Performance Considerations in Software Multicasts

J. Cordsen, H. W. Pohl, W. Schröder-Preikschat[†]

GMD FIRST
Rudower Chaussee 5
D-12489 Berlin, Germany
{jc,hans}@first.gmd.de

[†]University of Magdeburg
Universitätsplatz 2
D-39106 Magdeburg, Germany
wosch@cs.uni-magdeburg.de

Abstract

Parallel computation environments exploiting conventional processors offer the potential to achieve high efficiency at a low cost. In the future, even heterogeneous clusters of symmetric multiprocessors (SMPs) will supersede special purpose computers. Due to this trend, the availability of special hardware support for global communication will be more unusual. For such an environment, software-implemented multicasts and broadcasts are highly demanded to support a global dissemination of information over networks of processors.

This article introduces the theory and presents an algorithm for the implementation of an one–source/many–destination distribution of a message (multicast communication) based on a send-and-forget semantic, i.e. the event of sending a message performs asynchronously with respect to the blocking receive event. The performance of a multicast communication is sensitive to the underlying communication system. In order to achieve optimal results, the algorithm must consider the latencies at the sending and receiving sites. It is shown that computing systems with a low probability for contentions in the communication network offer optimal performance results when they consider generalized Fibonacci sequences. Experiments on a parallel computing system and comparisons with related work demonstrate the relevance of the proposed work.

1 Introduction

The efficiency of communication is important to the overall system performance. Especially, this is true in large-scale computing systems consisting of distributed memory computing resources. Many message-passing communication libraries (e.g. PVM or MPI) are available and allow for a portable programming of parallel applications.

Message-passing communication services can be grouped into two classes: point-to-point and collective operations. Point-to-point communications involves two communication partners in the form of various modes of send and receive operations, e.g. blocking or non-blocking semantics. A collective communication involves a group of processes and implements frequently used communication patterns in parallel programs, e.g. broadcast or reduction communications.

In the area of high-performance computing, the success of a message-passing system is a measure of the performance capabilities of the communication subsystem. It is vital to provide both a low latency and a high data throughput rate for point-to-point communication (unicast) and, even more important, for collective communication (multicast). In order to accomplish these goals, it is important to minimize the interactions with the memory subsystem, i.e. avoiding intermediate buffering in message delivery, and to optimally match an application's communication pattern to the capabilities of the underlying communication subsystem.

A barrier synchronization includes both a many-to-one and a one-to-many communication. The former is necessary to implement a reduction phase, whereas the latter is used in the distribution phase to inform the processes to proceed execution. Due to load imbalance that may affect the processes participating in a barrier synchronization, it is rather impossible to propose a theory providing optimal performance results for the implementation of the reduction phase. Therefore, this paper only discusses performance aspects of software-implemented multicasts used in the distribution phase of a barrier synchronization.

The runtime behavior of a multicast communication can be optimized by organizing the communicating processes into a tree-structured (i.e., hierarchical) communication topology (multicast tree). In a multicast tree, each process uses the point-to-point communication service to forward a message to each of its children in turn. This lets communication proceed in parallel if non-blocking primitives are used to send messages to the childrens.

The rest of this article is organized as follows. Section 2 presents an overview of multicast trees proposed in the literature and being used in various barrier implementations. Thereby, the disadvantages of their designs are discussed. Afterwards, section 3 analyzes the impact of sending and receiving latencies and provides a theory on the performance results of a multicast communication. In section 4, the design aspects and an algorithm for the construction of an optimal multicast tree is presented. Section 5 shows runtime results of multicast communications using different multicast trees on a parallel computing system. Finally, conclusions are drawn and some directions for further investigations are discussed.

