Tree Search on an Atomic Model for Message-passing*

Pangfeng Liu
Department of Computer Science
and Information Engineering
National Chung Cheng University
Chiayi, Taiwan 621, R.O.C.

William Aiello
AT&T Labs, Research
AT&T Shannon Center
180 Park Avenue
Florham Park, NJ 07932

Sandeep Bhatt
Akamai Technologies, Inc.
Cambridge, MA 02139

Abstract

This paper presents a simple atomic model of message-passing multicomputers. Within one synchronous time step each processor can receive one atomic message, perform local computation, and send one message. When several messages are destined to the same processor then one is transmitted and the rest are blocked. Blocked messages cannot be retrieved by their sending processors; each processor must wait for its blocked message to clear before sending more messages into the network. Depending on the traffic pattern, messages can remain blocked for arbitrarily long periods.

The model is conservative when compared with existing message-passing systems. Nonetheless, we prove linear message throughput when destinations are chosen at random; this rigorously justifies an instance of folklore. Based on this result we also prove linear speedup for backtrack and branch-and-bound searches using simple randomized algorithms.

1 Introduction

The message-passing style of programming is widely used on almost all parallel computers. The primitives to *send* and *receive* messages hide low-level architectural details and are ideal for programming many large applications. While message-passing systems have been in use for over a decade, relatively few results concerning the complexity of message-passing protocols are available. One reason for this discrepancy is the lack of theoretical models that appropriately capture issues related to communication; as stated in [5], most theoretical models "encourage exploitation of formal loopholes, rather than rewarding development of techniques that yield performance across a range of current and future parallel machines."

We propose an atomic model [18] to study the performance of message-passing programs. The model is simple and much more restricted in its capabilities in comparison with existing systems. Nevertheless, we show that it allows simple and efficient solutions (linear speed-up) for message scattering, backtrack and branch-and-bound searches.

1.1 Message-passing Systems

Message-passing instructions appear in two varieties: *blocking* and *non-blocking*. Blocking instructions require synchronization between the sender and receiver: a send instruction terminates only when the corresponding receive is executed by a remote process. One advantage of blocking instructions is that no system buffering is required. However, the delay in waiting for a send instruction to complete means that computation and communication cannot overlap; this can reduce overall performance

*A preliminary version of this paper appeared in the 5th Annual ACM Symposium on Parallel Architectures and Algorithms.

significantly. Another disadvantage is that the programmer must carefully arrange send/receive instruction pairs to avoid deadlock.

Non-blocking instructions allow a process to execute multiple send instructions before any of the corresponding receive instructions is executed. This allows for the possibility of increased efficiency since communication and computation can overlap. However, more system resources, buffering and bandwidth for example, are required for a non-blocking scheme otherwise *pending* messages (those sent but not yet received) will be excessively delayed or potentially lost. Moreover, since system resources are finite, the programmer must ensure that the number of pending messages is bounded at all times.

Underneath the message-passing abstractions, a message goes through several phases before it is absorbed at its destination. During each phase it requires some critical system resource to continue its journey. For example, a memory buffer is required to compose a message. When a message buffer is sent, it goes through the network interface connecting the processor to the network. Before the message arrives at the destination, it travels in the network and occupies network buffers. On reaching its destination the message occupies a buffer at the network interface before it is removed and processed. Whenever a message cannot get the critical resource it needs, it must wait. When messages wait a long time, there is the danger that communication delays can cause processor idling, thereby reducing overall performance greatly.

In many applications it is also common practice to reduce communication costs (due primarily to system overheads) by aggregating data into fewer but longer, atomic, messages [3].[1] First the sender notifies the receiver of the message length. Upon receipt of this notification, the receiver allocates sufficient buffer space and sends back an acknowledgment. This establishes a link between the sender and the receiver and the message is transmitted in the third step. Once again, there is ample opportunity for delay from the time the protocol is initiated to the time the data is actually transferred.

Given the limited resources of multicomputer systems, it is natural to ask whether the efficiency gained by using non-blocking instructions is lost if the number of pending messages is limited. Nevertheless, we show that non-blocking communication can still achieve high performance, even with very limited communication resources.

We investigate this question formally within the atomic model which permits only one pending message per processor. In brief, each processor is given one send buffer and one receive buffer, each capable of holding one atomic message. The system alternates between message transmission and computation cycles. During a computation cycle a processor retrieves a message from its receive buffer, performs a computation, enqueues newly generated messages into a message queue, and writes the first message in the queue into the send buffer if the send buffer is empty. During the transmission cycle, the network attempts to transmit every message in each send buffer to the receive buffer of the destination. If more than one message is destined for the same processor, exactly one is successfully transmitted. The rest remain in their send buffers. The one which is transmitted is chosen by a *network arbiter*. The *worst-case* arbiter makes choices to maximize the running time. The *FIFO* arbiter gives priority to messages with smaller time-stamp; messages with the same time-stamp can be delivered in arbitrary order.

The atomic model is motivated by the desire to analyze the performance of message-passing programs in an architecture-independent manner. For this reason, we have chosen to abstract the network as an arbiter which takes one unit of time to transfer messages from send buffers to receive buffers at the destination. We believe this is reasonable in applications that involve the atomic transfer of large data sets. Unit-delay assumptions are also made in the literature on PRAMs and complete networks [13, 14]. Unlike these models however, we explicitly account for message contention and do not allow multiple messages to be received in one step by a processor. The issue of contention at the receiving module is also addressed in models for optical communication [8] and module parallel computers [12, 20]. A key feature which distinguishes the atomic model is that once a message has been sent it cannot be retrieved; the sending processor must wait for the network to clear the send buffer after the message has been copied into the receive buffer at the destination. Finally, the atomic model can be viewed as the limiting version of the LogP model [5]; with long messages of equal length the latency, overhead and gap parameters of the LogP model can be lumped into a single, unit time delay.

[1] Atomic messages travel through a critical resource as a single entity; different messages do not co-exist inside the critical section.

Communication contention is an important issue in modeling parallel computations. Valiant's Bulk Synchronous Parallel (BSP) model [24] divides the parallel computation into supersteps in which processors perform local computation and exchange data. All the outstanding communication requests will be serviced before the next superstep starts. The memory contention is characterized by the length of the time interval a processor must wait before sending the next message. To model the fact that many shared memory machines now have a large number of memory banks in order to serve relatively much faster processors, Blelloch et al. extended BSP into (d, x)-BSP [4] by adding two parameters – the memory bank delay (the minimum interval length a memory bank can serve memory requests), and the ratio of the number of memory banks to processors. The QSM model [9] also characterizes the contention problem by a bandwidth parameter g so that a processor can issue memory requests only once every g steps. All these models characterize the contention by limiting the communication capability on a *per processor* basis. In contrast the QSM(m) model by Alder et al. [1] added another parameter to describe the limitation on the communication capability for the *entire* system.

Besides characterizing the memory contention by a memory bandwidth parameter, it is also possible to model the contention by allowing atomic access to shared resource. Dwork et al. [6] proposed a model for shared memory access in which simultaneous accesses to a single memory location are serialized and only one will succeed at a time. However, a process may have multiple pending operations due to trying to access different memory locations. Gibbons et al. [11] proposed a queue-read, queue-write (QRQW) model that allows concurrent reading and writing to the same memory location to be queued. Similar to BSP, the computation is divided into supersteps and all the queued memory requests in one superstep will be served before the next one. The asynchronous variant of QRQW (AQRQW) [10] relaxes the bulk synchronous requirement of QRQW and BSP, and more accurately captures the memory contention phenomenon in modern shared-memory parallel computers [10]. The atomic model we propose differs from all the previous *message-passing* models that it only allows one pending outstanding "request" *per processor*[2].

Despite of the restriction on the rate at which the network can deliver messages to a destination, as well as the adversarial nature of the arbiter, we show that simple randomized algorithms can attain linear speed-up for branch-and-bound and backtrack tree searching. Furthermore, all-to-some message passing can finish within a constant factor of the optimal time with high probability if the destinations are uniformly distributed among the processors.

