An atomic model for message-passing

Pangfeng Liu[1] William Aiello[2] Sandeep Bhatt[2]

[1] Dept. of Computer Science, Yale University, New Haven CT 06520.

[2] Bell Communications Research, Morristown NJ 07960.

Abstract

This paper presents a simple atomic model of message-passing network systems. 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 exisiting message-passing systems. Nonetheless, we prove linear speedup for backtrack and branch-and-bound searches using simple randomized algorithms.

1 Introduction

Many parallel computers support the message-passing programming model. Send and receive primitives hide low-level architectural details related to the network. Such abstractions are ideal for programming many large applications. We propose an atomic model 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 efficient solutions (linear speed-up) for backtrack and branch-and-bound searches.

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 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. It is natural to ask whether the efficiency gained by using non-blocking instructions is lost if the number of pending messages is severely limited.

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, enques 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 [3]. Unlike these models however, we do not allow multiple messages to be received in one step by a processor. A related model for optical communication was investigated in [2]. 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.

Inspite 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 algorithms can attain linear speed-up.

Main results. This paper studies three problems: 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 rough 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.

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 [3], 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 [3] 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 [5] 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.*

The remainder of this abstract is organized as follows. Section 2 defines the model. In Section 3 we introduce and analyze the problem of message scattering. Section 4 gives a brief outline of the technical approach. Sections 5 and 6 give the proofs of Theorems 1 and 2 respectively. Section 7 concludes with comments on further work.

2 The atomic message-passing model

We model a message-passing multicomputer as a collection of p nodes connected via an interconnection network [6].

For convenience of analysis we require that the system be synchronous, and operate in discrete time steps. This assumption simplifies the analysis of throughput; we do not exploit it in the design of correct algorithms.

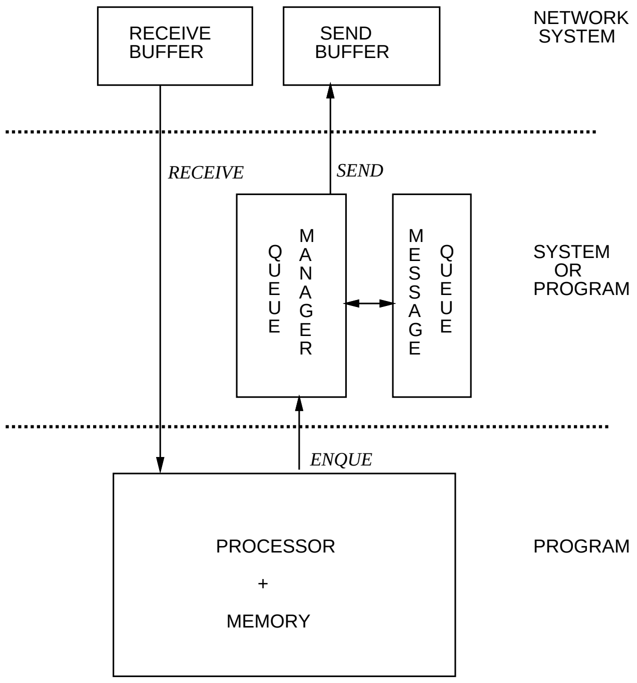

Figure 1: The structure of a node.

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 enque 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:

1. **The send phase** (performed by the processor or system):
    - MAINTAIN QUEUE: Put newly enqueued messages into an appropriate place in the message queue.
    - SEND: Inject the message at the head of the queue into the send buffer if empty.

2. **The transmission phase** (peformed by the network system):

   - TRANSMIT: 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.

3. **The computation phase** (performed by the processor):

   - RECEIVE: Probe the receive buffer to receive an incoming message, if any, into local memory.
   - COMPUTE: Perform local computation, possibly on the newly received message, and generate new messages.
   - ENQUE: Pass the newly generated messages to the queue manager.

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

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 [4]: "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 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.

For message scattering and backtrack search our analysis is valid under worst-case arbitration, for branch-and-bound we require FIFO arbitration to prove optimal speedup.

3 Message scattering

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 current paper is motivated by our ongoing project on the large scale astrophysical simulations of galaxies [1].

The off-line version of this problem in which the lists can be reordered is easily solved using standard edge-coloring techniques. If r is the maximum number of messages received by any node, then $\max\{r, m\}$ steps are necessary and sufficient.

However, the distributed version of the problem, without reordering, is not as simple. We will show in the full paper 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 in this section 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 $O(m)$. This means that the average throughput is $\Omega(p)$ messages received per time step, asymptotically the maximum possible. 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 k sufficiently large.*

Proof. We adapt Ranade's proof [5] of the result of Karp and Zhang [3].

