Computer Science
Technical Report

Colorado State University

Job Scheduling for Torus Connected Networks

Sandeep K. S. Gupta and Pradip K. Srimani
Department of Computer Science
Colorado State University
Ft. Collins, CO 80523

January 17, 1997

Technical Report CS-97-103

Computer Science Department
Colorado State University
Fort Collins, CO 80523-1873

Phone: (970) 491-5792 Fax: (970) 491-2466
WWW: http://www.cs.colostate.edu

Job Scheduling for Torus Connected Networks

Sandeep K. S. Gupta and Pradip K. Srimani
Department of Computer Science
Colorado State University
Ft. Collins, CO 80523

January 17, 1997

Abstract

In this paper we investigate the problem of how to schedule n independent jobs on an $m \times m$ torus based network. We develop a model to to quantify the effect of contention for communication links on the dilation of job execution time when multiple jobs share communication links; we then design an efficient algorithm to schedule a set of n independent jobs with different torus size requirements on a given torus with an objective to minimize the total schedule length. We also develop a feasibility algorithm for preemptively scheduling a given set of jobs on a torus of given size with a given deadline. We provide analysis for both the algorithms.

1 Introduction

The mesh and torus networks have been recognized as versatile interconnection networks for massively parallel computing. Mesh/torus-like low-dimensional networks have recently received a lot of attention for their better scalability to larger networks, as opposed to more complex networks such as hypercubes [BP95]. Examples of machines with such topologies include the MasPar MP-1 [Mas], Intel Paragon, MIT J-Machine [DDF$^+$89], Tera HORIZON [TS88], Cray T3D [Cra93, Oed93], Polymorphic Torus [LM89], Fujitsu AP-1000, and iWarp [BCC$^+$88].

A torus is a mesh with wrap-around links. Although meshes and torii are generally regarded as close families, there are still some distinctions: (i) As opposed to a mesh, all nodes of a torus are topologically symmetric, (ii) a torus has a smaller (about half) diameter compared to that of an equal-size mesh, and (iii) although the ratio of the number of links in a torus to that in a mesh is close to one, the *bisection bandwidth*[1] of a torus is twice that of a mesh.

Several schemes have been proposed for processor allocation in mesh connected multiprocessor networks [LC91, Zhu92]. Some work has also been done for processor allocation in a partitionable torus connected multiprocessor [QN95]; however, this scheme allocates submeshes in a torus. In this paper, we propose a scheme for allocating subtorii in a torus network. The

[1] Bisection bandwidth is the minimum number of links across any hyper-plane that cuts a network in half.

motivation for doing this is that the algorithms which are designed for torus networks would run faster under this allocation strategy [dCVGG95].

Our scheme is targeted towards wormhole routed networks. The message propagation time in wormhole routed networks is insensitive to routing distance in the absence of contention for the links as long as the routing distance does not exceed a threshold. However, contention for communication links severely degrades the performance. It is thus important that communication link contention should be avoided or minimized. This has motivated design of many algorithms for wormhole routed systems which perform the required communication as sequence of contention free phases. Further in order to fully utilize the available communication bandwidth the number of communication phases are minimized by scheduling as many communications in a phase as possible. Such is also the case for algorithms designed for wormhole-routed torus networks [TG96, TLGP97]. However, programs based on such algorithms would not be able to get maximum benefit from the underlying torus network if the jobs are allocated on a submesh rather that a subtorii.

Our purpose in the present paper is to investigate job scheduling in 2D torus connected networks under different models. The problem of job scheduling on torus connected networks can be formulated as follows. We are given a set of n independent jobs $J = \{J_i : 1 \leq i \leq n\}$ and a torus of dimension m (i.e., $T_{m \times m}$. Each job $J_i = (d_i, t_i), 1 \leq i \leq n$ requires a torus of dimension d_i (i.e., a d_i-subtorus) for t_i units of time where $0 \leq d_i \leq m$ and t_i is a rational number, $t_i > 0$. The problem is to compute a schedule such that the finish time (the time when all jobs are finished) is minimized (we call this an *optimal schedule*). A schedule is called *preemptive* if a job may be preempted before completion and can resume at a later time, possibly on a different subtorus. We also assume, for the sake of simplicity (without any loss of rigor) that the jobs are ordered, i.e., $\forall i, 1 \leq i \leq n, d_i \geq d_{i+1}$.

We present job scheduling algorithms under two different models: with and without contention. The scheduling with contention uses a *contention model* to determine the dilation in job execution time in the presence of contention for communication links. On the other hand, scheduling without contention must use some link-disjoint decomposition of the torus to eliminate contention from other jobs in the system. However, link-disjoint decomposition precludes the use of all the processors in the system.

2 Job Scheduling on Torus with Contention

2.1 Partitioning a Torus

A torus is a mesh with wraparound links. Formally, a $N_1 \times N_2$ torus $T_{N_1 \times N_2} = (V, E)$, where the vertex set $V = \{(i, j) | 0 \leq i < N_1, 0 \leq j < N_2\}$ and the edge set $E = \{((i_1, j_1), (i_2, j_2)) | (i_2 = (i_1 + 1) mod N_1 \wedge j_1 = j_2) \vee (j_2 = (j_1 + 1) mod N_2 \wedge i_1 = i_2)\}$. We assume that each edge represents a bidirectional communication link between its end nodes.

Consider the following subtorus of $T_{N_1 \times N_2}$: $T_{N_1/K_1, N_2/K_2} = (V_1, E_1)$, where $V_1 = \{(i, j) | (i, j) \in V \wedge i \; mod \; K_1 = a, 0 \leq a < K_1, j \; mod \; K_2 = b, 0 \leq b < K_2\}$ for some constants a and b and the edge set E_1 is defined similar to E. To distinguish

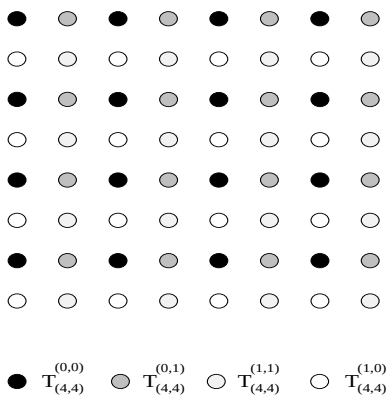

Figure 1: Partitioning of $T_{8,8}$ into four 4×4 Subtorii (Links are not shown for clarity).

this subtorus from other subtorus we would refer it as $T^{(a,b)}_{N_1/K_1,N_2/K_2}$. Notice that each edge in E_1 spans multiple edges of E. In particular, an edge $((i_1, j_1), (i_2, j_2)$ spans the edges in the set $\{((i_1, j_1), (i_1 + 1, j_2)), \ldots, ((i_2 - 1, j_1), (i_2, j_2))\}$ if $j_1 = j_2$, otherwise it spans the edges in the set $\{((i_1, j_1), (i_2, j_1 + 1)), \ldots, ((i_1, j_2 - 1), (i_2, j_2))\}$. We would refer the set of edges spanned by an edge e of a subtorus as $span(e)$.