2 The Atomic Message-passing Model

We model a message-passing multicomputer as a collection of p nodes connected via an interconnection network [23]. For convenience of analysis we require that the system be synchronous, and operate in discrete time steps.[3] Each time step is divided between one communication step and one node computation step.

Each node consists of a receive buffer, a processor, local memory, a queue manager, a message queue, and a send buffer. Each buffer can hold one atomic message. Every node can perform local computation using its processor and local memory. It can also receive a message using the receive buffer and enqueue messages into the message queue. The message queue is maintained by a queue manager which may be under the control of the processor or the system. A message from the message queue is injected into the network by placing it into the send buffer. For our purposes, it is convenient to model the actions at a node as repeated executions of the following *reactive cycle* which occurs during one synchronous time step. Notice that we consider the communication as a sequence of time steps, and the timing of an event is expressed as the time step number the event occurs, not a particular point in the absolute time line. Similarly, a time interval refers to a contiguous set of time steps, not an interval in terms of wallclock time.

1. **The send phase** (performed by the processor):
   Transfer the message at the head of the queue into the send buffer if it is empty.
2. **The transmission phase** (performed by the network system):
   Take messages from send buffers to receive buffers according to message destinations. If more than one message is destined for the same receive buffer, the one which succeeds is selected by the network arbitration policy.

[2]The SIMD-QRQW model in [11] also allows only one pending request per processor in shared memory.
[3]This assumption is not required for termination, but simplifies the analysis of throughput.

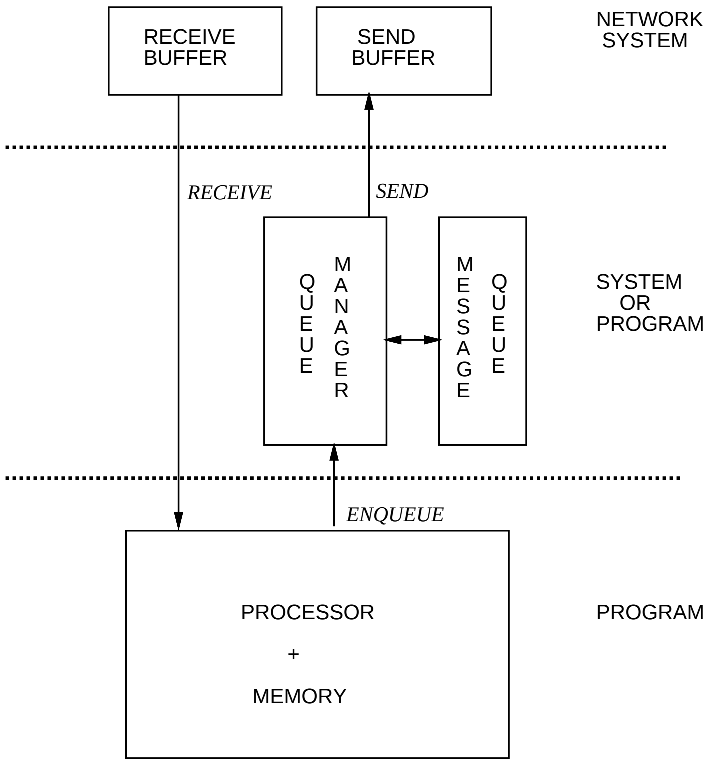

Figure 1: The structure of a node.

3. **The receive phase** (performed by the processor):
   Probe the receive buffer to receive an incoming message, if any, into local memory.
4. **The local computation phase** (performed by the processor):
   - COMPUTE: Perform local computation, possibly on the newly received message, and generate new messages.
   - ENQUEUE: Pass the newly generated messages to the queue manager, which will place it into an appropriate place in the message queue.

Observe that there are two ways a message can be delayed. First, a message may have to wait in the message queue until it is selected to be placed in the send buffer. Second, once a message is in the send buffer, it may be delayed in the network. We call the second kind a *receive delay.*

When more than one message, occupying send buffers of different nodes, are simultaneously destined for the same node, the network must deliver one message. Since every node executes a RECEIVE instruction during its reactive cycle, this requirement of the network satisfies the network contract of the CM-5 [17]: "The data network promises to eventually accept and deliver all messages injected into the network by the processors as long as the processors promise to eventually eject all messages from the network when they are delivered to the processors." With the reactive cycle and the network contract we are assured that deadlocks cannot occur.

We wish to make as few assumptions as necessary on the message queue. Our results for backtrack search are independent of queue maintenance. Our result for branch-and-bound depends on maintaining the message queue as a priority queue.

We also wish to make as few assumptions as necessary about the network arbitration policy when multiple messages are destined for the same node. We will consider two different network arbitration policies. The *worst-case* policy selects the message which maximizes the overall time to complete the task at hand. The *FIFO* policy dictates that, for any pair of messages with the same destination, they will be accepted in the order of earliest occupancy of their respective send buffers. In other words, if the messages reach their send buffers at different time steps, then the earlier one will be delivered first. If two messages reach their send buffers at the same time, then the order of delivery is arbitrary. Optimal speedup for backtrack search can be achieved even with worst-case arbitration; whereas, we require FIFO arbitration to prove optimal speedup for branch and bound search.

3 Overview of Results

We will study three problems under the atomic message setting: all-to-some message scattering, backtrack search, and branch-and-bound search. For each of these problems we analyze the case when all messages are destined for independently chosen random nodes. Our intuition is that when messages are headed for random destinations, the number of conflicting messages is unlikely to become too large. However, when the size of the computation is much larger than the number of processors, this is not always true and one has to prove that the effects of the conflicts do not add up significantly.

The message scattering problem is informally stated as follows: suppose that each node has a list of m messages to send (in order) to remote nodes. How much time does it take, under the worst-case (adversarial) arbitration policy, until all messages are received at their destinations? This problem arises naturally in several applications. In fact, the message scattering problem and the atomic message model are motivated by the "all-to-some" communication in our parallel N-body implementation [19].

In the backtrack search problem, each internal vertex of a search tree $\mathcal{T}$ corresponds to a partial solution to a problem while each leaf represents a solution with a certain cost. The goal of backtrack search is to find the minimum cost leaf in the search tree. The search tree is not given in advance, rather it is spawned on-line as the search proceeds. The search begins with the root of the tree in a given node; when each internal vertex is expanded two (or any bounded number of) children are spawned and must each be examined. When a leaf is examined, the cost is calculated and no further expansion along that branch is possible. If the total number of vertices in the search tree is n, and the maximum depth of any leaf is h, it is easy to see that the time to examine all leaves is at least $\Omega(n/p + h)$, where p denotes the number of processors.

Branch-and-bound search is similar to backtrack search, except that only a subtree of the search tree must necessarily be explored. Following Karp and Zhang [13, 14], we model a branch-and-bound tree as a binary search tree, each of whose vertices has an associated cost. The cost of each vertex is strictly less than the cost of each of its children (for simplicity we assume that all vertex costs are distinct). The problem is to find the leaf with minimum cost in the tree. Clearly, every tree vertex whose cost is less than the minimum cost leaf must be expanded because one of its children could potentially be the minimum cost leaf. These vertices form a critical subtree, call it $\mathcal{T}$ of the overall search tree.

As before, the time to complete the search is $\Omega(n/p + h)$ where n is the number of vertices in the *critical* subtree, and h is the height of the critical subtree. Non-critical vertices can, in principle, be pruned by the search process and need not be explored.

Tight upper bounds for branch-and-bound, and hence for backtrack search, were given by Karp and Zhang [13] on the complete network which allows multiple messages to be simultaneously received at each node, and on the concurrent PRAM which essentially allows unsuccessful writes to be detected. The basic idea was to send each node to a random processor for further exploration. Ranade [21] gave an elegant alternative proof of the Karp-Zhang result. By extending Ranade's techniques we show that the random destination strategy yields linear speedup for backtrack search in the atomic model.

