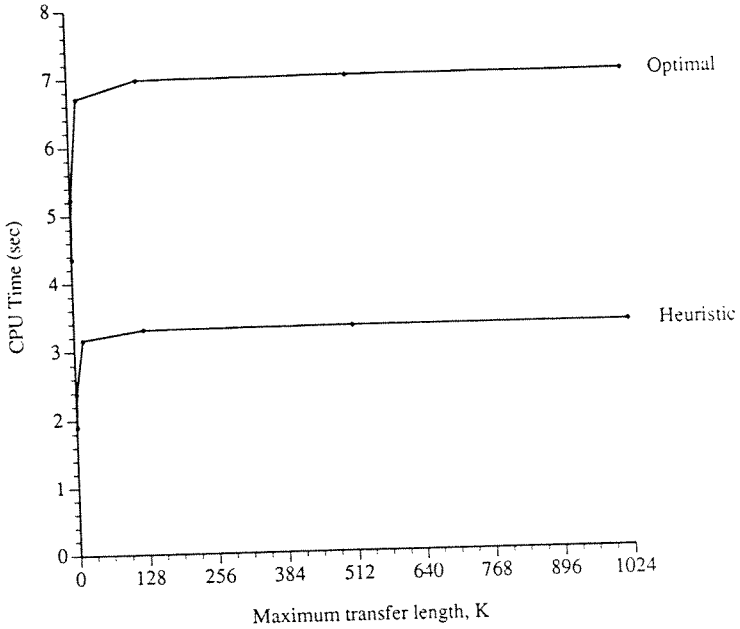

Figure 6.3: CPU time versus maximum transfer length for **GRA** and **Tree** for complete binary tree architectures with 64 senders

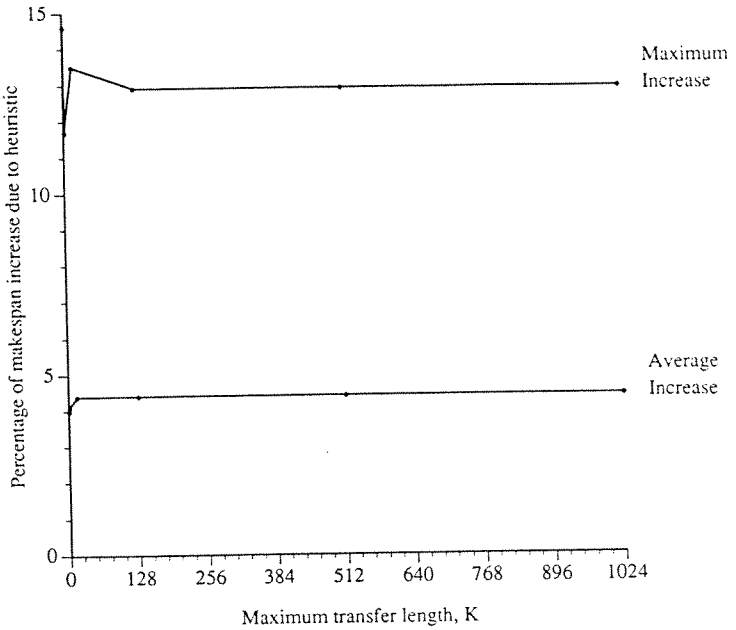

Figure 6.4: Penalty paid for using **GRA** versus maximum transfer length for complete binary tree architectures with 64 senders

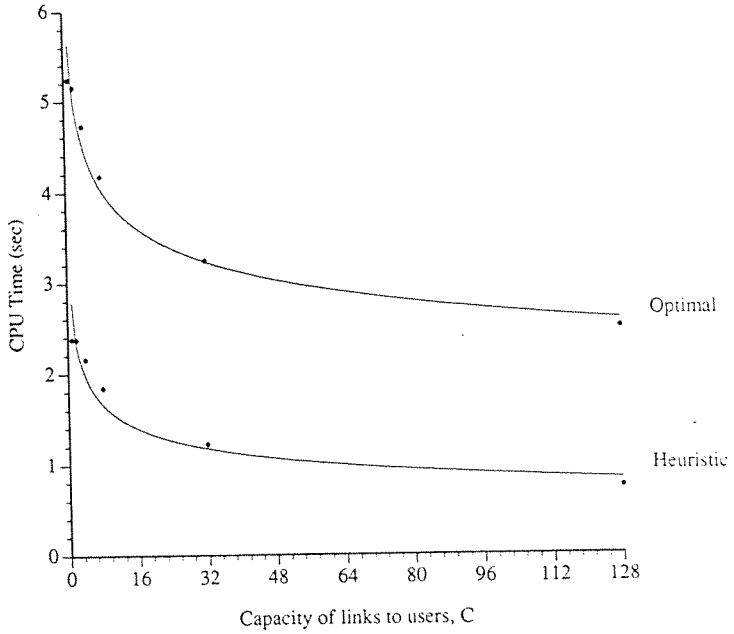

Figure 6.5: CPU time versus user link capacity for **GRA** and **Tree** for complete binary tree architectures with 64 senders

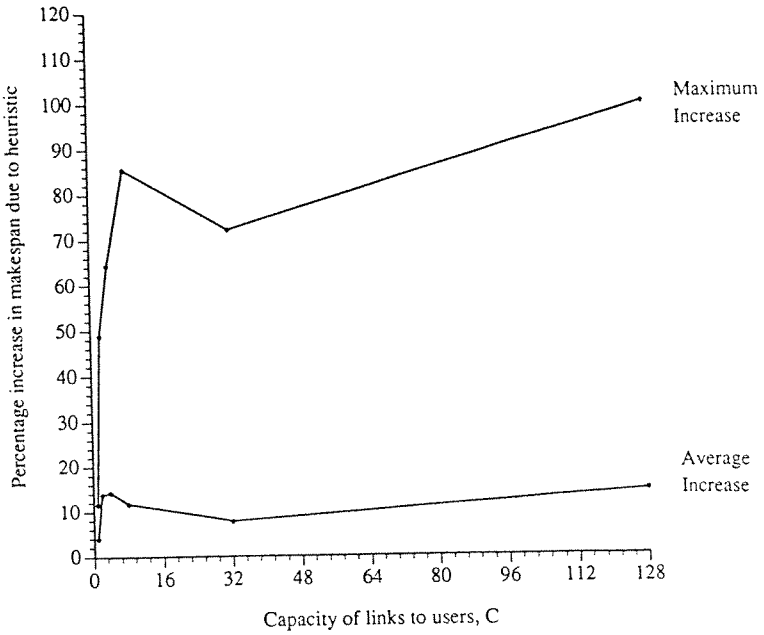

Figure 6.6: Penalty paid for using **GRA** versus user link capacity for complete binary tree architectures with 64 senders

6.3 Discussion

The experiments described in the previous section show that as the number of users is increased, the **GRA** algorithm can provide substantial savings in average execution time over **Tree** (upto 50%, or an asymptotic factor of n improvement) for a relatively small average penalty (about 5% increase in schedule length). This behavior is almost independent of the lengths of the transfers involved. The insensitivity to transfer lengths is as expected, since although longer transfer lengths will lead to longer schedules, they will in general not increase the switching complexity, or number of feasible transfers that are calculated; each feasible transfer will simply be re-used for a greater number of time slots.

The running time of **GRA** follows that of **Tree** as the user link capacities are incresed. The discussion in sec. 5.6 applies to **GRA** also. It is interesting to note that as user link capacities are increased, the maximum penalty, in terms of schedule length, for using **GRA** increases quite sharply, while the average penalty remains in the 10-15% range. Clearly as link capacities increase the optimal algorithm has greater opportunities for maximizing the number of parallel data transfers at every time slot, while the heuristic does not backtrack to try to take advantage of these opportunities once a set of feasible transfers has been found. Further study is needed to fully understand the differences between the average and maximum penalties, and the effect of increasing the user link capacities even further.

Taking a broader view, the results of this chapter constitute one exploration of a design space for constructing heuristics to solve $TreeDTS$. It is possible to

design a different heuristic, for instance simply by changing the ordering that is applied to the edges of the resource graph in the Greedy Tree Algorithm. As a concrete example, consider the following promising heuristic: sort the edges by their congestion, i.e., calculate for each edge the ratio of the transfer length for that edge to the minimum of the capacities of its end users, and examine those edges with the highest congestion ratios first. Call this heuristic the Maximum Congestion First, or **MCF** heuristic. The **MCF** heuristic is myopic in the same sense that **GRA** is: only part of the input instance information is examined. **MCF** is also simplistic in the same sense that **GRA** is: no backtracking is done to improve on the initial choices made by the algorithm. However, **MCF** would give rise to different, probably better, performance in terms of schedule length, while paying a greater penalty in terms of execution time, than **GRA**. An experimental evaluation of **MCF**, very similar in style to that described in this chapter for **GRA**, could then be carried out to test and quantify its behavior.

This example shows a general characteristic of the process of designing heuristics to solve optimization problems. There are two classes of design parameters: the amount of the (input) state space that is examined, and the complexity of the function applied to it. In the case of the **GRA** and **MCF**, both these parameters are changed: **MCF** examines more of the state space, as well as applies a slightly more complex function to it, than **GRA**. One can envision a range of heuristics that can be systematically designed, each appropriate for different input parameters and applications, as the design parameters are systematically varied.

Chapter 7

Scheduling in Extended Hierarchical Architectures

In this chapter we extend the range of problems that can be solved still further. The first extension solves a generalization of the problem of scheduling in tree architectures that was studied in Chapter 5. In the following section we consider tree architectures in which some transfers do not traverse the root of the tree, i.e., both local and remote transfers are permitted. When this possibility is specified in our model, the architecture graph is no longer a tree. This problem has applications both for parallel I/O and satellite communications scheduling. We obtain an approximation algorithm that solves more general cases of the problem than the best previous heuristic, has the same performance guarantee, but a better time complexity.

The second extension in this chapter considers tree architectures in which preemptions may take place arbitrarily, and not only at pre-specified integer boundaries. This problem is applicable to data transfers in systems with continuous media, such as those in current and proposed multimedia systems. We show that the **Tree** algorithm can be modified slightly to solve this problem. The last extension we mention is that of tree-structured architectures in

which the users communicate via transceivers. This problem has applications in packet radio networks. Sasaki [125] has proved that an approximation algorithm can be designed for this problem. The **Tree** algorithm modified to handle arbitrary preemptions is used as a subroutine in the solution.

