On the Furthest-Distance-First Principle in Scattering and Gathering

Jyh-Pin Tsai
Department of Computer Science and Information Engineering
National Taiwan University

Contents

Chapter 1

Introduction

In many parallel algorithms, factors such as the message size, **packetization** (how to split a message into packets to various sizes), the set-up time for sending a packet, queueing delays, and the order of dispatching packets can interact with one another in complicated ways to affect performance of transmission. In this thesis, we investigate these issues that arise from the scattering problem. In the **data scattering** problem a designated processor sends distinct packets of arbitrary sizes (including zero) to every other processor in the network [29]. This is an important parallel primitive and has been studied under various names such as one-to-all personalized communication [15] and distributing problem [30]. A more general operation is multiscattering, where a scattering operation is performed by every node [29]. **Data gathering** problem is another major subject besides data scattering. The reverse of the data scattering problem, the data gathering problem means that every other processors except the designated processor transmit distinct packets of arbitrary sizes to the designated processor. It is called all-to-one communication problems in the literature.

A realistic and widely accepted model assumes the time to transmit a packet of size L (measured in, say, bytes) across a link is $\beta + L\tau$, where β is the **set-up** time, which accounts for the start-up and termination time, and τ is the bandwidth of the link expressed in, say, bytes per second [9], [15], [28], [29]. Typically, $\beta \gg \tau$. We will use **flit** instead of byte to denote the unit of transmission. The concept of flit (flow control digit) comes from the work on wormhole routing [6]. From now on, the size of a packet or message will be measured in flit, and τ in flits/second. Without loss of generality, we may assume τ to be unity, for we can multiply the whole expression by $1/\tau$ to make it true and multiply the final timing result by a factor of τ. Hence, from now on, our timing model is simply this:

it takes $\beta + L$ time to transmit a packet of size L across a link.

In the **single-port** (also called **processor-bound** [30]) model, a node can send and receive information from only one of its immediate neighbors during each communication step. This is in contrast to the **multi-port** model (also called **link-bound** [30]) where a node can exchange information along all its links at the same time. We shall assume the single-port model, which is a more realistic one. We also assume the store-and-forward or packet-switched model, where a node receiving a packet must finish receiving it before any of its contents can be sent; this is in contrast to the circuit-switched model.

A well-known heuristic for efficient data scattering is the **furthest-distance-first** (FDF) principle whereby the messages, without any splitting or regrouping during the process, are sent in reverse order of their distance from their destinations. This is proposed for scattering data on a ring by Saad and Schultz [29]. Fraigniaud, Miguet, and Robert [9] show that the above scheme is optimal, even if one allows arbitrary splitting or regrouping in the process, as long as each message is of equal size and every node has to receive a message. Bhatt, Pucci, Ranade, and Rosenberg [2] show that every FDF scattering scheme on a tree is optimal. Note that a linear array is a special kind of tree. They allow messages to vary in sizes (including zero). However, they disallow buffering and queueing delays by requiring that each message be split into single-flit packets and that flits from the same packet be send contiguously without interruption. Under such assumptions, of course, set-up time is irrelevant (indeed, $\beta = 0$ is assumed for simplicity). Scattering on the hypercube has been studied by Johnsson and Ho [15].

In the data gathering problem, **the nearest-receive-first** (NRF) principle is benefitial. The NRF principle suggests that the dispatch time (dispatch order) of those packets sent by the processors other than the designated processor is decided by the distance of the path from the source processor to the destination processor (the designated processor). The nearer the distance of the path is, the earlier the packet of that source processor is sent. Bhatt, *et al.* also show that the NRF principle is optimal on a linear array. But it doesn't work on a general tree. As a result, they propose a new algorithm — Transmission Certification — to solve it [2]. That algorithm is also useful when setup time and packetization are involved.

Whatever the problems, our results can be summarized as follows. The issue of queueing delay can be handled easily, and the maximum buffer size needed for any scheme is at most that of the largest packet, the tightest bound possible. We show that not only the FDF principle but also the NRF principle no longer produces optimal running time on trees when the set-up time and packetization are allowed. On the other hand, the FDF and NRF principles still work when applied to linear arrays. We develop a general approach that used the theory of continued fractions to compute packetizaiton efficiently, taking into account the set-up time, so that a message uses the least amount of time to reach its destination in the absence of other messages. We consider a special class of data scattering problems where every node receives a message of the same length (this is more general that the situation considered by Fraigniaud, *et al.* [9]) and show that messages going to contiguously located nodes have to be split into the same number of packets. If, in addition, nodes that receive a message are *all* contiguously located on the linear array, then the problem can be solved in polynomial time. Packetization and queueing delays are allowed. Finally, given an arbitrary network, among all its spanning trees, the FDF and NRF principle can be applied to any breadth-first ones to produce the optimal schedule, again in the absence of set-up time, thus extending the result of Bhatt, *et al.* [2]

Clearly, the set-up time is never zero, as it reflects the fixed cost of setting up communication between two neighboring nodes. Indeed, set-up time can easily be two orders of magnitude larger than the time to transmit a single byte; see Table 1 of [25] for data on the NCUBE, Table 2 of [28] and Figure 4.2 of [8] for data on the Intel iPSC and the Ametek S/14, and [3] for data on the Connection Machine CM-5 and the Intel Paragon. As the message gets bigger, such as a file, splitting it is also a good strategy, which is why it is used in the APARNET [31]. When multiple paths exist between two nodes, packetization is also beneficial [15].

The set-up time affects how a message is to be **split** into smaller **packets**, a process called packetization. We will not allow information from *different* messages to form a packet. The set-up time and the ability to split packets can complicate things a lot.

Intuitively, transmitting smaller packets instead of the original longer message can lead to shorter total transmission time due to the pipelining effect. However, since *each* packet incurs a set-up time of magnitude β, the benefit of splitting tends to diminish as the set-up time or the number of packets a message is split into increases. A simple example can make this point clear. Suppose we are required to send a message of size L to a node located l nodes away. Without splitting , this obviously takes $l(\beta + L)$ time. Now, suppose we split it into a packets of equal size, L/a (forget about the complex integrality issue for simplicity now). The running time becomes $(l + a - 1)(\beta + L/a)$, which is minimized when a is equal to $\sqrt{L(l-1)/\beta}$. One can easily see that, if a is chosen optimally as above, the resulting running time can be smaller or larger than $l(\beta + L)$, which corresponds to the case where $a = 1$, depending on the values of L, l, β; see Figure 1.1. The general trend is that, if β is big, making each transmission across a link very expensive to set up , then a message should not be split into too many packets (i.e., a smaller a is favored). On the other hand, if either L or l is big, then the opposite is true. Similar observations have been made by Saad and Schultz [29]. There, as here, the issue of integrity is ignored; in fact, most, if not all, previous authors ignore the effects of rounding or truncating [15], [29], [30]. Indeed, as in the case of linear and integer programming [18], [26], the choice of optimal a under the integrality constraint gives rise to complications. The integrality problem is basically the optimization of a linear function on a convex set defined by a quadratic function, all in two variables. We shall show that this problem is polynomial-time solvable.

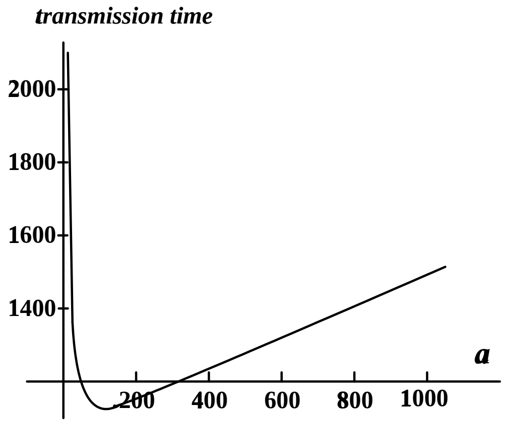

Figure 1.1: The trade-off between large packets (smaller a) and small packets (larger a). When $a = 1$, it corresponds to the case where the message is not split. In this particular example with $L = 1000$, $l = 10$, and $\beta = 0.5$, such does not produce minimal transmission time; nor does very large a.

This thesis is organized as follows. In Chapter 2, definitions and terminology are given. We prove in Chapter 3 several lemmas that turn out to be useful later. We also prove that the FDF principle can produce optimal performance on a linear array as well as analyze the queueing delay and buffer congestion. Finally, we prove that any breadth-first tree can be used with the FDF principle to produce optimal results as long as the set-up time is zero. In Chapter 4, the non-optimality of the FDF principle for trees is demonstrated by a counter-example, and we show that, in the absence of set-up time, allowing queueing delays and splitting a message into packets of size greater than one flit do not help in lowering the running time. In Chapter 5, several lemmas are proved to lead ultimately to an integer optimization problem, which is equivalent to the packetization problem when there is only one message to send. In Chapter 6, the packetization problem for a special class of data scattering problems is analyzed and solves in polynomial time. After solving the data scattering problem, we focus on the data gathering problems in Chapter 7. Those issues appeared in previous chapters will be discussed again about gathering problem on a general trees, such as packetization, set-up time and so on. The Transmission-Certification algorithm in [2] will be adjusted to the new conditions, especially with regards to the packetization. As a consequence, we can also find the special solution for the special case about linear arrays. Finally, a conclusion is given in Chapter 8.

Chapter 2

Definitions and Terminology

Let processors P_0, P_1, $\cdots$, P_n form the nodes of a tree with P_0 as the root. Node P_0 is to send message M_i of size L_i to node P_i with each L_i being a non-negative integer. Before the scattering operation begins, a message of positive size may be split into small packets. We do not consider the more general case, unless noted otherwise, where those packets may be grouped with other packets from other messages to form a single packet or be split further in the process. A packet of size one, or one flit, is the smallest unit of communication; that is, the size of each transmitted packet must be some positive integer. As mentioned before, a packet of size L take $\beta + L$ time to cross a link, where β is the set-up time.