ICS 97 Vienna Austria
Copyright 1997 ACM 0-89791-902-5/97/7..$3.50

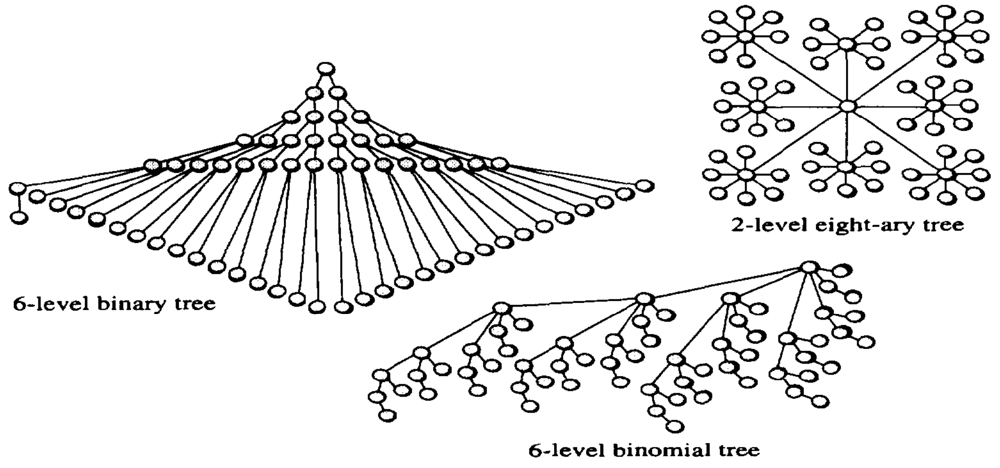


Figure 1: Examples of different multicast trees for 64 processes

2 Multicasts in Barrier Synchronizations

Barriers were introduced in 1978 by H. Jordan as the fundamental synchronization mechanism for the Finite Element Machine at NASA Langley [13]. A barrier synchronization is a point in the control flow of an algorithm at which all processes must arrive before any of the processes are allowed to proceed. The time in which a barrier service function receives arrival messages is the reduction phase of a barrier synchronization. When all arrival messages are received, the barrier service will initiate a distribution phase to let the processes continue their work. A programmer may use a barrier synchronization to ensure that all processes executing a particular parallel loop have finished the loop before continuing other program execution. The dissemination of the message to continue program execution requires a multicast communication, thus making a barrier synchronization dependent on the performance of the communication system.

In medium-range massively parallel processing systems, hardware implementations can provide excellent performance results. The special communication hardware used to implement the synchronization requires significant design and engineering effort but can be obtained at reasonable costs. The scalable parallel processing system T3D from Cray Research, Inc. [6] uses a synchronization network to realize a hardware-assisted barrier synchronization. Measurements on a 64-node T3D system [2] indicate that barrier synchronization is never slower then four microseconds.

However, today's trend of building massively parallel supercomputers using clusters of conventional workstations make the availability of hardware assisted global communication more unusual. The problem of a software-implemented barrier synchronization lies not in the communication of data (i.e. it depends not on the bandwidth results). Rather, the process of sending, routing, and receiving data (i.e. the communication latency) is vital to the performance results. Compared to other system software activities, sending and receiving of data must require execution of only a very few instructions. Moreover, routing of messages through the communication network must be as efficient as possible.

2.1 Software Implementations — Some Examples

Parallel programming environments provide support for barrier synchronization. For example, the message-passing communication libraries PVM [16] and MPI [11] comprise functions implementing barrier synchronization. Whenever possible, these communication libraries will use the fast hardware provided by the vendor. If such dedicated support is not available, there is a range of different approaches implementing the multicast communication.

The simplest implementation is a centralized approach. In this approach, only the barrier server sends distribution messages. Individual processes have no duties other than to receive a notification and to continue work. For small numbers of processes, the centralized approach may perform quite well, but in systems with larger numbers of processes, the barrier server will act as a bottleneck and, thus, cause unnecessary delays for the worker processes.

A more promising approach minimizes communication latency through the use of a multicast tree which is constructed atop the unicast communication service. In the distribution phase, a barrier server sends notification messages to a subset of the worker processes. When these processes receive their notifications, they start to distribute the message to a different subset of worker processes.

An optimal distribution phase then requires that every worker process which receives a notification helps to distribute it to other processes until all processes have received the notification. The duration of a distribution phase is the *distribution latency*. This phase begins when the barrier server initiates the first notification and ends when the last worker process has received the message.

Of course, in order to obtain optimal runtime results, it is necessary to account for architectural properties of the system. The topology of the communication network is most significant to the runtime results and, thus, should be considered in the construction scheme of the multicast tree. Various types of multicast trees have been proposed. Figure 1 presents three different types that are used in several implementations.

Frequently, a binary tree is used to implement a parallel distribution of messages. In case of MPI implementations, at least Convex [9] and IBM SP2 [10] use a binary tree to implement the barrier synchronization. The binary multicast tree in figure 1 shows an example constructed for the 64 processor MIT Alewife machine [14]. Using a binary multicast tree for this machine requires a 6-level binary tree, that is, the longest communication distance from the root to a leaf involves six point-to-point communications. The single processor at level 6 indicates that the 6-level 64-processor binary multicast tree is incomplete. In contrast, a 5-level 63-processor binary multicast tree would be complete.

A special feature of the Alewife machine is that interprocessor communication can be effected either through shared-memory or message-passing. While the barrier implementation for shared-memory communication used a binary multicast tree, the alternative message-passing implementation was based on the 2-level eight-ary tree [14]. Both multicast tree structures are implemented in software only. Their execution results significantly gain through the hardware assisted dedicated communication support. A barrier synchronization with 64 processes executes in 50 μs through shared-memory communication and in 20 μs by way of a message-passing communication.

In absence of any special hardware communication support, the execution times grow by more than a factor of ten [8]. For example, in case of a four-ary multicast tree used in a KSR-machine with 10 processes, a barrier synchronization executes in 223 μs, that is $(10, 223)$. For larger configurations, the performance results are $(16, 338)$ and $(32, 635)$ [8]. The difference to the Alewife results is significant, but, on the other hand, the numbers for a centralized KSR implementation are even slower and reported to be $(10, 254)$, $(16, 451)$ and $(32, 920)$.