Definition 1 *(Link-disjoint Subtorii) We say that two subtorii* $T^{(a_1,b_1)}_{N_1/K_1,N_2/K_2} = (V_1, E_1)$ *and* $T^{(a_2,b_2)}_{N_1/K_1,N_2/K_2} = (V_2, E_2)$ *are link-disjoint iff* $span(e_1) \cap span(e_2) = \phi$ *for all* $e_1 \in E_1$ *and* $e_2 \in E_2$.

Remark 1 *It is easy to see that any two subtorus* $T^{(a_1,b_1)}_{N_1/K_1,N_2/K_2}$ *and* $T^{(a_2,b_2)}_{N_1/K_1,N_2/K_2}$ *are link-disjoint iff* $a_1 \neq a_2$ *and* $b_1 \neq b_2$.

For a given torus, we can obtain $K_1 \times K_2$ distinct subtorii. Out of these $K_1 \times K_2$ distinct subtorii we are interested in a set of pairwise link-disjoint subtorii. A maximal set of pairwise link-disjoint subtorii has $min(K_1, K_2)$ subtorii. Further, we can partition the $K_1 \times K_2$ subtorii into $max(K_1, K_2)$ maximally link-disjoint sets.

Example: Consider $T_{8\times8}$ with $K_1 = K_2 = 2$. Then the the four subtorii of $T_{8\times8}$, namely, $T^{(0,0)}_{4\times4}$, $T^{(0,1)}_{4\times4}$, $T^{(1,0)}_{4\times4}$, $T^{(1,1)}_{4\times4}$ are shown in Fig.1. We can see that $T^{(0,0)}_{4\times4}$ and $T^{(1,1)}_{4\times4}$ are link-disjoint and so are $T^{(1,0)}_{4\times4}$ and $T^{(0,1)}_{4\times4}$. However, $T^{(0,1)}_{4\times4}$ and $T^{(1,1)}_{4\times4}$ are not link-disjoint. Hence, the set of 4×4 subtorii can be partitioned into two pair-wise link disjoint subsets with cardinality of each set to be two.

2.2 Model of Contention

In wormhole routed systems communication contention degrades the performance of the jobs. In this section we present a model of contention which we will use to dialate the finish time of the jobs that are allocated to subtorii sharing some communication links and which overlap in time.

Definition 2 *(Contention Model) If a job* J_k *of duration* t_k *is allocated to a subtorus which has a communication link which is shared by* $(n - 1)$ *other subtorii for a time duration of* Δ *then the duration of job* J_k *is dilated by an additive factor of*

$(n-1)\Delta/m$, where m is the maximum number of subtorii which can share a communication link. Hence, under this model job J_k will occupy the subtorus to which it is allocated for a duration of $t_k + (n-1)\Delta/m$.

Definition 3 *(Job Load) The time duration by which a job gets dilated is called the load of that job.*

Example: Suppose jobs J_1 and J_2 are allocated to subtorii $T_{4,4}^{(0,0)}$ and $T_{4,4}^{(0,1)}$ of torus $T_{8\times 8}$ and they overlap in time then these jobs will compete for the row links. According to the contention model the job duration of these jobs would become $t_1 + \Delta/2$ and $t_2 + \Delta/2$, respectively, assuming that the duration of overlap is Δ.

The contention model can be justified as follows. Suppose that a job requires Δ communication time without any contention. Hence,

$$\Delta = V/B,$$

where V is the data volume communicated and B is the bandwidth of each link. Then, in the presence of contention from $(k-1)$ other jobs the communication time can be approximated to be:

$$\Delta^{'} = V/(B/k) = k(V/B) = k\Delta,$$

assuming that each job gets equal share of the bandwidth. Hence, the job duration would be dilated by

$$\Delta^{'} - \Delta = (k-1)\Delta.$$

We dilate the duration of each job involved with this amount[2]. We further normalize this dilation by m, the maximum number of jobs which can compete for a communication link.

2.3 Load Update

In the following, we would be using this model to update the time duration of all the competing jobs. The jobs will be allocated to subtorii one by one. Placement of a job J_k on a free subtorii T increases the load of all the jobs allocated to subtorii sharing communication links with T. The loads of these jobs needs to be updates. Further, the current load of job J_k needs to be computed. The load of job J_k and all the competing jobs are updated according to the following theorem.

Theorem 1 *Let job J_k overlap with job J_i for duration Δ_i, $1 \leq i < k$ when it becomes active. Assuming that all the k jobs compete for same communication link, the load l_i of job J_i, $1 \leq i < k$ needs to be incremented by Δ_i/m, i.e,*

$$l_i = l_i + \Delta_i/m.$$

Further, the current load of job J_k is $l_k = \sum_{i=1}^{k-1} \Delta_i/m$, and the dilated finish time t'_k of J_k is:

$$t'_k = t_k + l_k.$$

[2]We are assuming that all the jobs communicate equal amount volume of data in same time duration. Further note that, for simplicity, we are assuming that the jobs are communicating the entire duration they occupy a subtorus.

Proof : ssume that job J_i was competing with n other jobs for a communication link during duration Δ_i before job J_k was introduced. With the presence of job J_k, job J_i now competes with $n+1$ other jobs. The difference between adjusted duration of job J_i after job J_k was introduced and before it was introduced is:

$$(t_i + (n+1)\Delta_i/m) - (t_i + n\Delta_i/m) = \Delta_i/m.$$

As opposed to uniformly competing with n jobs during the duration Δ_i, it can so happen that job J_i competes with different number of jobs. For example, suppose job J_1 competes with n_1 jobs during duration $\Delta_{i,1}$ and n_2 jobs during duration $\Delta_{i,2}$, where $\Delta_i = \Delta_{i,1} + \Delta_{i,2}$. So, the difference between the adjusted duration for job J_i after and before introduction of job J_k is:

$$(t_i + (n_1+1)\Delta_{i,1}/m + (n_2+1)\Delta_{i,2}/m) - (t_i + n_1\Delta_{i,1}/m + n_2\Delta_{i,2}/m) = \Delta_{i,1}/m + \Delta_{i,2}/m = \Delta_i/m.$$

This can be easily generalized. Hence, each job J_i's load has to be incremented by Δ_i/m.

Now consider job J_k. Note that if two jobs contend for a communication link their duration is dilated by equal amount of time. Hence, it can be easily concluded that t_k's time duration should be dilated by $\sum_{i=1}^{k-1} \Delta_i/m$. □

2.4 Job Scheduling with Contention

In this section, we will look at non-preemptive scheduling. Further we assume that all the requests are for square subtorus and that the dimension are power of two.

Problem: We are given a set of n independent jobs $J = \{J_i : 1 \leq i \leq n\}$ and a torus of dimension $m = 2^p$ (i.e., $T_{m \times m}$. Each job $J_i = (d_i, t_i), 1 \leq i \leq n$ requires a torus of dimension $d_i = 2^{s_i}$ for t_i units of time where $0 \leq d_i \leq m$ and t_i is a rational number, $t_i > 0$. The problem is to compute a schedule such that the finish time (the time when all jobs are finished) is minimized in presence of link contention.

Our algorithm attempts to minimize the overall contention (under the contention-model presented before) experienced by the jobs. It is based on the following greedy strategy: *A job is scheduled on an earliest available subtorus with minimum load*. The algorithm is given in Fig.2.

