Alcom-FT Technical Report Series
ALCOMFT-TR-01-173

A Bandwidth Latency Tradeoff for Broadcast and Reduction*

Peter Sanders and Jop F. Sibeyn

Max-Planck-Institut für Informatik
Im Stadtwald, 66123 Saarbrücken, Germany.
{sanders,jopsi}@mpi-sb.mpg.de.
http://www.mpi-sb.mpg.de/{~sanders,~jopsi}

Abstract. The "fractional tree" algorithm for broadcasting and reduction is introduced. Its communication pattern interpolates between two well known patterns — sequential pipeline and pipelined binary tree. The speedup over the best of these simple methods can approach two for large systems and messages of intermediate size. For networks which are not very densely connected the new algorithm seems to be the best known method for the important case that each processor has only a single (possibly bidirectional) channel into the communication network.

1 Introduction

Consider P processing units, *PUs*, of a parallel machine. *Broadcasting*, the operation in which one processor has to send a message M to all other PUs, is a crucial building block for many parallel algorithms. Since it can be implemented once and for all in communication libraries such as MPI [9], it makes sense to invest into algorithms which are close to optimal for all P and all message lengths k. Since broadcasting is sometimes a bottleneck operation, even constant factors should be considered. In addition, by reversing the direction of communication, broadcasting algorithms can usually be turned into reduction algorithms. *Reduction* is the task to compute a generalized sum $\bigoplus_{i<P} M_i$, where initially message M_i is stored on PU i and where "$\oplus$" can be any associative operator. Broadcasting and reduction are among the most important communication primitives. For example, some of the best algorithms for matrix multiplication or dense matrix-vector multiplication have these two functions as their sole communication routines [5].

We study broadcasting long[1] messages for a simple synchronous, symmetric communication model which is intended as a least common denominator of practical protocols able to support high bandwidth for long messages: It takes time $t + k$ to transfer a message of size k regardless which PUs are involved.

* Partially supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

[1] For very short messages, different algorithms based on trees with large degree near the root are better, also a synchronous communication model is less attractive.

This is realistic on many modern machines where network latencies are small compared to the start-up overhead t. Both sender and receiver have to cooperate in transmitting a message. We are considering two variants. Our default is the *duplex* model where a PU can concurrently send a message to one partner and receive a message from a possibly different partner. We use the name send|recv to denote this parallel operation in pseudo-code. The more restrictive *simplex* model permits only one communication direction per processor. The broadcasting time for simplex is at most twice that for duplex communication for half as many PUs.[2] We note the cases where we can do better.

We begin our description in Sec. 2 by reviewing simple results on *pipelined* broadcasting algorithms. By arranging the PUs in a simple chain, execution time

$$T^*_\infty = k\left(1 + \mathcal{O}\left(\sqrt{tP/k}\right)\right) + \mathcal{O}(tP) \tag{1}$$

can be achieved. Except for very long messages, a better approach is to arrange the PUs into a binary tree. This approach achieves broadcasting time[3]

$$T^*_1 = k\left(2 + \mathcal{O}(\sqrt{t\log(P)/k})\right) + \mathcal{O}(t\log P) \tag{2}$$

(replace "2" by "3" for the simplex model). We also give lower bounds.

The main contribution of this paper is the *fractional tree* algorithm described in Sec. 3. It is a generalization of the two above algorithms and achieves an execution time of

$$T^*_* = k\left(1 + \mathcal{O}\left(\left(\frac{t\log P}{k}\right)^{1/3}\right)\right) + \mathcal{O}(t\log P), \tag{3}$$

i.e., it combines the advantage of the chain algorithm to have a $(1 + o(1))$ factor in the k dependent term with the advantage of the binary tree algorithm to have a logarithmic dependence on P in the t dependent term of the execution time. For large P and medium k the improvement over both simple algorithms approaches a factor two (3/2 for the simplex model).