Each node has a buffer for storing packets which have to await other packets to cross the link. A node extracts those of the incoming packet's contents that are destined to it first before depositing the rest of its contents into the buffer. A **queueing strategy** at a node is simply a procedure that decides which bits in its buffer to form the next packet and when to send it. A queueing strategy is **first-come-first-serve** (FCFS) if incoming packets are treated as atomic without being split or grouped with other packets' contents. From now on, unless stated otherwise, we shall only concern ourselves with FCFS queueing strategies.

Root node P_0 sends out a sequence of packets, S_1, S_2, $\cdots$, where each S_i is a packet from some message, say M_j, but cannot contain information from more than one message(i.e., no **grouping**). Let $\mathcal{S} = \{S_1, S_2, \cdots\}$ be the **dispatching strategy** and $\mathcal{M} = \{M_1, M_2, \cdots\}$ denote the set of messages to be sent by the root. A **schedule** for scattering M is completely described by the dispatching strategy and the queueing strategy at each node. We use $|P|$ to denote the size of packet P. A packet's **dispatch** time at a node is defined to be the time when it starts being set up for transmission. A packet's **departure** time at a node is the time when it is completely received. Hence the departure time of a packet at a node is *equated* with its **arrival** time at the next node. A packet's arrival time will mean the time it arrives at its destination. Clearly, packet S_i's departure time from the root is $\sum_{j=1}^{i}(\beta + |S_j|)$.

A schedule satisfies the FDF **principle** if a packet to a further node is dispatched before a packet to a nearer node. Note that, with FCFS queueing strategies, if the FDF strategy is applied at the root, then the FDF principle is automatically satisfied for each node as well. A dispatching or a queueing strategy is **greedy** if it sends out the next packet the moment the link is free. Apparently, greedy strategies do not increase the running time; hence, without loss of generality, all strategies are greedy in this paper. We say packets S_i and S_{i+1} are **out of order** or **violate** the FDF principle if the former's destination is nearer the root than the latter's ; see Figure 2.1.

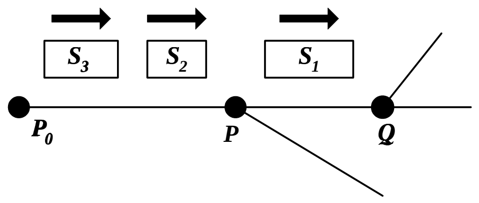

Figure 2.1: Packets S_2 and S_3 violate the FDF principle if P is S_2's destination and Q is S_3's.

As a packet travels, it may encounter arbitrary queueing delays. A packet of size L with dispatch time t will reach a node located l nodes away from the root at time $t + l(\beta + L)$ and be dispatched from there at the same time, if there are no delays. We say it is **delayed** by T(**steps**) at a node P if its departure time is $T + t + (\beta + L)$, where t is its arrival time at P and L its size. Here we allow T to be zero or even negative, to be general. It is delayed by packet b by the amount of T if, furthermore, it is dispatched immediately after b at P and $T \geq 0$. Finally, we simply say it is delayed by packet b at P if $T > 0$. We shall assume S_1's dispatch time from the root is 0.

Chapter 3

Optimality of FDF and Some Preliminaries

This section contains results that will be used later. These results are mostly about how queueing and the order of dispatching packets affect the running time.

3.1 FDF and optimal schedules

The following lemma show we can exchange the dispatch order of two adjacent packets that violate the FDF principle without increasing the running times, as long as they lie on the same branch of the tree.

Lemma 1 *If S_i's destination is between the root and S_{i+1}'s destination, violating the FDF principle, their dispatch order can be exchanged without increasing the running time.*

Proof:

Suppose in schedule ρ, packet S_i and S_{i+1} violate the FDF principle; that is, S_i's destination is P and S_{i+1} is Q, but P is closer to the root than Q and $P_0 \to P_1 \to P_2 \to \cdots \to P_k$, where $P = P_j$, $Q = P_k$, and $j < k$.

Consider schedule ρ^* that is the same as ρ except that S_i's and S_{i+1}'s dispatch order are switched.

We consider that ρ^*'s running time does not increase. Apparently, those packets S_l with $l < i$ are not affected by such a switch, as they are dispatched earlier and the queueing strategy is FCFS. So we only have to concern ourselves with the arrival times of those S_l with $l \geq i$.

Case 1: $|S_{i+1}| \geq |S_i|$ (see Figure 3.1). We first claim that

(a) in both ρ and ρ^*, the arrival time of S_{i+1} at its destination is at least as large as that of S_i at its destination.

Clearly, in ρ, this is the case, as S_{i+1} is dispatched later than S_i and has to go further. For ρ^*, on the other hand, although S_{i+1} is dispatched ahead of S_i, it arrives at a node P_m at least as late as S_i arrives at a previous node P_{m-1} for $2 \leq m \leq j+1$, because its size is as least as big. Hence, when S_i arrives at P_j, S_{i+1} can at best arrive at P_{j+1}.

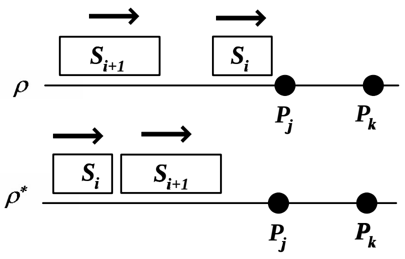

Figure 3.1: Case 1 of Lemma 1's proof

We further claim that

(b) S_{i+1}'s departure time from a node P in ρ is exactly S_i's departure time from P in ρ for $P < P_j$, S_i's destination,

and

(c) S_{i+1}'s departure time at a node P in ρ^* is exactly $\beta + |S_i|$ less than S_{i+1}'s departure time at P in ρ for $P < P_j$.

As the size of S_i is at most that of S_{i+1}, it is clear that, in ρ^*, S_i's arrival time at a node is at most S_{i+1}'s departure time at the same node; that is , S_i follows S_{i+1} without "gaps", intuitively speaking. Therefore, in ρ^*, the departure time of S_i at a node is exactly that of S_{i+1} plus $\beta + |S_i|$, the time needed to transmit S_i. So, we only have to argue for (c), and (b) will follow.

Now, (c) certainly holds at the root node P_0. Consider, using induction, node P_m for $m < j$. If S_{i+1} in ρ^* is delayed by earlier packets at that node, then, as it cannot move faster than S_i of ρ because it is not shorter, S_i is also delayed there. In such a case, both will be dispatched from that node at the same time when earlier packets are cleared, and S_{i+1} of ρ arrives at the node exactly $\beta + |S_i|$ after S_{i+1} of ρ^*. Suppose, on the other hand, S_{i+1} in ρ^* is not delayed at P_m. By the induction hypothesis, S_{i+1} departs from P_{m-1} in ρ exactly $\beta + |S_i|$ after S_{i+1} of ρ^* does. Hence, when it arrives at P_m, packet S_i is already cleared, which is true because, not longer than S_{i+1},packet S_i in ρ is dispatched at least as early as S_{i+1} in ρ^*. As a consequence, S_{i+1}'s departure time from that node in ρ is again exactly $\beta + |S_i|$ less than S_{i+1}'s in ρ^*.

Claims (b) and (c) say up to the time when S_{i+1} of ρ arrives at node P_j, the very time when S_i of ρ^* arrives at the same node, packets that are dispatched *later*, that is $\{S_{i+2}, S_{i+3}, \cdots\}$, have exactly the *same* departure times up to node P_j in either schedule. From that point on, since S_{i+1} of ρ^* arrives at node P_j earlier than S_{i+1} of ρ (by the amount of $\beta + |S_i|$), it will be dispatched no later. This means it will reach its destination, P_k, no later and packets dispatched later than S_i in ρ^* that go beyond P_j will be dispatched no later than those same packets in ρ and reach their destinations no later. This combined with (a) establishes the lemma for Case 1.

Case 2: $|S_{i+1}| < |S_i|$ (see Figure 3.2). We first claim that

(d) S_{i+1}'s departure time from a node P in ρ is exactly S_i's departure time from P in ρ^* for $P < P_j$, S_i's destination.

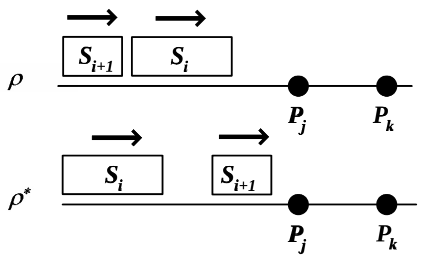

Figure 3.2: Case 2 of Lemma1's proof

Using the same argument as leads to (c), we can easily show that S_i's departure time at a node P in ρ^* is exactly $\beta + |S_{i+1}|$ larger than S_i's departure time at P in ρ for $P < P_j$. Now, (d) is clear as S_{i+1} follows S_i in ρ without "gaps", it takes $\beta + |S_{i+1}|$ time to transmit.

Claim (d) and the simple observation that S_{i+1} of ρ is dispatched later than S_{i+1} of ρ^*

mean the arrival time of S_{i+1} in ρ is no earlier than that of S_{i+1} in ρ^*. Furthermore, we have seen that the arrival time of S_i in ρ^* at its destination, P_j, is exactly that of S_{i+1} in ρ at P_j. Hence we know that the larger of the arrival times of S_i and S_{i+1} in ρ^*. To complete the proof for Case 2, we only have to show that packets dispatched after S_i in ρ^*, $\{S_{i+2}, S_{i+3}, \cdots\}$, reach their destinations at least as early as those in ρ. Surely, (d) guarantees that those packets in either schedule have the same departure times from any node $P < P_j$. From node P_j and beyond, as S_{i+1} of ρ^* is at least as ahead as S_{i+1} of ρ, those packets under consideration will be less restrained in ρ^*, in the sense that it is easier for them to catch up with S_{i+1} and get delayed in ρ than in ρ^*.

So in either case, we find we can exchange out-of-order dispatches without increasing the running time. This proves the lemma. □

We note that the above lemma is applicable to trees and can be easily employed to prove that, for any message set $\mathcal{M}$, there exists an optimal FDF schedule on linear arrays.