2.4.1 Availability Matrix

The algorithm uses *Availability Matrix* A to keep track of the availability of each subtorii. Recall that $T_{2^p,2^p}$ torus can be partitioned into $2^{p-s} * 2^{p-s}$ of size $2^s \times 2^s$. We will use the notation σ_s to denote 2^{p-s}. Hence, if all the job requests are of same size, say $2^s \times 2^s$ then we need an A of size $\sigma_s \times \sigma_s$, since entry $A[i,j]$ can track of the availability of subtorus $T^{(a,b)}_{2^s,2^s}$, $0 \leq a, b < \sigma_s$. In particular, if $s = p$ then we need an A with just one entry and if $s = 0$ (i.e. each request is for single processor) then we need an A of size $2^p \times 2^p$. Since job requests can be for subtorii of different dimensions we

Algorithm CNP_Alloc
Input:
J: job list $\{J_1, \ldots, J_n\}$, where $J_i = (2^{s_i}, t_i)$, $1 \leq i \leq n$.
Output:
S: Job schedule with job J_k start time to be $S[k].start$
on subtorus $T_{d_k,d_k}^{(S[k].row,S[k].col)}$.
$make\_span$: Schedule length.
begin
1) $A[i,j].tm = 0$, $A[i,j].sz = s_1$, $A[i,j].id = 0$, $0 \leq i,j < m$.
2) $time = 0$, $s_0 = s_1$.
3) $done(i,j) = false, 1 \leq i,j \leq n$.
4) FOR $k = 1, n$ /* Schedule job J_k */
5) If ($s_k \neq s_{k-1}$) /* Expand A to next level */
6) FOR $i = 0, \sigma_{s_{k-1}}$
7) FOR $j = 0, \sigma_{s_{k-1}}$
8) $A[i + \sigma_{s_{k-1}}, j]$ =$A[i, j + \sigma_{s_{k-1}}]$ = $A[i + \sigma_{s_{k-1}}, j + \sigma_{s_{k-1}}]$ = $A[i,j]$
9) ENDFOR ENDFOR
10) Endif
11) **Determine Free subtorii set B:**
12) $min = min(A[0 : \sigma_{s_k-1}, 0 : \sigma_{s_k-1}])$.
13) $A[0 : \sigma_{s_k-1}, 0 : \sigma_{s_k-1}] = A[0 : \sigma_{s_k-1}, 0 : \sigma_{s_k-1}] - min$.
14) $time+ = min$.
15) $B = \{(i,j) | A[i,j] = 0\}$.
16) **Determine minimum loaded subtorii from B:**
17) $((i,j), load) = min\_loaded(B)$.
18) **Assign J_k to (i,j).**
19) $(S[k].start, S[k].row, S[k].col) = (time, i, j)$.
20) **Update Availability matrix:**
21) $A[i,j].tm = t_k + load$.
22) $A[i,j].sz = s_k$, $A[i,j].id = k$.
23) FOR $r = 0, \sigma_{s_k} - 1$.
24) $s = A[i,r].sz$,$i' = i \bmod \sigma_s$, $r' = r \bmod \sigma_s$.
25) IF($\neg done(A[i',r'].id, k)$)
26) $A[i',r'].tm = A[i',r'].tm + min(t_k, A[i',r'].tm) * 1/\sigma s_k$.
27) $A[i' + x * \sigma_s, r' + y * \sigma_s] = A[i,r].tm, 0 \leq x,y < 2^{s-s_k}$.
28) $done(A[i',r'].id, k) = true$
29) ENDIF
30) ENDFOR
31) FOR $c = 0, \sigma_{s_k} - 1$.
32) ....
33) ENDFOR
34) ENDFOR
35) **Compute Schedule Length:**
36) $make\_span = time + max(A[0 : \sigma_{s_n} - 1, 0 : \sigma_{s_n} - 1])$.
end

Figure 2: Algorithm for Scheduling Subtorii with Contention.

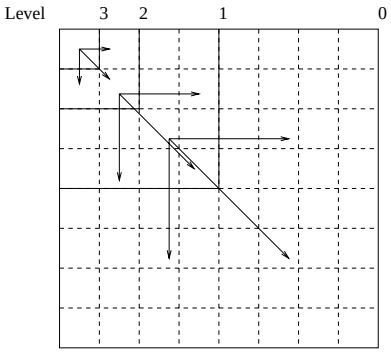

Figure 3: Availability Matrix Structure for $T_{8,8}$.

need an availability matrix of maximum possible size, i.e. $\sigma_{s_{min}} \times \sigma_{s_{min}}$, where $s_{min} = min(s_1, \ldots, s_n)$. We will call the availability matrix for requests for subtorii of size $2^s \times 2^s$ as the *availability matrix at level s*: Besides, the availability of subtorii the availability matrix is also used to keep track about the size of the job and the id of the job assigned to a subtorii:

Definition 4 *The Availability Matrix A at Level s, $0 \leq s \leq p$, is a $\sigma_s \times \sigma_s$ matrix. Each entry $A[i,j]$, $0 \leq i,j < \sigma_s$ of matrix A has three fields:*

- *$A[i,j].tm$: is the time when the subtorii $T^{(i,j)}_{2^s,2^s}$ is available.*
- *$A[i,j].sz$: is the size of the job active on this subtorii.*
- *$A[i,j].id$: is the id of the job active on this subtorii.*

Remark 2 *Note that $A[i,j].sz \geq s$. If $t = A[i,j].sz > s$ then job $J_{A[i,j].id}$ which was assigned to $T^{(i \bmod \sigma_t, j \bmod \sigma_t)}_{2^t,2^t}$ is also occupying $T^{(i,j)}_{2^s,2^s}$ since $T^{(i,j)}_{2^s,2^s}$ is a subtorus of $T^{(i \bmod \sigma_t, j \bmod \sigma_t)}_{2^t,2^t}$.*

If the job requests are ordered in the decreasing order of their dimensions then we need to maintain an availability matrix at a fixed level as long as the job request dimension remains the same. And whenever the job requests size decreases from $2^s \times 2^s$ to $2^t \times 2^t$ we need to expand the availability matrix from level s to level t. Note that the availability matrix at level t is $2^{s-t} * 2^{s-t}$ times the availability matrix at level s. If $t = s-1$ then the availability matrix doubles in each dimension. Further note that each subtorus $T^{(a,b)}_{2^s,2^s}$, $s \geq 1$, of dimension $2^s \times 2^s$ consists of four subtorii $T^{(a,b)}_{2^{s-1},2^{s-1}}$, $T^{(a+\sigma_s,b)}_{2^{s-1},2^{s-1}}$, $T^{(a,b+\sigma_s)}_{2^{s-1},2^{s-1}}$, and $T^{(a+\sigma_s,b+\sigma_s)}_{2^{s-1},2^{s-1}}$ each of dimension $2^{s-1} \times 2^{s-1}$. Hence, when expanding the availability matrix from level s to level $s-1$ each entry $A[a,b]$ is replicated to $A[a+\sigma_s, b]$,$A[a, b+\sigma_s]$, and $A[a+\sigma_s, b+\sigma_s]$. That is to say, subtorii $T^{(a,b)}_{2^{s-1},2^{s-1}}$, $T^{(a+\sigma_s,b)}_{2^{s-1},2^{s-1}}$, $T^{(a,b+\sigma_s)}_{2^{s-1},2^{s-1}}$, and $T^{(a+\sigma_s,b+\sigma_s)}_{2^{s-1},2^{s-1}}$ are (not) available only if $T^{(a,b)}_{2^s,2^s}$ is (not) available. Figure 3 illustrates the structure of the availability matrix for $T_{8\times 8}$. It also shows the pattern of expansion from each level to next lower level.