7.1 Systems with local and remote data transfers

In this section we consider an extension to the $TreeDTS$ problem in which the architecture graph permits both local and remote transfers i.e., transfers that pass through the root of the tree architecture (remote transfers), as well as transfers that only traverse the root of a subtree (local transfers). This is an important extension as it occurs frequently both for the parallel I/O application as well as the communications switching application. In the case of the parallel I/O application it occurs in shared-bus systems such as the Sequent [100] and potentially also in systems such as Hector [138] and interconnection networks such as KYKLOS [102]. In the case of communications networks it occurs for intersatellite transfers [7, 53, 54].

We first define the problem formally in our model, and discuss how it arises in the parallel I/O and intersatellite communications application. We then show how the graph-theoretic nature of our specification facilitates the systematic *decomposition* of the problem into a number of subproblems, allowing us to obtain a heuristic solution. This solution also applies to a special case of the intersatellite communication problem studied earlier by Bertossi et al [7], and provides the same upper bound on makespan but better time complexity than the best heuristic available previously.

7.1.1 Specification of the problem

A formal specification of the problem in the model is as follows. See Fig. 7.1 for an example.

LocalRemoteDTS = (PG, AG, RG, f, Preempt)

where the 5-tuple is defined as follows.

$PG = (T, Ep, Lp)$ has $|T| = m$ tasks, of length $Lp(t) \in N$ for all $t \in T$, and $|Ep| = 0$, i.e., no precedence constraints.

$AG = (R, Ea, La)$ has $|R| = n + 3b$ vertices, with n vertices of type $SUSER$ and $RUSER$ b vertices each of type MUX and $DMUX$, and b vertices of type $NULL$. Each $NULL$ vertex is the root of a three-level tree which can be specified exactly as the architecture graph of *TreeDTS*, with N leaves of each type $SUSER$ and $RUSER$. (Hence $n = 2bN$). One of the $NULL$ vertices, called the *system vertex*, has an incoming arc from the root of every other $DMUX$ subtree to the root of its MUX subtree and an outgoing arc from the root of its $DMUX$ subtree to the root of every other MUX subtree. (These arcs are called *inter-bus links*, a name suggested by the parallel I/O application). The capacity is $La(e) = 1$ for all $e \in Ea$ except those arcs incident on the $NULL$ vertices; for the latter arcs $La(e) \geq 1$ and, for the parallel I/O application, represents the bus capacity.

$RG = (R, Er, Lr)$ is a bipartite graph, where Er is a set of arcs from $SUSER$ to $RUSER$ vertices only, leaving the assignment of other resources implicit,

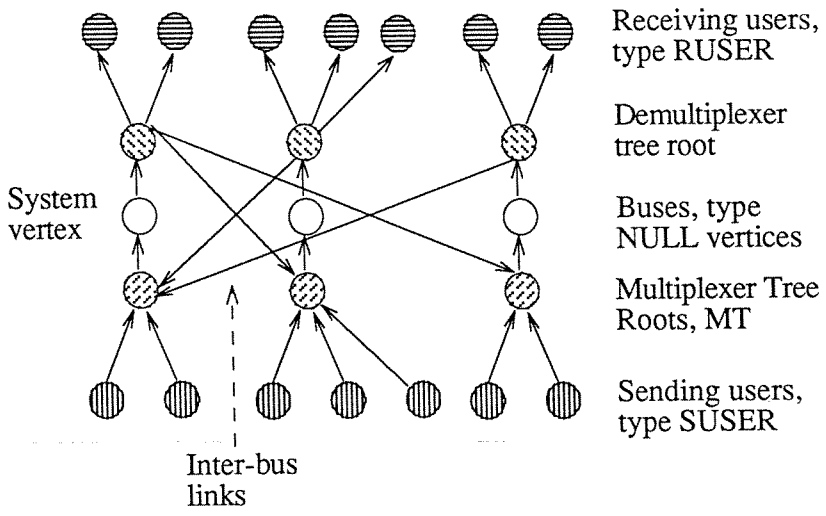

Figure 7.1: AG for the scheduling problem with local and remote transfers, *LocalRemoteDTS*

and Lr is a bijection from Er to T. RG is further restricted in that if the only path between a pair of $SUSER$ and $RUSER$ vertices in AG contains more than one inter-bus link, there is no arc between those vertices in RG.

f is makespan.

Preempt is true.

We discuss two applications which give rise to the *LocalRemoteDTS* problem.

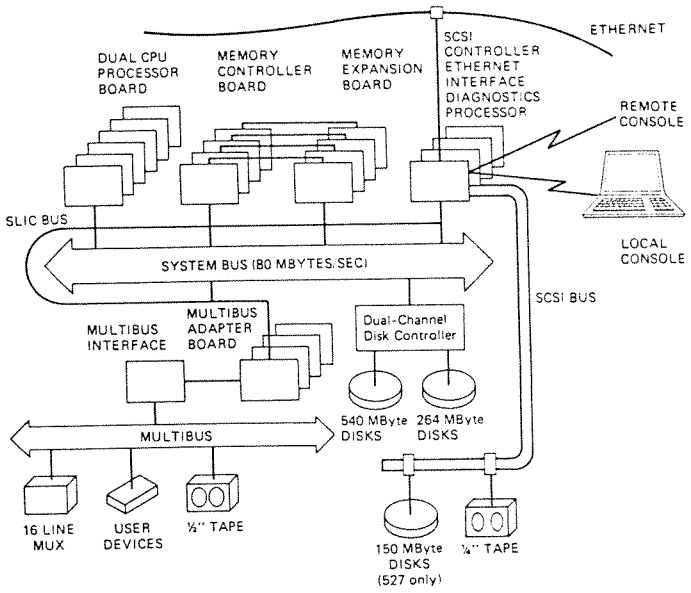

Figure 7.2: Sequent architecture

7.1.2 The parallel I/O application

The bus I/O scheduling problem consists of scheduling the data transfers among processors and peripheral devices connected via a collection of hierarchically organized buses. Examples of such architectures include the commercially succesful Sequent [100] and the research prototype Hector [138]. The Sequent system (Fig. 7.2) has a two-tier arrangement of buses. A single fast system bus connects processors and main memory. Several relatively slow I/O buses, e.g. SCSI or Multibus (see [68]), connect I/O devices to the system bus. Each bus permits at most one data transfer to be in progress at any given time.

Two types of I/O transfers may take place. Data transfers from one I/O device to another on the same bus are called *local* transfers, while those among

main memory and I/O devices on separate buses are called *remote* transfers. A remote transfer requires simultaneous posession of an I/O device, a system bus, an I/O bus, and either a memory unit or another I/O device. A local transfer requires two I/O devices and a local bus. An example of an application requiring local transfers is 3D visualization of scientific data (e.g. [144]), for which a fast disk may supply data to a high-performance graphics workstation connected to a common local bus. A more mundane example is the periodic backup of files from disks to tape. Each transfer consists of a number of fixed-size units (e.g. disk blocks) which we call packets, and transfers can be preempted at packet boundaries. It is assumed that each I/O device or memory unit has one port and is either a sender or a receiver of data.

As mentioned in Chapter 3, given the increasing data transfer demands of new applications programs, there are likely to be multiple parallel buses in future bus-oriented parallel architectures, as in the IBM RP3 [115]. Even if only one bus is available, however, if the bus bandwidth is high enough it can be time-shared to effectively provide many parallel I/O transfers. An example of a bus for which this is possible is the Sequent system bus, which has a bandwidth of 53 MB/s.

7.1.3 The intersatellite communications application

The intersatellite communications scheduling problem [7] consists of scheduling the data transfers among a set of *zones* on the ground via a network of satellites, ground-satellite links and intersatellite links (ISL). Each zone communicates with one satellite. Each ground-satellite link and each ISL is

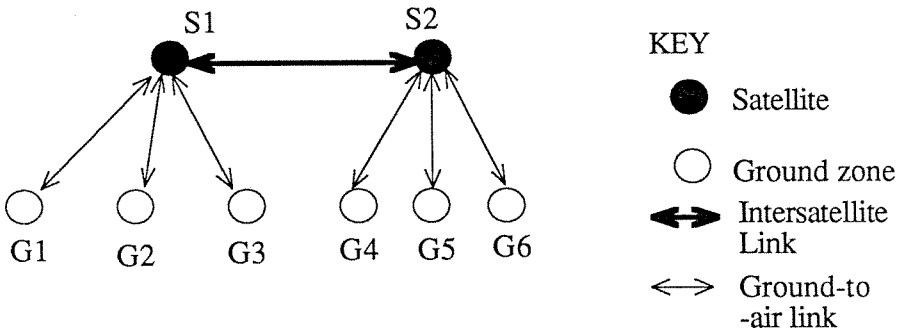

Figure 7.3: Example ISL communications system

bidirectional and allows at most one transfer in each direction at any given time. Both local transfers between zones connected to the same satellite, as well as intersatellite transfers between zones connected to different satellites, are possible (see Fig. 7.3). Data for a single transfer is transmitted as fixed-length packets which may be interspersed on the communications links with packets from other transfers. It is assumed that there is no intersatellite transfer required between two zones if their respective satellites are not connected by an ISL.

Bertossi et al [7] have shown that the ISL communication scheduling problem is NP-complete for an arbitrary number of satellites even for highly restricted ISL network topologies, and zero ISL propagation delay. They conjecture [7] that the special case of just two satellites and zero ISL delay, which we call the *SimpleISL* problem, is also NP-complete, and propose two suboptimal

heuristics. Both heuristics generate schedules of at most twice the optimal schedule length. That is, the upper bound on the makespan, $UB(1) = 2\ LB$, where LB is a lower bound on the makespan defined as follows. Let $L(u,v)$ be the total intersatellite traffic from satellite u to satellite v, $ST(u)$ the total traffic sent from zone u, and $RT(u)$ the total traffic received at zone u. Then,

$$LB = max\{max\{L(i,j) : 1 \leq i,j \leq b \wedge i \neq j\},$$
$$max\{ST(i) : 1 \leq i \leq n\}, max\{RT(i) : 1 \leq i \leq n\}\}$$

The time complexity of the first heuristic is $O(n^{4.5})$, and of the second is $O(n^{8.5})$.

The ISL problem has also been formulated as a modified open-shop by Ganz and Gao [53] for the case of arbitrary ISL propagation delay, and a heuristic has been proposed. The modified open-shop formulation models each uplink as a processor and each downlink as a job. Since each local data transfer requires an uplink and a downlink simultaneously it is modeled as one operation of a job on a processor. Since the transfers through a downlink may occur in any order, the operations of jobs on processors are modeled as not having any technological cobstraints, i.e., as an open shop.

