This paper was presented at the 1987 International Conference on Parallel Processing, Aug 17-21 1987.

Yale University
Department of Computer Science

The Communication Efficiency of Meshes, Boolean Cubes and Cube Connected Cycles for Wafer Scale Integration

Abhiram G. Ranade
S. Lennart Johnsson

YALEU/DCS/TR-579
November 1987

This work has in part been supported by the Office of Naval Research under contracts N00014-84-K-0043 and N00014-86-K-0564. Approved for public release: distribution is unlimited.

The Communication Efficiency of Meshes, Boolean Cubes and Cube Connected Cycles for Wafer Scale Integration *

Abhiram G. Ranade and S. Lennart Johnsson

Yale University
Department of Computer Science
New Haven, Connecticut

Abstract

In this paper we analyze the emulation of two-dimensional meshes, butterfly networks, and spanning trees on meshes, Boolean cubes, and Cube Connected Cycles (CCC) networks. We consider three timing models for signal propagation along a wire: constant delay, capacitive delay, and resistive delay. We also present novel layouts for hypercubes and CCCs that offer better performance for some problems, while essentially maintaining the performance for other problems. The mesh interconnection performs better on all emulations for all delay models, if the communication throughput determines the performance. With resistive delay model, meshes also offer the best latency for all emulations. The hypercube and CCC layouts yield lower latency for emulating butterfly networks and spanning trees for the constant delay and capacitive delay models.

1 Introduction

In Wafer Scale Integration, the cost and performance of a parallel computer are critically dependent upon the network interconnecting processing elements. This is because signal propagation delays along a wire do not scale as well as switching speeds with decreased feature sizes; and the wiring area for most networks dominates the area for logic. In this paper we compare the performance of the two dimensional mesh, the binary hypercube and the Cube Connected Cycles network [5] on 3 model problems. Following Dally [2], we assume the same number of processors for all networks, the same memory size per processor, and the same total area i.e. the hypercube and the CCC have narrower communication paths than a mesh. The model problems we consider are the emulation of two dimensional meshes, butterfly networks and spanning trees. We shall refer to the networks being emulated as guest architectures, and the networks on which the emulation occurs as the host architectures.

We consider three timing models. In a purely **capacitive model** the signal delay across a wire of length l is proportional to $\log l$ if driven by a sequence of optimized drivers, viz. "exponential horn" [4]. In the **resistive model** both resistance and capacitance of wires are accounted for. The delay is proportional to l^2 [1,7], but can be reduced to become proportional to l by introducing a sequence of "repeaters" along the wire. The delay would be proportional to the length of the wire also when the speed of light is a determining factor. A third model is a **constant delay model**, which represents the case where the clock period is determined by some other piece of the design.

*This work has in part been supported by the Office of Naval Research under contracts N00014-84-K-0043 and N00014-86-K-0564.

2 Host Architectures

We first establish a numbering of the nodes of the host architectures. Let $a_0|_{l_0}a_1|_{l_1}...a_k|_{l_k}$ represent the number obtained by concatenation of the l_i bit long numbers a_i. In what follows the length subscripts and the bars may be omitted if they are clear by context. Gray(x) is used to denote the representation of x in the binary reflected Gray code. We shall assume that each processor is layed out in a square area of side s, has a local memory of size M^2, and that all communication channels lie outside this area. We assume that all the channels enter and leave the processor at a corner.

Each network has $P^2 = 2^{2p}$ processors. The processors in all the networks are numbered 0 through $P^2 - 1$. In the mesh, a processor numbered $x|_p y$ can communicate with processors $x+1|y$, provided $x + 1 < P$, with $x - 1|y$ provided $x > 0$, with $x|y + 1$ provided $y + 1 < P$, and with $x|y - 1$ provided $y > 0$. In the hypercube, a processor numbered $a|_i b|_1 c$ can communicate with processor $a|\bar{b}|c$, for all i. For the CCC, let $P^2 = (P'^2 \log P'^2)$, with $\log P' = p'$. To simplify the following discussion, we assume that p' is an integer, and $2p'$ a perfect square. It should be observed that if either is not true then our estimates are only affected by small constant factors. In the CCC a processor numbered $a|_{j-1}b|_1 c|_{p'-j}j$ communicates with processors $a|_{j-1}\bar{b}|_1 c|_{p'-j}j$ (along the intercycle channel) and $a|_{j-1}b|_1 c|_{p'-j}(j \pm 1 \bmod 2p')$ (along the intracycle channels). The processors communicate with one another using channels of width w_g, w_h and w_c for the mesh, the hypercube, and the CCC. We shall assume that there are separate channels for communication in either direction.