Hence, for simplicity, we make the following assumptions:

1. Jobs are sorted in non-increasing order of their dimensions, i.e. $s_1 \geq s_2 \geq \ldots \geq s_n$.

2. A job request for each s, $s_1 \geq s \geq s_n$, is present in the job list[3]. This ensures that we need to expand the availability matrix by at most one level for each job.

In Fig.2, lines 5-9 expand the matrix A to next lower level whenever the size of the subtorii requested by the current job is not the same as the size of the subtorii requested by the previous job.

2.4.2 Determining Earliest Available Minimum Loaded Subtorus

When scheduling J_k, it is assigned to a minimum loaded subtorus from among the earliest available subtorii.

Definition 5 *(Load on a Subtorus) The load on a subtorus T (with respect to job J) is the load job J would experience if it were assigned to T.*

Once the availability matrix of the right level has been obtained, the next thing the algorithm does is that it determines the set of free subtorii B. The variable $time$ is used to keep track of the time when there will be a free subtorii of the desired size. The updation of $time$ and determination of set B is done by first computing the minimum min of the $A[i,j].tm$, $0 \leq i,j < \sigma_{s_k}$, where s_k is the size of subtorii requested by the current job J_k. Then, min is added to $time$ and decremented from all $A[i,j].tm$, $0 \leq i,j < \sigma_{s_k}$. The set B is determined to $\{(i,j)|A[i,j].tm = 0\}$.

Remark 3 *If $(a,b) \in B$ then $T^{(a,b)}_{2^{s_k},2^{s_k}}$ is available at time $time$.*

Job J_k is allocated to the minimum loaded subtorii in set B. The load of each subtorus $T^{(i,j)}_{2^{s_k},2^{s_k}}$ is computed as follows:

1) load = 0
2) For $r = 0, \sigma_{s_k - 1}$
3) $\quad load = load + min(t_k, A[r,j].tm)$
4) Endfor
5) For $c = 0, \sigma_{s_k - 1}$
6) $\quad load = load + min(t_k, A[i,c].tm)$
7) Endfor
8) $load = load/\sigma_{s_k}$.

Recall that $T^{(i,j)}_{2^{s_k},s_k}$ and $T^{(r,c)}_{2^{s_k},s_k}$ are not link-disjoint iff $i = r$ or $j = c$. Hence, a job on $T^{(i,j)}_{2^{s_k},s_k}$ competes for row links with all the jobs assigned to subtorii $T^{(i,c)}_{2^{s_k},s_k}$, $0 \leq c < \sigma_{s_k} \wedge c \neq j$. Similarly, a job on $T^{(i,j)}_{2^{s_k},s_k}$ competes for column links with all the jobs assigned to subtorii $T^{(r,j)}_{2^{s_k},s_k}$, $0 \leq r < \sigma_{s_k} \wedge r \neq i$. In the above code, $min(t_k, A[r,j].tm)$ gives the duration of overlap between job J_k (if it were to be assigned to $T^{(i,j)}_{2^{s_k},s_k}$) and the job assigned to $T^{(r,j)}_{2^{s_k},s_k}$. The r-loop (c-loop) does not check for $r = i$ ($c = j$) since $A[i,j].tm = 0$. Line 8 normalizes the computed load according to our contention model.

[3]If some s is missing in the original job list then introduce a dummy job $(2^s, 0)$ in the job list.

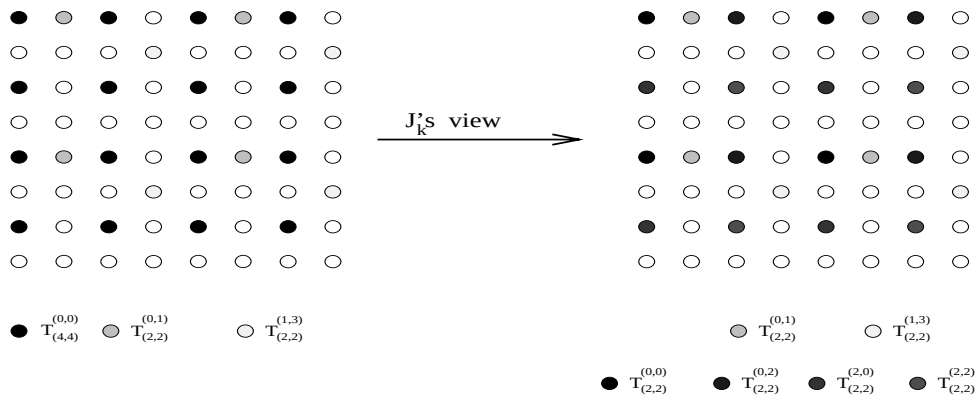

Figure 4: Load computation for interfering jobs on $T_{4,4}^{(0,0)}$ and $T_{2,2}^{(0,1)}$.

In Fig.2, the function call $min\_loaded(B)$ returns the entry (i, j) of B for which the $load$ computed was minimum. Job J_k is assigned to torus $T_{s_k,s_k}^{(i,j)}$ and the schedule is accordingly updated. The algorithm then adjusts the loads of all the jobs which get affected by J_k and accordingly updates the availability matrix.

2.4.3 Updating Availability Matrix

A job J_k assigned to subtorus $T_{2^{s_k},2^{s_k}}^{(i,j)}$ competes with all jobs (overlapping in time with J_k) assigned to subtorii $T_{2^{s_k},2^{s_k}}^{(r,j)}$, $r \neq i$, for the column links and all jobs (overlapping in time with J_k) assigned to subtorii $T_{2^{s_k},2^{s_k}}^{(i,c)}$, $c \neq j$, for row links. Let $\Delta_r^{(row)}$ be the overlap duration between job J_k and the job assigned to $T_{2^{s_k},2^{s_k}}^{(r,j)}$ and $\Delta_c^{(col)}$ be the overlap duration between job J_k and the job assigned to $T_{2^{s_k},2^{s_k}}^{(i,c)}$. Then according to Theorem 1, the time of job J_k should be dilated by $load = \sum_{i \neq r} \Delta_r^{(row)} + \sum_{c \neq j} \Delta_c^{(col)}$. This is exactly the value of $load$ returned by $min_loaded()$. Hence, the time of J_k is dilated by $load$ and $A[i, j].tm$ is assigned $t_k + load$.

At any given time jobs on different dimension subtorii can be active simultaneously. In our algorithm a job on subtorus $T_{2^s \times 2^s}^{(a,b)}$ is treated as 4 jobs (with same finish time) of dimension $2^{s-1} \times 2^{s-1}$ on subtorii $T_{2^{s-1} \times 2^{s-1}}^{(a,b)}$, $T_{2^{s-1} \times 2^{s-1}}^{(a+\sigma_s,b)}$, $T_{2^{s-1} \times 2^{s-1}}^{(a,b+\sigma_s)}$, and $T_{2^{s-1} \times 2^{s-1}}^{(a+\sigma_s,b+\sigma_s)}$. In general a job on $T_{2^s \times 2^s}^{(a,b)}$ is treated as 4^{s-t} jobs of dimension $2^t \times 2^t$ on the 4^{s-t} subtorii $T_{2^t \times 2^t}^{(a+i*\sigma_s,b+j*\sigma_s)}$, $0 \leq i, j < 2^{s-t}$.