The difficulty with this formulation arises in its modeling of intersatellite transfers, which require three resources (uplink, downlink and ISL) simultaneously. Each direction of an ISL is modeled as a processor and a job consisting of only one operation which executes on that processor. Thus an intersatellite transfer assumes implicitly that two processors are used simultaneously, and

this assumption is enforced when each time slot is scheduled by the heuristic solution [53].

From our viewpoint the problem specification in our model is preferable to the open-shop formulation as the former clearly and explicitly specifies precisely which transfers require which three resources simultaneously. (This approach subscribes to software engineering principles which advocate a clean separation between the specification of a program and its implementation).

An assumption made by Ganz and Gao [53] is that the ISL propagation delay δ is the same for all ISL. In addition, an intersatellite transfer of length t time units is assumed to require the uplink and downlink for $t + \delta$ time units and the ISL for t units. We assume that an intersatellite transfer requires all three links for time $t + \delta$.

SimpleISL is the same as *LocalRemoteDTS* except that $b = 2$, and, since a zone may send and receive data simultaneously, each zone is represented as one $SUSER$ vertex and one $RUSER$ vertex. In the following section we describe a heuristic to solve *LocalRemoteDTS* and *SimpleISL*.

7.1.4 The decomposition heuristic

The heuristic we use to solve *LocalRemoteDTS* is to decompose the architecture graph into trees and apply the solution used for *TreeDTS*. One set of trees allows only local transfers to take place, and one tree allows the remote

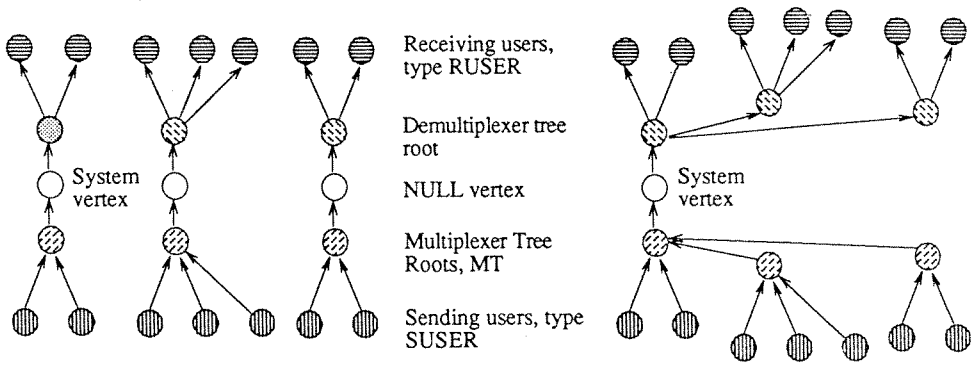

Figure 7.4: Applying the decomposition heuristic

transfers to take place. An example of the decomposition is shown in Fig. 7.4.

Decomposition Heuristic.

Input: Scheduling problem *LocalRemoteDTS.*

Output: A schedule satisfying *LocalRemoteDTS.*

Step A. Decompose LocalRemoteDTS to b Local data transfer problems

1. Let $AG' = (R, Ea', La')$ with $Ea' = Ea - \{e : e \text{ is an inter-bus link}\}$ to to obtain a forest $AG' = \{AG'(1), AG'(2), ..., AG'(b)\}$. La' is La restricted to Ea'.
2. For $1 \leq i \leq b$, let $RG'(i) = (R'(i), Er'(i), Lr'(i))$ be RG restricted to the vertices in $AG(i)$.

3. For $1 \leq i \leq b$, let $PG'(i) = (T'(i), Ep'(i), Lp'(i))$ with $T'(i)$ restricted to the arcs of $Er'(i)$, i.e., $T'(i) = \{t : t \in Lr'(i)\}$, and $Ep'(i)$ and $Lp'(i)$ restricted to $T'(i)$.
4. For $1 \leq i \leq b$, there is a scheduling problem *LocalDTS(i) = (PG(i), AG(i), RG(i), f, Preempt).*

Step B. Decompose LocalRemoteDTS to one Remote data transfer problem

1. Let $AG'' = (R'', Ea'', La'')$ be constructed as follows. Denote the $NULL$ system vertex as s, its MUX child as MT and its $DMUX$ child as DT. Then $R'' = R - \{v : v \in NULL - \{s\}\}$. Let E be Ea restricted to R'' and with all inter-bus links deleted. Let $F = \{(u, MT) : u \neq MT \wedge u$ *is a* MUX *vertex*$\}$. Similarly let $G = \{(DT, u) : u \neq DT \wedge u$ *is a* $DMUX$ *vertex*$\}$. Then $Ea'' = E \cup F \cup G$. La'' is La restricted to Ea'' and with unit capacities for arcs in $F \cup G$.
2. Let $RG'' = (R', Er'', Lr'')$ with $Er'' = Er - \{e : e \in Er'(i), 1 \leq i \leq b\}$, and Lr'' restricted to Er''.
3. Let $PG'' = (T'', Ep'', Lp'')$ where $T'' = \{t : t = Lr''\}$, and Ep'' and Lp'' are restricted to T''.
4. Obtain a scheduling problem *RemoteDTS = (PG", AG", RG", f, Preempt).*

Step C. Schedule Local data transfers. For each *LocalDTS(i)*, call the **Tree** algorithm.

Step D. Schedule Remote data transfers. Call the **Tree** algorithm to solve *RemoteDTS.*

End Heuristic.

Theorem 7.1 *The decomposition heuristic solves an instance of LocalRemoteDTS in time $O(n^4)$, providing a schedule with an upper bound on the makespan of $UB(2) = 2\ LB$.*

Proof. The heuristic clearly terminates, and from the construction it is plain that the $b+1$ scheduling problems $LocalDTS(i)$ and $RemoteDTS$ are special cases of *TreeDTS*. Recall that the time complexity of the **Tree** algorithm is $O(n^4C)$ where n is the number of user nodes and C is the average capacity of the user links For *LocalRemoteDTS* the user nodes are $SUSER$ and $RUSER$ vertices and the user links are the arcs incident upon them, so $C = 1$. For $LocalDTS(i)$ the number of user nodes is $2N = n/b$ while for $RemoteDTS$ it is n, so that the time complexity of the decomposition heuristic is $O(n^4)$. The length of the schedule generated by the heuristic has an upper bound given by

$$\begin{aligned} UB(2) &= max\{L(i,j) : 1 \leq i,j \leq b \wedge i \neq j\} \\ &\quad + max\{max\{ST(i) : 1 \leq i \leq n\}, max\{RT(i) : 1 \leq i \leq n\}\} \\ &\leq 2\ LB \\ &= UB(1) \end{aligned}$$

□

The decomposition heuristic is a generalization of the heuristics of Bertossi et al [7]. For the *SimpleISL* problem the decomposition heuristic compares favorably with those Bertossi et al, which have the same upper bound on schedule length and have time complexities of $O(n^{4.5})$ and $O(n^{8.5})$.

Our approach to the scheduling problem for local and remote transfers also illustrates the use of the scheduling model to obtain effective solutions to scheduling problems by graphical decomposition of the abstract specification.

7.2 Systems allowing arbitrary preemption

In this section we briefly note that the **Tree** algorithm can be modified to solve problems in which the traffic demand is non-integer or the system allows preemption at arbitrary boundaries. This may be useful for data transfers involving continuous media, which is becoming more commonly used in multimedia applications.

It turns out that the proof of correctness, and the time complexity, of **Tree** derived in need to be modified only slightly. The lower bound on the schedule length becomes:

$$L'(r) = \max_{e \in E} \frac{t_r(e)}{c(e)}$$

The definition of *network link lower bound* and *network link capacity* are changed to:

$$b_h^*(e) = \begin{cases} c(e), & \text{if } e \in E \text{ and } \frac{t_r(e)}{c(e)} = L'(r) \\ 0, & \text{otherwise} \end{cases}$$

$$c_h^*(e) = \begin{cases} 0, & \text{if } e \in E \text{ and } t_r(e) = 0 \\ c(e), & \text{otherwise} \end{cases}$$

Similar changes to other definitions and proofs, most of which are straightforward, show that **Tree** remains optimal for solving $TreeDTS$ with arbitrary preemptions, and the time complexity remains $O(Cn^4)$. For details, see Sasaki and Jain [125].

7.3 Applications to packet radio and transceiver systems

An important extension to the $TreeDTS$ problem is the case where users communicate through *transceivers*. A transceiver is a device that can transmit and receive, but not both at the same time. This problem has applications for packet radio networks and other communications networks. Special cases of the problem have been studied by Hajek and Sasaki as well as Choi and Hakimi [66, 25].

The paper by Sasaki and Jain [125] shows that the **Tree** algorithm modified to handle arbitrary preemptions, mentioned above, can be used as a subroutine to approximately solve a special case of $TreeDTS$ with transceivers. The modified algorithmm has a time complexity of $O(Cmn^2)$. The proof is due to Sasaki.

7.4 Conclusions and future work

The previous work related to the results in this chapter has already been discussed; it can be found in [7, 66, 25, 53, 54, 125].

Our contributions in this chapter can be summarized as follows. Firstly, we have found an approximation algorithm for the problem of data transfers in tree architectures when both local and remote transfers are allowed. This problem has important applications for parallel I/O as well as intersatellite communications. Our algorithm generalizes previous work that was done in the context of intersatellite communications, provides a performance guarantee at least as good as that provided by the previous best heuristic, and has better time complexity. Secondly, we have shown that the **Tree** algorithm can be modified to apply to tree architectures in which preemption can occur arbitrarily. Such systems may become more prevalent in the future with the spread of continuous media in multimedia systems. Finally, we mention that the **Tree** algorithm has been shown to provide a heuristic for data transfer scheduling in systems with transceivers.

For future work, we suggest two questions. The first is to determine the operating parametres for which the heuristic for local and remote transfers, as well as the **Tree** algorithm for continuous media, provide practical solutions for the parallel I/O application. The second is to resolve the open question of whether *SimpleISL*, the intersatellite communication problem with two satellites for which we have provided a heuristic, is NP-complete.

Chapter 8

Scheduling Tasks Under Mutual Exclusion and Precedence Constraints

A natural extension to the data transfer problem DTS is to allow the specification of logical constraints between tasks. In this chapter we will consider two types of constraints: mutual exclusion constraints and precedence constraints. Informally, a mutual exclusion constraint between two tasks means that in any legal schedule they are not permitted to execute simultaneously. Note that precedence constraints are logically stronger than mutual exclusion constraints, as they additionaly specify the order in which tasks must occur. While the scheduling of tasks with precedence constraints has been studied extensively (e.g. see Chapter 2 and [56, 112]), to our knowledge the previous work on mutual exclusion constraints is very limited, despite its practical applicability to parallel computing. Thus we first concentrate on developing results for scheduling tasks under mutual exclusion constraints, and in sec. 8.2 we present some results on the NP-completeness of precedence-constrained scheduling.