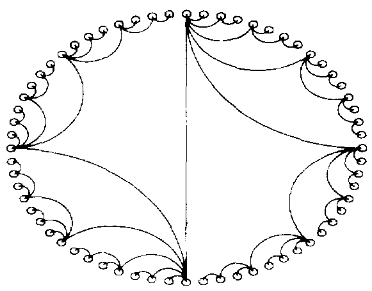


Figure 2: Recursive Doubling in a Binomial Multicast Tree

Binomial trees or spanning trees are related to the theoretical work on priority queues [3]. Since then, various implementations and analysis work, as in [8, 12], have demonstrated that binomial trees gain from an optimal scheduling of notified processes. During the first communication step, the barrier server is able to distribute the message to only one destination. In each subsequent communication step, each processor holding a copy can send it to exactly one new processor. At most, the number of processors is doubled in each communication step. Therefore, this technique is known as the *recursive doubling* procedure. Beside of the complete 6-level 64-processor binomial multicast tree in figure 1, figure 2 shows another way to view the recursive doubling in a binomial multicast tree.

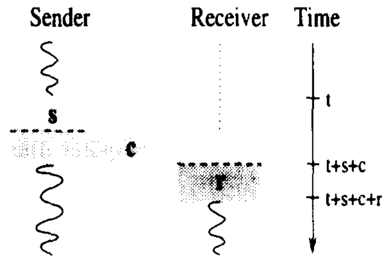
2.2 Analysis

Performance measurements have demonstrated that the scheduling times in multicast trees significantly deviate from theoretical considerations [18]. Recently, work has been carried out to model communication loads [7] and to construct optimal multicast trees accounting the communication latencies of a target system [1, 15]. In [15], dynamic programming is used to compute the optimal multicast tree for given communication latencies and the number of processes. In [1], the postal model introduces generalized Fibonacci sequences for the construction of optimal multicast trees. Their communication model bears similarities to the model of our work but has the drawback that it only allows to compute boundaries on the performance results of multicast communication being based on an optimal multicast tree.

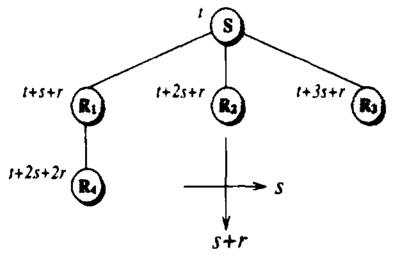
Section 2.2.1 introduces our communication cost model based on a sending *and* a receiving communication latency. Afterwards, section 2.2.2 uses this model to analyze the multicast trees of figure 1 in respect to their suitability in computing systems with different communication latencies.

2.2.1 A Communication Model

The communication model presented here can be applied to communications between a process using a non-blocking function to send a message and a process using a blocking function to receive the message. It is assumed that the receiver has called the receive function in advance.



Communication model



Activation times

Figure 3: Communication Model and Activation Times

Figure 3 illustrates the activities performed within a communication. At time t, a sender sends a message to another process (the receiver) which is located on a different processor. The latter is blocked and waits on the receipt of that message. In the time interval $[t, t+s+c]$, the processor hosting the sender is busy transfering the message to the communication partner. At the receiving site, the processor is busy in the time interval $[t+s, t+s+c+r]$.

Time s is significant to the sender only and depends on the local message startup time. On the other hand, time r is relevant only to the receiving site and depends on the activation time of a receiving process when the message has arrived. Finally, time c is the time in which both processes are busy transfering data from the sender to the receiver. For communication with large amount of data, e.g. several MBs, the time c will dominate the costs of a communication and time s as well as time r can be ignored. This is not the case in a distribution phase of a barrier. Rather, the opposite is true. Sending a notification requires only a very small message and minimizes c to an extent that it can be ignored in comparison to the communication latencies of s and r.

The simplified communication model (without parameter c) is depicted in figure 3. It shows the effects of parallel communication during a multicast communication. Process S sends three messages at time $t, t+s$, and $t+2s$. Processes R_1, R_2 and R_3 will be activated at time $t+s+r, t+2s+r, t+3s+r$ respectively. Process R_1 is activated at time $t+s+r$ and immediately sends a copy of the message to another process R_4 which receives the message at time $t+2s+2r$. The example shows that, when moving from left to right, an increase in time of unit s is accounted, whereas moving towards the leaves of the tree adds an additional time $s+r$.

2.2.2 Suitability

Based on the communication model introduced in the previous section, it is possible to compute the distribution latency of a barrier distribution phase as a function of the send and receive latencies (s and r). In a binary tree, the right subtree is notified through the second message and the depth of the tree is $\log_2(p)$ with p being the number of processes. Adding the receive latencies, a complete binary tree with $p = 2^t - 1$ ($t \geq 1$) processes has a distribution latency of $(2*s+r)*(\log_2(p+1)-1)$. In general, a complete l-level n-ary tree comprises $p = \frac{n^{l+1}-1}{n-1}$ processes and has a distribution latency of

$$(n*s+r)*(\log_n((n-1)*p+1)-1) \quad (1)$$

communication steps. A complete 2-level eight-ary tree comprises 73 processes and has a distribution latency of $(8*s+r)*(\log_8((8-1)*73+1)-1)$. Finally, due to the recursive doubling in each communication step, a binomial tree has a distribution latency of $(s+r)*\log_2(p)$ communication steps.