Theorem 1 *Using random destinations, the probability that a binary backtrack search tree of size n and depth h takes time more than $k(n/p + h)$ in the atomic transmission model with worst-case arbiter is polynomially small in n, for k sufficiently large.*

Achieving linear speedup for branch-and-bound in the atomic model is a little harder. The subtle distinction is that pending non-critical vertices can delay pending critical vertices. In the Karp-Zhang model this can never happen. Since we have no control over the number of non-critical vertices, and we do not know the shape of the critical subtree, it is conceivable that the delays can become arbitrarily large under the worst-case arbiter which consistently favors non-critical vertices over critical vertices. However, under a FIFO arbiter we establish the following result.

Theorem 2 *Let the critical subtree, $\mathcal{T}$ of a branch-and-bound search tree have size n and depth h. Using randomized destinations, the probability that the time, in the atomic model with FIFO arbiter, exceeds $k(n/p + h)$ is polynomially small in n when $n > p^2 \log p$, and k is sufficiently large.*

We present the proof of Theorem 1 and 2 in Section 6 and 7.

4 Message Scattering

The off-line version of the message scattering problem in which the lists can be reordered is easily solved using standard bipartite graph edge-coloring techniques [2, 7]. If r and m are the maximum numbers of messages received and sent by any node respectively, then $\max\{r, m\}$ steps are necessary and sufficient.

However, the distributed version of the problem, without reordering, is not as simple. We show that with each of p nodes sending m messages (m can be arbitrarily larger than p) the worst-case time is $\Omega(mp)$. In other words, the average throughput of the system is $O(1)$ messages received per time step, independent of the size of the system.

On a positive note, we show that when each of the messages is destined for a randomly chosen node (all destinations independent and uniformly drawn) then, with high probability, the time to completion is $O(m)$. As a result the average throughput is $\Omega(p)$ messages received per time step, asymptotically the maximum possible.

4.1 Lower Bounds

Suppose every processor sends n messages to every processor in ascending processor index order. We show that a simple FIFO network arbiter increases the communication time to $\Omega(np^2)$ so that on average only a constant number of messages are received in one time step. The network arbiter ensures that the messages are received in FIFO order; the message sent first is received first. The messages sent at the same time are received in increasing processor index order.

Figure 2 shows the history of four processors sending two messages to each destination in ascending processor index order. A square in the intersection of row i and column j indicates that processor p_i successfully sends a message at time step j. The numbers in the squares are the processor index of the destination. Notice that two successful sends to the same destination are p time steps apart because the messages are received in FIFO order. The total number of time steps is therefore $((n-1)p+1)p+(p-1) = \Omega(np^2)$.

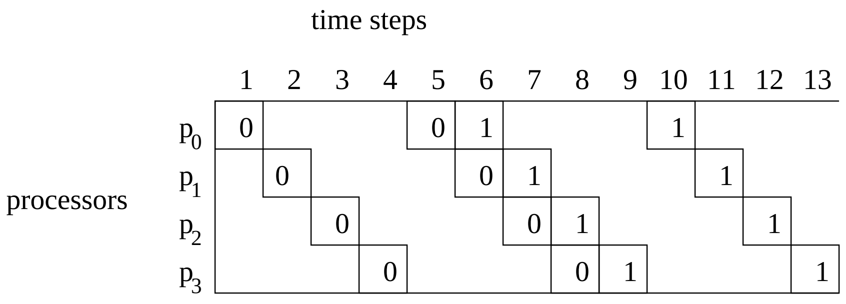

Figure 2: The situation when the messages are sent in ascending processor address order.

4.2 Randomized Scattering

Formally, we establish the following theorem.

Theorem 3 *Suppose that each node sends m messages, and that for each message all destinations are equally likely and independent of choices of all other messages. The probability that the time until all messages have been received exceeds km is bounded by $O(e^{-m})$, for sufficiently large k and $m > \log p$.*

Proof. We adapt Ranade's proof [21] of the result of Karp and Zhang [13].

Let T be the completion time of the protocol, the last time step at which a message is received. Let message $\mathcal{M}_m$ be a message received at time step T, and let S be the node which was the source of message $\mathcal{M}_m$. Let $\mathcal{M}_i$ denote the ith message sent by node S, and let T_i denote the time step at which $\mathcal{M}_i$ was received at its destination Q_i.

Definition. Suppose that message $\mathcal{M}$ is selected for transmission, i.e., $\mathcal{M}$ enters the send buffer at time step τ, and is destined for node q. Then we say that $\mathcal{M}$ became *ready for q at time step τ.*

Lemma 1 *There exists a partition $\Pi = \Pi_1, \ldots, \Pi_m$ of the interval $[1, T]$ and a set R of $T - m$ messages (not including those sent by S) each of which satisfies the following property: if the message became ready during Π_i its destination node is Q_i.*

Proof of Lemma. Message $\mathcal{M}_i$ is received at Q_i at time step T_i. Let $T_i^{nr} < T_i$ be the maximum time step at which Q_i does not receive a message. At each time step of the interval $\Delta_i = [T_i^{nr} + 1, T_i]$ Q_i receives a message. Each of these messages became ready during the same interval Δ_i.

Observe that message $\mathcal{M}_{i-1}$ was received at time step T_{i-1}, and message $\mathcal{M}_i$ became ready at time step $T_{i-1}+1$. Therefore, $T_i^{nr} \leq T_{i-1}$. As a result there is no gap between any pair of consecutive intervals Δ_i, Δ_{i+1}. Given the intervals $\Delta_1, \ldots, \Delta_m$, we construct a partition Π as follows:

$$\begin{aligned} \Pi_m &= \Delta_m \\ \Pi_i &= \Delta_i - \bigcup_{j>i} \Pi_j, \quad 1 \leq i < m. \end{aligned}$$

By construction, it follows that every message received by Q_i during Π_i became ready during Π_i, and at least $T - m$ messages received by Q_i's during Π_i's were not sent by S. This establishes the lemma. ■

To complete the proof of the theorem, we sum, over all possible partitions, choice of source S, and choice of $T - m$ messages, the probability that these $T - m$ messages chose their destinations in accordance with the partition.

The probability that a message which becomes ready during Π_i chooses Q_i as its destination equals $\frac{1}{p}$. The probability that each of $T - m$ messages makes the right choice is $p^{-(T-m)}$. The number of choices for S, the partitions and the $T - m$ messages equals $p\binom{T+m}{m}\binom{(p-1)m}{T-m}$. The probability that $T \geq km$ is at most

$$\begin{aligned} & p\binom{T+m}{m}\binom{(p-1)m}{T-m}p^{-(T-m)} \\ \leq\ & 2^{T+2m}\binom{(p-1)m}{(k-1)m}p^{-((k-1)m)} \\ \leq\ & 2^{(k+2)m}(\frac{e}{k-1})^{(k-1)m} \\ \leq\ & (2^{k+2}(\frac{e}{k-1})^{k-1}))^m \end{aligned}$$

The first two inequalities follow from the assumption that $m > \log p$, and the fact that $\binom{x}{y} < (xe/y)^y$ (the proof can be found in [16, page 165]. For k sufficiently large, this quantity is smaller than $O(e^{-m})$. ■

5 Techniques for Tree Searches

5.1 Algorithmic Issues

This section outlines the algorithmic and proof strategies for backtrack and branch-and-bound search in the atomic message model. The branch-and-bound strategy is essentially that of Karp and Zhang [13]; their model allows any number of messages to be received at a node in one time step. Our technical contribution is to extend their result to the atomic transmission model. The proofs of both results extend the techniques of the previous section.

While the goal of both search procedures is to find the minimum-cost leaf, there is an essential difference. Backtrack search examines *every* vertex of the search tree. In branch-and-bound search the cost associated with each vertex increases monotonically with the distance from the root, so that only the *critical subtree*, consisting of vertices with cost no greater than the minimum-cost leaf, need be examined. We call such vertices *critical vertices.* For efficient branch-and-bound search, the time devoted to examining non-critical vertices must not dominate that for examining the critical subtree.