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 m is selected for transmission, i.e., m enters the send buffer at time step τ, and is destined for node q. Then we say that 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, $T_i^{nr} \le T_{i-1}$. This means that 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:

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

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 $T > km$ is at most $p\binom{T+m}{m}\binom{(p-1)m}{T-m}p^{-(T-m)}$. For k sufficiently large, this quantity is smaller than $O(e^{-m})$. ■

4 Techniques for tree searches

4.1 Algorithmic issues

This section outlines the algorithmic and proof strategies for backtrack and branch-and-bound search in the atomic model. The branch-and-bound strategy is essentially that of Karp and Zhang [3]; 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 weaker atomic model. The proofs of both results build on the ideas 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 monotonically increases 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. 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 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 observe 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.

4.2 Proof techniques

In this section we describe some of the ideas 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[1] 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.

[1]Every vertex in backtrack search is critical.

In the analysis of the running time we proceed as follows. At time step $t = 1$ the root is assumed to be expanded in node 1, without loss of generality, and the children enqueued with random destinations. 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 step 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.

The signature set R is either large or small, where large and small are defined by a threshold αT, where α is some suitably chosen constant. We first show, just as in the proof of Lemma 1, that, with high probability, R is small when T is large.

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. Omitted in this abstract.

The second part of the proof argues that the event that T is large and R is small is polynomially small. The intuition is that the receive-delay of a vertex is expected to be a small constant, so that it is unlikely for a small number of signature vertices to suffer a large total receive-delay. Unfortunately, since the delays of the children of the signature vertices are not independent random variables, 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 signature set can essentially be treated as a martingale, thereby allowing us to use Chernoff bounds.

5 Analysis of backtrack search

This section presents the proof of linear speedup for backtrack search (Theorem 1) under the assumption that all nodes are sent to independently and uniformly chosen random destinations.

We begin with some terminology and definitions. As before, the set $S = \{s_1, \ldots, s_h\}$ denotes the special vertices along a maximally delayed root-to-leaf path. Let T_i be the time step at which vertex s_i is received at its destination node Q_i, with boundary conditions $T_1 = 1$ and $T_h = T$.

Definition. A node Q is *empty at time step t* if neither the send buffer nor the message queue of Q contains a vertex *at the end of time step t.*

Observe that if Q is empty at t then Q could not have received an internal vertex at time step t; however, it is possible that Q received a leaf at time step t.

Definition. An interval $I = [a, b]$, denotes the sequence of time steps between a and b, inclusive. Let I^+ denote the interval $[a+1, b+1]$.

Let T_i^e be the maximum time step $t \le T_i$ such that Q_i is empty at t. Let N_i denote the interval $[1 + T_i^e, T_{i+1} - 1]$. By definition, Q_i is non-empty throughout N_i.

Also, for $i < h$, let T_i^{nr} denote the maximum time step $t \le T_i^e$ at which Q_i does not receive a vertex. Let A_i denote the interval $[1 + T_i^{nr}, 1 + T_i^e]$. By definition, Q_i receives a vertex at every time step of A_i; we call A_i an *arrival window* for Q_i.

Next, we define a partition $\Pi = \{\Pi_1, \ldots, \Pi_{h-1}\}$ of the interval $[1, T-1]$ as follows. Let $\Delta_i = A_i \cup N_i = [1 + T_i^{nr}, T_{i+1} - 1]$, $1 \le i < h$, and

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

Let R_i be the set of non-special internal vertices which become ready for Q_i during the interval Π_i, and let $R = \cup_{i<h} R_i$. The set R is the signature set we will consider. First, note that by Lemma 2 the probability that $T > k(\frac{n}{p} + h)$ and $|R| \ge \alpha T$ is overwhelmingly small. In what follows we bound the probability that $T > k(\frac{n}{p} + h)$ and $|R| < \alpha T$.

Lemma 3 *For every $1 \le i < h$: (i) Q_i receives an internal vertex at time $1 + T_i^e$. (ii) A_i is an arrival window for Q_i, and (iii) at every time step in the interval N_i^+, the output buffer of Q_i attempts to inject a vertex into the network.*

Proof. Follows from definitions, omitted.

From part (iii) of the Lemma, it follows that the send buffer of Q_i contains a tree vertex during each time step of the interval $N_i^+ \cap \Pi_i^+$, $1 \le i < h$. Let $C_i = \{c_{ij} : 1 \le j \le m_i\}$ be the set of tree vertices that appear in the send buffer of Q_i during $N_i^+ \cap \Pi_i^+$. We claim that the parent of every vertex in C_i must be in the signature set (in fact in R_i). Otherwise, Q_i would have received a message at time step T_i^{nr} which, by definition, is not possible.