Figure 4 shows the distribution latencies of the different tree types. The execution time is computed as a function of the number of processes and the ratio of the receive and send latencies ($\frac{r}{s}$). Unit s is fixed to 20 μs and the four triple sets of measurements are made for ratios $\frac{r}{s} = \frac{1}{5}, \frac{3}{2}, \frac{3}{1}$ and $\frac{5}{1}$. The leftmost set of measurements are made with values of $s = 20\mu s$ and $r = 4\mu s$, the rightmost with values of $s = 20\mu s$ and $r = 100\mu s$. In the first case, the binomial tree performs best, while the eight-ary tree provides the worst results. Due to the dominance of the send latency costs in the distribution latency of the eight-ary tree, this result is what one would expect.

For increasing values of r, the performance results of an eight-ary tree increasingly get better in comparison to the results of binary and binomial trees. This observation is of great importance because computing systems normally have a receive latency which is higher than the send latency. In other words, due to the concrete values of r and s, the construction scheme of a multicast tree must be adapted. For example, if $r > s$, the tree structures must be broader and less deep.

3 Impact of Communication Latencies

The previous section demonstrated that an optimal multicast tree used to implement the distribution phase in a barrier synchronization depends on the send and receive latencies. In this section, we consider the impact of communication latencies on the progress of distributing a message in a multicast tree. Section 3.1 introduces a function $f(t)$ which expresses the number of processes in a Fibo-tree that have received the message at time t. Afterwards, section 3.2 shows some examples of $f(t)$ for different values of s and r.

3.1 Speed of Message Distribution

At a given time $t, t \geq 0$, the number of processes holding a copy of the message is at least one, because the root of the multicast tree (i.e., the process implementing the barrier service) holds a copy. Therefore, the initial definition can be written as:

$$f(t) = 0 \; for \; t < 0 \; and \; f(0) = f(1) = \ldots = f(s+r-1) = 1.$$

A duration of $s+r$, i.e. a complete communication, is required to get a copy of the message to a second process. In general, for values of $t > 0$, $f(t)$ is defined as:

$$f(t) = 1 + f(t-s-r) + f(t-2s-r) + f(t-3s-r) + \ldots$$

or

$$f(t) = 1 + \sum_{i=1}^{\infty} f(t - i*s - r).$$

The formulation of a recursive definition for $f(t)$ requires to determine two terms. First, it is relevant how many processes were active at a past time and, second, how many processes were activated since then. Function $f(t)$ can be expressed as $f(t) = f(t-s) + f(t) - f(t-s)$. The determination of $f(t-s)$ for $t-s > 0$ is:

$$f(t-s) = 1 + \sum_{i=1}^{\infty} f(t-s-i*s-r) = 1 + \sum_{i=2}^{\infty} f(t-i*s-r).$$

This equation can be used to compute the term $f(t)-f(t-s)$ with the result:

$$\begin{aligned} f(t) - f(t-s) &= 1 + \sum_{i=1}^{\infty} f(t-i*s-r) - \\ &\quad (1 + \sum_{i=2}^{\infty} f(t-i*s-r)) \\ &= f(t-s-r) \end{aligned}$$

In total, this defines $f(t)$ as a simple recursive definition:

$$f(t) = \begin{cases} 0 & : \; for \; t < 0 \\ 1 & : \; for \; 0 \leq t < s+r \\ f(t-s) + f(t-s-r) & : \; for \; t \geq s+r \end{cases} \quad (2)$$

The result is a generalized Fibonacci sequence, similar to the definition in [1]. The classical and simplest Fibonacci sequence is defined as $f(t) = f(t-1) + f(t-2)$ and is used to predict population growth.

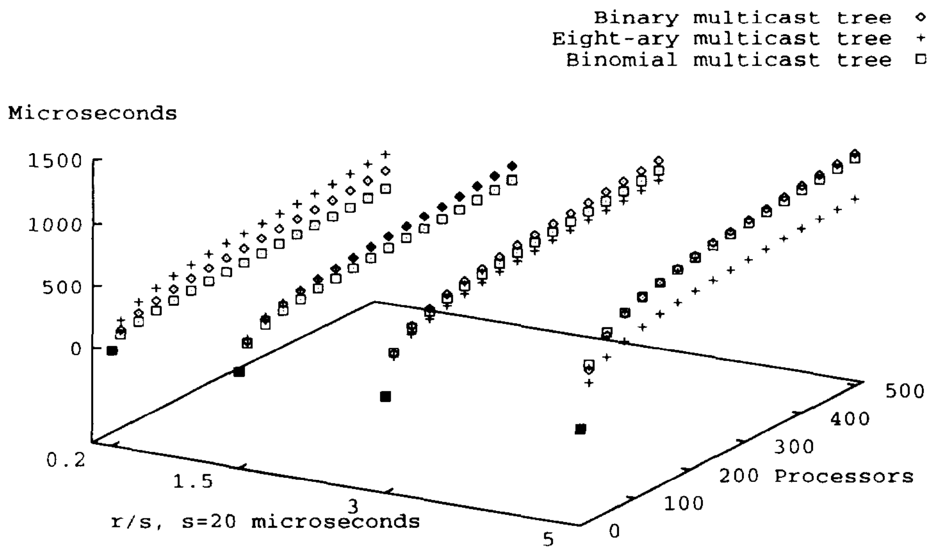


Figure 4: Execution results for different ratios of s and r

3.2 A few Examples

The original Fibonacci sequence of $f(t) = f(t-1) + f(t-2)$ requires $s = 1$ and $r = 1$. Then, the sequence of numbers is $1, 1, 2, 3, 5, 8, 13, 21, \ldots$. It is unlikely that a computing system offers both a send and receive latency in the order of a microsecond. On the other hand, choosing a time unit of s, all computing systems with $\frac{r}{s} \approx 1$ approximate the sequence of numbers presented above. However, for reasons of clarity, the following examples always are normalized to $s = 1$.