One of the consequence of this is that a job on smaller dimension subtorus is penalize more than a job on larger dimension subtorus. This is justifiable since the larger job has more processors active and so can use up larger bandwidth (in proportion to its relative size).

Example: Consider $T_{8\times 8}$ and let job J_l on $T_{4,4}^{(0,0)}$ overlaps with job J_k on $T_{2,2}^{(0,1)}$ for Δ time duration (see Fig.4). Then, t_l will be dilated by $\Delta/4$ but t_k will be dilated by $2\Delta/4 = \Delta/2$ since job J_l on $T_{4,4}^{(0,0)}$ is treated as four sub-jobs on $T_{2,2}^{(0,0)}$, $T_{2,2}^{(0,2)}$, $T_{2,2}^{(2,0)}$, and $T_{2,2}^{(2,2)}$. Out of these four sub-jobs, the sub-jobs on $T_{2,2}^{(0,0)}$ and $T_{2,2}^{(0,2)}$ overlap with J_k for Δ duration. So, when computing the dilation for t_k, $\Delta/4$ is counted twice. However, when computing the dilation of J_l, $\Delta/4$ is counted only once.

In order to keep the entries for subtorii corresponding to sub-jobs of a job J_l consistent in availability matrix, once the dilation of J_l is computed it is propagated to all the relevant entries in the availability matrix. In Fig.2, lines 23-30 propagate the load updation for each job J_l which interferes with J_k. The flag $done(l,k)$ is used to ensure that propagation for job J_l is done only once for each J_k. No such propagation needs to be done for job J_k when it is being scheduled, since it cannot interfere with a smaller dimension job as the jobs are scheduled in non-increasing order of their dimension.

2.4.4 An Example

Consider $T_{8\times 8}$ and a job set $J = \{J_1, J_2, J_3, J_4, J_5, J_6\}$, where $J_1 = (8,2)$, $J_2 = (4,2)$, $J_3 = (4,4)$, $J_4 = (4,4)$, $J_5 = (4,1)$, and $J_6 = (2,4)$. Note that the jobs are arranged in non increasing order of required torus size.

Initially, the availability matrix has a single element $A[0,0] = (0,0,0)$ [4]. Job J_1 is scheduled to start at $time = 0$ on the entire torus. The availability matrix is updated to $A[0,0] = (2,8,1)$. Next J_2 is scheduled which requires a 4×4 torus. Since, $s_2 \neq s_1$ the availability matrix is expanded to size 2×2: $A[i,j] = (2,8,1), 0 \leq i,j < 2$. After computing minimum of $A.tm$ and adjusting $time$ to 2, all $A[i,j].tm$'s become 0 indicating that all the 4×4 subtorii are free at $time = 2$. Hence, J_2 is scheduled on $T_{4,4}^{(0,0)}$ and $A[0,0]$ is updated to be $(2,4,2)$.

In scheduling J_3, set B is determined to be $\{(0,1),(1,0),(1,1)\}$ with loads of respective subtorii to be 2, 2, and 0. Since, $T_{4,4}^{(1,1)}$ is minimum loaded J_3 is assigned to it. The availability matrix is updated to be [5]:

(2,4,2)	(0,*,*)
(0,*,*)	(4,4,3)

For $J_4 = (4,4)$, set B has two elements $\{(0,1)$ and $(1,0)\}$ each with $load = max(2,4)/2 + max(4,4)/2 = 3$. Assuming job J_4 is assigned to $(0,1)$, the availability matrix after load updation looks like:

(3,4,2)	(7,4,4)
(0,*,*)	(6,4,3)

Now, job J_5 can only be scheduled on $T_{4,4}^{(1,0)}$. The availability matrix after scheduling J_5 becomes:

(3.5,4,2)	(7,4,4)
(2,4,5)	(6.5,4,3)

Since $s_6 \neq s_5$ the availability matrix is expanded to:

[4] Each entry $A[i,j]$ is represented as a 3-tuple $(A[i,j].tm, A[i,j].sz, A[i,j].id)$.

[5] A * denotes a don't care value.

(3.5,4,2)	(7,4,4)	(3.5,4,2)	(7,4,4)
(2,4,5)	(6.5,4,3)	(2,4,5)	(6.5,4,3)
(3.5,4,2)	(7,4,4)	(3.5,4,2)	(7,4,4)
(2,4,5)	(6.5,4,3)	(2,4,5)	(6.5,4,3)

After computing minimum of $A.tm$ to be 2 and adjusting $time$ to $time + 2 = 4$ we get:

(1.5,4,2)	(5,4,4)	(1.5,4,2)	(5,4,4)
(0,*,*)	(4.5,4,3)	(0,*,*)	(4.5,4,3)
(1.5,4,2)	(5,4,4)	(1.5,4,2)	(5,4,4)
(0,*,*)	(4.5,4,3)	(0,*,*)	(4.5,4,3)

Set B is determined to be $\{(1,0),(1,2),(2,0),(2,2)\}$ with loads on each subtorii: $load = 2 * max(4,1.5)/4 + 2 * max(4,4.5)/4 = 2.75$. Assuming J_6 is assigned to $T_{2,2}^{(1,0)}$ we get:

(1.875,4,2)	(5,4,4)	(1.875,4,2)	(5,4,4)
(6.75,2,6)	(5.5,4,3)	(0,*,*)	(5.5,4,3)
(1.875,4,2)	(5,4,4)	(1.875,4,2)	(5,4,4)
(0,*,*)	(5.5,4,3)	(0,*,*)	(5.5,4,3)

Since, $max(A) = 6.75$ and $time = 4$, the schedule length is 10.75. If the dilation due to contention is ignored, then the schedule length would have been 10.

The schedule computed by algorithm is:

index	1	2	3	4	5	6
start	0	2	2	2	2	4
finish	2	5.875	8.5	9	4	10.75
(i,j)	(0,0)	(0,0)	(1,1)	(0,1)	(1,0)	(1,0)

2.4.5 Complexity of the Algorithm

Assuming that n jobs are being scheduled on $T_{m \times m}$, the time complexity of the algorithm is $O(n.log(n) + n.m^2)$. Since, $O(n.log(n))$ time is required to sort the jobs in non-increasing order of their dimension and for scheduling each job at most m^2 entries of the availability matrix are referenced/updated.

3 Job Scheduling on Torus Without Contention

In this section we consider contention free scheduling of jobs, i.e., when we decompose a given torus into subtorii we need to do so in a link disjoint fashion. We consider the problem of preemptive scheduling jobs on a given torus without contention with an objective to minimize the schedule length.

Each torus $T_{m\times m}$ contains 2 link-disjoint $T_{m/2\times m/2}$s. This decomposition can be done in the way already mentioned 2.1. In this section, we want to consider an alternative link disjoint decomposition of the torus for convenience of description. Let the rows and columns of a torus $T_{m\times m}$ of dimension m, be numbered from 0 to $m-1$. The processors are uniquely numbered from 0 through m^2-1; any processor $P(i,j)$i, that belongs to i-th row and j-th column is assigned a unique id $(i-1)*m+j-1$. In order to specify the subtorii within the given torus T_m, we need a new definition.