For some powerful network topologies, somewhat better algorithms are known. For Hypercubes, there is an elegant and fast algorithm which runs in time $T^*_{\mathrm{HC}} = k(1 + \sqrt{t\log(P)/k})^2 = k(1 + \mathcal{O}(\sqrt{t\log(P)/k})) + \mathcal{O}(t\log P))$ [1,4]. However, no similarly good algorithm was known for networks with low bisection[4] bandwidth, e.g., meshes. Even for fully connected networks the best known algorithms for arbitrary P are quite complicated [2,8]. The fractional tree algorithm does not have this problem. In Sec. 4 we explain how it can be adapted to several sparse topologies like hierarchical networks and meshes.

[2] A *couple* of simplex PUs emulate each communication of a duplex PU in two substeps. In the first substep one partner acts as a sender and the other as a receiver for communicating with other couples. In the second substep the previously received data is forwarded to the partner.

[3] Throughout this paper $\log x$ stands for $\log_2 x$.

[4] The *bisection width* of a network is the smallest number of connections one needs to cut in order to produce two disconnected components of size $\lfloor P/2\rfloor$ and $\lceil P/2\rceil$.

2 Basic Results on Broadcasting Long Messages

Lower Bounds. All non-source PUs must receive the k data elements, and the whole broadcasting takes at least $\log P$ steps. Thus in the duplex model there is a lower bound of

$$T_{\text{lower}} = k + t \cdot \log P \ . \tag{4}$$

In the simplex model all non-source PUs must receive the k data elements. Hence the communication volume is at least $(P-1) \cdot k$. This observation can be combined with Eq. (4) yielding the following Lemma:

Lemma 1.

$$T_{lower,\ one\text{-}port} = 2 \cdot (1 - 1/P) \cdot k + t \cdot (\log P - 3).$$

Proof: Our proof is based on the following observations:

1. After T time, for $T < t \cdot \log P$, at most $2^{\lfloor T/t \rfloor}$ PUs are sending.
2. If there are a active PUs, then in T time units at most $a \cdot T$ packets can be transferred.
3. If at time T the sum over all PUs of the amount of data that the PUs have not yet received equals S, then the broadcasting cannot be completed before time $T' = T + 2 \cdot S/P$.

Observation 1 follows from the fact that in any t time units the number of active PUs can at most double. Observation 3 follows because a PU can either receive or send but not both, and the S data have to be sent as well as received.

Combining observation 1 and 2 implies that during the first $t \cdot (\log P) - 1$ time units, at most $\sum_{i=0}^{\log(P)-1} t \cdot 2^i < t \cdot P$ packets can be transferred (this is a course an overestimate). In total, $(P-1) \cdot k$ packets have to be transferred. So, according to observation 3, the remaining broadcasting still takes at least $2 \cdot ((P-1) \cdot k - t \cdot P)/P$ time. ■

These lower bounds hold in full generality. For a large and natural class of algorithms, we can prove a stronger bound though. Consider algorithms that divide the total data set of size k in s packets of size k/s each. All PUs operate synchronously, and in every step they send or receive at most one packet. So, until step $s-1$, there is still at least one packet known only to the source. Thus, for given s, at least $s - 1 + \log P$ steps are required in the duplex model. Each step takes $k/s + t$ time. For given k, t and P, the minimum is assumed for $s = \sqrt{k \cdot t / \log P}$:

$$T^*_{\text{lower}} = k \left(1 + \sqrt{t \log(P)/k}\right)^2 \ . \tag{5}$$

Two Simple Pipelined Algorithms. For $k \gg t$, a central idea for fast broadcasting is to chop the message into s packets of size k/s and to forward these packets in a pipelined fashion. The simplest pipelined algorithm arranges all PUs into a chain of length $P-1$. The head of the chain feeds packets downward. Interior PUs receive one packet in the first step and then in each step receive the next packet and forward the previously received packet. Fig. 3-d gives an example. It is easy to see that one gets an execution time of $T_\infty^s := (P - 2 + s) \cdot (t + \frac{k}{s})$. The optimal choice for s is $\sqrt{k(P-2)/t}$. Substituting this into T_∞^s yields $T_\infty^* := k\left(1 + \sqrt{t \cdot (P-2)/k}\right)^2 = k\left(1 + \mathcal{O}\left(\sqrt{tP/k}\right)\right) + \mathcal{O}(tP)$.