The basic idea of a binomial tree is to double the number of notified processes in each communication step. With $s = 1$, the goal is defined as $f(t+s) = 2 * f(t)$. To solve this, it is required that r is zero ($r = 0$) and leads to $f(t) = f(t-1) + f(t-1)$ with the solution $f(t) = 2^t$. Note, that the speed of message distribution in a binomial multicast trees is simply defined by a single generalized Fibonacci sequence.

However, it is rather impossible to develop a message-passing environment that exploits zero execution time to deliver and to schedule a message to the receiving process. The choice of a binomial tree as the communication structure in a multicast communication therefore cannot effect optimal execution results. Nevertheless, as shown in section 2.2.2, the binomial tree can produce execution results superior to the results of a binary tree. Moreover, when all leaf processes can start the computation simultaneously, the binomial tree structure provides optimal support for many-to-one communications (e.g. in the reduction phase of a barrier synchronization).

The final pair of values ($s = 1, r = 3$) approximate the latencies of the execution platform used for the performance measurements in section 5. This leads to the definition $f(t) = f(t-1) + f(t-4)$ and produces a sequence with the numbers $1, 1, 1, 1, 2, 3, 4, 5, 7, 10, 14, 19, 25, 36, 50, 69, \ldots$.

4 Fibo-trees

The previous section introduced generalized Fibonacci sequences describing the speed of message distribution in a multicast tree. Section 4.1 presents a construction scheme for a multicast tree, namely the Fibo-tree, which allows to execute a multicast communication at the speed described by the generalized Fibonacci sequences for the given communication latencies. Section 4.2 presents an outline of a proof of the optimal performance results achieved by Fibo-trees. Finally, section 4.3 presents an algorithm that uses generalized Fibonacci sequences to construct a Fibo-tree.

4.1 Construction Scheme of Fibo-trees

A Fibo-tree $F(t)$ for values of $t < 0$ is empty, for $t \geq 0$ it consists of a root node with successors $F(t-s-r), F(t-2s-r), \ldots$. The correspondence to the recursive definition scheme of $f(t) = 1 + f(t-s-r) + f(t-2s-r) + \ldots$ is obvious. Figure 5 illustrates the recursive construction scheme for the Fibo-tree $F(5)$ for $s = 1$ and $r = 1$.

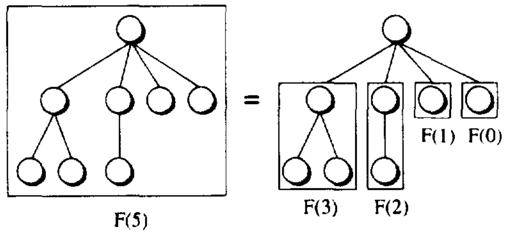


Figure 5: Construction Scheme of a Fibo-tree

4.2 The Speed of Message Distribution

The speed of the message distribution in a Fibo-tree can be determined. For sake of simplicity, the greatest common divisor of s and r is assumed to be 1. It can be shown that for the only real positive solution λ of the characteristic polynomial $x^{s+r} = x^r + 1$ it holds that $f(t)$ is proportional to λ^t or more precisely

$$\lim_{t\to\infty} \frac{f(t)}{\lambda^t} = a \qquad (3)$$

for a real positive value a.

An outline of the proof comprises three parts, but is left out due to space limitations. First, there are no multiple (complex) solutions of the characteristic polynomial. Therefore, $f(t)$ can be represented as a linear combination of the λ_i^t with every λ_i being a solution of the characteristic polynomial. Then, it remains to show that the only real positive solution λ has the greatest absolute value and, finally, that the coefficient of λ^t in the linear combination for f is greater than zero.

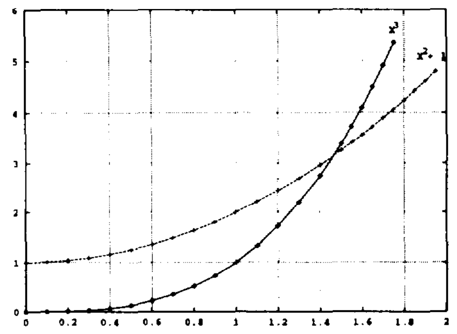


Figure 6: Determination of λ through an intersection of x^3 and $x^2 + 1$

Figure 6 shows an example with values of $s = 1$ and $r = 2$. The intersection of the lines x^3 and $x^2 + 1$ defines the value of λ. In this case, λ is about 1.47. If the receive latency is a factor of two of the send latency, due to equation 3, a distribution of a message requires approximately $\log_{1.47}(p)$ communication steps to notify p processes. This is a significant deviation of an ideal multicast tree with its theoretical result of $\log_2(p)$ time steps.