Definition 6 *A torus $T(a,b)$, (a sub torus of $T_{m\times m}$), where $0\le a,b\le m-1$ and $a\le b$, consists of the processors $P(i,j)$ such that $a\le i,j\le b$; note that the sub torus $T(a,b)$ has a size $b-a+1$.*

Thus, the given torus is denoted by $T(0,m-1)$ and it can be decomposed into two link disjoint sub torii $T(0,m/2-1)$ and $T(m/2,m-1)$. It is to be noted that in this link disjoint decomposition of the torus, exactly half of the processors in the original torus remain unused and secondly, in the sub torii $T(0,m/2-1)$ or $T(m/2,m-1)$, the distance between the processors is not exactly uniform; but assuming wormhole routing that does not affect routing time; wormhole routing is distance insensitive.

Definition 7 *For any a,b with $0\le a,b\le m$ $a<b$, let $[a,b]$ denote the set of processors belonging to the subtorus $T(a,b)$ (see Definition 6). We call $[a,b]$ a* processor interval *or a* p-interval. *We say this p-interval $[a,b]$ has a size $(b-a+1)$ (consisting of $(b-a+1)^2$ processors.*

Remark 4 *For any given $\ell,\ell>0$, an m-torus T_m can be divided into ℓ consecutive p-intervals $[a_1,b_1],\cdots[a_\ell,b_\ell]$ where $a_1=0, b_\ell=m-1$ and $(\forall i:1\le i<\ell:a_{i+1}=b_i+1)$.*

Remark 5 *Note that all p-intervals of size x^2 (x is a positive integer) are x-subtorus; in this paper we are interested only in those p-intervals which are valid link disjoint sub torii and hence we use the terms p-interval and subtorus interchangeably.*

Definition 8 *The* profile *[AZ90, ZA93] of a schedule is defined to be a function F that maps a processor $p\in V$ to a time $f=F(p)$ such that the processor p has been busy until time f and $T-f$ denotes the time when the processor p is available for more work.*

So, if T denotes the given deadline for the job set, $r=T-f$ denotes the *Remaining Processing Time* or the RPT of the processor p. If we attempt to find the schedule one job at a time, we need to know the finish time of all the processors for the

existing schedule and this information is stored in the profile. We use $S(i)$ to denote the schedule after job J_i is scheduled and use $P(i)$ to denote the corresponding profile.

When we schedule the jobs on a torus (each job needs a subtorus of some dimension), the profile function maps a p-interval (all the processors in the interval) to a time. Thus, the profile of the complete schedule on the torus is a sequence of ordered pairs of p-intervals and finish times

$$P = ([a_1, b_1], f_1), ([a_2, b_2], f_2), \cdots, ([a_y, b_y], f_y)$$

for some integer y where the y intervals divide the given m-torus in a link-disjoint fashion. Again, we logically extend the concept of RPT to the intervals; RPT of an interval is the RPT of its processors; more specifically, for a give deadline of the jobs, $r_j = T - f_j$ will denote the RPT of the p-interval $[a_j, b_j]$.

Remark 6 *If a p-interval has zero RPT in a schedule, it cannot be used for scheduling further jobs and will be deleted from the profile.*

Definition 9 *A profile P is called* stair-like *[ZA93] if* $\forall i : f_{i+1} < f_i$.

Preemptive Schedule – Feasibility Algorithm

Given a set of jobs $J = \{J_1, J_2, \cdots, J_n\}$, where $J_i = (d_i, t_i)$ as explained earlier and an m-torus, the feasibility algorithm computes if the given jobs can be scheduled on the m-torus, to meet a given deadline T. Obviously, if the given deadline T is feasible, we must have $\forall i : 1 \leq i \leq n : T \geq t_i$ and $T \geq \frac{1}{m^2} \sum_{i=1}^{n} t_i d_i^2$. We can safely assume that the given T satisfies both of these requirements or we can declare the deadline to be infeasible.

We assume that the job set J is sorted in descending order of dimensions of the subtorii needed, as explained earlier. We attempt to schedule the jobs in this order one at a time. Let $S(i)$ and $P(i)$ denote respectively the schedule and the profile after the job J_i is scheduled. $S(0)$ is the initial schedule (null) and $P(0)$ is the initial profile (before any job is scheduled). So, $P(0) = ([0, m-1], 0)$. We use k to denote the number of p-intervals with nonzero RPT in the profile $P(i-1)$. If $k = 0$, job J_i cannot be scheduled; otherwise $P(i-1)$ will look like

$$P(i-1) = ([a_1, b_1], f_1), ([a_2, b_2], f_2), \cdots, ([a_k, b_k], f_k)$$

[Note: if this profile is stair-like, the p-intervals in $P(i-1)$ are ordered in increasing order of their RPTs.]

The Algorithm to schedule $J_i = (d_i, t_i)$

Step 1: If $t_i > r_k$, then return "infeasible" (Job J_i cannot be scheduled).

Step 2: If $t_i < r_1$, then schedule job J_i *entirely* on the sub-torus (p-interval) $[a_1, a_1 + d_i - 1]$ from time f_1 to time $f_1 + t_i$.

Step 3: If there exists an integer j such that $t_i = r_j$, then schedule the job J_i entirely on the sub-torus $[a_j, a_j + d_i - 1]$ to use up all its RPT.

Step 4: Compute an integer j such that $t_i > r_j \wedge t_i \leq r_{j+1}$; schedule the job J_i on the sub-torus $[a_j, a_j + d_i - 1]$ to use up all its RPT r_j and schedule the remaining time $t_i - r_j$ of job J_i on the sub-torus $[a_{j+1}, a_{j+1} + d_i - 1]$ from time f_{j+1} to time $f_{j+1} + (t_i - r_j)$.

Remark 7 *For any job J_i if Step 1 does not apply, our algorithm is able to schedule the job by either one of the 3 steps 2, 3 or 4.*

Remark 8 *Note that application of the steps of the algorithm involves appropriate update of the profile; scheduling of a job J_i may split a particular p-interval into two or may necessitate deletion of a p-interval (due to its RPT being completely used up). This updating of the profile P will depend on the data structure used and is not relevant to the correctness of the scheduling algorithm.*