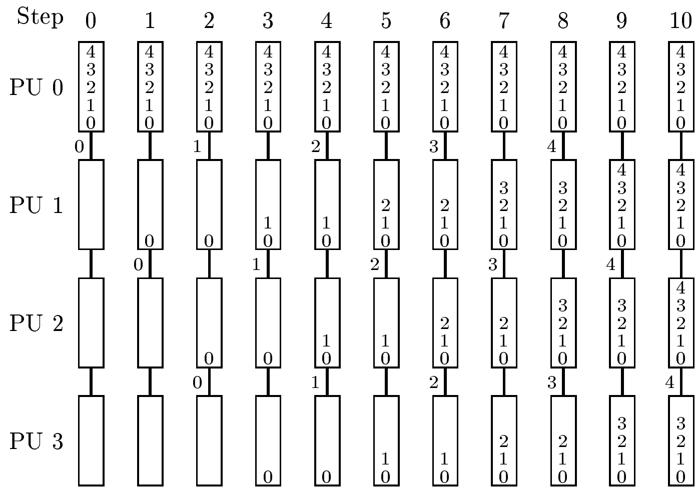

Fig. 1. The sequential pipeline algorithm for $P = 4$ and $s = 5$.

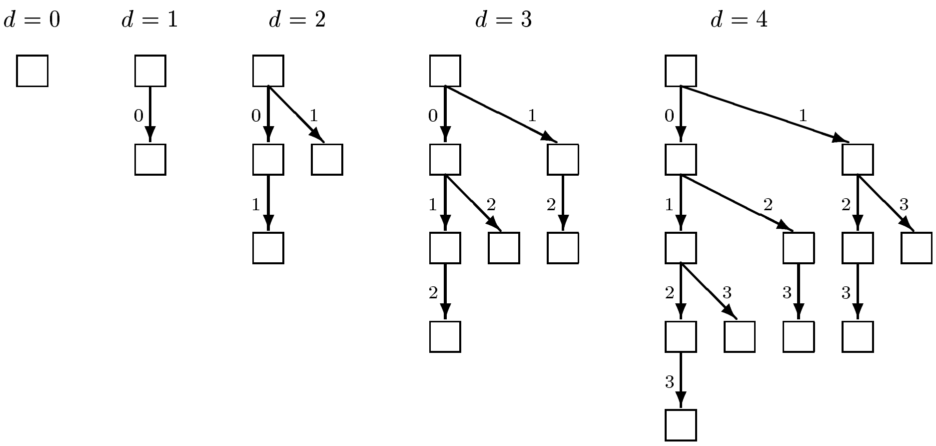

Fig. 2. The optimal tree structures used for pipelined binary tree broadcasting for trees of various depths d. The tree for depth d is constructed out off the trees for $d-1$ and $d-2$. The numbers indicate the routing steps in which a packet coming from the root is routed over the connections. Every three steps the root injects one packet.

For $k \gg tP$ the performance of this algorithm is quite close to the lower bound (4). However, since t is usually a large constant, on systems with large P we only get good performance for messages which are extremely large. We can reduce the dependence on P by arranging the PUs into a binary tree. Now every interior node forwards every packet to both successors. This needs two steps per packet. The execution time is $T_1^s := (d + 2s) \cdot (t + \frac{k}{s})$ where d is the time step just before the last leaf receives the first packet; d is defined by the recurrence $P_i = 1 + P_{i-1} + P_{i-2}$, $P_0 = 1$, $P_1 = 2$. We have $d = \min\{i : P_i \geq P\} - 1 \approx \log_{1.62} P$. For our purposes it is sufficient to note that $d = \mathcal{O}(\log P)$. Fig. 3-a shows the tree with $P_5 = 18$ PUs. Choosing $s = \sqrt{2k \cdot d/t}$, one gets $T_1^* := k\left(\sqrt{2} + \sqrt{d \cdot t/k}\right)^2 = k\left(2 + \mathcal{O}(\sqrt{t \log(P)/k})\right) + \mathcal{O}(t \log P)$. (For the simplex model replace the two by a three.) For small and medium k this is much better than the chain algorithm, yet for large k it is almost two times slower.