Theorem 2 *On a linear array, there exists an optimal FDF schedule for any data scattering problem.*

Proof:

Apply Lemma 1 to every pair of out-of-order packets. After $O(|\mathcal{S}|^2)$ exchanges, we will get an FDF schedule that is as good as the original schedule. □

3.2 How delays happen and manifest themselves

We say $\langle T_1, T_2, T_3, ..., T_m \rangle_i$ forms a **delay** sequence if T_j delays T_{j+1} at node i. Note that all packets in such a delay sequence must pass through node i in the order as specified in the sequence on their way to their destinations. Note also that, although it means T_{j+1} must be dispatched by node i right after T_j, this may not be true at other nodes that they both pass through, as a packet may be dispatched between them but to a different branch, and T_{j+1} catches up with T_j later at node i. A packet is a **leader** at node i if it is not delayed by any other packet at node i. Clearly, if a packet P is a leader at a node, then it will reach the next node exactly $\beta + |P|$ time after it arrives at the current node.

Lemma 3 *Let $\langle T_1, T_2, T_3, ..., T_m \rangle_i$ be a delay sequence. If T_1 is dispatched from node i at time t, then T_j is dispatched from node i at time $t + \sum_{k=1}^{j-1}(\beta + |T_k|)$.*

Proof:

By the definition of delay, node i will be continuously sending out packets from time t. Surely, at time $t + \sum_{k=1}^{j-2}(\beta + |T_k|)$, it just dispatches packet T_{j-1}, which departs at time $t + \sum_{k=1}^{j-1}(\beta + |T_k|)$, at which time T_j will be dispatched. □

Corollary 4 *Let $\langle T_1, T_2, T_3, \cdots, T_m \rangle_i$ be a delay sequence where T_1 is a leader. (a) T_j will depart from node i at most $|T_1| - |T_j|$ after it arrives there. (b) This number is exact if $\langle T_1, T_2, T_3, \cdots, T_m \rangle_l$ is also a delay sequence, T_1 arrives at i from l, no packets dispatched between T_1 and T_m at l leave it without going to i.*

Proof:

We only have to prove the corollary for (b), as any packet leaving l without going to i can only delay some packet T_j's arrival time at node i, potentially shortening its stay, hence delay, at i.

Let T_1 be dispatched from node i at time t. Then T_1 is dispatched from node l at time $t' = t - \beta - |T_1|$ because it is a leader. Lemma 3 says T_j is dispatched from node l at time $t' + \sum_{k=1}^{j-1}(\beta + |T_k|)$, hence arriving at node i at time $t' + \sum_{k=1}^{j}(\beta + |T_k|)$. Lemma 3 also says T_j will be dispatched from node i at time $t + \sum_{k=1}^{j-1} \beta + |T_k|$. Hence the delay is exactly

$$t + \left(\sum_{k=1}^{j-1}(\beta + |T_k|)\right) - \left(t' + \sum_{k=1}^{j}(\beta + |T_k|)\right) = |T_1| - |T_j|$$

□

We have the following simple yet useful corollary, whose validity is immediate from Corollary 4.

Corollary 5 *If $\langle T_1, T_2, T_3, ..., T_m \rangle_i$ is a delay sequence and T_1 is a leader, then $|T_1| > |T_j|$ for $1 < j \leq m$.*

3.3 Maximum congestion at a buffer

We show that if $\mathcal{S} = (S_1, S_2, ...)$ is the dispatching sequence, and greedy FCFS queueing strategy is used (for trees), then a buffer has at most $\max_j |S_j|$ flits at any point in time, which surely is the tightest bound possible.

Apparently, we only have to show it to be true for any node i and any delay sequence $\langle T_1, T_2, T_3, ..., T_m \rangle_i$ with T_1 a leader. Furthermore, we can assume that every T_{j+1} is dispatched by some node l, neighboring i, right after T_j, for if a packet dispatched between T_1 and T_m leaves l without going to i, then it can only delay some packet's arrival time at node i, making i's buffer potentially less congested.

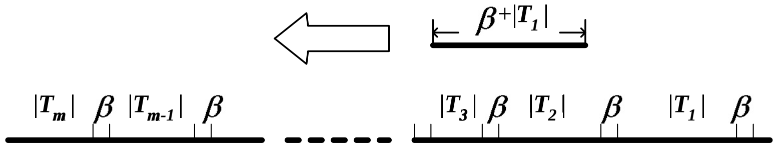

Figure 3.3: Analyzing the maximum buffer size, used in the proof of Theorem 6.

As T_1 arrives, it occupies $|T_1|$ flits. It is sufficient that we show the buffer size never contains more than $|T_1|$ flits throughout this sequence. In the next β time, it sets up T_1 for transmission without delay as T_1 is a leader. Similarly, it does not receive any flits yet, as T_2 is also being set up for transmission. From there on, when i is sending out real flits, surely the flit rate from the input end cannot be higher; hence the buffer size does not increase. On the other hand, when i is working on the set-up, the input may keep coming in. This can be envisioned in Figure 3.3, where we move a stick of length $\beta + |T_1|$ to the left on top of a stick of length $\sum_{k=1}^{m}(\beta + |T_k|)$ arranged as in the figure, denoting the total amount of work to be done at node i for the delay sequence. The right tip of this stick denotes what the output end of node i is doing, while the left tip of this stick denotes what the input end of this node is doing. When a tip falls within a region denoted by β, it means a packet is being set up ("to be send out of node i" for the right tip and "to be sent to node i" for the left tip). When it falls within a region denoted by $|T_j|$, it means a packet is being transmitted. Clearly, the number of flits (fraction allowed) falling under the stick is exactly the number of flits in the buffer at that moment. From Corollary 5, this stick contains more than β amounts of work devoted to the set-up time, leaving less than $|T_1|$ flits for storing data. This proves the following theorem.

Theorem 6 *For any schedule with an FCFS greedy queueing strategy, the maximum buffer size required is the size of the biggest packet.*

3.4 Handling queueing delays for FDF schedules on linear arrays

Allowing packets to have various sizes means queueing delays may develop. Fortunately, we do not have to worry about them as the next lemma shows, because we can make a schedule do away with queueing delay as far as possible without increasing its running time.

Lemma 7 *For any FDF schedule on a linear array, if packet P is delayed by packet Q by T, and P is followed by another packet R going to the same destination as P, then one can transfer up to T flits from R to P without increasing the running time. (P's length may be shorter than Q's and Q may go to the same destination as P.)*

Proof:

Since the leader of the delay seuence that contains P will not be changed before P reaches its destination by Corollary 4(b), such transfer does not change P's departure time from any node. As R is now shorter, packets dispatched after R can only benefit by having, possibly, earlier departure times. □

Remark 8 *This procedure can certainly be applied repeatedly until either such an R no longer exists, or P is no longer delayed by Q. In the process, we may actually decrease the number of packets into which the message P belongs to is split, thus potentially shortening the running time. Although a more sophisticated scheme for balancing is given in Section 5.1, this lemma will be sufficient alter in Chapter 6 when we consider a special case where each message has the same length.*

3.5 Any breadth-first tree is optimal

In Bhatt, et. al.'s scheme, we recall, each message is split into 1-flit packets to be sent out according to the FDF principle. Note that a packet's arrival time at its destination only depends on its dispatch time at the root and its destination's distance from the root because there is no queueing delay. We show below that any breadth-first tree created out of a general network will run as fast under Bhatt, *et al.*'s scheme.

Lemma 9 *The running time of any FDF schedule which packetizes a message into single-flit packets is the same and optimal for any breadth-first tree.*

Proof:

Observe that every breadth-first tree contains the same set of nodes at the same level by definition. Furthermore, due to the FDF principle, the last flit to reach any node at a given level is the same for any breadth-first tree. This shows the scheme has the same running time for any breadth-first tree.

The next step is to transform any spanning tree of a network into a breadth-first tree without increasing its running time, even if the schedule is not changed. This done, the claim is proved.

Let T be a spanning tree of a given network G and not a breadth-first tree. Note that if a node's distance to the root is T is larger than that in the original network (call such a node **misaligned**), then the same holds for all of its descendants in T. Find a misaligned node v such that its distance to the root in T is the least among all misaligned nodes. Then v must have a neighbor u is G that is not mialigned and closer to the root in G as well. Detach the v-rooted subtree from T and attach it to u to form a new tree.

Now, observe that the arrival time of any packet to a destination not in the v-rooted subtree is not changed. Secondly, a packet whose destination is in the v-rooted subtree now reaches its destination earlier, because the distance between its destination and the root is shortened. hence, the overall running time is not increased.

We can repeat the above detach-attach operation until a breadth-first spanning tree develops. Such a tree will eventually appear because every such operation brings at least one more node to its correct level. Hence, after a finite number of steps, every node will be at the right level. □

The above result shows we can start with a network, find any of its breadth-first trees, then apply an FDF scattering scheme to guarantee optimal running time, as long as each packet is required to be only one flit long as in [2]. Such a strategy can be justified when the set-up time is zero or negligible.

Chapter 4

Data Scattering On Trees

We first restate a known result due to Bhatt, *et al.*

Fact 10 *([2]) Every FDF schedule for scattering from the root of a tree is optimal if the set-up time is zero and each packet has exactly one flit.*

Their scheme is very simple. The root splits each message into single-flit packets before sending those packets according to the FDF principle.

The first subsection to follow shows that the above fact no longer holds if the set-up time is not zero and the packets may not be on flit long. Then, in the following subsection, we show that if either requirement is dropped, then the above fact still holds. Hence their result holds if we drop the first requirement (as is done in [2]) or the second on (see Claim 11), but not both.

4.1 Scattering on trees and the FDF principle

The general FDF principle no longer holds when scattering on trees is concerned. The following is a counter-example. Let S_i's destination be P and S_{i+1}'s destination be Q. Let the arrangement of the root, P, and Q be as shown in Figure 4.1. Now let the root have three flits for node P and one flit for node Q. Any FDF schedule then has to send the flit to Q before any of the flits for P. We will construct a non-FDF schedule that out-performs any FDF schedule ρ for some value of β. There are four possible FDF schedules: (i) send one flit to Q, then one flit to P, then one flit to P, then one flit to P, using $6\beta + 6$ time; (ii) send one flit to Q, then one flit to P, then two flits to P, using $5\beta + 8$ time; (iii) send

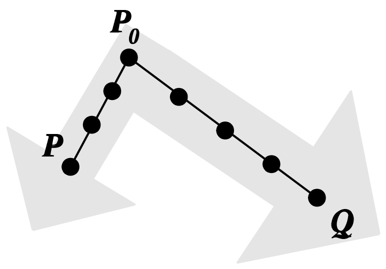

Figure 4.1: The root P_0 has three flits to send to node P and one flit to send to Q. This is part of the construction of a counter-example to the optimality of the FDF principle for scattering on a tree.

one flit to Q, then two flits to P, then one flit to P, using $5\beta + 8$ time; (iv) send one flit to Q, then three flits to P, using $4\beta + 10$ time. Consider the following non-FDF schedules, ρ^*: send three flits on P, then send one flit to Q, using $\max(3\beta + 9, 5\beta + 7)$ time.