Within each synchronous reactive cycle, each processor: (1) receives a tree vertex, if any, from its receive buffer, (2) examines and expands the vertex, and (3) puts the children onto the message queue, headed for an independently chosen random destination. For backtrack search we place no requirements on the message queue discipline. However, for branch-and-bound search we require that the message queue be a priority queue, so that the tree vertex selected for transmission is the one with minimum cost.

Using priority message queues for branch-and-bound search means that non-critical vertices cannot be selected for transmission when there is at least one critical vertex inside the message queue. However, a critical vertex can arrive inside the message queue while a non-critical vertex occupies the send buffer. In this case, the critical vertex will have to wait for selection, but it is easy to see that a critical vertex can be delayed by a non-critical vertex in this manner at most once.

Once a message has been selected for transmission, it is still subject to receive delays. Receive delays depend on the network policy and are beyond the control of the programmer, so we would like to make as few assumptions as necessary. For backtrack search we are able to carry out the analysis without making any assumptions on network arbitration. For branch-and-bound however, our analysis requires that the network observes a FIFO arbitration policy.

In conclusion, our analysis for branch-and-bound search makes stronger assumptions on both the message queue discipline, and the network arbitration policy. The first assumption is required to guarantee that progress is made on the critical subtree and is reasonable from an algorithmic viewpoint. The second assumption, concerning network arbitration, is required for technical reasons: we bound the running time as a function of the size of the critical subtree, not the entire search tree which can be arbitrarily larger. Currently we do not know if the FIFO assumption can be weakened, and it is conceivable that it can.

5.2 Proof Techniques

In this section we describe some of the ideas and terminology common to the analysis for both backtrack and branch-and-bound search. In both problems our goal is to analyze the time to expand a critical tree[4] of size n and depth h on a p-node system. For branch-and-bound search the quantities n and h can each be much smaller than the size and depth of the complete search tree.

In the analysis of the running time we proceed as follows. At time $t = 1$ the root is assumed to be in the send buffer of some node, and is received at its destination within that time step. Suppose that the running time is T, i.e., the last time step at which a critical vertex is received. Pick one of the critical vertices received at time T – it must be a leaf in the critical subtree. Call the path $s_1, s_2, \ldots, s_h$ from the root (s_1) to this critical leaf (s_h) the *special path* and the vertices along this path the *special vertices.* Let Q_i denote the destination queue of special vertex s_i.

The first step of the proof is similar to the proof of Lemma 1. For a fixed run of the algorithm we construct a partition, $\Pi = \{\Pi_1, \ldots, \Pi_h\}$, of the time interval $[1, T]$. Next, we construct a *signature set* R of non-special, critical vertices each of which became ready for some Q_i during the corresponding time interval Π_i. Roughly speaking, the signature set, R, is constructed such that the receive delay periods of its children are disjoint, and the sum of these receive delays is large, i.e., close to T.

There are two cases to consider: the signature set R is either large or small, compared with the threshold αT, where α is some suitably chosen constant. We first show that it is unlikely that R is large when T is large. The proof closely follows the proof of Theorem 3.

Lemma 2 *For suitable constants k, α, the probability that $T > k(\frac{n}{p} + h)$ and $|R| \geq \alpha T$ is polynomially small in n.*

Proof.

We estimate $Pr(|R| \geq \alpha T)$ by summing the probability of the event $|R| \geq \alpha T$ under all possible combinations of partition Π and queue sequence Q. Given Π and Q each critical vertex appears in

[4]Every vertex in backtrack search is critical.

R with probability $1/p$, independent of other vertices. The probability that $|R| \geq \alpha T$ and all the special nodes go to destinated Q_i is therefore bounded by $\binom{n}{\alpha T} p^{-(h+\alpha T)}$.

The number of choices for Π, Q and the special path S is no more than $\binom{T+h}{h} p^h n$. Thus, the probability that $|R| \geq \alpha T$ is bounded by

$$\begin{aligned}
& \binom{n}{\alpha T} p^{-\alpha T} \binom{T+h}{h} n \\
< \;& (\frac{ne}{\alpha T p})^{\alpha T} ((1+T/h)e)^h n \\
< \;& (\frac{e}{\alpha k})^{\alpha k (\frac{n}{p}+h)} ((k(\frac{n}{ph}+1)+1)e)^h n, \quad \text{when } T > k(\tfrac{n}{p}+h) \\
< \;& [(\frac{e}{\alpha k})^{\alpha k (\frac{n}{ph}+1)} ((k(\frac{n}{ph}+1)+1)e)]^h n \\
< \;& [(\frac{e}{\alpha k})^{\alpha k} ((2k+1)e)]^h n
\end{aligned}$$

which, for appropriately chosen k, is polynomially small in n, the size of the critical subtree since the height of the tree h is at least $\log n$. ■

The second part of the proof argues that it is unlikely that R is small when T is large. The intuition is that the expected receive-delay of any vertex is a small constant; therefore, it would seem unlikely for the children of a small number of signature vertices to suffer a large total receive-delay. Unfortunately, the delays of the children of the signature vertices are not independent random variables, so that Chernoff bounds cannot be immediately invoked.

Briefly, in analyzing backtrack search we track the destinations of the children of the signature vertices to construct a new set of queues, a new partition of time, and a new signature set. The new signature set is guaranteed to be large; consequently, the remainder of the proof follows the proof of Lemma 2. The analysis of branch-and-bound is based on the observation that, under FIFO arbitration, the delays of the children of the signature set can essentially be treated as a martingale, thereby allowing us to use Chernoff bounds.

6 Analysis of Backtrack Search

In this section we demonstrate that when each vertex chooses its destination randomly and independently, then with high probability the completion time of backtrack search is optimal within a constant factor.

Following the outline of the previous section, we proceed in two stages. In the first stage we identify the required signature set R; the second stage establishes the unlikelihood of the event that T is large while R is small.

6.1 Signature Set

We begin with some terminology and definitions. As before, let $S = \{s_1, \ldots, s_h\}$ denote the special vertices along the special root-to-leaf path and Q_i be the destination queue which receives special vertex s_i at time T_i.

Definition. A node Q is *empty at time t* if neither the send buffer nor the message queue of Q contains a vertex *right after the send phase of time step t*. By definition a node cannot be empty at time t if it receives an internal vertex at time $t-1$. However, it is possible that a node which is empty at t receives a leaf at time $t-1$. Note that any node which is non-empty throughout an interval I attempts to inject a vertex into the network at every time step of I.

Definition. Suppose that node Q receives a vertex at each time-step during time interval W. We call the interval W an *arrival window* for node Q. Recall that a receive delay is the time interval during which a message waits in the send buffer due to destination congestion. Note that if time interval W is the receive delay of vertex v, then W is an arrival window for the destination queue of v.

Definition. For $1 \leq i < h$, let T_i^e denote the maximum time $t < T_{i+1}$ such that Q_i is empty at t, and T_i^{nr} denote the maximum time $t \leq T_i^e$ at which Q_i does not receive a message. Finally, let N_i denote the interval $[T_i^e + 1, T_{i+1}]$, $A_i = [T_i^{nr} + 1, T_i^e]$, and $\Delta_i = A_i \cup N_i$.

The following lemma summarizes three properties which are a straightforward consequence of the definitions above.

Lemma 3

1. *Q_i receives an internal vertex at time T_i^e,*
2. *Q_i is non-empty throughout N_i and Q_i attempts to inject a vertex at every step of N_i, and*
3. *A_i is an arrival window for Q_i.*

Let c_i denote the set of vertices that are injected into the network from Q_i during N_i. We obtain the following lemma.

Lemma 4

1. *The parent of every vertex in c_i becomes ready for Q_i during Δ_i,*
2. *Δ_i can be partitioned into a set of arrival windows (not necessarily for all Q_i), and*
3. *the interval Δ_i contains $[T_i, T_{i+1}]$, and $\bigcup_{i=1}^{h-1} \Delta_i = [1, T]$.*

Proof.