3 Fractional Tree Broadcasting

The starting point for this paper was the question whether we could find a communication pattern which allows a more flexible tradeoff between the high bandwidth of a chain (i.e., a tree with degree one) and the low latency of a binary tree. We give a family of communication pattern we call *fractional trees* which have this property. The algorithm for the duplex model, described in Sec. 3.1, is rather simple. An efficient implementation in the simplex model is explained in Sec. 3.2.

3.1 Duplex Model

The idea for fractional trees is to replace the node of a binary tree by a group of r PUs forming a chain. The input is fed into the head of this chain. The data is passed down the chain and on to a successor group as in the single chain algorithm. In addition, the PUs of the group cooperate in feeding the data to the head of a second successor group. Fig. 3 shows the structure of a group and several examples.

All PUs execute the same code shown in Fig. 4. All timing considerations are naturally handled by the synchronization implicit in synchronous point-to-point communication. The input is conceptually subdivided into s/r runs of r packets each. The only nontrivial point is that the i-th member of a group is responsible for passing the i-th packet of every run of r packets to the right. The effect is that every $r+1$ steps the head of the right successor gets a run of r packets in the right order. The pause after this run is used to pass the last packet downward. Packets are passed right while the next run arrives.

As in the special case of binary trees ($r = 1$), the right successors receive data one step later than the downward successors. Therefore, optimal tree layouts are somewhat skewed. The number of nodes reachable within $d + 1$ steps is governed by the recurrence $P_i = r + P_{i-r} + P_{i-r-1}$ ($P_i = i + 1$ for $i \leq r$)

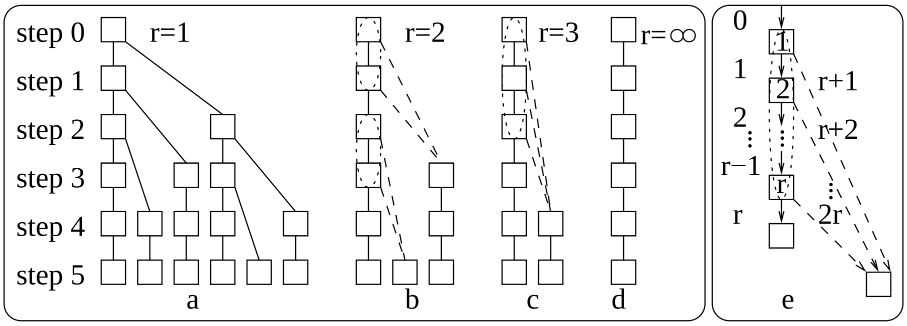

Fig. 3. Examples for fractional trees with $r \in \{1, 2, 3, \infty\}$ where the last PU receives its first packet after 5 steps. The case $r = 1$ corresponds to plain binary trees and pipelines can be considered the case $r = \infty$. Part e) shows the communication pattern in a group of r PUs which cooperate to supply two successor nodes with all the data they have to receive. Edges are labeled with the first time step when they are active.

so that $d = \min\{i : P_i \geq P\} - 1$. This implies $d = \mathcal{O}(r \log(P/r))$. Using this recurrence each processor can find its place in the tree in time $\mathcal{O}(d)$ and without any communication.