If we pick $1 < \beta < 3$, then ρ^*'s running time is $5\beta + 7$, and one can easily show that

$5\beta + 7 < 6\beta + 6$

$5\beta + 7 < 5\beta + 8$, and

$5\beta + 7 < 4\beta + 10$

Hence ρ^* out-performs each of the four FDF schedules.

For every non-FDF schedule on a tree where S_i's destination, P, is nearer the root than S_{i+1}'s destination, Q, we can exchange their dispatch order without increasing the running time if P is on the path from the root to Q. This observation follow from Lemma 1. So, in a limited sense, the FDF principle still works if we are only concerned with packets going to destinations lying on the same *branch* of the tree.

4.2 Zero set-up time

Claim 11 *Bhatt, et al.'s scheme remains optimal if we allow queueing and packets of more that one flit, as long as $\beta = 0$.*

Proof:

We prove this claim by showing that, if there is any packet of size more than on flit, then we can split it into two smaller packets without increasing the running time. Clearly this shows their scheme remains optimal when packets are allowed to be of size greater than one, if the set-up time β, remains zero as in their paper.

Let S_i be a packet in schdule ρ with $|S_i| > 1$. Construct another schedule ρ^* such that S_i is replaced by two packets, one with size one, call it S', and another one with size $|S_i| - 1$, call it S'', which we send first; see Figure 4.2.

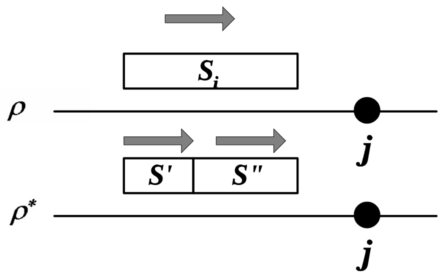

Figure 4.2: Splitting of S_i into two shorter packets.

Any packet dispatched before S_i will reach its destination at the same time as before, because it is not affected by the splitting. S_i's arrival time at its destination cannot be earlier than S''s, which can be argued as follows. Surely, S_i and S' depart from the root at the same time. Inductively, look at a node j between S_i's destination and the root. If S_i is delayed at j, then S'' is too (since it is shorter, hence running at least as ahead), in which case S'' and S_i will be dispatched from j at the same time; hence S_i and S' arrive at the next node at the same time since S',being at most as long, follow S'' without "gaps." If, on the other hand, S_i is not delayed at j, then, by the induction hypothesis, S_i arrives at j no earlier than S'. However, S' can only be delayed by no more that $|S''|$ time. Consequently, again, S' will arrive at the next node no later than S_i. This shows the flits that S_i contains will reach their destination no later in ρ^* than in ρ.

Finally, as S' arrives at any node no later than S_i does, surely packets that follow them will reach their destinations in ρ^* no later than in ρ. □

Chapter 5

How To Do Packetization

This chapter discusses how to split a message into packets to minimize its delivery time on trees. We show that each message can be split into packets as evenly in size as possible without loss of generality. The decision, in the absence of other messages, ultimately depends on optimizing a linear objective function over a convex set. The packetization problem for the most general case considered in Theorem 2 seems intractable.

5.1 Best distribution of packet sizes under the FDF discipline

What is the best way to packetize messages of various sizes to send in an FDF manner on trees? We can simplify the matter a lot by proving first that making the packets as even in size as possible is one of the best strategies.

With a message of m flits to be sent in r packets , we want to distribute the flits among these r packets (we will discuss how to pick r later). Suppose the i^{th} packet has size $m_i > 0$ for $1 \leq i \leq r$. Recall that the i^{th} packet is the i^{th} of these r packets to leave the root. Certainly, $\sum_{i=1}^{r} m_i = m$. Call $(m_i)_{1 \leq i \leq r}$ a **size distribution**.

Let $b_1 = \lceil m/r \rceil$, $b_2 = \lfloor m/r \rfloor$, and $0 \leq r_1 \leq r$ be the remainder of the division of m by b_2: $m = rb_2 + r_1$. Define $r_2 = r - r_1$; see Figure 5.1 for illustration. Consider a schedule such that the i^{th} packet is of size b_1 for $1 \leq i \leq r_1$ and b_2 for $r_1 + 1 \leq i \leq r$. Call such a size distribution $\rho\langle m, r\rangle$. In general, a message is said to use the **even-size distribution** if it is packetized according to $\rho\langle m, r\rangle$ for some r where m is the message's size.

Suppose we split a message into r packets according to the even-size distribution principle, and this message is to be sent to a node that is l nodes away. Its last packet's departure time from the root is $m + r\beta$. By Corollary 4(b), the last packet will be delayed by $b_1 - b_2$ at each node for the next $l-1$ hops, and its arrival time, hence the message's, is

$$\alpha_l(m,r) \stackrel{\text{def}}{=} m + r\beta + (l-1)(\beta + b_2 + (b_1 - b_2)) = m + r\beta + (l-1)(\beta + b_1)$$

We summarize our finding as follows.

Lemma 12 *If a message is packetized according to $\rho\langle m,r\rangle$, where m is the message's size, and there are no inter-message delays, then it will arrive at a node that is l nodes away $\alpha_l(m,r)$ steps after it is dispatched.*

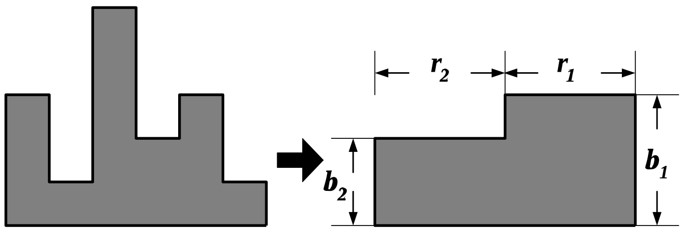

Figure 5.1: For any size distribution for the packets of a message of length m, we can transform it into another one, which is basically a rectangle with area m (except possibly for a "dent" of height one) which runs at least as fast; see Lemma 12.

Consider an FDF schedule ρ and look at one of its messages, P, which is split into r packets such that the i^{th} packet is of size $m_i > 0$. We shall show in the next lemma that if we, instead, split it according to $\rho\langle m,r\rangle$, where m is P's size, to produce another schedule ρ^*, then the last packet of ρ^* will depart from any node no later than that of P in ρ from the same node. As a consequence, for FDF schedules, we only have to consider the case where each message is split according to the even-size distribution principle.

Lemma 13 *For any FDF schedule on a tree, we can produce another schedule as efficient such that each message is packetized according to the even-size distribution principle.*

Proof:

We proceed as follows. Let us focus on an arbitrary message, P, that does not satisfy the even-size distribution principle. We can find a pair of packets sent in tandem, A and B, where B is sent right after A by the root, such that either (i) $|A| < |B|$ or (ii) $|A| > |B|$. We show that in (i) we can move one flit from B to A without increasing the running time and in (ii) we can move one flit from A to B without increasing the running time (see Figure 5.2). Apparently, this proved, we can step by step transform P's size distribution so that it satisfies the even-size distribution principle without increasing its running time, and the lemma is prove.[1]

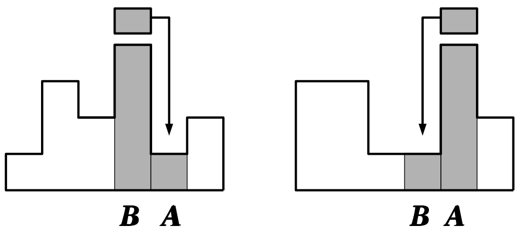

Figure 5.2: When adjacent packets to the same destination have different sizes, balancing their sizes by moving one flit to the shorter one does not increase the running time.

We use induction to prove first that balancing works for (i). B departs from the root at the same time in both schedules. By the induction hypothesis, suppose B of the new schedule arrives at a node at least as early as B of the original schedule. Since B's size has been decremented by one, it will depart at least one step earlier from that node than the original longer B, and (i) is proved, unless it is delayed by A. So suppose it is delayed by A in the new schedule. If A (of the new schedule) is also delayed at the same node, then the original A, being shorter, must also be delayed there and both are dispatched at the same time, in which case both Bs will depart from that node at the same time, too. On the other hand, if A is not delayed, then B, delayed by A, must be shorter than A by one flit after the balancing, which means B will be delayed on step at that node, hence still departing from it at least as early as the original, longer B.

We now turn to (ii). B departs from the root at the same time in both schedules. By the induction hypothesis, suppose B of the new schedule arrives at a node at least as early as B of the original schedule. Now suppose A of the new schedule is delayed at that

[1]We comment that, in fact, this can be used to prove that, given r, any size distribution will do as long as the largest packet and the smallest one differ in size by at most one.

node. A, being shorter, will be dispatched no later than the original A. Hence, in the worst case where both As are dispatched at the same time, B will depart at the same time as the original B. Suppose, on the other hand, A of the new schedule is not delayed. There are two cases to consider. If $|A| > |B| + 1$, then, although B's size has been incremented by one, it will depart at least one step earlier from that node than the original shorter B because B is delayed by A by an amount two flits less than the original shorter B is delayed by the original longer A. If, on the other hand, $|A| = |B| + 1$, then, although B of the new schedule takes one more step to transmit, B of the old schedule is delayed by at least one flit by A, hence arriving at the next node no earlier. □

Remark 14 *This lemma also proves that, given a packetization, we can always re-packetize any message according the even-size distribution principle using the same number of packets without increasing the running time of the resulting scheme.*

We comment that Lemma 13 can be applied to broadcasting, where the root sends the same message to every other node in the tree. If one assumes the multi-port model and hot potato forwarding [12], in which once a packet is received by a node, it is transmitted to all of its other neighbors in the tree, then of course the time is dominated by the arrival time of the message's last packet to the furthest node from the root. If the message size be m and the depth of the tree be d, then what remains is to determine the r such that $\alpha_d(m, r)$ is minimized. This kind of optimization is formalized in the next subsection.