Let w be the parent of a vertex $v \in c_i$. In contradiction to (1), suppose w is ready at or before T_i^{nr}. Two cases follow: either w is received before T_i^{nr} or after T_i^{nr}.

In the first case, if w is received before T_i^{nr}, then v will stay in the message queue of Q_i until it becomes ready. However, from the definition, v cannot become ready until $T_i^e + 1$ or later. This contradicts the fact that Q_i is empty at T_i^e.

In the second case, if w is received after T_i^{nr}, then the receive delay of w is an arrival window for Q_i. This contradicts the fact that Q_i does not receive a vertex at T_i^{nr}. As a result w must be ready after T_i^{nr}.

From part 2 of Lemma 3 N_i can be partitioned into arrival windows for the destinations of c_i. Therefore, Δ_i can be partitioned into arrival window A_i for Q_i and a set of arrival windows in N_i for the destinations of c_i.

Finally, for part 3 we observe that $s_{i+1} \in c_i$, so s_i must be ready after T_i^{nr} (from part 1 of this lemma) and is received at $T_i > T_i^{nr}$. Therefore Δ_i contains $[T_i, T_{i+1}]$ and part 3 follows. ■

From part 3 of Lemma 4 the union of all Δ_i covers the time interval $[1, T]$, consequently we can define a partition $\Pi = \{\Pi_1, \ldots, \Pi_{h-1}\}$ of the interval $[1, T]$ as follows.

$$\begin{aligned} \Pi_{h-1} &= \Delta_{h-1} \\ \Pi_i &= \Delta_i - \bigcup_{j>i} \Pi_j, \quad 1 \leq i < h-1. \end{aligned}$$

Definition. Let R_i be the set of critical vertices which are not special ($v \notin S$) but are ready for Q_i during the interval Π_i. Also, let $R = \cup_{i<h} R_i$. We call the set R the *signature set.*

Having identified the signature set, it remains to estimate the probability that the signature set is small when T is large.[5] This estimation is completed in the following section.

6.2 A Refined Partition

As mentioned in Section 5, our strategy will be to find a new partition of the interval $[1, T]$ and a corresponding signature set which is guaranteed to be large.

In this section we identify $O(|R| + h)$ arrival windows which cover the interval $[1, T]$. We will find arrival windows to cover each Π_i ($1 \leq i < h$) and argue that the sum of number of windows in each Π_i is $O(|R| + h)$. The next section will identify the new partition and signature set.

[5]The case when R and T are both large is covered by Lemma 2.

Definitions. Let C_i denote those k_i children of $R_i \cup s_i$ that are ready within $N_i \cap \Pi_i$. We sort C_i into a list $v_{i,1}, \ldots, v_{i,k_i}$ according to the time they become ready. Also let $Q_{i,j}$ denote the destination of vertex $v_{i,j}$ and $W_{i,j}$ denote the receive delay of $v_{i,j}$.

Each Π_i is the union of two disjoint intervals $N_i \cap \Pi_i$ and $A_i \cap \Pi_i$. The interval $N_i \cap \Pi_i$ is an initial segment of N_i which, from part 2 of Lemma 4, can be partitioned into arrival windows. Each such arrival window W is the receive delay of a vertex v which is enqueued after T_i^e and becomes ready in Q_i at the beginning of W. From part 1 of Lemma 4 the parent of v must be ready for Q_i after T_i^{nr}. In other words, the parent of v must be ready during Π_i and consequently $v \in C_i$. As a result the interval $N_i \cap \Pi_i$ can be partitioned into the receive delays of the k_i vertices in C_i[6]. Each such receive delay is an arrival window $W_{i,j}$ for $Q_{i,j}$ $(1 \leq j \leq k_i)$.

From part 3 of Lemma 3 $A_i \cap \Pi_i$ is an arrival window for Q_i. Let $W_{i,0} = A_i \cap \Pi_i$ and $Q_{i,0} = Q_i$, so that the interval Π_i can be partitioned into arrival windows $W_{i,j}$ for $Q_{i,j}$ $(0 \leq j \leq k_i)$. Finally we consider Π_i for all i and it follows that $\bigcup_{1 \leq i < h} \bigcup_{0 \leq j \leq k_i} W_{i,j} = [1, T]$.

From the discussion above the receive delays, $W_{i,j}$, of the k_i vertices $v_{i,j} \in C_i$, and $W_{i,0}$ cover Π_i. Moreover the parent of every vertex in $C = \bigcup_{1 \leq i < h} C_i$ is in either R or S; consequently, the number of arrival windows is at most $h + |C| \leq h + 2(|S| + |R|) = 3h + 2|R|$ since we assume a binary search tree.

We need one more definition to derive the refined partition Π^* and queue sequence Q^* where we can find T vertices that become ready for Q^* according to Π^*.

Definition. For the arrival window $W_{i,j} = [t_1, t_2]$ for $Q_{i,j}$, let $t \leq t_1$ be the maximum time step at which $Q_{i,j}$ does not receive a message. Define $W^*_{i,j} = [t+1, t_2]$ so that $(W^*_{i,j}, Q_{i,j})$ is the *maximal backward extension* of the arrival window.

Let Q^* be the sequence of all the queues $Q_{i,j}$, $1 \leq i < h$, $1 \leq j \leq k_i$. Notice that every message received by $Q_{i,j}$ during the interval $W^*_{i,j} = [t+1, t_2]$ necessarily becomes ready for $Q_{i,j}$ during the same interval. Otherwise $Q_{i,j}$ would have received a message at time t, a contradiction.

From the extended windows $W^*_{i,j}$, $1 \leq i < h$, $0 \leq j \leq k_i$, we next obtain the partition Π^* as follows:

$$\begin{aligned}
\Pi^*_{h-1,k_{h-1}} &= W^*_{h-1,k_{h-1}} \\
\Pi^*_{i,j} &= W^*_{i,j} - \bigcup_{l > i \vee (l = i \wedge m > j)} \Pi^*_{l,m}
\end{aligned}$$

Now we can find T vertices that become ready for Q^* according to Π^* since every vertex received by Q^* must be ready in the corresponding Π^* interval, and the union of Π^* is $[1, T]$. Let $X_{i,j}$ denote the set of vertices v such that $v \notin C \cup R \cup S$, and v is received by $Q_{i,j}$ during $\Pi^*_{i,j}$. From the discussion above every vertex in $X_{i,j}$ must become ready for $Q_{i,j}$ during $\Pi^*_{i,j}$. Finally, let $X = \cup X_{i,j}$ and $V = C \cup R \cup S$. Since the arrival windows cover the interval $[1, T]$, it follows that $|X| \geq T - |V|$.

6.3 Execution Templates

Our goal in this section is to estimate the probability of the event that T is large and R is small. We proceed in two stages; first we characterize the completion time in terms of an *execution template*. Then we show that execution templates corresponding to large completion times are unlikely. This follows the delay-sequence arguments used in the literature [21, 22].

Definition. An execution template $\mathcal{E}$ is an octuple $(S, R, C, \Pi, \Pi^*, X, Q, Q^*)$ whose elements are defined as follows.

- $S = \{s_1, \ldots, s_h\}$ denotes the set of vertices along a path from the root to a leaf,
- R_i, $1 \leq i < h$, are disjoint sets of non-special critical vertices that become ready for Q_i during Π_i, and $R = \cup_{i=1}^{h-1} R_i$ is the signature set,
- C_i, $1 \leq i < h$, are disjoint sets of tree vertices that are children of $s_i \cup R_i$ and become ready within $N_i \cap \Pi_i$; $|C_i| = k_i$, and $C = \cup C_i$,
- $\Pi = \{\Pi_1, \ldots, \Pi_{h-1}\}$ is a partition of $[1, T]$,

[6]The receive delay of the last vertex v_{i,k_i} may not fall entirely within $N_i \cap \Pi_i$ in which case we truncate it accordingly.