Performance Analysis. Having established a smooth timing of the algorithm the analysis can proceed analogously to that of the simple algorithms from the introduction. Every communication step takes time $(t + k/s)$ and $d + s \cdot (1 + 1/r)$ steps are needed until all s/r runs have reached the last leaf group. We get a total time of

$$T_r^s := \left(d + s\left(1 + \frac{1}{r}\right)\right)\left(t + \frac{k}{s}\right) \quad . \tag{6}$$

Using calculus one gets $s = \sqrt{kdr/(t(r+1))}$ as an optimal choice for the number of packets. Substituting this into Eq. (6) yields

$$T_r^* := k\left(1 + \tfrac{1}{r}\right)\left(1 + \sqrt{\tfrac{drt}{k(r+1)}}\right)^2 = k\left(1 + \tfrac{1}{r} + \mathcal{O}\left(\sqrt{\tfrac{rt\log P}{k}}\right)\right) + \mathcal{O}(rt\log P). \tag{7}$$

Since d depends on r in a complicated way, there seems to be no closed form formula for an optimal r. But we get a close to optimal value for r by setting $d = d' \cdot (r+1)$ and ignoring that d' depends on r.[5] We get $r \approx (k(r+1)/(d \cdot t))^{1/3}$. After rounding, these values make sense for $k(r+1) \geq d \cdot t$. For smaller k one should use $r = 1$ or even a non-pipelined algorithm. Substituting r and s into T_r^s we get a broadcasting algorithm with execution time

$$T_*^* \leq k\left(1 + \left(\frac{d \cdot t}{k(r+1)}\right)^{1/3}\right)^3 = k\left(1 + \mathcal{O}\left(\left(\frac{t\log P}{k}\right)^{1/3}\right)\right) + \mathcal{O}(t\log P) \ .$$

[5] In the program one can efficiently solve the equations numerically, e.g., using golden section search [7].

```
Procedure broadcastFT(r, s, 0 ≤ i < r:Integer; var D[0..s − 1]:Packet)
    recv(D[0])                                  −− wait for first packet
    pipeDown(r, 0, D)                           −− First phase
    for k := r to s − r step r do               −− Remaining phases
        sendRight|Recv(D[k − r + i], D[k])
        pipeDown(r, k, D)
    sendRight(D[s − r + i])

    (* send down packets D[k..k + r − 1] and receive packets D[k + 1..k + r − 1] *)
    Procedure pipeDown(r, k:Integer; var D[..]:Packet)
    for j := k to k + r − 2 do sendDown|Recv(D[j], D[j + 1])
    sendDown(D[k + r − 1])
```

Fig. 4. Pseudocode executed on each PU for fractional tree broadcasting, where i is the index of the PU within its group, s is a multiple of r, and the array D is the input on the root and the output on the other PUs. For the root PU, receiving is a no-op. For the top PU of a group receiving means receiving from any PU in the predecessor group. For the other PUs it means receiving from the predecessor in the group. Sending down means sending to the next PU in the group respectively sending to the top PU of the successor group. Sending right means sending to the top PU of the right successor group. If the successor defined by this convention does not exist, sending is a no-op.

For $k \gg t \log P$ algorithm performs quite close to the lower bound (4).

Performance Examples. How does the algorithm compare to the two simple algorithms? For example, for $P = 1024$ and $k/t = 4096$ we choose $d' = d/(r + 1) \approx \log P - 1 = 9$ and get $r \approx (k/(d' \cdot t))^{1/3} \approx 8$. This yields $d = 57$ and we get $s = \sqrt{4096 \cdot 57 \cdot 8/9} \approx 456$. With these values $T_r^s \approx 1.389k$. These choices are quite robust. For example, a better approximation of the optimal r yields $r = 10$ and $s = 503$ but the resulting $T_r^s \approx 1.387k$ is less than 0.2 % better.

Fig. 5 plots the achievable speedup for three different machine sizes. Even for a medium size parallel computer ($P = 64$) an improvement of up to a factor 1.29 can occur. For very large machines ($P = 16384$) the improvements reach up to factor of 1.8 and a significant improvement is observed over a large range of message lengths.