8.1 Mutual exclusion constraints

Mutual exclusion constraints on tasks are very common in parallel programs, and represent a natural and practical means of expressing synchronization requirements. The CODE parallel programming environment, for instance, provides a limited form of mutual exclusion constraints for this purpose [15], [140], [141]. We surmise that mutual exclusion constraints may also be useful for expressing scheduling constraints on problems drawn from many other application areas, including the communications switching application.

An advantage of our model (see Chapter 2) over simpler classification schemes is the ability to perform graph transformations to the problem specifications so as to prove that two problems are related in the sense that a solution to one is a solution to the other. In this section we demonstrate this by showing the scheduling problem for data transfers in systems with tree-structured architectures (*TreeDTS*) is related to scheduling data transfers in the presence of limited mutual exclusion constraints. The limited mutual exclusion constraints are a superset of those allowed in the CODE 1.2 parallel programming environment [140]. We thus obtain an algorithm producing optimal-length schedules for this application of parallel I/O scheduling in a parallel programming environment.

8.1.1 Problem definition

We consider the data transfer scheduling problem in which each data transfer operation involves a distinct pair of resources drawn from two disjoint resource

sets, and each transfer may be required to be logically mutually exclusive with a (restricted) set of other transfers. For simplicity we assume in the following that the architecture provides a direct dedicated link between every pair of communicating entities.

In our model the problem is specified as follows:

$$LimMutex = (PG_1, AG_1, RG_1, f_1, Preempt_1)$$

where the elements of the 5-tuple are defined as follows.

$PG_1 = (T, Ep, Lp)$ where $|T| = n$, $Lp(t)$ is the length of task $t \in T$, and Ep is a set of hyperedges. Recall that in the model hyperedges represent mutual exclusion constraints between tasks, i.e., no two tasks included in the same hyperedge may execute simultaneously. We will introduce a hierarchy of restrictions on Ep below.

$AG_1 = (R, Ea, La)$ with $|R| = 2n$ contains vertices of type $SUSER$ and $RUSER$ corresponding to sending users and receiving users respectively. For ease of exposition we assume the number of both sending users and receiving users is equal to n, although this restriction can easily be relaxed. Ea forms a complete directed bipartite graph from vertices of type $SUSER$ to vertices of type $RUSER$, and $La(e) = 1$ is the link capacity for all $e \in Ea$.

$RG_1 = (R, Er, Lr)$ where Er is a set of arcs from vertices of type $SUSER$ to those of type $RUSER$ such that no two edges share a vertex. Lr is a bijection from Er to T.

f_1 is makespan.

$Preempt_1 = true$.

Terminology. The abbreviation *mutex* stands for "mutual exclusion constraint"; the plural is *mutexes*. Vertices and hyperedges in PG will sometimes be called the entities they represent, i.e., tasks and mutexes, and vice versa. A set of vertices in PG (i.e., tasks) connected by a hyperedge is called a *mutex set*. A task $t \in T$ is said to *participate in a mutex* if it is a member of a mutex set in PG. The degree of a graph G is denoted $degree(G)$.

8.1.2 Limited Mutual Exclusion Constraints

Allowing arbitrary mutexes between tasks can lead to very complex restrictions which are difficult for run-time systems to enforce efficiently, let alone schedule optimally; to our knowledge there are no published scheduling results in this area. It is necessary to consider limited mutexes that provide a trade-off between expressibility and efficiency. We consider three successively looser restrictions.

R1. A task may participate in at most one mutex, i.e., $degree(PG_1) \leq 1$.

R2. There are no hyperedge cycles of odd length in PG_1 and $degree(PG_1) \leq 2$.

R3. A task may participate in at most 2 mutexes, i.e., $degree(PG_1) \leq 2$.

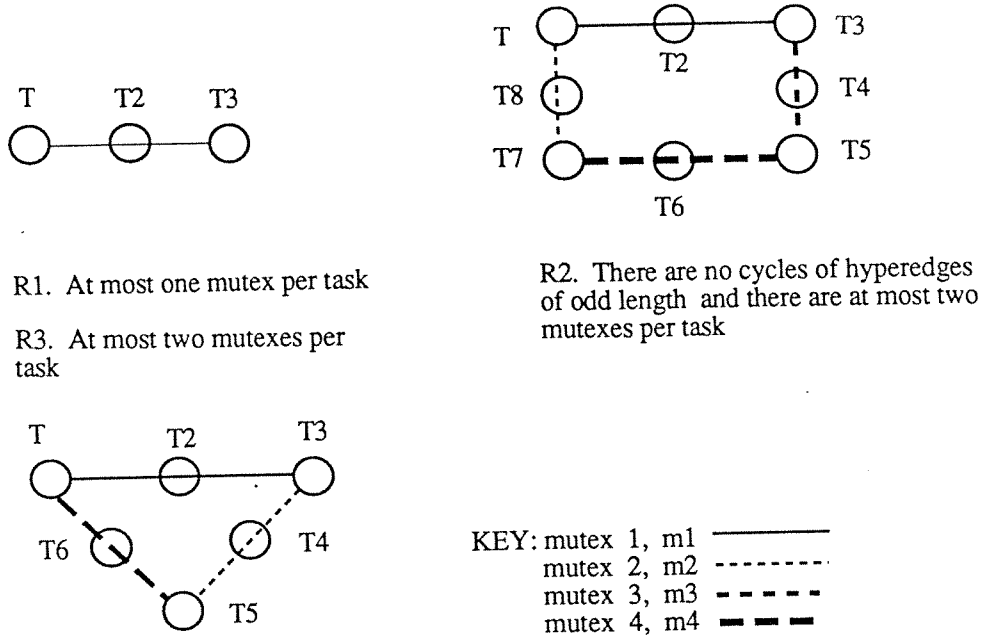

Figure 8.1: Limited mutual exclusion constraints

Clearly, $R1 \Rightarrow R2$, and $R2 \Rightarrow R3$. In Fig. 8.1 we show examples of precedence graphs satisfying *R1 - R3*. Restriction *R1* is implemented in the CODE 1.2 parallel programming environment [140]. We will show that there exist instances of the *LimMutex* problem satisfying *R3*, but not *R2*, which cannot be scheduled using the **Tree** algorithm of Chapter 5.

8.1.3 Transformation

We will transform *LimMutex* into the problem *Tree** defined below, which is an instance of *TreeDTS*. The basic idea is that the mutual exclusion constraints in *LimMutex* are converted to architecture constraints. For an example of this transformation, see Fig. 8.2.

$$Tree^* = (PG_2, AG_2, RG_2, f_2, Preempt_2)$$

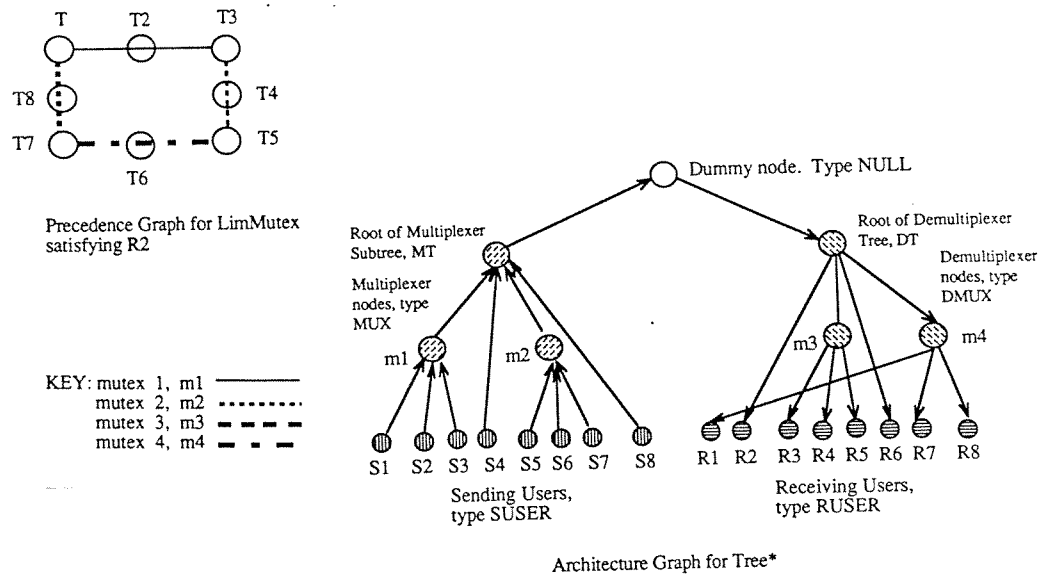

Figure 8.2: Example mutex transformation

where

$PG_2 = (T, Ep, Lp)$ has $|T| = n$ and $Ep = \{\}$, as in *TreeDTS*. $Preempt_2 = Preempt_1$ and $f_2 = f_1$, and both variables are as in *TreeDTS*. $RG_2 = RG_1$, and hence this is a special case of *TreeDTS* where no two arcs share a vertex since there are no resource assignment conflicts. $AG_2 = (R, Ea, La)$ is a special case of *TreeDTS* (see Fig. 5.1) in which all arcs not connected to the root have capacity 1, and the arcs entering and leaving the root have capacity n.

An informal explanation of the transformation is given here (see Theorem 8.1 below for the proof). The basic idea is that the two sets of leaves of the architecture graph of *Tree** represent data senders and receivers respectively, and the interior vertices model the mutual exclusion constraints of *LimMutex*.

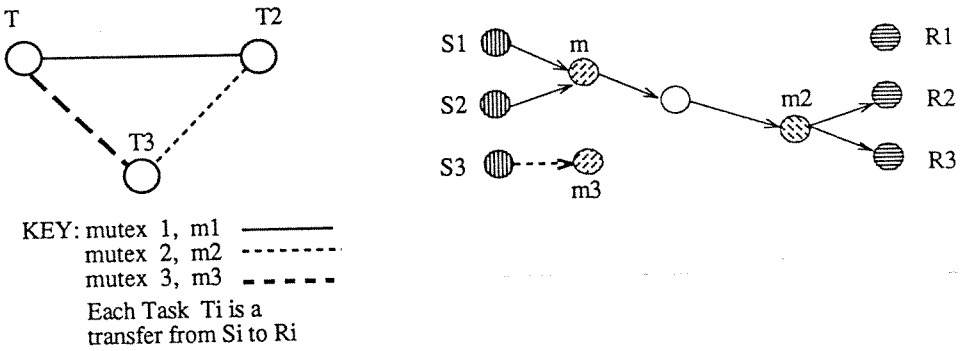

Figure 8.3: Example PG satisfying $R3 \wedge \neg R2$

For instance, a MUX vertex with two incoming unit-capacity arcs from two senders, but a single outgoing arc of unit capacity, will allow only one of the senders to transfer data at any given time.