5.2 Finding the best size distribution when there are no inter-message delays

Previously, we showed that for any given r, the number of packets a particular message is split into, we can choose a size distribution for a packet of size m just by dividing and rounding to get $b_1 = \lceil m/r \rceil$, and $b_2 = \lfloor m/r \rfloor$; see Figure 5.1 again. Now we have to find the right r for each message.

Consider each message in isolation as if there were no other messages, i.e., assume there are no inter-message delays. We want to minimize the following over all integers r between 1 and m:

$$\alpha_l(m,r) = m + r\beta + (l-1)(\beta + b_1)$$

But it is easy to see that this problem is equivalent to minimizing

$$r\beta + (l-1)b + \beta(l-1) + m$$

with the constraint $r\beta \leq m$ where both r and b are positive integers not exceeding m. We use the following notation to denote the optimal value (note that the last two terms above are constant)

$$\alpha_l^*(m) \overset{\text{def}}{=} \min_{\substack{m \geq r > 0 \\ m \geq b > 0 \\ rb \geq m}} \left\{ r\beta + (l-1)b + \beta(l-1) + m \right\} \tag{5.1}$$

We shall assume β to be a rational number.

This problem in principle can be solved in polynomial time by the basis-reduction algorithm and the Ellipsoid method. In fact, this is an easy corollary of Theorem 2.5.1 of [20] (p.62). In the next section we shall describe a more direct and practical method with some ideas from [4] and continued fractions approximation to real numbers [10], [11], [14]. In this subsection, we are mainly interested in formulating the problem.

We mention that problem (5.1) becomes trivial when r and b can be real numbers, as the solutions are simply

$$\hat{r} = \sqrt{m(l-1)/\beta} \quad \text{and} \quad \hat{b} = \sqrt{m\beta/(l-1)} \tag{5.2}$$

(Here we assume neither exceeds m; if either exceeds it, just make it m and the other one.) As mentioned before in the Introduction, previous authors who consider set-up time make real numbers acceptable solutions; see; for example, [15], [28], and [39]. Surely, this is not satisfactory as packets cannot contain fractional number of bits.

As $\beta(l-1) + m$ is constant given l and m, we can assume without loss of generality that we are minimizing

$$r\beta + (l-1)b \tag{5.3}$$

subject to $rb \geq m$ with both r and b begin positive integers not exceeding m. We note that the restriction on r and b not exceeding m is redundant. This is because, if either variable exceeds m, then we can always make it m without disturbing the other variable's value and produce an even smaller value for (5.3). The special class of data scattering problems we shall consider in Chapter 6 can also be reduced to the above optimization problem.

This optimization problem has a simple geometric interpretation. Let $L(x, y)$ denote the equation $\beta r + (l-1)b = \beta x + (l-1)y$, i.e., the line with slope $-\beta/(l-1)$ that passes through point (x, y). What we want is simply a lattice point (points with integral coordinates), (r^*, b^*), on the upper-right of the r-b plane such that $r^*b^* \geq m$ and there are no other such lattice points to the left of the line $L(r^*, b^*)$; see Figure 5.3. In other words, we slide a line with slope $-\beta/(l-1)$ from positive infinity towards the origin until it hits a lattice point to the right of, or on, the hyperbola, $rb = m$, such that there are no more lattices points satisfying $rb \geq m$ to the left.

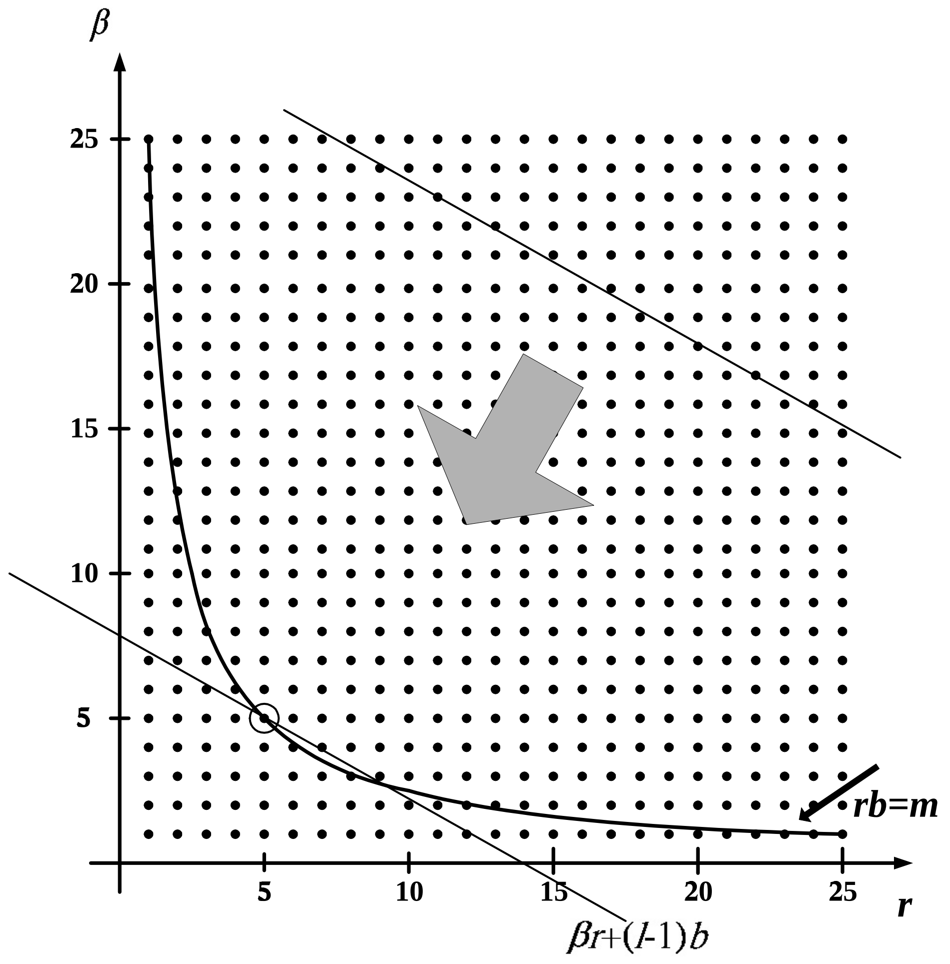

Figure 5.3: Geometric interpretation of our optimization problem. In this figure we let m be 25. In this particular example, point (5,5), which we circle, is the optimal choice.

Chapter 6

A Polynomial-Time Solution to a Special Class

Consider the following scattering problem on a linear array: The root P_0 wants to send a message M_i of length m to node P_i where $s \leq i \leq t$. That is, all the messages have the same length, and they go to contiguous nodes on the linear array. This generalizes the situation considered in [9] and [28]. Our goal is to find the packetization, i.e., a size distribution $\left(m_j^i\right)_{1 \leq j \leq m}$ for each message M_i, and the schedule that minimizes the transmission time. We will consider a slightly more general case at the end of this section.

By Lemma 13 of Section 5.1, we can restrict ourselves to even-size distributions for each message. So assume each $\left(m_j^i\right)_{1 \leq j \leq m}$ is an even-size distribution $\rho\langle m, r_i \rangle$ for some $r_i > 0$. Furthermore, by Theorem 2 of Section 3.1, we can restrict ourselves to FDF schedules. Hence assume packets are sent according to their distance from the sender, the root P_0.

We first prove that we can without loss of generality consider only size distributions where a message is split into at most as many packets as its predecessor going to a further node.

Lemma 15 *There is an optimal FDF schedule such that* $r_s \leq r_{s+1} \leq \cdots \leq r_t$.

Proof:

Consider any two messages destined to two adjacent nodes: M_b and M_{b+1}, but $r_b > r_{b+1}$. The first packet of M_b , say P, must be delayed by the last packet of M_{b+1} by an amount at least as large as the difference in size between P and M_b's first packet, say Q (see Corollary

4(b)). Recall that the first packet of a message of an even-size distribution is always the longest of packets. Apply Lemma 7 to P to make its length as large as Q, which is always doable because each message has the same length. Apply Lemma 7 to the rest of M_b's packets. Because each message has the same length, in the end, the number of packets that M_b is split into will be r_{b+1} , that is $r_b = r_{b+1}$. Finally, apply Remark 14 to M_b to make it conform to the even-size distribution principle. As each operation does not increase the running time and we can apply them iteratively to any offending pairs, we have proved the lemma. □

As a consequence, we assume the schedule further satisfies Lemma 15; see Figure 6.1.

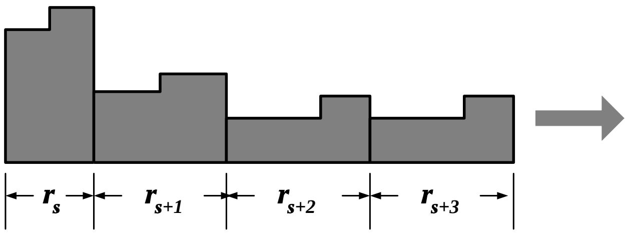

Figure 6.1: After many transformations, we narrow down to FDF schedules such that each message is split into no more packets than its predecessor going to a further node.

Corollary 16 *In any optimal FDF schedule that satisfies Lemma 15, M_s is the last message to reach the destination.*

Proof:

From Lemma 15, it is clear that there is no inter-message delay. As the messages go to contiguous nodes and the schedule is FDF, it is easy to see that M_{b+1} reaches its destination as least as early as M_b for $s \leq b < t$. Hence the last packet to issue is the last to arrive at its destination. □

Now we present the main theorem, showing that one of the best schedules is to split every message into the same number of packets (see Figure 6.2(a)).