The impact of a growing receive latency is shown in figure 7. Again, the greatest common divisor of s and r is assumed to be 1. Starting with the theoretical optimum, i.e. no receive latency at all, the curve shows that a ratio of $\frac{r}{s} = 3$ already forces down the base of the logarithm function to less than 1.4. At a ratio of 8, the base is less than 1.2 and the performance wins in comparison to a sequential implementation will only be possible in large scale computing systems, i.e. for high numbers of p.

In comparison, the authors of the postal model [1] give exponential lower and upper bounds to characterize the speed of message distribution. While their upper bound is approximately the square of the lower bound, we are able to give the exact base of the exponential function approximating the speed of growth. In the case of the original Fibonacci sequence, their bounds are:

$$1.31607^t \leq f(t) \leq 1.73205^t$$

compared to the exact value of $\lambda = 1.61803$.

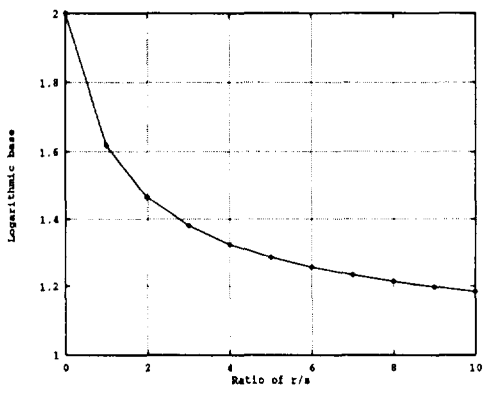


Figure 7: Base to the logarithm in the cost function for various r and $s = 1$

4.3 The Algorithm

The implementation of a Fibo-tree is straightforward (Figure 8). This section presents an almost complete implementation. The algorithm is implemented in the programming language *C* and creates a Fibo-tree for the arguments s, r and p, describing the send latency, receive latency and the number of processes respectively. Function `fibo` is equipped with two additional arguments `pool` and `id`. The former is an array of references to the processes which have to be sorted into the Fibo-tree, while the latter is a logical identification number of the running process.

```
01: void fibo (int s, int r, int p, PROCS pool[], ID id) {
02:   int i, j, k, steps, rem, max;
03:   int pop[1024]; int children[1024];
04:     /* initialization */
05:   for (i = 0; i < r+s; i++) pop[i] = 1;
06:   for (i = s+r; i < 1024; i++) {
07:     pop[i] = pop[i-s] + pop[i-s-r];
08:     if (pop[i] > 1024*1024) break;
09:   }
10:   for (steps = 0; pop[steps] < p; steps++);
11:   rem = pop[steps] - p;
12:     /* computation */
13:   i = 0;
14:   while (1) {
15:     k = steps - (i+1)*s - r;
16:     if (k < 0) break;
17:     if (k > 0) max = pop[k] - pop[k-1]; else max = 1;
18:     if (rem <= max) {
19:       children[i] = pop[k] - rem; rem = 0;
20:     } else {
21:       children[i] = pop[k] - max; rem -= max;
22:     }
23:     i++;
24:   }
25:     /* delegation */
26:   j = 0;
27:   while (i-- > 0) {
28:     pool[j].who = id;
29:     max = children[i] - 1;
30:     if (max > 0) {
31:       fibo (s, r, max+1, &pool[j+1], pool[j].id);
32:       j += max;
33:     }
34:     j++;
35:   }
36: }
```

Figure 8: Creating a Fibo-tree for given s, r and p

In figure 8, lines 2-3 declare a couple of variables. Array pop is used to compute the population growth of a Fibo-tree

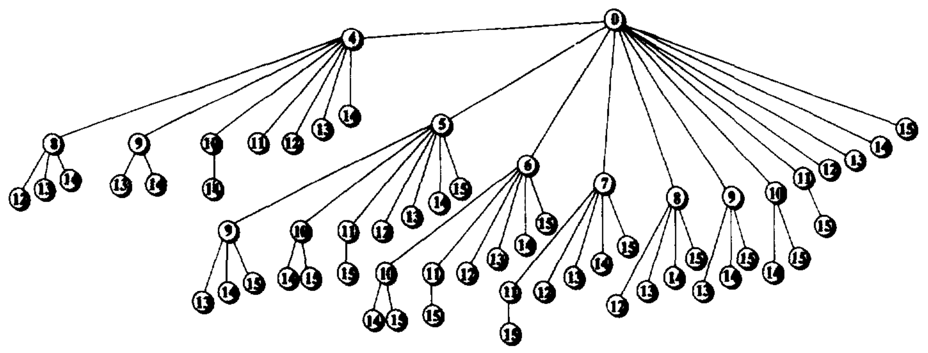


Figure 9: Fibo-tree for $s = 1$, $r = 3$, $p = 64$

for given values of s and r. The second array **children** is necessary to determine the sizes of the subtrees which the process executing function **fibo** will refer to. The computation of the population growth is implemented in lines 5-9. A termination check in line 8 prevents an overflow condition. Variable **steps** (line 10) determines the number of communication steps which a distribution phase needs to notify p processes. In most cases, the value of p will not match any of the values in array **pop**. Therefore, variable **rem** is initialized with the difference between the element being higher or equal to p and p itself.