Our conclusion is that fractional tree broadcasting yields a small improvement for "everyday parallel computing" yet is a significant contribution to the difficult task of exploiting high end machines such as the ones currently build in the ASCI program. For example, Compaq plans to achieve 100TFlops with 16384 Alpha processors by the year 2004 [3].

3.2 Simplex Model

Our algorithm for the simplex model is an implementation of the duplex algorithm using unidirectional communication. We just replace every parallel send|Recv operation by a sequence `s`(end) `r`(eceive) or `r`(eceive) `s`(end). The difficulty is that fractional trees contain cycles so that it is not trivial to decide which order

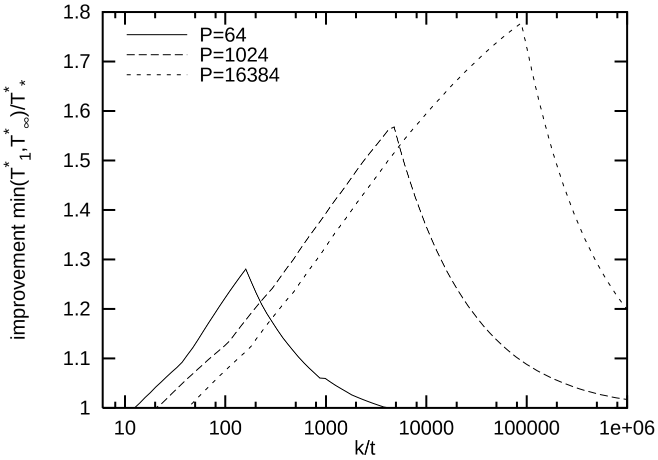

Fig. 5. Improvement of fractional tree broadcasting over the best of pipelined binary tree and sequential pipeline algorithm as a function of k/t.

to choose without incurring delays or even deadlocks. It turns out that we have to distinguish two kinds of groups, type A and type B.

An *A-group* consists of PUs which all execute the same periodic schedule in the inner loop. Namely, $(\texttt{srrs})^{r/2}, \texttt{s}$ for even r and $\texttt{rs}(\texttt{srrs})^{\lfloor r/2 \rfloor}, \texttt{s}$ for odd r, i.e., the PUs alternate between the order `rs` and `sr` for emulating a send|Recv call of the duplex algorithm. Observe, that the last send/Recv always uses the order `rs`. This property remains true for B-groups.

An A-schedule works efficiently if the head of the group is fed with packets at steps $j_0 + (2r+1)j + x$ with $x \in \{2, 3, 6, 7, \ldots, 2r-6, 2r-5, 2r-2, 2r-1\}$ where r is even, j_0 is some fixed phase shift and where j takes arbitrary integer values. For odd r we get the same situation except that $x \in \{1, 4, 5, 8, 9, \ldots, 2r-6, 2r-5, 2r-2, 2r-1\}$.

The downward output of an A-group is of the same form so that the downward successor group can be an A-group again. However, the rightward output has offsets $x \in \{1, 3, 5, 7, \ldots, 2r-7, 2r-5, 2r-3, 2r-1\}$. We therefore design a *B-schedule* which works efficiently for this case so that it can be used for the rightward successor group of an A-group. The first PU in a B-group uses the schedule $(\texttt{rs})^r, \texttt{s}$, whereas the last PU uses the same schedule as the PUs in A-groups. The schedules for the remaining PUs can be viewed as interpolating between these two schedules. More precisely, the i-th PU in a B-group $(0 \leq i < r)$ schedules the last i send|Recv operations according to the schedule of an A-group whereas the first $r - i$ send|Recv operations are scheduled `rs` for odd i and `sr` for even i. Both successor groups of a B-group can be A-groups. Fig. 6 gives an example for $r = 6$.