Theorem 17 *There is an optimal FDF schedule with $r_s = r_{s+1} = \cdots = r_t$*

Proof:

By Lemma 15, we consider an optimal FDF schedule that satisfies $r_s \leq r_{s+1} \leq \cdots \leq s_t$. Let b be the least integer such that $r_{b-1} < r_b$. If we now apply $\rho\langle m, r_{b-1}\rangle$ instead of $\rho\langle m, r_b\rangle$

to M_b, the dispatch time of the last message will be earlier by $(r_b - r_{b-1})\beta$ and, hence, it will arrive earlier by the same amount of time as there is still no inter-message delay. By Corollary 16, this schedule runs faster, a contradiction. Hence no such b exists. $\square$

Equipped with the above theorem, we can now proceed to analyze the running time of an optimal schedule. Let each message be split into r packets. The last packet's dispatch time is

$$(t-s)(r\beta+m)+(r-1)\beta+(m-\lfloor m/r \rfloor)$$

From there on, it takes $\beta + \lfloor m/r \rfloor$ time for it to traverse a link. Hence it reaches its destination at time

$$(t-s)(r\beta+m)+(r-1)\beta+(m-\lfloor m/r \rfloor)+(\beta+\lfloor m/r \rfloor)s = Ar + B\lfloor m/r \rfloor + C$$

where $A = (t-s+1)\beta$. $B = (s-1)\lfloor m/r \rfloor$, and $C = (t-s+1)m + (s-1)\beta$. As C is a constant, the problem can be reduced to solving

$$\min_{\substack{m \geq x > 0 \\ m \geq y > 0 \\ xy \geq m}} \left\{ Ax + By \right\}$$

where x and y are positive integers.

The above theorem generalizes Fraigniaud, *et al.*'s result for the linear array in that they consider the more limited case where *every* node receives a message of equal size. On the other hand, they prove the optimality of FDF schedules under a more general condition which allows a packet to contain flits from various messages [9]. It is not hard to see that this relaxation can lead to better running time; see Figure 6.2.

Lemma 15 actually holds without modification in a more general context where the messages may not be destined to contiguous nodes on a linear array, as long as they have the same length. That is, P_0 sends a message M_i of length m to node P_i where

$$s_1 \leq i \leq t_1, \text{ or } s_2 \leq i \leq t_2, \text{ or}, \cdots, s_k \leq i \leq t_k.$$

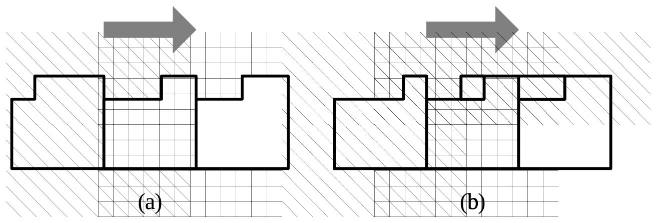

Figure 6.2: Allowing packets to contain flits from different messages may lead to better running time. (a) shows an optimal FDF schedule. In schedule (b), the first two messages's packets carry some of the third one's flits, and it is clear that it runs faster because it has less packets in total.

In this case, the messages can be partitioned into k sets, $\mathcal{M}_j = \left\{ M_i | s_j \le i \le t_j \right\}$, where $1 \le j \le k$, such that the messages in the same set go to contiguous nodes. Corollary 16 now needs nominal changes, and now reads:

> "in any optimal FDF schedule that satisfies Lemma 15, message M_{s_j} is the last one in $\mathcal{M}_j$ to reach its destination, where $1 \le j \le k$."

This clearly generalizes the older version and uses the same proof. Finally, Theorem 17 can now be generalized as follows (see Figure 6.3).

Theorem 18 *There is an optimal FDF schedule with* $r_{s_j} = r_{s_{j+1}} = \cdots = r_{t_j}$ *for* $1 \le j \le k$

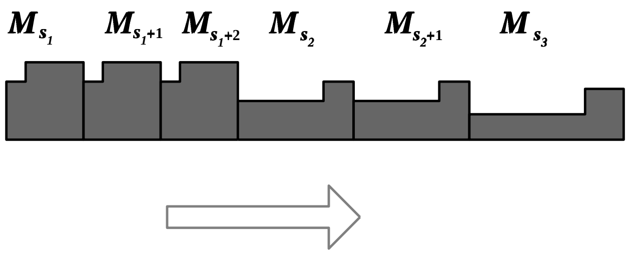

Figure 6.3: If every message has the same length, one optimal FDF schedule will split the messages that go to contiguously located nodes into the same number of packets.

Chapter 7

Data Gathering on Trees

In this chapter, it introduces whether the similar result for the data gathering problem would be achieved. Fortunately, it seems that every lemma and theorem about the NRF (near-receive-first) principle still works for the data gathering problem. For a more general tree, a new algorithm “Transmission Certification II Algorithm ” will be applied to get the optimal running time.

7.1 The Nearest-Receive-First Principle

The data gathering problem is the opposite of data scattering problem. By running an FDF scattering algorithm “backwards” (nearest-receive-first principle). It is one way of implementing gathering operation. The nearer distance between the root node and some node is, the earlier the root node receives the packet from the node. The term “PE” shall denote the other nodes different from the root node.

Of course, one can not literally run an FDF scattering algorithm “backward,” because in the scattering operation, the PEs other than P_0 are passive, while in the gathering operation, they are active; they must initiate their message transmissions. To compensate for this fact, any algorithm for a bufferless gathering operation must precede the transmission of messages by a distributed protocol that schedules the dispatch times of the messages so that no two collide in transit. We will introduce “Transmission Certification II Algorithm” [2], which is readily adapted to general tree structures, but only at the cost of added time for separate synchronization and scheduling activities.

At a result, the running time for gathering problem is more than the time for scattering problems. It is obviously that each PE cannot safely begin transmitting its message until that node gets the transit permission; otherwise, packets may collide at some node or when crossing some link.

Theorem 19 *For a linear array, there is only one schedule — the NRF — in which every packet will be received in order, such as $S_1, S_2, S_3, \ldots, S_{k-1}, S_k$. (see Figure 7.1*

Proof:

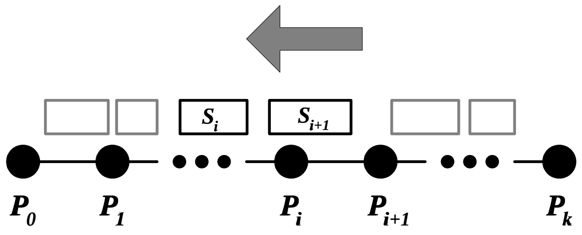

Figure 7.1: The root node P_0 receives packets S_1,S_2, . . . , S_i, S_{i+1}, . . . in order.

For our basic structure network, the buffer size of every node is the size of the biggest packet. When gathering operation starts, all nodes is ready to transmit their own packet. For this reason, every packet should be delivered in order to avoid the insufficient buffer size.

Noticeable, some overhead must be considered in order to prevent the conflict of packets. For example, every node can't deliver its packet to its parent node until its parent node informs it. □

Theorem 20 *For any schedule with an FCFS greedy queueing strategy, the maximum buffer size required is the size of the biggest packet.*

Proof:

In a single-port network, any node can only transmit one packet over some link at one time. By using the new Transmission-Certification algorithm (see section 7.3), the root will give each message an individual free time gap stream. In other words, the situation won't happen that two packets arrive some intermediate node PE simultaneously. Note that no matter when some packet begins transmitting at some node, it can't be stopped until it reach the destination node - the root P_0. As a result, the maximum buffer size required is the size of the biggest packet. $\square$

Lemma 21 *The running time of any NRF schedule which packetizes a message into single-flit packets is the same and optimal for any breadth-first tree.*

Proof:

This proof is similar to Lemma 9's proof.

Observe that every breadth-first tree contains the same set of nodes at the same level by definition. Furthermore, due to the NRF principle, the last flit to reach the root is the same for any breadth-first tree. This means that the scheme has the same running time for any breadth-first tree.

Next, we want to prove that we can transform any spanning tree of a network into a breadth-first tree without increasing its running time, even if the schedule is not changed.

Let T be a spanning tree of a given network G and not a breadth-first tree. Note that if a node's distance to the root in T is larger than that in the original network (call such a node misaligned), then the same holds for all of its descendants in T. Find a misaligned node v such that its distance to the root in T is the least among all misaligned nodes. Then v must have a neighbor u in G that is not misaligned and closer to the root in G as well. Detach the v-rooted subtree from T and attach it to u to form a new tree.

Now, observe that the receive time of any packet from the source node not in the v-rooted subtree is not changed. By the NRF principle, each packet get its own gap-free stream initially. Furthermore, the distance between the source node and the root is shortened. By detaching the v-rooted subtree from T and attaching it to u, the free time slot are also kept. We assume that the distance (or depth on that tree) between the root and some node P_δ in v-rooted subtree is l and packet S_i for P_δ is delivered at time t. As we see, no node except P_δ at the l^{th} depth can use this time slot $[t, t + \beta + |S_i|]$ on the tree, otherwise some collision will occur at some node whose depth is less than l. Similarly, no node whose depth is $(l-1)^{th}$ except P_δ can use this time slot $[t+(\beta+|S_i|), t+2(\beta+|S_i|)]$. Step by step, no node whose depth is 1^{st} except P_δ can use the time slot at $[t+(l-1)(\beta+|S_i|), t+l(\beta+|S_i|)]$.

Now, we can trace back on the new path. The packet S_i whose depth becomes l' can just dispatch at time $t+(l-l')(\beta+|S_i|)$ where $l > l'$. Hence, we maintain the running time of the new spanning time whose schedule is no longer the NRF principle.

By increasing the dispatch time of those packet whose depth is larger than l' and decreasing the dispatch time of those packet whose position is changed in new spanning tree, the NRF principle will be reborn. Therefore, the running time of the new one using NRF is the same as the old one.

We can repeat the above detach-attach operation until a breadth-first spanning tree develops. After a finite number of steps, every node will be at the right level. $\square$

7.2 The counter-example violating NRF with packetization

In this section, we will show that the NRF principle would not work well in presence of packetization and setup time for a general tree, but for a linear array.