The computation phase in lines 13-24 determines the values for the array **children**. An entry of **children** defines the size, i.e. the number of processes, of all the subtrees which are notified by this process. In this calculation, it must be considered that the number of processes in the subtrees must be reduced by **rem** processes. The strategy of removal is to remove only those processes that would receive the notification in the last communication step. In comparison to the removal of one or several subtrees, this maximizes the number of processes which are notified early in the distribution phase. This offers the opportunity to compensate some deviations that might occur in the execution times of the communications and, thus, makes the execution results of the Fibo-tree more stable.

In line 17, the number of processes in a subtree which are going to be activated in the final communication step is computed. If there are processes to remove, this number is subtracted from the maximum size of that subtree. Finally, the array **children** is used to create references to the subtrees. The element **who** in the array **pool** indicate the responsibility of notification, i.e. the process identification of the process that sends the message. Therefore, the process assigns its identification into the element for that subtree (line 28). If that subtree holds references to other processes, a recursive call to function **fibo** is performed.

Figure 9 shows the Fibo-tree which is constructed for the arguments $s = 1$, $r = 3$ and $p = 64$. In the circles, the number denotes the communication step in which a process receives a copy of the message. The final process is activated in step 15, whereas a complete 6-level binary tree requires $(2 * 1 + 3) * (\log_2((2 - 1) * 127 + 1) - 1) = 25$ steps. For 64 processes, a binomial tree implements the distribution in $(1+3) * \log_2(64) = 24$ steps and a complete 2-level eight-ary tree requires $(8 * 1 + 3) * (\log_8((8 - 1) * 73 + 1) - 1) = 22$ communication steps.

5 Runtime Measurements

The execution platform used for validation is the communication support system VOTE [5]. VOTE is part of the PEACE operating system family [17] and runs on the parallel distributed memory system MANNA [4] which is based on hierarchies of crossbar communication networks. The VOTE system implements the idea of a coexistence of communication paradigms, i.e. VOTE provides runtime support for both message-passing and shared-memory communication. A global address space is provided by means of a virtual shared memory (VSM). Communication is possible via calls to message-passing functions or, alternatively, performed automatically through the on-demand data movement of memory pages to processes that request data causing a memory access fault

Program execution on top of VOTE is not limited to a SPMD (single program/multiple data) programming style, but most shared-memory based parallel applications run in that mode and coordinate the execution of parallel loops through the use of a barrier synchronization.

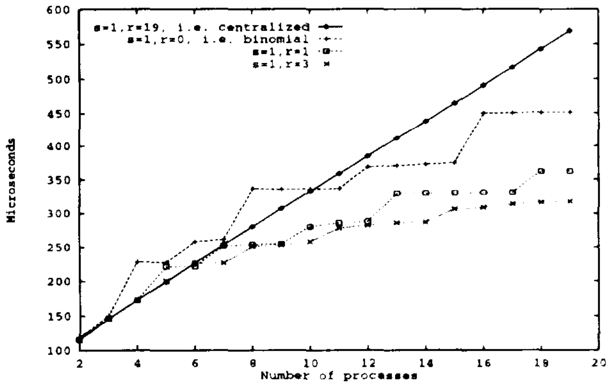


Figure 10: Distribution latency of different Fibo-trees in a $s = 1, r = 3$ system

To support parallel program execution with the less possible communication and synchronization overhead, VOTE uses a Fibo-tree to implement the distribution phase of a barrier synchronization. Figure 10 show the distribution latency of a barrier synchronization for various pairs of s and r values on the MANNA parallel computing system.

The dotted line presents the runtime of a centralized approach where the barrier server distributes the messages to all processes. The time until the first process receives a notification is about $115\mu s$, whereas the second process is activated after $142\mu s$. With these two equations $s + r = 115$ and $2s + r = 142$ follows that $s = 27$ and $r = 88$.

Reasonably, these two numbers are represented by $s = 1$ and $r = 3$. The graph for these values show the best performance results, although, at least for the small scale parallel machine available for testing, the graph for $s = 1, r = 1$ approximates the optimal results reasonably. In contrast, a binomial tree ($s = 1$, $r = 0$) produces runtime results which are far from optimal. For process numbers less than ten, the sequential approach outperforms the binomial tree. For 19 processes, the time to receive notifications was measured to be 569 μs in a sequential implementation and 451 μs in a binomial tree. The runtime could be dropped to 318 μs using a Fibo-tree with parameters $s = 1$ and $r = 3$.

6 Conclusion

This article presented a new multicast tree (named Fibo-tree) for a one-to-many communication based on a multicast tree. The Fibo-tree provides optimal performance results if the underlying communication network supports uniform communication latencies. The strength of the model is the distinction of both a send *and* a receive latency. This is in contrast to other approaches which are based on the notion of a communication latency only, and, thus ignore the specific latencies at the sending and the receiving sites.

It was demonstrated that the distribution latency of a barrier is significantly improved when the multicast tree considers both a send and receive latency. An algorithm was presented which implements the mathematical model of Fibo-trees. Moreover, the example is parameterizable and allows a customization to any execution platform simply through the specification of the communication setup time (s latency) and the further costs of receiving a fixed sized and small message (r latency).