Both MUX and $DMUX$ interior vertices can be introduced to model mutexes, and each sender and each receiver can be connected to an interior vertex. Since each data transfer task involves a distinct sender-receiver pair, it would thus seem that each task can be allowed to participate in two mutexes; in other words, restriction $R3$ would seem to suffice. However, the tree topology of the architecture graph requires that every path from a sender to a receiver include the root vertex. This necessitates the stronger restriction $R2$ on mutexes. Fig. 8.3 shows an example for which $R3$ is satisfied but $R2$ is violated, and an instance of $Tree^*$ cannot be constructed.

Theorem 8.1 *An instance of LimMutex satisfying R2 can be transformed to an instance of Tree*.*

Proof. (Follows from Lemmas 8.1 - 8.3 below.) For convenience PG_1, the precedence graph of *LimMutex*, is converted to a *mutex graph* containing edges but no hyperedges. Condition *R2* can then be stated as an equivalent condition *R2'* on the mutex graph (see Lemma 8.1). The key step of the proof is contained in Lemma 8.2, which shows that, provided the mutex graph satisfies *R2'*, each vertex in the mutex graph (i.e., each mutex in PG_1) can be systematically represented as an interior vertex of type MUX or $DMUX$ in AG_2 so that adjacent vertices in the mutex graph are assigned different types. This process is called *typing*, and an algorithm for performing it is given. Once the interior vertices have been consistently typed, it is a straightforward construction to obtain AG_2 (see Lemma 8.3). □

Def. Given a precedence graph $G = (V, E, L)$ containing only hyperedges, the corresponding mutex graph is $M_G = (V', E', L')$ where $V' = \{m_i : e_i \in E\}$, $E' = \{(m_i, m_j) : e_i, e_j \in E \wedge e_i \cap e_j \neq \{\}\}$, and $L' : V' \rightarrow 2^V$ is a labeling function such that $L(m_i) = \{u : e_i \textit{ is incident on } u\}$.

Vertices of M_G may also be called mutexes. We state a condition *R2'* on the mutex graph.

R2'. There are no cycles of odd length in the mutex graph MG corresponding to PG_1.

Lemma 8.1 $R2 \equiv R2'$.

Proof. Clearly $degree(PG_1) \geq 3 \Rightarrow \neg R2'$, and if there are cycles of odd length in PG_1 so are there in MG. Hence $\neg R2 \Rightarrow \neg R2$. Proving $\neg R2 \Rightarrow \neg R2$ is equivalent to showing that if MG has an odd cycle and $degree(PG_1) \leq 2$ then PG_1 also has an odd cycle, which follows from the construction of MG. □

We assign a type MUX or $DMUX$ to each vertex in the mutex graph MG using the Typing Algorithm given below. We will see that in order to subsequently construct a tree, the typing algorithm must assign different types to adjacent vertices in the mutex graph, which can only be done if the mutex graph satisfies *R2'*.

Def. A vertex set $M \subseteq V'$ of a mutex graph $MG = (V', E', L')$ is said to be *consistently typed* if for all $m, m' \in M$ such that $(m, m') \in E'$, $type(m) \neq type(m')$. A mutex graph is said to be consistently typed if the set of all its vertices is consistently typed.

Typing Algorithm.

Input. Mutex graph $MG = (V', E', L')$ satisfying *R2'*.

Output. Type function $type : V' \rightarrow \{MUX, DMUX\}$.

Let $type(m_1) = MUX$ for some $m_1 \in V'$

Let $A_1 = (V_1, E_1)$ with $V_1 = \{m_1\}$.

$i = 1$

Repeat

Choose some $m \in V' - V_i$ such that for some $m' \in V_i$, $(m, m') \in E'$.

Let $type(m)$ be the opposite of $type(m')$

Let $A_{i+1} = (V_{i+1}, E_{i+1})$ with$V_{i+1} = V_i \cup \{m\}$ and $E_{i+1} = E_i \cup \{(m, m')\}$.

$i = i + 1$

Until all vertices in V' are typed.

End algorithm.

Lemma 8.2 *Given a mutex graph satisfying R2', the Typing Algorithm produces a consistently typed mutex graph.*

Proof. By induction on the sequence $A_1, ..., A_p$. Wlog assume that the mutex graph $MG = (V', E', L')$ is strongly connected. Then by the construction so are all the A_i.

basis. Clearly A_1 is consistently typed.

hyp. A_i is consistently typed.

ind. Let m be the unique element of $V_{i+1} - V_i$. By construction there exists a vertex $m' \in V_i$ such that $(m, m') \in E'$ and $type(m) \neq type(m')$. Suppose there exists $m'' \in V_i$ such that $(m, m'') \in E'$ and $type(m) = type(m'')$. Then there exists a path of mutexes $(m' = n_1, n_2, ..., n_q = m'')$ in A_i since A_i is

strongly connected; in addition, $type(n_i) \neq type(n_{i+1})$, for $1 \leq i < q$, since A_i is consistently typed. The path $(n_1, n_2, ..., n_q)$ followed by m is a cycle of odd length, contradicting *R2'*. □

Lemma 8.3 *An architecture graph AG_2 satisfying Tree* can be constructed from a consistently typed mutex graph $MG = (V', E', L')$.*

Proof. By construction of $AG_2 = (V_2, E_2, L_2)$.

Vertices of AG_2. Let $V_2 = S \cup R \cup M \cup D \cup \{MT, DT, r\}$ defined as follows. S and R are sets of n vertices each of type $SUSER$ and $RUSER$ respectively. Set $M = \{m_i : m_i \in V' \wedge type(m_i) = MUX\}$, and $D = \{d_i : d_i \in V' \wedge type(d_i) = DMUX\}$. Set the types of the roots of the multiplexer tree, the demultiplexer tree, and of AG_2 as $type(MT) = MUX$, $type(DT) = DMUX$, and $type(r) = NULL$.

Arcs of AG_2. Let $E_2 = E_S \cup E_R \cup E_M \cup E_D \cup E_X \cup \{(MT, r), (r, DT)\}$ defined as follows. Add arcs from sending users to a vertex $m \in M$ if transfers from that user participate in m, i.e., let $E_S = \cup_{m \in M} E_m$ where $E_m = \{(s_i, m) : s_i \in S \wedge v_i \in L'(m)\}$. Similarly let $E_R = \cup_{d \in D} E_d$ where $E_d = \{(d, r_i) : r_i \in R \wedge v_i \in L'(d)\}$. Add arcs from the MUX vertices to the root of the multiplexer subtree, i.e., let $E_M = \{(m, MT) : m \in M\}$. Similiarly, let $E_D = \{(DT, d) : d \in D\}$. Finally, connect sending users whose transfers do not participate in mutexes directly to the root of the multiplexer subtree, and similarly for receivers, i.e., let $E_X = \{(s, MT) : s \in S \wedge degree(s) = 0\} \cup \{(DT, s) : s \in R \wedge degree(s) = 0\}$.

Labels of AG_2. Set the arc capacities of all links except those incident on the $NULL$ vertex to 1, i.e., for all $e \in E_2 - \{(MT, r), (r, DT)\}$ let $L_2(e) = 1$. Let $L_2((MT, r)) = L_2((r, DT)) = n$. □

8.2 Precedence constraints

In this section we consider the problem of scheduling data transfers under the presence of precedence constraints. Clearly, this is an especially important problem for the parallel I/O application. We will show, however, that even for situations in which tasks are of unit length and the precdence constraints are restricted to be in the form of a tree, the problem is NP-complete. In the following we specify the problem and review some well-known related NP-completeness results. We then observe the NP-completeness of our problem, and suggest avenues for further work.

We specify a restricted form of the precedence-constrained data transfer scheduling problem which we will later show to NP-complete.

$TreePrecDTS = (PG, AG, RG, f, Preempt)$

where

$PG = (T, Ep, Lp)$ has, for all $t \in T$, $Lp(t) = 1$, and is a tree.

$AG = (R, Ea, La)$ is a complete bipartite graph.

$RG = (R, Er, Lr)$ is a bipartite graph with $|Er| = |T|$.

f is makespan, $Preempt$ is true.

We contrast the $TreePrecDTS$ problem with two related scheduling problems, one of which is the well-known *Resource Constrained Scheduling* problem for multiprocessors [55, 56], a special case of which we call $TreeRCMS$, specified as follows.

$$TreeRCMS = (PG, AG', RG', f, Preempt)$$

where PG, f and $Preempt$ are as for $TreePrecDTS$.

$AG' = (R, Ea, La)$ is a bipartite graph, i.e., R is partitioned into a set of 'processors', R_p, and a set of 'other resources', R_r, but $Ea = \{\}$.

$RG' = (R, Er, Lr)$ has $R = R_p \cup R_r$ and $Er = \{\}$, i.e., the assignment of tasks to resource instances is not specified. The task resource requirement tr is that each task requires one processor and some number of other resources, i.e., for all $t \in T$, there exists k, $0 < k \leq |R_r|$, such that $tr(t) \in R_p \times R_r^k$.

It is known that $TreeRCMS$ is NP-complete [55]. However, we note that in $TreeRCMS$ the assignment of tasks to resources has to be computed as well as a schedule minimizing the makespan. Therefore it might be possible that if the assignment is fixed, finding the schedule is not NP-complete. In fact,

this turns out not to be the case, as shown by considering a related problem, *Processor-Bound Multiprocessor Scheduling* with tree precedences [64], which we call $TreePBMS$.

$TreePBMS = (PG, AG'', RG'', f, Preempt)$

where PG, f and $Preempt$ are as for $TreePrecDTS$ and $TreeRCMS$.

$AG'' = (R, Ea, La)$ consists of $|R|$ distinguished vertices, with $Ea = \{\}$.

$RG'' = (R, Er, Lr)$ has $|Er| = |T|$ consisting of self-loops on each vertex.

Thus $TreePBMS$ differs from $TreePrecDTS$ and $TreeRCMS$ in that each task requires only one resource, but also differs from $TreeRCMS$ in that the assignment of tasks to resource instances is known. Goyal [64, 56] has shown, using a reduction very similar to that used by Garey and Johnson [55], that $TreePBMS$ is NP-complete.