The following is a counter-example that violates the NRF principle. Let S_i's departure node be P and S_{i+1} be Q. Let the arrangement of the root, P, and Q be as shown in Figure 7.2. Now let node P have three flits for the root and node Q have one flit for the root. Any NRF schedule then has to send the flits of P to the root before the flit of Q. We will construct a non-NRF schedule that is much better than any NRF schedule for some β. There are four possible NRF schedules:

(i) the root receives one flit from P, then one flit from P, then one flit from P, then one flit from Q, using $6\beta+6$ time.

(ii) the root receives two flits from P, then one flit from P, then one flit from Q, using $5\beta+8$ time.

(iii) the root receives one flit from P, then two flits from P, then one flit from Q, using $5\beta+8$ time.

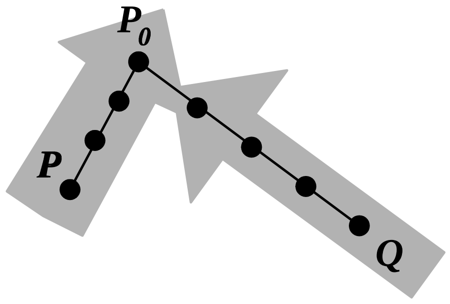

Figure 7.2: The node P sends three flits to the root P_0 and the node Q sends one flit to the root P_0.This is part of the construction of a counter-example to the optimality of the NRF principle for gathering on a tree.

(iv) the root receives three flits from P, then one flit from Q, using $4\beta + 10$ time.

Now, consider the following non-NRF schedules, ρ^*, which receives one flit from Q, then three flits from P, using $\max\{3\beta + 9, 5\beta + 7\}$ time. If we pick $1 < \beta < 3$, then ρ^*'s running time is $5\beta + 7$, and it is easy to show that

$$5\beta + 7 < 6\beta + 6$$

$$5\beta + 7 < 5\beta + 8$$

$$5\beta + 7 < 4\beta + 10$$

Hence the performance of ρ^* is better than each of the four NRF schedules.

7.3 Packetization & set-up time

The following algorithm is modified from the transmission certification algorithm developed by Bhatt, *et al.* The concept of the new one is similar to the old one. The only difference between two algorithms is the method computing the lag time, which means that some node can start delivering their own message after the lag time. At the same time, the message length must be adjusted because of packetization.

Transmission Certification II Algorithm

Phase 1: the root broadcasts "synchronization token" to all other nodes. This wakeup call lets the PE's know that P_0 is ready to "gather" their messages.

Phase 2: Each PE P_i responds to the synchronization token by sending a "transmission certificate" to its parent PE. The "transmission certificate" packet will give its parent PE some useful and important information about its message size and lag time.

Phase 3: When P_i receives its children's certificates, it initiates a wave of TRANSMIT-MESSAGE orders. P_i must decide the receive order about its children because of single-port network. We can just use the sorting algorithm for this phase, and the parameter from its children. Figure 7.3 illustrates it.

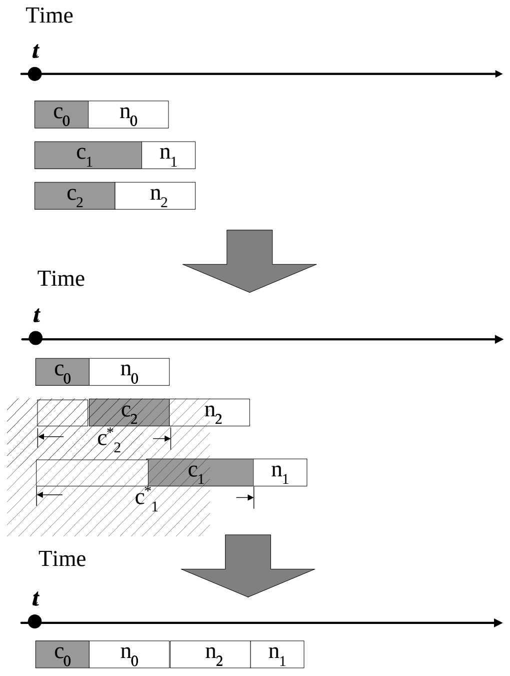

Figure 7.3: Sorting the lag time of its children and Combining those messages into one message and one lag time.

The TRANSMIT-MESSAGE orders :

$(c_{i,1}, n_{i,1})$, $(c_{i,2}, n_{i,2})$, $(c_{i,3}, n_{i,3})$,.... (c_{i,d_i}, n_{i,d_i})

INITIAL: the leaf node $c_i = 0$

At node P_i : we sort the lag time of its children to get the new lag time c_i. It takes $d_i(\beta + 1)$ time for P_i to transmit the new lag time to its children node. Also, because of different message size, some changes must be considered.

$$c^*_{i,k} = c_{i,k} + \max_{\forall k,j}(\beta + |n_{i,k,j}|) + d_i(\beta + 1),$$

where $n_{i,k,j}$ means the j^{th} packet of the $n_{i,k}$ of the i_{th} child. Also, we combine those messages of its children and itself into one message.

$$n_i = r_i(\beta + \lceil M_i/r_i \rceil) + \sum_{j=1}^{d_i} n_{i,j} \quad , \quad c_i = d_i(\beta + 1),$$

where d_i means the number of P_i's children.

The time that last packet arrives the root is no more than

$$c_0' + \sum_{i=1}^{n} \left\{ \beta + |M_i| \right\} = c_0' + \sum_{j=1}^{n} \left\{ r_i \times (\beta + \lceil M_i/r_i \rceil) \right\},$$

where c_0' is denoted to the receive time of the first packet received by the root P_0.

The n-number sorting problem is concerned, so the time complexity about sorting d_i number is

$$d_i \log_2 d_i.$$

Phase 3.1: Repeat **Phase 3**, we can't stop until the root P_0 also repeat one time. After the computation for the root P_0, we can get the final lag time for each children of the root.

After $d_0 \log_2 d_0$ times, each child of the root will be given a new dispatch time(lag time s_i), in other words, a free time interval, which the child can transmit. So the root will transmit the exact time interval to each child. Corollary, the children of root will correct their initial lag time. Repeat this phase until all information reach all leaf node.

$$c_{0,j}' = c_{0,j}^* + s_0 - c_0,$$

where c_0 means the previous lag time at node P_0.

$$c_{i,j}' = c_{i,j} + s_i - c_i \tag{7.1}$$

Earlier, when P_i's old certified lag time c_i and its children lag time $c_{i,j}$. After this phase, P_i's parent gives P_i the new lag time s_i, so we must adjust the lag time for optimality.

Phase 4: Finally, the PE's follow the schedule of *phase 3*, transmitting messages in a free gap time slot toward P_0, via their parents.

Lemma 22 *For general gathering problems, the running time equals* $2B + C + L + M$

Proof:

Let P_i be some node of the tree and M_i be the message of that node. And let the degree of the node P_i be d_i, the maximum degree of the tree be d ,and the depth of the tree be λ. It is clear that each node can transmit the message "Synchronization token" to its children nodes after it receives and each node at one time can only transmit one packet. According the topology of the tree, each edge must be traveled at least one time, or some node may not receive the "Synchronization token". Let the length of the "synchronization token" message be 1 flit. By the concept of the pipeline that two different level edges can be used to transmit at same time, the broadcast time for *Phase 1* at most:

$$B = (\lambda \times\ d)(\beta + 1)$$

The number of children of node P_i be d_i. Also, the complexity of sorting n-number problems is $n \log_2 n$. So, the computation time for each node P_i : $c_i = d_i \log_2 d_i$. Also, it takes $d_i \times (\beta + 1)$ for P_i to receive its children response packets. As a result, the total time spending on computation is at most

$$C = d_i \times (\beta + 1) + \lambda \times\ (d \log_2 d)$$

Each message M_i is split into r_i packets. And the setup time of each packet is β. The transmission time slot for message M_i will be reconstructed from $\beta + M_i$ to $r_i \times \left\{\beta + \lceil M_i/r_i \rceil\right\}$. As a consequence, the total message transmission time slot :

$$M = \sum_{i}^{n} \left\{ r_i \times (\beta + \lceil M_i/r_i \rceil) \right\}$$

According to the transmission certification II algorithm, we can easily estimate the running time of our gathering problem by the final lag time and the total message transmission time slot. The Figure 7.3 is part of some simple example. The lag time means that the receive time of the first packet at root node. It takes at least $d_0(\beta + 1) + (\beta + \lceil M_i/r_i \rceil)$ and at most $\lambda \times (\beta + \lceil M_i / r_i \rceil)$ for the first packet to reach the root node. So, the worst case of the lag time at the root node P_0 :

$$L = \lambda \times \left\{ (\max_i d_i) \times (\beta + 1) + \max_i \lceil M_i / r_i \rceil \right\}$$

As a consequence, the running time for data gathering problem:

$$2B + C + L + M$$

$\square$

7.4 A special class of gathering problems

Consider the following gathering problem on a linear array: the root P_0 wants to receive each message M_i of length m from node P_i where $s \leq i \leq t$. That is, all messages have the same length, and the root receives messages from contiguous nodes on the linear array. All we do is find the packetization,i.e., a size distribution $(m^i_j)_{1 \leq j \leq m}$ for each message M_i, and the schedule that minimizes the transmission time.

And, we can restrict ourselves to even-size distribution for each messages. So assume each $(m^i_j)_{1 \leq j \leq m}$ is an even-size distribution $\rho\langle m, r_i \rangle$ for some $r_i > 0$. According the Theorem xx, we can restrict ourselves to NRF schedules.

Without loss of generality, we first prove that we can consider only the size distributions where a message is split into at most as many packets as its predecessor going to a further node.

Lemma 23 *There is an optimal NRF schedule such that* $r_s \geq r_{s+1} \geq \cdots \geq r_t$

Proof:

Consider any two messages dispatched from two adjacent nodes: M_b and M_{b+1}. Now, at node M_b, the first packet of M_{b+1} must be delayed by the last packet of M_b because of the NRF principle. As a consequence, if we want to minimize the running time, the relation between r_b and r_{b+1} must conform to the inequation $r_b \geq r_{b+1}$. By the same way, we can prove the other inequation like $r_{b+1} \geq r_{b+2}$, $r_{b+2} \geq r_{b+3}$, $\cdots$, $r_{t-1} \geq r_t$.