Denote by W_{ij} ($1 \le j \le m_i$) the portion of the time interval $N_i^+ \cap \Pi_i^+$ during which vertex c_{ij} occupies the send buffer of Q_i. Since c_{ij} is blocked throughout all (except possibly the last time step) of W_{ij}, it follows that W_{ij} is an arrival window for the destination node Q_{ij} of vertex c_{ij}.

We are now ready to bound the desired probablity. First, note that $|R| < \alpha T$ implies that $|C| \le 2(|R| + |S|) < 2(h + \alpha T)$. We next refine the partition Π into arrival windows as follows: for each interval Π_i the initial segment $W_{i0} = \Pi_i \cap A_i$ is an arrival window for Q_i; beyond that W_{ij} is an arrival window for Q_{ij}, $1 \le i < h$, $1 \le j \le m_i$. All the arrival windows taken together form a partition of the interval $[1, T]$.

We need one more definition. For the arrival window (W_{ij}, Q_{ij}), where $W_{ij} = [t_1, t_2]$, let $t \le t_1$ be the maximum time step at which Q_{ij} does not receive a message. Define $W_{ij}^* = [t + 1, t_2]$ so that (W_{ij}^*, Q_{ij}) is the *maximal backward extension* of the arrival window. Note that every message received by Q_{ij} during the interval W_{ij}^* must have become ready for Q_{ij} during the same interval. Let Q^* be the sequence of all the queues Q_{ij}, $1 \le i < h$, $1 \le j \le k_i$.

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

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

Let X_{ij} denote the set of vertices v such that $v \notin C \cup R \cup S$, and v is received by Q_{ij} during Π^*_{ij}. From the discussion above any vertex in X_{ij} must become ready for Q_{ij} during Π^*_{ij}. Finally, let $X = \cup X_{ij}$. Since the arrival windows cover the interval $[1, T]$, it follows that $|X| \ge T - |V|$.

To sum up the discussion above, an execution template $\mathcal{E}$ is an octuple $(S, R, C, X, Q, Q^*, \Pi, \Pi^*)$ whose elements are defined as before. Each execution template fixes the destinations of vertices in S, R, C, and X.

We bound the probability of the event $T > k(\frac{n}{p} + h)$ and $|R| < \alpha T)$ by summing the probabilities of all possible execution templates. The probability of a fixed execution template is

$$\begin{aligned} & p^{-|C \cup R \cup S|} p^{-(T-|C \cup R \cup S|)} \\ = \; & p^{-h} p^{-|R|} p^{-|S-(R \cup C)|} p^{-(T-|C \cup R \cup S|)}. \end{aligned}$$

Next, we count the number of different execution templates. The destinations of vertices from C and R are specified by Q, so the number of unspecified queues in Q^* is $|S - (R \cup C)|$ and the number of ways to choose Q and Q^* is $p^h p^{|S-(R \cup C)|}$. The total number of execution templates is therefore no more than

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

Bounding the product of the above two terms yields the following lemma.

Lemma 4 *For suitable constants k, α the probability that $T > k(\frac{n}{p} + h)$ and $|R| < \alpha T$ is polynomially small in n.*

Finally, Theorem 1 follows from Lemmas 2, 4.

6 Branch-and-bound search

This section presents the proof of linear speedup for branch-and-bound search (Theorem 2) under the assumption that the network arbiter observes the FIFO policy. Under this policy, incoming vertices are received in FIFO order, i.e. the vertex that is ready first is received first, while vertices that are ready at the same time are received in arbitrary order. Termination detection can be handled in a manner similar to [5].

Our proof proceeds as follows: with every possible execution we associate a signature set R. As before, R is defined so that by Lemma 2 it is unlikely for both T and R to be large. For the second case, we argue that the delay is very likely to be bounded by $O(|R|)$ so that it is unlikely for T to be large and R to be small.

6.1 The signature set

As before, let special vertex s_i be received by Q_i at time T_i. From the definition s_{i+1} is generated in Q_i at T_i and received at Q_{i+1} at T_{i+1}, for all $1 \le i < h$. Our goal is to find a set of vertices whose receive-delay periods (time spent in send buffers) are disjoint and cover the interval $[T_i, T_{i+1}]$.

