at 55% starting at a low system load value of 0.56. Similarly the system utilization for FCFS with the Non-Equal Partition flattens around 60% at a relatively low system load value of 0.62. In contrary, the system utilization for backfilling with the Equal Partition starts saturation near 85% at a relatively high system load value of 0.8 and the system utilization for backfilling with the Non-Equal Partition exhibits a little saturation beyond a system load value of 0.9.

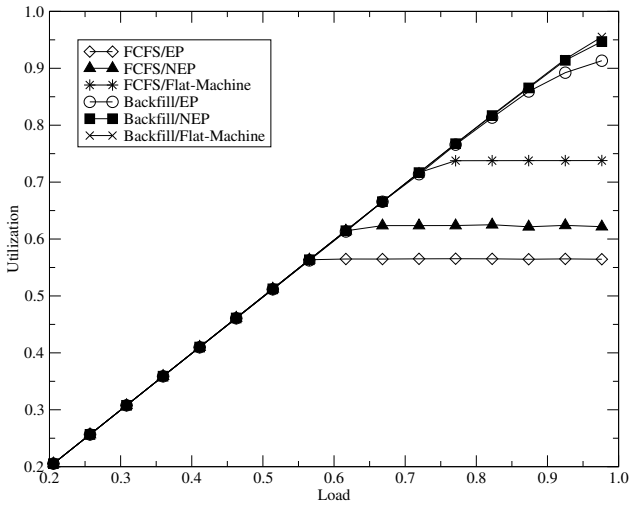

Figure 7: System utilization versus system load

Overall, the backfilling scheduling combined with the Non-Equal Partition allocation scheme is an effective way of improving quality of service by reducing the fragmentation effect and increasing system utilization. Our simulation shows that the system utilization achieved by this combination is approaching to the level delivered by the backfilling scheduling policy for fully connected machines. Meanwhile, the mean job slowdown values resulted from this combination are similar to the values offered by backfilling for flat machines. Specifically, the simulation results show that the system utilization on average is improved by 30% compared to FCFS by backfilling alone, and is increased by another 5% by the Non-Equal Partition compared to the Equal Partition.

5. Conclusions

Our work in this paper is motivated by applications in scientific computing where jobs that require parallel processors connected in a multi-dimensional torus are submitted to a multicomputer system to be executed. This multicomputer is a complex system of processors connected by communication links in the topology of a multi-dimensional torus. We first define the concept of a semitorus, whose topology is similar to that of a torus but may miss some wraparound edges. Under the reasonable assumptions that the original torus for the multicomputer has all-but-one dimension sizes to be powers of two and that each job requires

a power-of-two number of processors, we propose two partition schemes, the Equal Partition (EP) and the Non-Equal Partition (NEP), that partition a semitorus into a set of semitori and design two processor allocation algorithms based on these partition schemes. To evaluate our processor allocation algorithms, we incorporate them with two commonly used job scheduling algorithms, FCFS and backfilling, to obtain four combinations, FCFS/EP, FCFS/NEP, backfilling/EP, and backfilling/NEP. We conduct simulation experiments of these four algorithm combinations using two job logs from the Parallel Workloads Archive, the NASA iPSC/860 and the SDSC RS/6000 SP logs, on two currently running multicomputer systems, the QCDOC machine with 1024 nodes configured into a 6-D torus and the JLAB cluster with 384 nodes configured into a 5-D torus. We use two metrics to measure the quality of schedules generated, the system utilization and the mean job bounded slowdown. Our simulation results show that compared with using backfilling on a flat machine where processor allocation is a non-issue, our four algorithm combinations all produce schedules with acceptable quality measured by the system utilization and the mean job bounded slowdown. Furthermore, as expected, the backfilling/NEP combination consistently produce schedules with the best quality compared with the other three combinations. It even performs comparably with the backfilling algorithm on a flat machine.

Currently we are integrating our research results into production batch systems. For future research, we are interested in efficient processor allocation schemes satisfying job requests which require specific multi-dimensional toroidal configurations.

Acknowledgment

This work is supported in part by the Department of Energy, Contract DE-AC05-84ER40150.

References

[1] Y. Aridor, T. Domany, O. Goldshmidt, E. Shmueli, J. E. Moreira, and L. Stockmeyer, Multi-Toroidal Interconnects: Using Additional Communication Links to Improve Utilization of Parallel Computers, *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, 2004.

[2] M. M. Bae and B. Bose, Spare Processor Allocation for Fault Tolerance in Torus-Base Multicomputers, *Proceedings of the International Symposium on Fault-Tolerant Computer*, pp. 282-291, 1996.

[3] P. A. Boyle, D. Chen, N. H. Christ, M. Clark, S. D. Cohen, C. Cristian, Z. Dong, A. Gara, B. Joo, C. Jung, C. Kim, L. Levkova, X. Liao, G. Liu, R. D. Mawhinney, S. Ohta, K. Petrov, T. Wettig, and A. Yamaguchi, Overview of the QCDSP and QCDOC Computers, *IBM Journal on Resaerch and Development*, Vol. 49, No. 2/3, pp.351-365, 2005.

[4] H. Choo, S.-M. Yoo, and H. Y. Youn, Processor Scheduling and Allocation for 3D Torus Multicomputer Systems. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 11, No. 5, pp. 475-484, 2000.

[5] P.-J. Chuang and N.-F. Tzeng, Allocating Precise Submeshes in Mesh Connected Systems, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, No. 2, pp. 211-217, 1094.

[6] W. Cirne and F. Berman, A Comprehensive Model of the Supercomputer Workload, *Proceeding of the IEEE Annual Workshop on Workload Characterization*, pp. 140-148, 2001.

[7] D. G. Feitelson and L. Rudolph, Metrics and Benchmarking for Parallel Job Scheduling, *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 1-24, 1998.

[8] D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, Parallel Job Scheduling – A Status Report, *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 1-9, 2004.

[9] D. G. Feitelson and A. M. Weil, Utilization and Predictability in Scheduling the IBM SP2 with Backfilling, *Proceedings of the International Parallel Processing Symposium*, pp. 542-546, 1998.

[10] D. G. Feitelson, Parallel Workloads Archive, *http://www.cs.huji.ac.il/labs/parallel/workload/index.html*.

[11] E. Krevat, J. G. Castanos, and J. E. Moreira, Job Scheduling for the BlueGene/L System, *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 38-54, 2002.

[12] K. Li and K. Cheng, A Two-Dimensional Buddy System For Dynamic Resource Allocation in a Partitionable Mesh Connected System, *Journal of Parallel and Distributed Computing*, Vol. 12, No. 1, pp. 79-83, 1991.

[13] V. Lo, K. Windisch, W. Liu, and B. Nitzberg, Non-Contiguous Processor Allocation Algorithms for Mesh-Connected Multicomputers, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 8, No. 7, pp. 712-726, 1997.

[14] S. Q. Moore and L. M. Ni, The Effects of Network Contention on Processor Allocation Strategies, *Proceedings of the International Parallel Processing Symposium*, pp. 268-273, 1996.

[15] W. Qiao and L. M. Ni, Efficient Processor Allocation for 3D Tori, *Proceedings of the International Parallel Processing Symposium*, pp. 466-471, 1995.