- $\Pi^* = \{\Pi_{1,1}, \ldots, \Pi_{1,k_1}, \ldots, \Pi_{h-1,1}, \ldots, \Pi_{h-1\ k_{h-1}}\}$ is a partition of $[1, T]$,
- $X_{i,j}$, $1 \le i < h$, $0 \le j \le k_i$ are sets of tree vertices that are disjoint from $V = C \cup R \cup S$ and ready for $Q^*_{i,j}$ during $\Pi^*_{i,j}$, $X = \cup X_{i,j}$ and $|X| \ge T - |V|$,
- $Q = \{Q_1, \ldots, Q_h\}$ is a set of queues, such that for every $1 \le i < h$, Q_i is the destination queue of s_i and also of every vertex in R_i and $X_{i,0}$, and Q_h is the destination of s_h,
- $Q^* = \{Q^*_{1,1}, \ldots, Q^*_{1,k_1}, \ldots, Q^*_{h-1,1}, \ldots, Q^*_{h-1,k_{h-1}}\}$ is a set of queues, such that for every $1 \le i < h, 1 \le j \le k_i$, $Q^*_{i,j}$ is the destination for the jth element in C_i and every vertex in $X_{i,j}$.

From the earlier discussion, when the backtrack search takes T time steps to complete, there exists an execution template where the destination of vertices in S, R, C, and X satisfy the following conditions (let $D(v)$ be the random destination of vertex v):

1. $D(s_i) = Q_i$, $1 \le i \le h$.
2. $\forall v \in R_i$, $D(v) = Q_i$, $1 \le i < h$.
3. Let $v_{i,j}$ be the jth element in C_i, $D(v_{i,j}) = Q^*_{i,j}$, $1 \le i < h, 1 \le j \le k_i$.
4. $\forall v \in X_{i,j}$, $D(v) = Q^*_{i,j}$, $1 \le i < h, 1 \le j \le k_i$.
5. $\forall v \in X_{i,0}$, $D(v) = Q_i$, $1 \le i < h$.

6.4 Estimating the Probability of Execution Templates

Let $\mathcal{L}$ be the event $T > k(\frac{n}{p} + h)$ and $|R| < \alpha T$. We bound the probability of event $\mathcal{L}$ by summing the probabilities of event $\mathcal{L}$ under all possible execution templates. For a fixed execution template the probability that all vertices in S, R, C, and X choose the right queue according to $\mathcal{E}$ is at most

$$p^{-|S \cup R \cup C|} p^{-(T-|S \cup R \cup C|)} = p^{-h} p^{-|R|} p^{-|C-(S\cup R)|} p^{-(T-|S \cup R \cup C|)}$$

Next, we count the number of different execution templates. The number of possible S is n since there are at most n leaves in the tree. After S and R are specified, there are at most $\binom{2(|S|+|R|)}{|C|}\binom{n}{T-|V|}$ ways to choose C and X. The destinations of vertices from S and R are specified by Q, so the number of unspecified queues in Q^* is $|C - (S \cup R)|$ and the number of ways to choose Q and Q^* is $p^h p^{|C-(S\cup R)|}$. Finally there are $\binom{T+|C|+h}{|C|+h}\binom{T+h}{h}$ ways to choose Π^* and Π, so the total number of execution templates is at most

$$n\binom{n}{|R|}\binom{2(|S|+|R|)}{|C|}\binom{n}{T-|V|} p^h p^{|C-(S\cup R)|} \binom{T+|C|+h}{|C|+h}\binom{T+h}{h}$$

Lemma 5 *For suitable constants $k > 1, 0 < \alpha < 1/3$ the probability that $T > k(\frac{n}{p}+h)$ and $|R| < \alpha T$ is polynomially small in n.*

Proof.

The probability of $\mathcal{L}$ is no more than the product of the number of different execution templates and the probability that every vertex in S, R, C, and X will actually choose the destination according to $\mathcal{E}$ when $T > k(\frac{n}{p} + h)$ and $|R| < \alpha T$.

$$\begin{aligned}
& n\binom{n}{|R|}\binom{2(|S|+|R|)}{|C|} p^{-|R|} \binom{n}{T-|V|} p^{-(T-|V|)} \binom{T+|C|+h}{|C|+h}\binom{T+h}{h} \\
\le\ & 2^{\log n} 2^{2(|S|+|R|)} \binom{n}{\lceil \frac{n-p}{p+1} \rceil} p^{-\lceil \frac{n-p}{p+1} \rceil} \left(\frac{ne}{(T-|V|)p}\right)^{(T-|V|)} 2^{T+|C|+h} 2^{T+h} \\
\le\ & 2^{(2\alpha+2)T+5h+|C|} \left(\frac{ne}{\lceil \frac{n-p}{p+1} \rceil p}\right)^{\lceil \frac{n-p}{p+1} \rceil} \left(\frac{ne}{(T-|V|)p}\right)^{T-|V|} \\
\le\ & 2^{(2\alpha+2)T+5h+2h+2\alpha T} \left(\frac{ne}{\frac{n}{2p} p}\right)^{\frac{n}{p}} \left(\frac{ne}{(T-3\alpha T-3h)p}\right)^{T-3\alpha T-3h} \quad \text{since } |V| \le 3(\alpha T + h) \\
\le\ & 2^{7h+(4\alpha+2)k(\frac{n}{p}+h)} (2e)^{\frac{n}{p}} \left(\frac{ne}{((1-3\alpha)k-3)(\frac{n}{p}+h)p}\right)^{((1-3\alpha)k-3)(\frac{n}{p}+h)} \\
\le\ & 2^{((4\alpha+2)k+7)(\frac{n}{p}+h)} (2e)^{\frac{n}{p}} \left(\frac{e}{((1-3\alpha)k-3)}\right)^{((1-3\alpha)k-3)(\frac{n}{p}+h)} \\
\le\ & \left[2^{(4\alpha+2)k+7} (2e) \left(\frac{e}{((1-3\alpha)k-3)}\right)^{((1-3\alpha)k-3)}\right]^{(\frac{n}{p}+h)} \\
\le\ & 2^{-(\frac{n}{p}+h)}, \text{ for suitably chosen constants } k, \alpha.
\end{aligned}$$

The first inequality follows from the observation that $\binom{n}{x}p^{-x}$ is maximized when $x = \lceil \frac{n-p}{p+1} \rceil$. The second inequality follows from the fact that $\log n < h$, $|S| = h$, and the assumption that $R \le \alpha T$. The probability in the fourth inequality is roughly $2^{k_1 T} k_2^{n/p} k_3^{k_4 T} < (2^{k_1} k_2 k_3^{k_4})^T$. As long as we keep $(2^{k_1} k_2 k_3^{k_4}) < 1$ (by choosing sufficiently large k) the probability will diminish when n (and T) is sufficiently large. Therefore when n is sufficiently large, we replace T with $k(n/p + h)$. Finally, since $h \ge \log n$, the bound in the last step is polynomially small in n. ■

From Lemmas 2, 5 we have the following theorem.

Theorem 4 *Let $\mathcal{T}$ be any binary backtrack search tree of size n and depth h. Let T be the total time for the random destination backtrack search algorithm to expand $\mathcal{T}$ in a p-node network. The probability that T exceeds $k(\frac{n}{p} + h)$, where k is suitably chosen, is polynomially small in n.*

7 Analysis of Branch-and-bound Search

The proof for backtrack search does not apply in the branch-and-bound search case because an adversarial network arbiter can delay a critical node by favoring non-critical nodes. In the backtrack case, every tree node has to be expanded. Therefore, no matter which tree node the arbiter chooses to be received, some progress is made. In the branch-and-bound case, although a critical node cannot be delayed by a non-critical node in the competition for the send buffer, it can be delayed by non-critical nodes in the competition for the same destination. An adversarial arbiter can work against the critical nodes so that they suffer long receive delays.

Our analysis of branch-and-bound search is based on the assumption that the network obeys the first-in-first-out (FIFO) scheduling policy. Under FIFO scheduling incoming vertices are received in time-stamp order; a vertex that is ready cannot be delayed by a vertex that becomes ready at a later time-step; vertices that become ready at the same time can be received in arbitrary order.