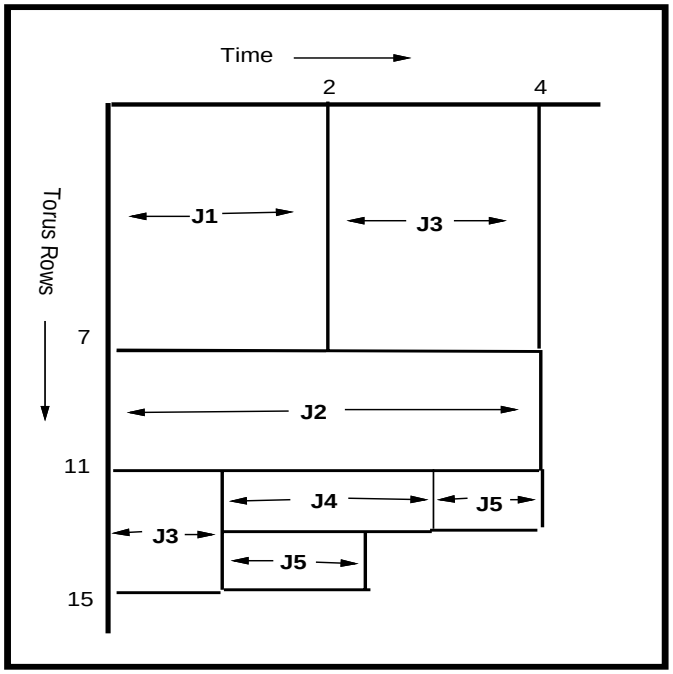

Figure 5: Schedule for the Example Job Set

Example: Consider a 16-torus or T_{16} (i.e., $m = 16$) and a deadline of $T = 4$, and a job set $J = \{J_1, J_2, J_3, J_4, J_5\}$, where $J_1 = (8, 2)$, $J_2 = (4, 4)$, $J_3 = (4, 3)$, $J_4 = (2, 2)$, $J_5 = (2, 2.5)$. Note that the jobs are arranged in non increasing order of required torus size. The initial profile is $([0, 15], 0)$ and Figure 5 shows the final schedule obtained by the algorithm. We show below the profiles generated after scheduling each job in the set:

$$([0,15],0) \xrightarrow[\text{Step 2}]{J_1 \text{ scheduled}} ([0,7],2),([8,15],0) \xrightarrow[\text{Step 3},j=2]{J_2 \text{ scheduled}} ([([0,7],2),([12,15],0)$$

$$\xrightarrow[\text{Step 4,}j=1]{J_3 \text{ scheduled}} ([12,15],1) \xrightarrow[\text{Step 2}]{J_4 \text{ scheduled}} ([12,13],3),([14,15],1) \xrightarrow[\text{Step 4, } j=2]{J_5 \text{ scheduled}} ([14,15],1.5)$$

Lemma 1 *The profiles $P(i), 0 \leq i \leq n$ are stair-like.*

Proof : The profile $P(0)$ is trivially stair-like. Assume that $P(i-1)$ is stair-like; we need to show that $P(i)$ is stair-like after the job J_i is scheduled by the algorithm.

- Assume Step 2 is executed to schedule J_i. There are two cases: If $b_1 - a_1 + 1 = d_i$, the profile $P(i)$ is obtained by replacing the first entry $([a_1, b_1], f_1)$ in $P(i-1)$ by an entry $([a_1, b_1], f_1 + t_i)$; else if $b_1 - a_1 + 1 > d_i$, then the profile $P(i)$ is obtained by replacing the first entry $([a_1, b_1], f_1)$ in $P(i-1)$ by two elements $([a_1, a_1 + d_i - 1], f_1 + t_i)$ and $([a_1 + d_i, b_1], f_1)$. In either case, the resulting profile $P(i)$ maintains the stair-like property.

- Assume Step 3 is executed to schedule J_i. Profile $P(i)$ is obtained by replacing the entry $([a_j, b_j], f_j)$ by a new entry $([a_j + d_i, b_j], f_j)$; the stair-like property is maintained. Note that if $b_j = a_j + d_i - 1$, then the original entry is simply deleted.

- Assume Step 4 is executed to schedule J_i. Profile $P(i)$ is obtained by deleting the entry $([a_j, b_j], f_j)$ and replacing the entry $([a_{j+1}, b_{j+1}], f_{j+1})$ by two entries $([a_{j+1}, a_{j+1} + d_i - 1], f_{j+1} + (t_i - r_j))$ and $([a_{j+1} + d_i, b_{j+1}], f_{j+1})$. Since $f_{j+1} + (t_i - r_j) \leq f_j$, the stair-like property is maintained in the profile $P(i)$.

□

Lemma 2 *The algorithm generates a feasible schedule iff one exists.*

Proof : We only need to prove that the algorithm generates a schedule if a feasible schedule exists. We use contradiction. Let S' be a feasible schedule of the job set J and the deadline T. Assume that the jobs $J_0, J_1, \cdots, J_{i-1}$ are scheduled in S' in the same way as in $S(i-1)$ and job J_i is scheduled differently in S' than it would be in $S(i)$. We show that S' can be modified so that J_i is scheduled in S' as in $S(i)$. Thus, the schedule S' can be transformed to $S(n)$, the schedule generated by the proposed algorithm. Let $P(i-1) = ([a_1, b_1], f_1), \cdots, ([a_k, b_k], f_k)$. Since the job J_i is scheduled in $S' - S(i-1)$, $f_k + t_i \leq T$ and hence our algorithm is able to schedule J_i and can generate $S(i)$. Assume our algorithm schedules J_i in $S(i)$ on subtorus $A = [a_j, a_j + d_i - 1]$ from time f_j to time $f_j + t_i$ ($= \tau$, say) (Step 2 or 3 of our algorithm); or on subtorus A from time f_j to T and on subtorus $B = [a_{j+1}, a_{j+1} + d_i - 1]$ from time f_{j+1} to time $f_{j+1} + (t_i - r_j)$ ($= \tau'$, say) (Step 4 of our algorithm). If the job J_i is scheduled in S' in the same way, we are done; if not, we rearrange jobs $J_i, J_{i+1}, \cdots, J_n$ in $S' - S(i-1)$ using the following procedure such that J_i is scheduled in S' just like in $S(i)$.

- Divide the entire time interval $[0, T]$ into equal length intervals of size δ (call those intervals δ-intervals) such that each job in S' is preempted or finished at the end of some δ-interval; this can always be done by choosing δ sufficiently small.

For an arbitrary δ-interval α, let $JS(\alpha)$ denote the set of jobs (from among $J_i, J_{i+1}, \cdots, J_n$) that are scheduled in S' in the δ-interval α, i.e., $JS(\alpha) = \{J_k : i \leq k \leq n,$ and J_k is scheduled in S' over $\alpha\}$.

- Divide the m-torus into m/d_i many d_i-subtorii across the entire interval $[0, T]$; line up jobs in $JS(\alpha)$ over each interval α such that no job is scheduled on two d_i-sub-torii – this is possible because $\forall J_k \in JS(\alpha) : d_k \leq d_i$.

- Let $T' = T - t_i$. Divide the schedule S' into two parts: left and right of T'. Let $I_1 = \{\alpha : \alpha$ is a δ-interval on left of T' and $J_i \notin JS(\alpha)\}$ and let $I_2 = \{\alpha' : \alpha'$ is a δ-interval on right of T' and $J_i \in JS(\alpha')\}$. Obviously, number of intervals in I_1 and I_2 are equal. Now we can think of a one-to-one function from I_1 to I_2. Consider an interval α in I_1 and the corresponding α' in I_2. Since the profile $P(i-1)$ is stair-like, number of d_i-subtorii over α in $S' - S(i-1)$ is at least as many as over α'. Thus, since J_i is over α' and not over α, there is at least a d_i-subtorus over α which is either an empty interval or occupied by a job in $JS(\alpha) - JS(\alpha')$ – thus we can interchange.