Observation. $TreePrecDTS$ is NP-complete.

The observation follows from the NP-completeness of $TreePBMS$, and noting that $TreePrecDTS$ is a special case of $TreePBMS$.

8.2.1 Further work

Since the $TreePrecDTS$ problem is of practical interest, it is useful to look for approximation algorithms for its solution. We are currently investigating

several such schemes [79].

A way of specifying the problem that may be useful for future work is to "merge" the resource and precedence graphs. Informally, the precedence graph is augmented by adding a hyperdge connecting any tasks that require the same resource instance (i.e., edges in the resource graph that have a common vertex). Then the edges of the resource graph can be deleted, since resource conflict information is already captured in the extended precedence graph. Note that the extended precedence graph can be simplified: if two vertices are connected by a directed edge, they need not be connected by a hyperedge even if they have a resource conflict. This extended precedence graph may be useful for designing heuristics that consider both the precedence and resource constraints between tasks simultaneously. (It can, of course, also be applied to problems in which there are no resource conflicts but tasks have logical mutual exclusion constraints as well as precedence constraints).

8.3 Discussion

8.3.1 Previous related work

The related work on precedence constrained scheduling [55, 64, 56, 98, and refernces therein] has already been reviewed in sec. 8.2 and Chapter 2.

To our knowledge, there has been no previous work on scheduling of tasks with explicit mutual exclusion constraints. Of course, implicit mutual exclusion constraints, such as situations where every task is mutually exclusive to

every other task since they all require the same resource, have been studied extensively. For instance, some of the recent work on scheduling in the presence of "exclusion constraints" between two tasks (e.g. [145]) actually refers to the restriction that if one task is executing, on a single processor, it may not be preempted by the other. Similarly, general resource constraints implicity define mutual exclusion constraints between tasks (e.g., see [130, 128] and references therein). However, *by specifying the mutual exclusion constraints implicitly, their logical structure is not apparent and cannot be exploited.* Thus general resource-constrained scheduling is NP-complete for the non-preemptive case and requires high-degree polynomial linear programming solutions in the preemptive case [56, 130]. In contrast, by considering the structure of explicit mutual exclusion constraints, we are able to specify a hierarchy of constraints that can occur in practice, and obtain a polynomial-time solution.

Almost all the previous work on logical constraints between tasks has focused on precedence constraints, which are logically stronger than mutual exclusion constraints and do not capture the synchronization requirements of tasks in some applications, particularly parallel programming. Some recent work has started to address this issue by weakening precedence constraints. Berger and Cowen [6] consider tasks which may be subject to precedence constraints (the usual partial order), as well as "concurrency constraints" (tasks that must be scheduled in the same time step) and "weak precedence constraints" (tasks that must be scheduled before, or at the same step as, some other task). They do not consider mutual exclusion constraints and simultaneous resource requirements, however.

8.3.2 Conclusions and further work

We have presented NP-completeness results for the problem of scheduling data transfers under the presence of tree-sructured precedence constraints.

We have also presented a solution to the problem of scheduling data transfer tasks when the tasks are subject to a restricted set of logical mutual exclusion constraints. Such constraints arise naturally in the parallel I/O application, and to our knowledge have not been previously systematically studied. While our results are limited to mutual exclusion constraints of a restricted class, they do apply to those allowed in the CODE 1.2 parallel programming environment. Further, the technique used in this chapter, of systematically transforming formal problem specifications, is promising and likely to be applicable to more general classes of mutual exclusion constraints, as well as other problems.

For further work, we suggest two questions. The first is whether the optimal algorithm we have developed, or a fast approximation algorithm based upon it, could be used in parallel programming environments such as CODE. The situation is quite promising since the mutual exclusion constraints are explicitly and deterministically specified by the user, and since an automatic programming system generates all the synchronization code, the algorithm could be used without the user having to be aware of it. The second question is whether tasks with mutual exclusion constraints could be combined with tasks under the constraints of Berger and Cowen's model [6], i.e., "concurrency constraints" and "weak precedence constraints". This would further

weaken the model of task interaction (compared to the usual model of partial orders, i.e., precedence constrained tasks) and allow scheduling in more realistic situations for applications such as parallel I/O.

Chapter 9

Conclusions and Further Work

We have studied the scheduling of data transfers, a problem of increasing importance in high-performance parallel computers and communications systems, particularly with the advent of advanced applications such as volume visualization and multimedia information systems.

Data transfer scheduling gives rise to an important and interesting class of simultaneous resource scheduling problems. We have defined a general graph-theoretical model for precisely specifying and classifying scheduling problems, and demonstrated its coverage of a wide range of traditional and simultaneous multiple resource scheduling problems. We have used the model for the recognition of the similarity of seemingly different problems from different application areas, for the systematic transformation of one problem specification into that of a seemingly different problem, and for the systematic decomposition of a problem specification into solvable subproblems.

We have obtained optimal and approximate algorithms for a wide range of problems, including communication architectures in which resources are fully

connected, communication architectures with a tree topology, and tree architectures in which both local and remote data transfers are permitted. We have also obtained results for scheduling data transfers under the presence of mutual exclusion constraints and precedence constraints. All these results either solve more general instances of the scheduling problem, or have better time complexity, or provide better approximations than previously known solutions, or all three. Finally, we have undertaken extensive experimenatal evaluations of our algorithms and determined the situations under which they operate best.

The results we have obtained are generally applicable to both parallel computers and communications systems. Specifically, they are applicable to certain types of shared-bus multiprocessor systems such as the Sequent [100], Encore [143], and the IBM RP3 [115]; bus-oriented local area networks such as the Ethernet; TDMA satellite switches [75]; hierarchical switching systems [39]; tree-structured multiprocessor architectures such as the Sequent [100], Tree Machine [133], KYKLOS [102], and Hector [138]; and intersatellite communications systems [7].

For future work, we have posed specific questions at the end of each chapter that relate to the topics studied in that chapter. Here we state some questions of broader practical and theoretical concern.

1. What is the range of architectures and exclusion constraints for which we can obtain optimal, polynomial-time solutions? In particular, can architectures such as the hypercube and mesh-based systems be covered? Is it possible to design and evaluate faster near-optimal solutions?

2. Is it possible to integrate data partitioning and allocation with data transfer scheduling so as to provide a better comprehensive approach to managing parallel I/O?
3. Is it desirable to integrate routing and scheduling of data transfers, in computer networks and multiprocessor architectures such as the hypercube? Is it possible to exploit the similarity of the solution techniques used for some routing and scheduling problems?
4. Can effective parallel algorithms be developed to perform data transfer scheduling?
5. Can the reasoning about the equivalence and transformation of scheduling problem classes using our scheduling model be formalized further into inference rules or general theorems?

BIBLIOGRAPHY

[1] J. Akella and D. P. Siewiorek. Modeling and measurement of the impact of Input/Output on system performance. In *Proc. 18th Intl. Symp. Comp. Arch.*, pages 390–399, 1991.

[2] M. Arrott and S. Latta. Perspectives on visualization. *IEEE Spectrum*, pages 61–65, Sep. 1992.

[3] Kenneth R. Baker. *Introduction to sequencing and scheduling.* John Wiley, 1974.

[4] H. Balan. Master's thesis, Dept. of Elect. and Comp. Eng., Univ. of Texas at Austin, 1990.

[5] Claude Berge. *Graphs.* North-Holland, 1985.

[6] B. Berger and L. Cowen. Complexity results and algorithms for $\{<, \leq, =\}$-constrained scheduling. In *Proc. Symp. on Discrete Alg.*, pages 137–147, 1991.

[7] A. A. Bertossi, G. Bongiovanni, and M. A. Bonuccelli. Time slot assignment in SS/TDMA systems with intersatellite links. *IEEE Trans. Comm.*, 35:602–608, June 1987.

[8] J. A. Bondy and U. S. R. Murty. *Graph theory with applications.* North-Holland, 1976.

[9] G. Bongiovanni, D. Coppersmith, and C. K. Wong. An optimum time slot assignment algorithm for an SS/TDMA system with variable number of transponders. Technical Report RC 8301 (# 35888), IBM T. J. Watson Research Center, 1980.

[10] G. Bongiovanni, D. Coppersmith, and C. K. Wong. An optimum time slot assignment algorithm for an SS/TDMA system with variable number of transponders. *IEEE Trans. Comm.*, 29(5):721–726, May 1981.

[11] M. A. Bonuccelli. A fast time slot assignment algorithm for TDM hierarchical switching systems. *IEEE Trans. Comm.*, 37:870–874, Aug. 1989.

[12] H. Boral and D. J. DeWitt. Database machines: An idea whose time has passed? A critique of the future of database machines. In *Third Intl. Workshop on Database Machines*, pages 166–187, 1983.

[13] H. Boral and P. Faudemay, editors. *Database machines.* Springer-Verlag, 1989.

[14] D. Bradley and D. A. Reed. Performance of the Intel iPSC/2 input/output system. In *Proc. Conf. on Hypercubes, Concurrent Comp. and Appl.*, pages 141–144, 1990.

[15] J. C. Browne, Muhammad Azam, and Stephen Sobek. CODE: A unified approach to parallel programming. *IEEE Software*, page 11, July 1989.

[16] J. C. Browne, A. Dale, C. Leung, and R. Jenevein. A parallel multistage I/O architecture with self-managing disk cache for database management applications. In *Fourth Intl. Workshop on Database Machines.* Springer-Verlag, 1985.

[17] J. C. Browne, G. E. Onstott, P. L. Soffa, Ron Goering, S. Sivaramakrishnan, Harish Balan, and Kiran Somalwar. Design and evaluation of external memory architectures for multiprocessor computer systems: Second quarter report to IBM Yorktown Heights Research Lab. Technical report, Univ. Texas at Austin, Dept. of Comp. Sci., 1987. Available from J. C. Browne.

[18] C. E. Catlett. Balancing resources. *IEEE Spectrum*, pages 48–55, Sep. 1992.

[19] S. Chalasani and A. Varma. Fast parallel time-slot assignment algorithms for TDM switching. In *Proc. Intl. Conf. Par. Proc.*, volume III, page 154, 1990.

[20] S. Chalasani and A. Varma. An improved time slot assignment algorithm for TDMA hierarchical switching systems. In *Proc. Fourth Intl. Conf. Data Comm. Sys. and their Perf.*, pages 116–132, 1990.

[21] W.-T. Chen and H.-J. Liu. An adaptive scheduling algorithm for TDM switching systems. In *Proc. IEEE Infocom*, pages 668–677, 1991.