The performance analysis can proceed analogously to the duplex case. Forwarding a run now takes $2r+1$ steps and after choosing good values for the

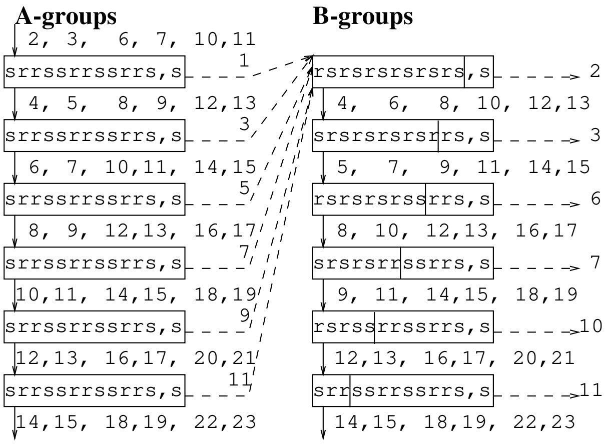

Fig. 6. Simplex schedule for fractional tree broadcasting for the case $r = 6$. The left part describes an A-group which feeds a B-group to its right. All other successor groups are A-groups. Nodes are labeled with their periodically executed schedule. Edges are labeled with offsets specifying when packets cross this edge. More precisely, is we set $j_0 = 12$ and pick any integer j then if an edge is labeled with an offset x this means that at time step $j_0 + 13j + x$ a packet traverses this edge.

parameters s and r one gets

$$T^*_* = k\left(2 + \mathcal{O}\left(\left(\frac{t\log P}{k}\right)^{1/3}\right)\right) + \mathcal{O}(t\log P) \ .$$

Again we are close to the lower bound (Eq. (1)). The improvement factor over binary tree broadcasting is now bounded by 1.5.

4 Sparse Interconnection Networks

Hierarchies of Crossbars. Compaq's above mentioned 16384 PU system is expected to consist of 256 SMP modules with 64 PUs each. We view it as unlikely that it will get an interconnection network with enough bisection width to efficiently implement the hypercube algorithm. Rather, each module will only have a limited number of channels to other modules. We call such a system a 256×64 *hierarchical* crossbar. Systems with similar properties are currently build by several companies.

We now explain how a fractional tree with group size r can be embedded into an $a \times b$ hierarchical crossbar if $b \geq r$ and if each module supports at least two incoming and outcoming channels to arbitrary other modules. A generalization to more than two levels of hierarchy is also possible.

First, one group in each module is connected to form a *global* binary tree with a nodes. Next, the $b - r$ remaining PUs in each module are connected to a form *local* fractional tree. What remains to be done is to connect the local trees by the global tree. Groups in the global tree with degree one can directly link with their local tree. Leaf groups in the global tree use one of their free links to connect to their local tree. The remaining free links are used to connect to the local trees of modules with a group in the global tree of degree two. There will be one remaining unused link which can be used to further optimize the structure. Fig. 7 gives an example where this link is used to build a tree where all PUs receive the first data at about the same time.

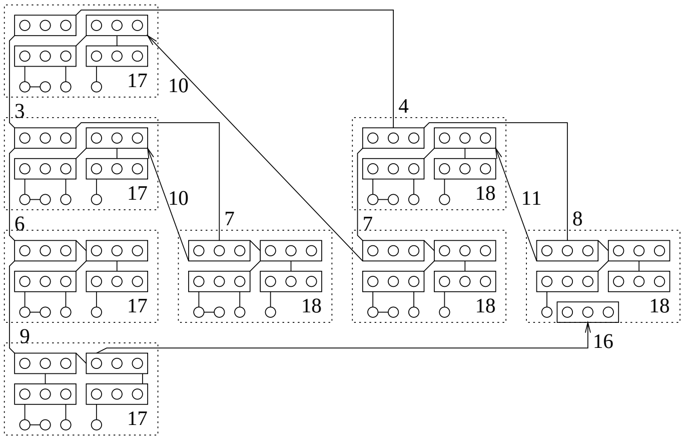

Fig. 7. Embedding of a fractional tree with $r = 3$ into an 8×16 hierarchical crossbar. After 18 steps all PUs receive the first data.