- Now the job J_i is in the interval $[T', T]$; we now move it to the desired subtorus and time intervals as is done in $S(i)$. We use the following rules:

  (1) If Step 2 is used to schedule J_i on $S(i-1)$ to produce $S(i)$, J_i is scheduled on subtorus $A = [a_1, a_1 + d_i - 1]$ from f_1 to $f_1 + t_i = \tau$. In this case, $T' > f_1$ and $\tau > f_1$. For each α in $[T', T]$ in S', we interchange J_i in its d_i-subtorus with jobs in A; we then swap J_i in A over $[T', T]$ with that in A over $[f_1, X]$. Because A extends from f_1 to T in $S' - S(i-1)$, the swapping can always be done.

  (2) If Step 3 is used to schedule J_i on $S(i-1)$ to produce $S(i)$, J_i is scheduled entirely on subtorus $A = [a_j, a_j + d_i - 1]$ from time f_j to time $f_j + t_i = \tau$. In this case, $T' = f_j$ and $T = \tau$. For each α in $[T', T]$ in S', we just interchange J_i in its d_i-subtorus with jobs in A.

  (3) If Step 4 is used to schedule J_i on $S(i-1)$ to produce $S(i)$, J_i is scheduled on subtorus A from time f_j to T and on subtorus $B = [a_{j+1}, a_{j+1} + d_i - 1]$ from time f_{j+1} to time $f_{j+1} + (t_i - r_j) = \tau'$. In this case, $f_j > T' > f_{j+1}$ and $f_j > \tau' > f_{j+1}$. For each α in $[f_j, T]$ in S', we interchange J_i in its d_i-subtorus with jobs in A; we interchange J_i in subtorus B over $[T', f_j]$ with that in B over $[f_{j+1}, \tau']$. Because B extends from f_{j+1} to T in $S' - S(i-1)$, the swapping can always be done.

□

Theorem 2 *The number of preemptions in a feasible schedule produced by the algorithm is upper bounded by* $n - 1$.

Proof : The first job J_1 is scheduled without any preemption and for each subsequent job we need at most one preemption; thus the result follows. □

Theorem 3 *The feasibility algorithm has a run time complexity of* $\mathcal{O}(n \log n)$.

Proof : The feasibility algorithm involves updating the profile by scheduling one job at a time from the job set starting from a profile of a single entry and assuming that the job set is ordered in non increasing order of dimension requirement. The jobs can be ordered in $\mathcal{O}(n \log n)$ time using a sorting algorithm like heapsort. The profile can be maintained by using some kind of a balanced tree structure like AVL trees. The initial tree contains only one node. Update of the tree for scheduling one job involves, in the worst case, one deletion and one insertion (i.e., one entry of the profile may need be deleted and one additional entry may need be inserted). Insertion and/or deletion in an AVL tree can be done in $\mathcal{O}(\log n)$ time where n is the number of elements in the tree. Thus, the entire operation of the profile updating can be done in $\mathcal{O}(n \log n)$ time. Lastly, to schedule each job, we need to decide on the particular step of the algorithm. There are only 4 steps in the algorithm; to decide if a particular step is applicable, we need to do a search on the tree which can take at most $\mathcal{O}(\log n)$ time and hence the decision process for all the n jobs will take $\mathcal{O}(n \log n)$ time. □

4 Conclusion

We have proposed a scheme to mathematically model the contention in the communication links when multiple jobs are scheduled on subtorii sharing communication links and then developed an efficient algorithm to schedule a given set of jobs (with different execution times and different subtorii requirements) with an objective to minimize the schedule length. We have also developed a feasibility algorithm to preemptively schedule a set of job with time and dimension requirements on a given torus with a given deadline. We have shown that the algorithm runs in $\mathcal{O}(n \log n)$ time where n is the number of jobs. Once we have the feasibility algorithm, minimum finish time for a given job set and a given m-torus can be easily computed by using binary search over a time interval $[t_{max}, \sum_{i=1}^{n} \frac{d_i^2 * t_i}{m^2}]$ where $t_{max} = \max\{t_1, t_2, \cdots, t_n\}$. It'd be interesting to design strategies to compute the minimal finish time of a job set for a given torus connected network without using the binary search.

References

[AZ90] M. Ahuja and Y. Zhu. An o(n log n) feasibility algorithm for preemptive scheduling of n independent jobs on a hypercube. *Information Processing Letters*, 35:7–11, June 1990.

[BCC+88] S. Borkar, R. Cohn, G. Cox, S. Gleason, T. Gross, H. T. Kung, B. Moore M. Lam, C. Peterson, J. Pieper, L. Rankin, P. S. Tseng, J. Sutton, J. Urbanski, and J. Webb. "iWarp: An integrated solution to high-speed parallel computing". In *Proceedings of Supercomputing '88*, pages 330–339. IEEE Computer Society and ACM SIGARCH, November 1988.

[BP95] G. Bilardi and F. P. Preprata. "Horizons of Parallel Computation". *J. of Parallel and Distributed Computing*, 27:172–182, 1995.

[Cra93] Cray Research, Inc. *Cray T3D System Architecture Overview*, 1993.

[dCVGG95] Luis Diaz de Cerio, Miguel Valero-Garcia, and Antonio Gonzales. A study of the communication cost of the fft on torus multicomputers. In *IEEE First Intl. Conf. on Algorithms and Architectures for Parallel Processing*, pages 131–140, 1995.

[DDF+89] W. J. Dally, R. Davison, J. A. Stuart Fiske, G. Fyler, J. S. Keen, R. A. Lethin, M. Noakes, and P. R. Nuth. "The J-Machine: A Fine-grain Concurrent Computer". In *Information Processing 89, IFIP*, pages 1147–1153, 1989.

[LC91] K. Li and K. H. Cheng. Job scheduling in a partitionable mesh using a two-dimensional buddy system partitioning scheme. *IEEE Trans. on Parallel and Distributed Systems*, 2(4):413–422, 1991.

[LM89] H. Li and M. Maresca. "Polymorphic-Torus Network". *IEEE Trans. on Computing*, 38(9):1345–1351, Sept. 1989.

[Mas] MasPar Computer Co. *"MP-1 Family Data-parallel Computers"*.

[Oed93] W. Oed. *Massively Parallel Processor System Cray T3D*. Cray Research GmbH, 1993.

[QN95] W. Qiao and L. M. Ni. Efficient processor allocation in 3D tori. In *Proceedings of International Parallel Processing Symposium*, April 1995.

[TG96] Y.-C. Tseng and S. K. S. Gupta. "All-to-All Personalized Communication in a Wormhole-routed Torus". *IEEE Trans. on Parallel and Distributed Systems*, 7(5):498–505, May 1996.

[TLGP97] Y.-C. Tseng, T.-H. Lin, S. K. S. Gupta, and D. K. Panda. Bandwidth-optimal complete exchange on wormhole-routed torus networks: a diagonal-propagation approach. *IEEE Trans. on Parallel and Distributed Systems*, 1997. To appear.

[TS88] M. R. Thistle and B. J. Smith. "A Processor Architecture for Horizon". In *Supercomputing*, pages 35–41, 1988.

[ZA93] Y. Zhu and M. Ahuja. On job scheduling on a hypercube. *IEEE Transactions on Parallel and Distributed Systems*, 4(1):62–69, January 1993.

[Zhu92] Y. Zhu. Efficient processor allocation strategies for mesh connected parallel computers. *J. of Parallel and Distributed Computing*, 16:328–337, 1992.