As a result, we can get $r_s \geq r_{s+1} \geq \cdots \geq r_t$ □

Corollary 24 *In any optimal NRF schedule that satisfies Lemma 23, M_t is the last message to reach its destination.*

Proof:

From Lemma 23, we observe that there is no inter-message delay. Applying NRF principle, it is easy to know that the destination of each message is the root node. So, the last packet that reaches the root must be dispatched from node M_t's. □

Theorem 25 *There is an optimal NRF schedule must have $r_s = r_{s+1} = \cdots = r_t$.*

Proof: By Lemma 23, we consider an optimal NRF schedule that satisfies $r_s \geq r_{s+1} \geq \cdots \geq r_t$. Let b be the least integer such that $r_{b-1} < r_b$. If we now apply $\rho\langle m, r_{b-1}\rangle$ instead of $\rho\langle m, r_b\rangle$ to M_b,the dispatch time of the last packet at node M_{b-1} be earlier by $(r_b - r_{b-1})\beta$ and, hence, it will arrive earlier at the root node by the same amount of time as there is still no inter-message delay. By the same way, we can show that any optimal NRF schedule must have $r_s = r_{s+1} = \cdots = r_t$. □

According the about theorem, we can now analyze the running time of an optimal schedule. Let each message be split into r packets. Let the depth of the linear array be λ. We observe that the degree of the linear array is 1. The last packet's reach time at the root node P_0 is

$$2B + C + L + M = 2B + C + \lambda(\beta + 1) + s(\beta + \lceil m/r \rceil) + (t - s)(r\beta + m)$$

,where B and C are constant. Let $A' = (t-s)\beta$, $B' = s$, and $C' = 2B + C + \lambda\,(\beta + 1) + s\beta + (t - s)m$. As C' is a constant, we can reduce this problems to

$$\min_{\substack{m \geq x > 0 \\ m \geq y > 0 \\ xy \geq m}} \Big\{ A'x + B'y \Big\},$$

where x and y are positive integers.

Theorem 26 actually holds without modification in a more general case where the messages sent by the contiguous nodes on a linear array, as long as each message has the same length. So the root P_0 will receive messages M_i of length m from node P_i where $s_1 \leq i \leq t_1$, or $s_2 \leq i \leq t_2$, or $s_3 \leq i \leq t_3$, or $\ldots$, or $s_k \leq i \leq t_k$.

Theorem 26 *There is an optimal NRF schedule with* $r_{s_j} = r_{s_j+1} = \cdots = r_{t_j}$ *for* $1 \leq j \leq k$.

Chapter 8

Conclusion

Generally speaking, we can transform an arbitrary network to a breadth-first spanning tree single-port network, which the FDF and NRF principle can be applied to produce the optimal schedule even in the presence of set-up time. For generalizing the data scattering and gathering problems on an arbitrary network, any factors that affect our result will be considered, such as packetization, multi-port, transmission rate, and et al..

When packetization is allowed, the data scattering and gathering problems would be reduced to find solution (x,y) satisfying $\min\limits_{\substack{m\geq x>0\\ m\geq y>0\\ xy\geq m}} \left\{Ax + By\right\}$. The method solving (x,y) is a huge work which can be resolved by the information and reference from the mathematics books.

The transmission rate of every link may be arbitrary on a account of the physical limits of transmission media, such as optical fiber, twisted pair, coaxial cable,and et al.. As the transmission media are different on our network, all we need to do is just time the measure of message length (M_i flits/sec) with some δ where δ is some ratio decided by the transmission media. The FDF and NRF principle still work on the spanning tree network as well as before. The only difference is that the running time varies according to the various transmission media.

Finally, the multiport play an important role in optimalization on the network. With the multiport, the running time of the data scattering seems no change, but the running time of data gathering problem will be improved. As we see, in phase 2 of transmission certification II algorithm, paralleling the receive of the TRANSMIT-MESSAGE packet can reduce the computation time C from $d_i(\beta + 1) + d_i \log_2 d_i$ to $d_i \log_2 d_i$ at most. Note that the tree structure is just part of the arbitrary network, so there must exist some unused links. One of the best way to improve the total running time is paralleling the message transmission by using the remainder links. Since the uncertainty of the network structure exists, there are two ideas which the one is transform the general network into several tree networks, and the other is that every unused link is considered as the shared link. It would be an interesting challenge for further research.

Bibliography

[1] Aho, A.C., J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms.* Reading, Massachusetts: Addison-Wesley, 1974.

[2] Bhatt, S.N., G. Pucci, A. Ranade, and A.L. Rosenberg. "Scattering and gathering Messages in Networks of Processors." In *Proc. 1992 Brown/MIT VLSI Conference,* 1992.

[3] Bozkus, Z., S. Ranka, and G. Fox. "Benchmarking the CM-5 Multicomputer." In *Proc. 4th Symp. on the Frontiers of Massively Parallel Computation,* 1992, pp. 100–107.

[4] Bultman, W.J., and W. Maass. "Fast Identification of Geometric Objects with Membership Queries." In *Proc. 4th Annual Workshop on Computational Learning Theory,* 1991, pp. 337–353.

[5] Chen, Y.C., W.-T. Chen and G.-H. Chen. "Efficient Median Finding and Its Application to Two-Variable Linear Programming on Mesh-Connected Computers with Multiple Broadcasting." *J. Parallel and Distributed Computing,* 15, No. 1 (May 1992), 79-84.

[6] Dally, W.J., and C.L. Seitz. "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks." *IEEE Trans. on Computers,* 36, No. 5 (May 1987), 547–553.

[7] Deng, X. "On the Parallel Complexity of Integer Programming." In *Proc. 1989 ACM Symp. on Parallel Algorithms and Architectures,* 1989, pp. 110-116.

[8] Fox, G.C., M.A. Johnson, G.A. Lyzenga, S.W. Otto, J.K. Salmon, and D.W. Walker. "Solving Problems on Concurrent Processors Vol. I: General Techniques and regular Problems." Englewood Cliffs: New Jersey: Prentice-Hall, 1988.

[9] Fraigniaud, P., S. miguet, and Y. Robert. "Scattering on a Ring of Processors." *Parallel Computing,* 13, No. 3 (March 1990), 377–383.

[10] Grötschel, M., L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization.* Berlin: Springer-Verlag, 1988.

[11] Hardy, G.H., and E.M. Wright. *An Introduction to the Theory of Numbers.* 5th ed. Oxford: Oxford University Press, 1979.

[12] Hedetniemi, S.M., S.T. hedetniemi, and A.L. Liestman. "A Survey of Gossiping and Boradcasting in Communication Networks." *networks,* 18, No. 4 (winter 1988), 319–349.

[13] Hirschberg, D.S., and C.K. Wong. "A Polynomial-Time Algorithm for the Knapsack Problem with Two Variables." *J. ACM,* 23, No. 1 (january 1976), 147–154.

[14] Hua, L.K. *Introduction to Number Theory.* Berlin: Springer-Verlag, 1982.

[15] Johnsson, S.L., and C.-T. ho. "Optimum Broadcasting and personalized Communication in Hypercubes." *IEEE Trans. on Computers,* 38, No. 9 (September 1989), 1249–1268.

[16] kannan, R. "A Polynomial Algorithm for the Two-Variable Integer Programming problems." *J. ACM,* 27, No. 1 (January 1980), 118–122.

[17] Kannan, R. "Algorithmic Geometry of Numbers." *Ann. Rev. Comput. Sci.,* 2(1987), 231–267.

[18] Karloff, H. *Linear Programming.* Boston: Birkhäuser, 1991.

[19] Knuth, D.E. *The Art of Computer Programming, Vol. II: Seminumerical Algorithms.* 2nd ed. Reading, Massachusetts: Addison-Wesley, 1981.

[20] Lová, L. *An Algorithmic Theory of Numbers, Graphs, and Convexity.* Philadelphia: SIAM, 1986.

[21] LYUU YUH-DAUH " On the Furthest-Distance-First Principle for Data Scattering with Set-up Time" In *Proceedings of the 5th IEEE Symposium on Parallel and Distributed Processing Dallas, TX, USA,* Dec 1993, 633–640.

[22] MEGIDDO, N. "Linear-time Algorithms for Linear Programming in $\mathcal{R}^3$ and Related problems." *SIAM J. Computing,* 12, No. 4(November 1983), 759–776.

[23] NEMHAUSER, G.L., AND L.A. WOLSEY. *Integer and Combinatorial Optimization.* New york: John Wiley, 1988.

[24] NEWMAN, M. *Integral Matrices.* New York: Academic press, 1972.

[25] NI, L.M., AND C.-T. KING. "On Partitioning and Mapping for Hypercube Computing. " *International Journal of parallel Programming,* 17, No. 6(December 1988), 475–495.

[26] PAPADIMITRIOU, C.H., AND K. STEIGLITZ. *Combinatorial Optimization: Algorithms and Complexity.* Englewood Cliffs, new Jersey: Prentice-Hall, 1982.

[27] PRESS, W.H., S.A. TEUKOLSKY, W.T. VETTERLING, AND B.P. FLANNERY. *Numerical Recipes in C: the Art of Scientific Computing.* 2nd. Cambridge: Cambridge University Press, 1992.

[28] REED, D.A., AND M.H. SCHULTZ. "The Performance of Multicomputer Interconnection Networks." *Computer,* 20, No. 6(June 1987), 63–73.

[29] SAAD, Y., AND M. H. SCHULTZ. "Data Communication in Parallel Architectures." it Parallel Computing, 11, No. 2 (August 1989), 131–150.

[30] STOUT, Q.F., AND B. WAGAR. "Intensive Hypercube Communication." *Journal of Parallel and Distributed Computing,* 10, No. 2(October 1990), 167–181.

[31] TANENBAUM, A.S. *Computer Networks.* Englewood Cliffs, New Jersey: Prentice-Hall, 1981.