By accepting an additional depth of $(r + 1) \log b$, we can work with one less connection per module: Use two global groups per module. The first one links to the second one and one other module. The second one links to the local tree and possibly to one other module.

Meshes. We only outline a simple case. Generalizations which are sufficient in practice should be relatively easy. A completely general treatment might turn out to be rather complicated. Assume we are given an $a \times b$ mesh and $r = r_1 \cdot r_2$ such that $a/r_1 = b/r_2$ is a power of two. We partition the mesh into submeshes of size $r_1 \times r_2$ each forming a group of the fractional tree. Now we can embed the binary tree of groups exploiting the substantial work on embedding binary trees into meshes (e.g., [10, 6]). Inside the group, the PUs are arranged in a snakelike fashion. In this way one gets an embedding with constant edge congestion. Often it is even possible to achieve edge congestion one for bidirectional meshes. Fig. 8 gives an example where it is exploited that *H-trees* yield a complete binary tree with one leaf in every 2×2 submesh.

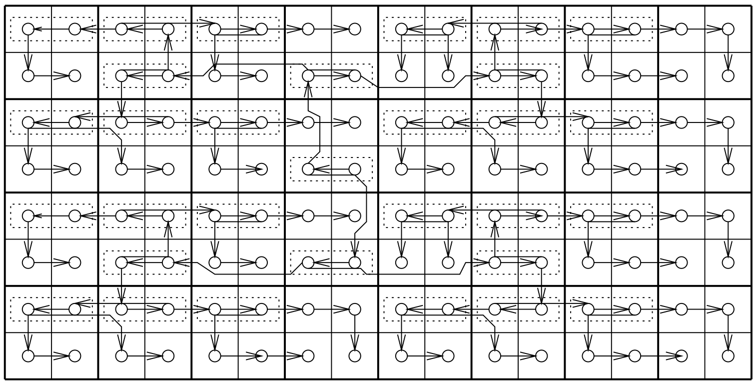

Fig. 8. Embedding of a fractional tree with $r = 2$ into an 8×16 mesh. Broadcasting on it gives edge congestion one even with x-y-routing.

References

1. V. Bala, J. Bruck, R. Cypher, P. Elustondo, A. Ho, C. Ho, S. Kipnis, and M. Snir. CCL: A portable and tunable collective communication library for scalable parallel computers. *IEEE Transactions on Parallel and Distributed Systems*, 6(2):154–164, 1995.
2. A. Bar-Noy and S. Kipnis. Broadcasting multiple messages in simultaneous send/receive systems. In *5th IEEE Symp. Parallel, Distributed Processing*, pages 344–347, 1993.
3. Compaq. AlphaServer SC series product brochure, 1999. `http://www.digital.com/hpc/news/news_sc_launch.html`.
4. S. L. Johnsson and C. T. Ho. Optimum broadcasting and personalized communication in hypercubes. *IEEE Transactions on Computers*, 38(9):1249–1268, 1989.
5. V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing. Design and Analysis of Algorithms*. Benjamin/Cummings, 1994.
6. J. Opatrny and D. Sotteau. Embeddings of complete binary trees into grids and extended grids with total vertex-congestion 1. *Discrete Applied Mathematics*, 98:237–254, 2000.
7. W. H. Press, S. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 2. edition, 1992.
8. Santos. Optimal and near-optimal algorithms for k-item broadcast. *JPDC: Journal of Parallel and Distributed Computing*, 57:121–139, 1999.
9. M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra. *MPI – the Complete Reference*. MIT Press, 1996.
10. P. Zienicke. Embedding of treelike graphs into 2-dimensional meshes. In *Graph Theoretic Concepts in Computer Science*, volume 484 of *LNCS*, pages 182–192. Springer, 1990.