We prove linear speedup in two steps. First, we prove that the aggregate delay of m non-overlapping receive delays is bounded by $O(m)$ with high probability under FIFO scheduling. Next, we show that for every execution there exists a signature set R and a set of $O(|R|+h)$ non-overlapping receive delays with aggregate delay $\Omega(T)$. As a result, it is very unlikely for T to be large and $|R|$ to be small. The other case, that of large T and large $|R|$ is already covered by Lemma 2.

7.1 Martingales

Lemma 6 *Let $X_1, \ldots, X_m$ be m random variables each in the range $[0..p-1]$, and let $X = \sum_{i=1}^{m} X_i$. Suppose that the conditional expectation $E(X_i \mid X_1 = x_1, \ldots,\ X_{i-1} = x_{i-1}) \le 1$, for all $1 \le i \le m$, and $0 \le x_1, \ldots, x_{i-1}, \le p-1$. Then, $Pr(X \ge \alpha m) \le (\frac{1}{2})^{\alpha \frac{m}{p-1}}$ when $\alpha \ge 2e$.*

Proof. The analysis is similar to the generalized Chernoff bound given by Leighton et al. in [15]. We first estimate the expectation of e^{tX}.

$$\begin{aligned} E(e^{tX}) &= E(e^{tX_1} e^{tX_2} \cdots e^{tX_m}) \\ &= \sum_{x=0}^{p-1} e^{tx} E(e^{tX_2} \cdots e^{tX_m} | X_1 = x) Pr(X_1 = x) \end{aligned}$$

We then choose a value x_1^* for X_1 so that $E(e^{tX_2} \cdots e^{tX_m} | X_1)$ is maximized.

$$\begin{aligned} E(e^{tX}) &\le \sum_{x=0}^{p-1} e^{tx} E(e^{tX_2} \cdots e^{tX_m} | X_1 = x_1^*) Pr(X_1 = x) \\ &\le (\sum_{x=0}^{p-1} e^{tx} Pr(X_1 = x)) E(e^{tX_2} \cdots e^{tX_m} | X_1 = x_1^*) \\ &\le E(e^{tX_1}) E(e^{tX_2} \cdots e^{tX_m} | X_1 = x_1^*) \\ &= E(e^{tX_1}) \sum_{x=0}^{p-1} e^{x} E(e^{tX_3} \cdots e^{X_m} | X_1 = x_1^*, X_2 = x) Pr(X_2 = x | X_1 = x_1^*) \end{aligned}$$

$$
\begin{aligned}
&= E(e^{tX_1})E(e^{tX_2}|X_1 = x_1^*)E(e^{tX_3}\cdots e^{tX_m}|X_1 = x_1^*, X_2 = x_2^*) \\
&\vdots \\
&\leq E(e^{tX_1})E(e^{tX_2}|X_1 = x_1^*)\cdots E(e^{tX_m}|X_1 = x_1^*, \ldots, X_{m-1} = x_{m-1}^*)
\end{aligned}
$$

Each of these expectations is maximized when the probability is non-zero only at 0 and $p-1$, the endpoints of the range of X_i. From Markov's inequality we can bound $E(e^{tX_i}|X_1 = x_1^*, \ldots, X_{i-1} = x_{i-1}^*)$ by $Pr(X_i = 0|X_1 = x_1^*, \ldots, X_{i-1} = x_{i-1}^*) + Pr(X_i = p-1|X_1 = x_1^*, \ldots, X_{i-1} = x_{i-1}^*)e^{t(p-1)} \leq (1 - \frac{1}{p-1}) + \frac{1}{p-1}e^{t(p-1)}$ since $E(X_i|X_1 = x_1^*, \ldots, X_{i-1} = x_{i-1}^*)$ is bounded by 1 from the assumptions. As a result we choose t so that $e^{t(p-1)}$ is larger than 1 and $E(e^{tX_i}|X_1 = x_1^*, \ldots, X_{i-1} = x_{i-1}^*)$ is maximized when $Pr(X_i = p-1|X_1 = x_1^*, \ldots, X_{i-1} = x_{i-1}^*)$ is maximized.

$$
\begin{aligned}
E(e^{tX}|X_1 = x_1^*, \ldots, X_{i-1} = x_{i-1}^*) &\leq ((1 - \frac{1}{p-1}) + \frac{1}{p-1}e^{t(p-1)})^m \\
&= (1 + \frac{e^{t(p-1)} - 1}{p-1})^m \\
&\leq e^{(\frac{e^{t(p-1)}-1}{p-1})m} \qquad (1 + y < e^y)
\end{aligned}
$$

Then we use Markov's inequality again to bound the probability that X is greater than αm.

$$
\begin{aligned}
Pr(X \geq \alpha m) &= Pr(e^{tX} \geq e^{t\alpha m}) \\
&\leq \frac{e^{(\frac{e^{t(p-1)}-1}{p-1})m}}{e^{t\alpha m}} \\
&= e^{-\frac{(\alpha \ln \alpha - \alpha + 1)m}{p-1}} \qquad (\text{when } t = \tfrac{\ln \alpha}{p-1}) \\
&\leq 2^{-\frac{\alpha m}{p-1}} \qquad (\text{when } \alpha \geq 2e)
\end{aligned}
$$

■

Lemma 7 *Let $V = \{v_1, \ldots, v_m\}$ be m vertices with non-overlapping delays. The probability that their aggregate delay exceeds βm is smaller than $(\frac{1}{2})^{\frac{(\beta-1)m}{p-1}}$ when $\beta \geq 2e + 1$.*

Proof. Let Q_i be the destination of v_i and X_i be the number of vertices that will be received by Q_i before v_i when v_i becomes ready. The receive delay of v_i is $X_i + 1$ and the aggregate receive delay of V is $m + \sum_{i=1}^{m} X_i$.

Every vertex chooses its destination independently and uniformly; therefore, given $X_1, \ldots, X_{i-1}$, v_i is equally likely to pick any destination. We will argue that, given $X_1, \ldots, X_{i-1}$, the expected value of X_i is no more than 1. When v_i makes its random choice there are at most $p-1$ other ready vertices in the system whose choices are independent of v_i's choice. Therefore, the conditional expectation of X_i is less than one.

For the aggregate delay to exceed βm, the sum of all X_i must exceed $(\beta - 1)m$. The bound on the probability of this event follows from Lemma 6. ■

7.2 The Signature Set

As before we consider the special vertices $S = \{s_1, \ldots, s_h\}$. Let s_i be received by Q_i at time T_i for $1 \leq i \leq h$. For every $1 \leq i < h$ we seek a set of receive delays which together cover the interval $[T_i, T_{i+1}]$.

Let T_i^{ns} be the largest time step smaller than T_{i+1} at which the send buffer of Q_i is not occupied by a critical vertex, $(1 \leq i < h)$. Note that at each time step during the interval $\Gamma_i = [T_i^{ns} + 1, T_{i+1}]$ the send buffer of Q_i is occupied by a critical vertex. Let c_i be the critical vertices that are injected into the network from Q_i during Γ_i. As a result, Γ_i can be partitioned into receive delays of vertices in c_i.

Among all the parents of vertices in c_i let f_i be the one that becomes ready at the earliest time step, say, T_i^f. Since $s_{i+1} \in c_i$ it follows that s_i is received no earlier than T_i^f.

It is possible for a gap to exist between the receive delays of f_i and vertices in c_i. In this case, the send buffer of Q_i must be occupied by a non-critical vertex, call it g_i, which is received at its destination at time T_i^{ns}. Observe that g_i cannot be received at its destination any earlier, for otherwise the send buffer of Q_i would have to be occupied by a critical vertex (the message queue contains at least one critical vertex, the child of f_i, and a critical vertex gets priority over non-critical vertices to enter the send buffer). But this contradicts the definition of T_i^{ns}.

Let T_i^g be the time step at which the parent of g_i becomes ready, and let $\Delta_i = [\min(T_i^f, T_i^g), T_{i+1}]$. The following lemma summarizes our observations.