From every T_{i+1} $(1 \le i < h)$ we trace back in time until T^{ns}_i, which is the largest time step smaller than T_{i+1} at which the send buffer of Q_i does not contain a critical vertex. By definition, the send buffer of Q_i is occupied by a critical vertex every time step during the interval $\Gamma_i = [T^{ns}_i + 1, T_{i+1}]$ $(1 \le i < h)$. Let C_i be the set of critical vertices that are injected into the network from Q_i during Γ_i. The interval Γ_i can be partitioned into intervals corresponding to the *receive delays* of the critical nodes C_i.

Among the parents of vertices in C_i, let f_i be one that is ready for Q_i earliest of all, and let T^f_i be the time step when f_i becomes ready. Since s_{i+1} is a member of C_i, its parent,s_i, cannot be ready for Q_i before T^f_i.

If there is a gap between the receive delay interval of f_i and those of nodes in C_i then it must be the case that, during that interval, the send buffer of Q_i contains a non-critical vertex until time step T^{ns}_i. Let g_i be this vertex, and let T^g_i be the time step at which it became ready. Let $\Delta_i = [\min(T^f_i, T^g_i), T_{i+1} - 1]$.

Lemma 5 *For every vertex in C_i, its parent must be ready for Q_i during Δ_i. Furthermore,* $\bigcup_{i=1}^{h-1} \Delta_i = [1, T-1]$.

Proof. From the preceding discussion, observe that Δ_i can be covered by the union of receive delays of f_i, C_i, and possibly g_i. Also, $T_i^f \leq T_i$ so that Δ_i contains the interval $[T_i, T_{i+1}]$. It follows that the Δ_i's, taken together, cover the interval $[1, T-1]$. ∎

With the Δ_i's as defined above we next construct a partition Π as follows:

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

Finally, we define the signature set to be the set of all non-special tree vertices that become ready for Q_i during Π_i.

There are three kinds of receive delays that cover Π_i: the earliest ready parent f_i, the non-critical node g_i, and those vertices in C_i that are ready during $\Gamma_i \cap \Pi_i$. We denote by F, G, C the set of chosen vertices whose receive delays cover the interval $[1, T-1]$. Since the parent of every vertex in C is either in S or in R, it follows that the total number of receive delays selected to cover the interval $[1, T-1]$ is at most $2h + 2(|S| + |R|) = 4h + 2|R|$.

As before, we use Lemma 2 to bound the probability that T and R are both large. In what follows, we bound the probablity that T is large when R is small.

6.2 Bounding large delays

Our intuition is that the total time covered by a small set of receive delays is likely to be small as well. In order to show that the sum of m non-overlapping delays is, with high probability, $O(m)$, we start with the following lemma.

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

Lemma 7 *Let $V = \{v_1, \ldots, v_m\}$ be m tree vertices with independent and uniformly chosen destinations whose receive delay periods are non-overlapping. The probability that the sum of the corresponding m receive delays is greater than βm is smaller than $(\frac{1}{2})^{\frac{(\beta-1)m}{p-1}}$ when $\beta \geq 2e + 1$.*

Proof. Let X_i be the number of vertices waiting to be received in front of v_i when v_i is ready. The total delay of V is $m + \sum_{i=1}^m X_i$.

We argue that given $X_1, \ldots, X_{i-1}$, the expected value of X_i is no more than 1. When v_i is about to make its random choice, it is given a distribution of the numbers of vertices waiting for each processor. This distribution is given *a priori* the random choice of v_i, so the distribution v_i can see is independent from the random choice v_i is about to make. The expected value of X_i is no more than one since there are at most $p-1$ other tree nodes waiting to be sent and v_i will pick a destination uniformly at random.

The event that the total delay is more than βm implies the sum of all X_i is more than $(\beta - 1)m$. From Lemma 6 this will happen with probability smaller than $(\frac{1}{2})^{(\beta-1)m}$ when $(\beta - 1) > 2e$. ∎

We will apply Lemma 7 to bound the total receive delays of the sets F, G, C of vertices. First, with every possible execution of the algorithm we associate a canonical choice of the sets F, G, C. If the total receive delay is T then the receive delay of vertices in one of the three sets must be $T/3$ or greater. Now, fixing any one of these sets gives no information about the destinations and so we can apply Lemma 7 to bound the total receive delay of the individual sets. Putting everything together, we obtain Theorem 2.

7 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.

8 Acknowledgments

We are grateful to Lennart Johnsson for encouragment and support, and to Marina Chen, Charles Leiserson, Abhiram Ranade and Peter Winkler for helpful discussions. The work of Sandeep Bhatt and Pangfeng Liu was 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.

References

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

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

[3] Richard M. Karp and Yanjun Zhang. A randomized parallel branch-and-bound procedure. In *29th Annual ACM Symposium on Theory of Computing*, 1988.

[4] 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.

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

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