[22] W.-T. Chen, P.-R. Sheu, and J.-H. Yu. Time slot assignment in TDM multicast switching systems. In *Proc. IEEE Infocom*, 1991.

[23] H.-A. Choi and S. L. Hakimi. Scheduling file transfers for trees and odd cycles. *SIAM J. Comput.*, 16(1):162–168, 1987.

[24] H.-A. Choi and S. L. Hakimi. Data transfers in networks. *Algorithmica*, 3:223–245, 1988.

[25] H.-A. Choi and S. L. Hakimi. Data transfers in networks with transcievers. *Networks*, 18:223–251, 1988.

[26] E. F. Codd. Multiprogram scheduling. *Comm. ACM*, 3, June 1960.

[27] E. G. Coffman, Jr., M. R. Garey, D. S. Johnson, and A. S. LaPaugh. Scheduling file transfers. *SIAM J. Comput.*, 3:744–780, 1985.

[28] E. G. Coffman, Jr. and R. L. Graham. Optimal scheduling for two-processor systems. *Acta Inf.*, 1:200–213, 1972.

[29] R. Cole and J. Hopcroft. On edge coloring bipartite graphs. *SIAM J. Comput.*, 11(3):540–546, 1982.

[30] R. W. Conway, W. L. Maxwell, and L. W. Miller. *Theory of Scheduling.* Addison-Wesley, 1967.

[31] D. de Werra. An introduction to timetabling. *European J. of Operational Res.* 19:151–162, 1985.

[32] T. A. DeFanti, M. D. Brown, and B. H. McCormick. Visualization: Expanding scientific and engineering research opportunities. *IEEE Computer*, pages 12–26, Aug. 1989.

[33] P. J. Denning. Effects of scheduling on file memory operations. In *Proc. AFIPS Spring Joint Comp. Conf.*, pages 9–21, 1967.

[34] N. Deo. *Graph Theory with Applications to Engineering and Computer Science.* Prentice-Hall, 1974.

[35] D. J. DeWitt. DIRECT - A multiprocessor organization for supporting relational database management systems. *IEEE Trans. Comp.*, June 1979.

[36] D. J. DeWitt, R. H. Gerber, G. Graefe, M. L. Heytens, K. Kumar, and M. Muralikrishna. GAMMA - A high performance dataflow database architecture. In *Proc. 12th Intl. Conf. on Very Large Data Bases*, Aug. 1986.

[37] G. Dobson and U. Karmarkar. Simultaneous resource scheduling to minimize weighted flow times. *Oper. Res.*, 37(4):592–600, 1989.

[38] E. W. Dusio, T. P. Murphy, and W. F. Cashman. Communications satellite software: A tutorial. *IEEE Computer*, pages 21–34, Apr. 1991.

[39] K. Y. Eng and A. S. Acampora. Fundamental conditions governing TDM switching assignments in terrestial and satellite networks. *IEEE Trans. Comm.*, COM-35:755–761, 1987.

[40] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM J. Comput.*, 5(4):691–703, 1976.

[41] S. Fiorini and R. J. Wilson. *Edge-colourings of graphs.* Pitman, London, U.K., 1977.

[42] L. R. Ford and D. R. Fulkerson. *Flows in networks.* Princeton University Press, 1962.

[43] E. A. Fox, editor. *CACM Special Issue on Digital multimedia systems.* ACM, Apr. 1991.

[44] J. C. French, T. W. Pratt, and M. Das. Performance meaurement of a parallel Input/Output system for the Intel iPSC/2 hypercube. In *Proc. SIGMETRICS*, pages 178–187, 1991.

[45] Simon French. *Sequencing and Scheduling.* John Wiley, 1982.

[46] A. M. Frieze. Probabilistic analysis of graph algorithms. In G. Tinhofer, E. Mayr, H. Noltemeir, and M. Syslo, editors, *Computational graph theory*, pages 209–233. Springer-Verlag, 1990. Also as *Computing Supp.*, vol. 7, Springer-Verlag, 1990.

[47] M. Fujii, T. Kasami, and K. Ninomiya. Optimal sequencing of two equivalent processors. *SIAM J. App. Math*, 17:784–789, 1969. Erratum, SIAM J. App. Math., vol. 20, p. 141, 1971.

[48] H. Gabow. Using euler partitions to edge color bipartite multigraphs. *Intl. J. Computer and Inf. Sci.*, 5:345–355, 1976.

[49] H. Gabow. An almost linear algorithm for two-processor scheduling. *J. Ass. Comp. Mach.*, 29:766–780, 1982.

[50] H. Gabow and O. Kariv. Algorithms for edge coloring bipartite multigraphs. *ACM Symp. Th. of Comp.*, pages 184–192, 1978.

[51] H. Gabow and O. Kariv. Algorithms for edge coloring bipartite graphs and multigraphs. *SIAM J. Comput.*, 11(1):117–129, 1982.

[52] H. Gabow and R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. In *Proc. 15th Ann. Symp. Theory of Comp.*, pages 246–251, 1983.

[53] A. Ganz and Y. Gao. Scheduling on SS/TDMA systems with intersatellite links. In *Proc. Intl. Conf. Comm.*, volume 1, pages 515 - 519, 1989.

[54] A. Ganz and Y. Gao. TDMA communication for SS/TDMA satellites with optical intersatellite links. In *Proc. Intl. Conf. Comm.*, pages 1081 - 1085, 1990.

[55] M. Garey and D. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM J. Comput.*, 4:397, Dec. 1975.

[56] M. Garey and D. Johnson. *Computers and intractability: A guide to the theory of NP-completeness.* Freeman, 1979.

[57] J. Ghosh and B. Agarwal. Parallel I/O subsystems for hypercube multicomputers. In *Proc. Intl. Par. Proc. Symp.*, pages 381–384, 1991.

[58] Alan Gibbons. *Algorithmic graph theory.* Cambridge University Press, 1985.

[59] G. A. Gibson. *Redundant disk arrays: Reliable, parallel secondary storage.* PhD thesis, Univ. of Calif., Berkeley, Comp. Sci. Div, 1990. Also available as Tech. Rep. UCB/CSD 91/613.

[60] Mario Gonzalez, Jr. Deterministic processor scheduling. *Computing Surveys*, 9:173, Sept. 1977.

[61] T. Gonzalez and D. B. Johnson. A new algorithm for preemptive scheduling of trees. *J. Ass. Comp. Mach.*, 27:287–312, 1980.

[62] T. Gonzalez and S. Sahni. Open shop scheduling to minimize finish time. *J. Ass. Comp. Mach.*, 23:665–679, 1976.

[63] C. C. Gotlieb. The construction of class-teacher timetables. In *Proc. IFIP Congress*, pages 73–77, 1962.

[64] D. K. Goyal. Scheduling processor bound systems. Technical Report CS-76-036, Washington State Univ., 1976.

[65] H. Hadimioglu and R. J. Flynn. The architectural design of a tightly-coupled distributed hypercube file system. In *Proc. Conf. on Hypercubes, Concurrent Comp. and Appl.*, pages 147–150, 1989.

[66] B. Hajek and G. Sasaki. Link scheduling in polynomial time. *IEEE Trans. Info. Th.*, 34:910–917, 1988.

[67] B. Hancock. Multiprocessors are NOT always better. *Digitial Rev.*, page 59, Dec. 2 1991.

[68] J. Hennessy and D. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann, San Mateo, CA, 1990.

[69] D. S. Hochbaum, T. Nishizeki, and D. B. Shmoys. A better than "best possible" algorithm to edge color multigraphs. *SIAM J. Comput.*, 7:79–104, 1986.

[70] I. J. Holyer. The NP-completeness of edge colourings. *SIAM J. Comput.*, 10:718–720, 1980.

[71] J. Hopcroft and R. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2:225–231. 1973.

[72] T. C. Hu. Parallel sequencing and assembly line problems. *Operations Research*, 9:841–848, 1961.

[73] IEEE. *Proc. Intl. Conf. Univ. Pers. Comm.*, 1992. Held Sep. 29 - Oct. 2, 1992, at Dallas, TX.

[74] T. Inukai. An efficient SS/TDMA time slot assignment algorithm. *IEEE Trans. Comm.*, COM-27:1449–1455, 1979.

[75] Y. Ito, Y. Urano, T. Muratani, and M. Yamaguci. Analysis of a switch matrix for an SS/TDMA system. *Proc. IEEE*, 65:411–419, 1977.

[76] Ravi Jain. Scheduling I/O in parallel computing environments. Unpublished manuscript, Dec. 1990.

[77] Ravi Jain and Galen Sasaki. Scheduling packet transfers in a class of TDM hierarchical switching systems. In *Proc. Intl. Conf. Comm.*, 1991.

[78] Ravi Jain, Kiran Somalwar, John Werth, and J. C. Browne. Scheduling parallel I/O operations in multiple-bus systems. *J. Par. and Distrib. Comp.*, Dec. 1992. Special Issue on Scheduling and Load Balancing.

[79] Ravi Jain and John Werth. Precedence constrained I/O scheduling. Unpublished manuscripts, 1992.

[80] Ravi Jain, John Werth, and J. C. Browne. A general model for scheduling of parallel computations and its application to parallel I/O operations. In *Proc. Intl. Conf. Par. Proc.*, 1991.

[81] Ravi Jain, John Werth, J. C. Browne, and G. Sasaki. A graph-theoretic model for the scheduling problem and its application to simultaneous resource scheduling. In *ORSA Conf. on Computer Science and Operations Research: New Developments in their Interfaces*, Jan. 1992. Available from Pergamon Press.

[82] W. Jilke. Disk array mass storage systems: The new opportunity. Technical report, Amperif Corp., Sep. 1986.

[83] C. V. Jones. The three-dimensional Gantt chart. *Oper. Res.*, 36(6):891–903, 1988.

[84] H. Jordan. Scalability of data transport. In *Proc. Scalable High Perf. Computing Conf.*, pages 1–8, 1992.

[85] A. Kandappan. Data allocation and scheduling for parallel I/O systems. Master's thesis, Dept. of Elect. and Comp. Eng., Univ. of Texas at Austin, 1990.

[86] H. J. Karloff and D. B. Shmoys. Efficient parallel algorithms for edge coloring problems. *J. Algorithms*, 8:39–52, 1987.

[87] K. N. Karna and E. W. Dusio. Communications satellite software. *IEEE Computer*, pages 15–16, Apr. 1983. Special Issue on communications satellite software.

[88] M. Y. Kim. Synchronized disk interleaving. *IEEE Trans. Comp.*, C-35, 1986.