In the experimental part, the significance of Fibo-trees was shown through runtime measurements of the distribution latency of a barrier synchronization. Compared to a centralized approach applied to a parallel application of 19 processes, the performance improvement was about 44.1 %. In case of a binomial multicast tree, the performance improvement still was about 29.5 %. That is, the binomial multicast tree, in theory implementing the maximal speed of message distribution (i.e. doubling the number of processes holding a copy of the message in each communication step), will never perform optimal in the presence of a system with non-zero receive latency.

These results were obtained by (1) determining the send and receive latencies of the target system (i.e. the hardware and software communication overheads) and (2) constructing an optimal multicast tree for this particular target system using the Fibo-tree algorithm provided with the latency parameters as input. In the near future, the model of Fibo-trees will be extended to cover additional aspects of communication. At first, the model will support hierarchies of communication networks with different send and receive latencies. Moreover, the impact of contentions on the communication network will be addressed, thus covering a broader range of communication networks. In total, this will make the optimal performance results of Fibo-trees applicable to most state-of-the-art computing systems.

References

[1] A. Bar-Noy, S. Knipis, "Designing Broadcasting Algorithms in the Postal Model for Message-Passing Systems", *Mathematical System Theory*, Vol. 27, No. 5, pp. 431–452, Sep./Oct., 1994.

[2] P. Beckman, D. Gannon, "A Portable Run-Time System for Object-Parallel Systems", *Technical Report*, Indiana University, USA.
URL: http://www.extreme.indiana.edu/hpc++/docs/RTS/

[3] M.R. Brown, "Implementation and Analysis of Binomial Queue Algorithms", In *Siam J. Computing*, pp. 298–319, Vol. 7, No. 3, Aug., 1978.

[4] U. Brüning, W.K. Giloi, W. Schröder-Preikschat, "Latency Hiding in Message Passing Architectures", In *Proceedings of the International Parallel Processing Symposium*, IPPS-8, pp. 704–709, Cancun, Mexico, Apr., 1994.

[5] J. Cordsen, W. Schröder-Preikschat, "On the Coexistence of Shared-Memory and Message-Passing in the Programming of Parallel Applications", *Proceedings of the HPCN Europe '97*, Lecture Notes in Computer Science, Springer Verlag, 1997.

[6] Cray Research Inc., "Cray Research Inc., Internet Information Server",
URL: http://www.cray.com

[7] D. Culler, R. Karp, D. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, T. von Eicken, "LogP: Towards a Realistic Model of Parallel Computation", In *Proceedings of the 4th SIGPLAN Symposium on Principles and Practices of Parallel Programming*, ACM, San Diego, CA, May, 1993.

[8] T.H. Dunigan, "Kendall Square Multiprocessor: Early Experiences and Performance", *Technical Report*, ORNL/TM-12065, Oak Ridge National Laboratory, Mar., 1992.
URL: http://www.epm.ornl.gov/~dunigan

[9] S. Fleischman, "MPI Collective Communication on the CONVEX Exemplar SPP-1000 Series Scalable Parallel Computer", Presented at *MPI Developers Conference*, University of Notre Dame, June 22+23, 1995.
URL: http://www.cse.nd.edu/mpidc95/proceedings/

[10] V. Georgitsis, J.S. Sobolewski, "Performance of MPL and MPICH on the SP2 System", Presented at *MPI Developers Conference*, University of Notre Dame, June 22+23, 1995.
URL: http://www.cse.nd.edu/mpidc95/proceedings/

[11] W. Gropp, E. Lusk, A. Skjellum, "Using MPI: Portable Parallel Programming with the Message-Passing Interface", MIT Press, 1994.

[12] S.L. Johnsson, C.-T. Ho, "Optimum Broadcasting and Personalized Communication in Hypercubes", In *IEEE Transactions on Computers*, Vol. C-38, pp. 1249–1268, Sep., 1989.

[13] H. Jordan, "Special Purpose Architecture for Finite Element Analysis". In *Proceedings of the IEEE International Conference on Parallel Processing*, pp. 263–266, 1978.

[14] D. Kranz, K. Johnson, A. Agarwal, J. Kubiatowicz, B.-H. Lim, "Integrating Message-Passing and Shared-Memory: Early Experience", In Proceedings of the 4th Symposium on Principles and Practice of Parallel Programming, pp. 54–63, May, 1993.

[15] J.-Y. L. Park, H.-A. Choi, N. Nupairoj, L.M. Ni, "Construction of Optimal Multicast Trees Based on the Parameterized Communication Model", *Technical Report*, Department of Computer Science, Michigan State University, Feb., 1996.

[16] A. Geist, A. Beguelin, J.J. Dongorra, W. Jiang, R. Manchek, V.S. Sunderam, "PVM3 User's Guide and Reference Manual", *Technical Report*, ORNL/TM-12187, Oak Ridge National Laboratory, Oak Ridge, USA, May, 1993.

[17] W. Schröder-Preikschat, "The Logical Design of Parallel Operating Systems", Prentice-Hall, ISBN 0-13-183369-3, 1994.

[18] H. Xu, P.K. McKinley, L.M. Ni, "Efficient Implementation of Barrier Synchronization in Wormhole-Routed Hypercube Multicomputers", *Technical Report*, MSU-CPS-ACS-47, Oct., 1991.