Lemma 8

1. *The parent of each vertex in c_i becomes ready for Q_i during Δ_i,*
2. *Δ_i is the union of receive delays of vertices f_i, c_i, and g_i (if it exists), and*
3. *$\bigcup_{i=1}^{h-1} \Delta_i = [1, T]$.*

Proof. The parent of each vertex v in c_i must become ready before v; furthermore, it cannot become ready before f_i. For (2), if there is a gap between the receive delay of f_i and c_i, then this gap will be covered by the receive delay of g_i from the discussion above. Finally s_i must be ready at or after T_i^f from the definition of f_i so it cannot be received before T_i^f; thus the interval Δ_i contains $[T_i, T_{i+1}]$, and (3) follows. ■

From (3) of Lemma 8 the union of all Δ_i cover the interval $[1, T]$. We can therefore define a partition Π of $[1, T]$ as follows,

$$\begin{aligned} \Pi_{h-1} &= \Delta_{h-1} \\ \Pi_i &= \Delta_i - \bigcup_{j>i} \Pi_j, \quad 1 \le i < h-1. \end{aligned}$$

Definition. As before, a critical vertex v is in the *signature set R* if v is not a special vertex ($v \notin S$) and v becomes ready for Q_i during Π_i.

There are three kinds of receive delays in Π_i: the earliest ready parent f_i, the non-critical vertex g_i, and those vertices in c_i that become ready during $\Gamma_i \cap \Pi_i$. We use F, G, and C to denote the sets of these three kinds of vertices from all Π_i. From (1) of Lemma 8 and the definition of R, the parent of every vertex in C is either in R or S. As a result the number of receive delays in $F \cup G \cup C$ is at most $2h + 2(|S| + |R|) = 4h + 2|R|$.

The receive delays identified thus far cover the interval $[1, T]$ and there are no more than $4h + 2|R|$ in number. They are not necessarily non-overlapping, however. Using a straightforward greedy procedure it is possible to produce a subset of no more than $2h + |R|$ intervals which are disjoint and whose union includes at least $T/2$ time steps. With this observation, we have the following theorem.

Theorem 5 *Let $\mathcal{T}$ be the critical branch-and-bound subtree of size n and depth h. Let T be the total time for the random destination algorithm to expand $\mathcal{T}$ in a p-node network under FIFO scheduling strategy. The probability that T exceeds $k(\frac{n}{p} + h)$ is polynomially small, for suitably chosen k, when $n > p^2 \log p$.*

Proof.

The probability that the signature set R exceeds αT in size is polynomially small by Lemma 2. From the above discussion we can identify a set of $2h + \alpha T$ non-overlapping receive delays whose aggregate delay is at least $T/2$. From the result of Lemma 7 this probability is bounded by $(\frac{1}{2})^{\frac{cT}{p}}$ for a suitable constant c. This quantity is polynomially small in n for $n > p^2 \log p$ and a suitably chosen k. ■

8 Conclusions

In this paper we have developed a simple model which captures some aspects of message passing systems. The model can be extended in several ways to include, for example, non-uniformity of routing times and more system buffering capacity.

We believe that the model is simple enough to carry out further algorithmic analysis which we expect will shed light on the limitations of bounded resources in parallel systems.

9 Acknowledgments

Thanks to Marina Chen, Charles Leiserson, Abhiram Ranade, Chuck Seitz and Peter Winkler for helpful discussions. We are grateful to Lennart Johnsson and Thinking Machines Corporation for their support. The work of Sandeep Bhatt at Rutgers University is supported by ONR Grant N00014-93-1-0944; Sandeep Bhatt and Pangfeng Liu were also supported in part by NSF/DARPA grant CCR-89-08285, DARPA contract DABT 63-91-C-0031 monitored by Army DOC and Air Force grant AFOSR-89-0382 at Yale University.

References

[1] M. Adler, P. Gibbons, Y. Matias, and V. Ramachandran. Modeling parallel bandwidth: Local versus global restrictions. *Algorithmica*, 24, 1999.

[2] C. Berge. *Graphs and Hypergraphs.* North-Holland, 1973.

[3] S. Bhatt, M. Chen, C. Lin, and P. Liu. Abstractions for parallel N-body simulation. In *Scalable High Performance Computing Conference SHPCC-92*, 1992.

[4] G. Blelloch, P. Gibbons, Y. Matias, and M. Zagha. Accounting for memory bank contention and delay in high-bandwidth multiprocessors. *IEEE Transaction on Parallel and Distributed Systems*, 8(9), 1997.

[5] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, and T. Eicken. Logp: Towards a realistic model of parallel computation. In *4th ACM PPOPP*, 1993.

[6] C. Dwork, M. Herlihy, and O. Waarts. Contention in sharded memory algorithms. *Journal of ACM*, 44(6), 1997.

[7] H. Gabow. Using Euler partitions to edge color bipartite multigraphs. *International Journal of Computer and Information Sciences*, 5, 1976.

[8] M. Gereb-Graus and T. Tsantilas. Efficient optical communication in parallel computers. In *4th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1992.

[9] P. Gibbons, Y. Matias, and V. Ramachandran. Can a shared-memory model servve as a bridging model for parallel compuations? In *proceedings of the 9th ACM Symposium on Parallel Algorithms and Architectures*, 1997.

[10] P. Gibbons, Y. Matias, and V. Ramachandran. The queue-read queue-write asynchronous pram model. *Theoretical Computer Science*, 196, 1998.

[11] P. Gibbons, Y. Matias, and V. Ramachandran. The queue-read queue-write pram model: Accounting for contention in parallel algorithms. *SIAM Journal on Computing*, 26(4), 1998.

[12] R. Karp, M. Luby, and F. Meyer auf der Heide. Efficient PRAM simulation on a distributed memory machine. In *24th Annual ACM Symposium on Theory of Computing*, 1992.

[13] R.M. Karp and Y. Zhang. A randomized parallel branch-and-bound procedure. In *20th Annual ACM Symposium on Theory of Computing*, 1988.

[14] R.M. Karp and Y. Zhang. Randomized parallel algorithms for backtrack search and branch-and-bound computations. *Journal of the ACM*, 40, 1993.

[15] F. T. Leighton, M. J. Newman, A. Ranade, and E.J. Schwabe. Dynamic tree embeddings in butterfly and hypercubes. In *1st Annual ACM Symposium on Parallel Algorithms and Architecture*, 1989.

[16] T. Leighton. *Introduction to Parallel Algorithms and Architectures.* Morgan Kaufmann Publishers, 1992.

[17] C. Leiserson, Z. Abuhamdeh, D. Douglas, C. Feynman, M. Ganmukhi, J. Hill, W. D. Hillis, B. Kuszmaul, M. St. Pierre, D. Wells, M. Wong, S. Yang, and R. Zak. The network architecture of the connection machine CM-5. In *4th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1992.

[18] P. Liu, W. Aiello, and S. Bhatt. An atomic model for message passing. In *5th Annual ACM Symposium on Parallel Algorithms and Architecture*, 1993.

[19] P. Liu and S. Bhatt. Experiences with parallel n-body simulation. In *6th Annual ACM Symposium on Parallel Algorithms and Architecture*, 1994.

[20] K. Mehlhorn and U. Vishkin. Randomized and deterministic simulations of PRAMs by parallel machines with restricted granularity of parallel memories. *Acta Informatica*, 21, 1984.

[21] A. Ranade. A simpler analysis of the Karp-Zhang parallel branch-and-bound method. Technical Report UCB/CSD 90/586, University of California, 1990.

[22] A. Ranade. Optimal speedup for backtrack search on a butterfly network. In *3rd Annual ACM Symposium on Parallel Algorithms and Architecture*, 1991.

[23] C.L. Seitz. Multicomputers. In *Developments in Concurrency and Communication, C.A.R Hoare (ed) Addison–Wesley, pp 131-201*, 1990.

[24] L. Valiant. A bridging model for parallel computations. *Comm. ACM*, 33(8), 1990.