[89] S. J. Kim and J. C. Browne. A general approach to mapping of parallel computations upon multiprocessor architectures. In *Proc. Intl. Conf. Par. Proc.*, pages 1–8, 1988.

[90] T. Kwok. Communications requirements of multimedia applications: A preliminary study. In *Proc. Intl. Conf. Univ. Pers. Comm.*, 1992.

[91] S. Lam and R. Sethi. Worst case analysis of two scheduling algorithms. *SIAM J. Comput.*, 6:518–536, 1977.

[92] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, 1976.

[93] E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Recent developments in deterministic sequencing and scheduling: A survey. In

Deterministic and Stochastic Scheduling, pages 35–73. D. Reidel Publishing, 1982.

[94] P. L'Eculyer. Efficient and portable combined random number generators. *Comm. ACM*, 31:742–774, June 1988.

[95] J. K. Lenstra and A. H. G. Rinnooy Kan. Complexity of scheduling under precedence constraints. *Oper. Res.*, pages 22–35, 1978.

[96] S. C. Liew. Comments on "Fundamental conditions governing TDM switching assignments in terrestrial and satellite networks". *IEEE Trans. Comm.*, 37:187–189, Feb. 1989.

[97] M. Livny, S. Khoshhafian, and H. Boral. Multi-disk management algorithms. In *Proc. SIGMETRICS*, May 1987.

[98] E. L. Lloyd. Concurrent task systems. *Oper. Res.*, 29:189–201, 1981.

[99] C. Lo, R. S. Wolff, and R. C. Bernhardt. An estimate of network database transaction volume to support universal personal communications services. In *Proc. Intl. Conf. Univ. Pers. Comm.*, 1992.

[100] T. Lovett and S. Thakkar. The Symmetry multiprocessor system. In *Proc. Intl. Conf. Par. Proc.*, pages 303–310, 1988.

[101] Weizhen Mao. Directed file transfer scheduling. Submitted for publication, 1992.

[102] B. Menezes and R. Jenevein. KYKLOS: A linear growth fault-tolerant interconnection network. In *Proc. Intl. Conf. Par. Proc.*, pages 498–502, 1985.

[103] E. Miller. Input/Output behavior of supercomputing applications. Technical Report UCB/CSD 91/616, Univ. California, Berekeley, 1991.

[104] W. D. Moren. Disk array: You know it when you see it. *Workstation News*, Apr. 1992.

[105] B. M. E. Moret, 1992. Private communication.

[106] B. M. E. Moret and H. D. Shapiro. *Algorithms from P to NP, Volume 1: Design and efficiency.* Benjamin-Cummings, 1991.

[107] T. N. Mudge, J. P. Hayes, and D. C. Winsor. Multiple bus architectures. *Computer*, 20(6):42–48, June 1987.

[108] R. R. Muntz and E. G. Coffman, Jr. Optimal preemptive scheduling on two-processor systems. *IEEE Trans. Comp.*, C-18:101, 1969.

[109] R. R. Muntz and E. G. Coffman, Jr. Preemptive scheduling of time tasks on multiprocessor systems. *J. Ass. Comp. Mach.*, 17:324–338, 1970.

[110] G. M. Nielson, editor. *IEEE Computer Special Issue on Scientific Visualization.* IEEE, Aug. 1989.

[111] R. G. Ogier. A decomposition method for optimal link scheduling. In *Proc. Allerton Conf. Comput. Comm*, pages 822–823, 1986.

[112] Krishna Palem. *On the complexity of precedence constrained scheduling.* PhD thesis, Univ. Texas at Austin, Dept. of Comp. Sci., 1986. Available as Tech. Rept. TR-86-11.

[113] D. A. Pattreson, G. A. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proc. SIGMOD*, 1988.

[114] P. G. Paulin and J. P. Knight. Force-directed scheduling for the behavioral synthesis of ASIC's. *IEEE Trans. Comp.-Aided Design*, pages 661–679, 1989.

[115] G. Pfister, W. C. Brantley, D. A. George, S. L. Harvey, W. J. Kleinfelder, K. P. McAuliffe, E. A. Melton, V. A. Norton, and J. Weiss. The IBM research parallel processor (RP3): Introduction and architecture. In *Proc. Intl. Conf. Par. Proc.*, pages 764–771, 1985.

[116] P. Pierce. A concurrent file system for a highly parallel mass storage system. In *Proc. Conf. on Hypercubes, Concurrent Comp. and Appl.*, pages 155–160, 1989.

[117] A. Pizzarello and F. Golshani. In-memory databases: An industry perspective. In *Proc. Workshop on Res. Iss. in Data Eng.*, pages 96–101, 1992.

[118] T. Pratt, J. French, P. Dickens, and Jr. S. Janet. A comparison of the architecture and performance of two parallel file systems. In *Proc. Conf. on Hypercubes, Concurrent Comp. and Appl.*, pages 161–166, 1989.

[119] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical recipes: The art of scientific computing.* Cambridge, 1986.

[120] W. Rash. Multimedia moves beyond the hype. *Byte*, pages 85–87, Feb. 1992.

[121] A. L. N. Reddy and P. Banerjee. Design, analysis and simulation of I/O architrctures for hypercube multiprocessors. *IEEE Trans. Par. and Distrib. Sys.*, pages 140–151, Apr. 1990.

[122] R. T. Rockafellar. *Network flows and monotropic optimization.* John Wiley, 1984.

[123] F. Rossi, C. Petrie, and V. Dhar. On the equivalence of constraint satisfaction problems. In *Proc. Euro. Conf. on Art. Intel. (ECAI90)*, 1990.

[124] K. Salem and H. Garcia-Molina. Disk striping. In *Proc. IEEE Intl. Conf. Data Eng.*, 1986.

[125] Galen Sasaki and Ravi Jain. Scheduling data transfers in preemptive hierarchical switching systems, 1991. Submitted to IEEE Trans. Comm.

[126] Galen Sasaki and Ravi Jain. Scheduling data transfers in preemptive hierarchical switching systems with applications to packet radio networks. In *Proc. Infocom*, 1991.

[127] R. K. Schultz and R. J. Zingg. Response time analysis of multiprocessor computers for database support. *ACM Trans. Database Sys.*, pages 14–17, 1984.

[128] Chia Shen, K. Ramamritham, and J. A. Stankovic. Resource reclaiming in real time. *IEEE Real-Time Sys. Symp.*, pages 41–50, 1990.

[129] A. Silberschatz and J. Peterson. *Operating systems concepts.* Addison-Wesley, 1988.

[130] R. Slowinski and J. Weglarz. *Advances in project scheduling.* Elsevier Science Pub., Amsterdam, 1989.

[131] J. E. Smith, W. C. Hsu, and C.Hsuing. Future general purpose supercomputer architectures. In *Proc. Supercomp. '90*, pages 796–804, 1990.

[132] Kiran Somalwar. Data transfer scheduling. Technical Report TR-88-31, Univ. Texas at Austin, Dept. of Comp. Sci., 1988.

[133] S. W. Song. A highly concurrent tree machine for database applications. In *Proc. Intl. Conf. Par. Proc.*, pages 259–268, 1980.

[134] J. D. Ullman. NP-complete scheduling problems. *J. Computer and Sys. Sci.*, 10:384–393, 1975.

[135] J. D. Ullman. Complexity of scheduling problems. In E. G. Coffman, Jr., editor, *Computer and job-shop scheduling theory.* John Wiley, 1976.

[136] A. Varma and S. Chalasani. An incremental time-slot assignment algorithm for TDM hierararchical switching systems. In *Proc. IEEE Intl. Conf. Comm.*, pages 1554–1558, 1991.

[137] V. G. Vizing. On an estimate of the chromatic class of a p-graph. *Diskret. Analiz.*, 3:25–30, 1964. In Russian. See Gabow, 1976.

[138] Z. G. Vranesic, M. Stumm, D. M. Lewis, and R. White. Hector: A hierarchically structured shared-memory multiprocessor. *Computer*, pages 72–79, Jan. 1991.

[139] S. B. Weinstein. *IEEE Spectrum*, 1985.

[140] John Werth, Dwip Banerjee, J. C. Browne, Ravi Jain, Steve Lin, Peter Newton, Ravi Rao, and Steve Sobek. CODE 1.2 User Manual and Tutorials. Technical Report TR-90-35, Univ. Texas at Austin, Dept. of Comp. Sci., November 1990.

[141] John Werth, J. C. Browne, Steve Sobek, T. J. Lee, Peter Newton, and Ravi Jain. The interaction of the formal and practical in parallel

programming environment development: CODE. Technical Report TR-91-09, Univ. Texas at Austin, Dept. of Comp. Sci., 1991.

[142] Jennifer Whitehead. The complexity of file transfer scheduling with forwarding. *SIAM J. Comput.*, 19(2):222–245, Apr. 1990.

[143] A. W. Wilson Jr. Hierarchical cache/bus architecture for shared memory multiprocessors. In *14th Intl. Symp. Comp. Arch.*, pages 244–252, 1987.

[144] R. H. Wolfe, Jr. and C. N. Liu. Interactive visualization of 3D seismic data: A volumetric method. *IEEE Comp. Graphics Appl.*, pages 24–30, July 1988.

[145] J. Xu and D. L. Parnas. Scheduling processes with release times, deadlines, precedence, and exclusion relations. *IEEE Trans. Soft. Eng.*, 16:360–369. Mar. 1990.

[146] W. Zhao, K. Ramamritham, and J. A. Stankovic. Preemptive scheduling under time and resource constraints. *IEEE Trans. Comp.*, page 949, Aug. 1987.

VITA

Ravi Jain was born on July 3, 1960, at Simla, India. After completing high school in Kitwe, Zambia, he received the B.Sc. in Electronics Engineering from The City University, London, in 1980. He obtained an M.S.E.E. from Penn State University in 1982, where his research focused on modeling energy deposition in the auroral ionosphere. From 1982 to 1985 he worked at Syntrex Inc. on communications and systems software for a microcomputer system, a local area network, and a fault-tolerant file server. He later worked at SES Inc. on performance modeling of communications systems, and at the Schlumberger Laboratory for Computer Science on high-level parallel programming.

Jain has been an MCD Fellow at the University of Texas at Austin. His research interests include resource management in parallel and distributed computers, communications protocols, discrete algorithms, and performance analysis. Jain has several refereed publications, and has served as a referee for numerous conferences and journals. Jain is a member of the Upsilon Pi Epsilon and Phi Kappa Phi honorary societies, as well as ACM, IEEE, and CPSR. Jain's current address is Bellcore, 445 South Street, Morristown, NJ 07962.

Permanent address: 114 East 31st St., #311
Austin, TX 78705