2.1 Normal Layout of the Hypercube and the Mesh

The processors are laid out in a square region of side S, in P rows and P columns. The processors are separated from each other by communication channels. Processor $x|_p y$ occupies position (x, y) of the layout. For both architectures each processor directly communicates only with processors in its row or column. For meshes, it is only necessary to have two tracks per row, one per channel in each direction. Thus $S = P(s + 2w_g)$. For hypercubes, each row of processors forms a smaller hypercube of p dimensions. The communication channels for the ith dimension can be accomodated in $2 \cdot 2^i$ tracks. Thus, the total number of tracks required is at most $\sum_{i=0}^{i=p-1} 2 \cdot 2^i = 2(P-1)$ tracks. Thus $S \approx P(s + 2Pw_h)$. This layout is known to have optimal area.

2.2 Layout for the CCC

We embed a cycle of length $2p'$ in a square grid of side $\sqrt{2p'}$, with dilation 1 or 2 ($\sqrt{2p'}$ odd). A point on the cycle is arbitrarily designated as 0, and the subsequent points are numbered 1 through $2p'-1$. Let this numbering be denoted by path(x, y) for grid point (x, y). The layout for the CCC can be obtained from the layout for the $2p'$ dimensional hypercube, each hypercube processor being replaced by the $2p'$ processors in each cycle of the CCC, using the path function. Thus, the processor at position $(x|_{p'}v|_{p-p'}, y|_{p'}w|_{p-p'})$ of the layout is numbered $x|y|$ path(v, w). The number of horizontal tracks required per row of cycles is at most $2P'$. Each row of cycles requires at most $2\sqrt{2p'}$ horizontal tracks to layout the intracycle channels. Given that there are P' rows of cycles, the total height of the layout is $S = P'(\sqrt{2p'}s + 2\sqrt{2p'}w_c + 2P'w_c) = Ps + 2Pw_c + P^2 w_c/p' \approx Ps + P^2 w_c/p$. Because the cycle can be embedded with dilation 2, the maximum distance between adjacent processors in a cycle can be at most $2S/P$. The maximum distance between arbitrary processors is $S/2$, when the cycles differ in the most significant bit.

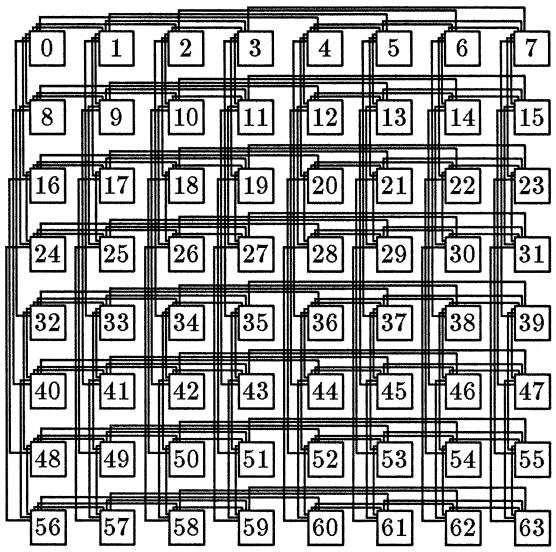

Figure 1: Normal layout for a hypercube

2.3 Gray code derived layouts

Gray code derived layouts have better performance for mesh like communication, without significantly sacrificing performance for other communication patterns. The total area and maximum wire length are the same as for the previous layout. For hypercubes, the processor occupying position (x, y) is numbered $\text{Gray}(x)|\,\text{Gray}(y)$. Processors only communicate with others in the same row or column. Since $\text{Gray}(x)$ and $\text{Gray}(x+1)$ $(x < P)$ differ only in one bit, the processor at (x, y) is directly connected to the processor at $(x+1, y)$. Thus, this layout contains an embedded mesh for which all the channels have the same length, figure 2. For CCCs, the position $(x|_{p'}v|_{p-p'}, y|_{p'}w|_{p-p'})$ holds the processor numbered $\text{Gray}(x)|\,\text{Gray}(y)|\,\text{path}(v, w)$.

In the layout of section 2.1 all the channels corresponding to a given dimension in the hypercube have the same length; $\frac{S}{P}2^{i \bmod p}$ for dimension i. However, in the Gray code layout channels along the ith dimension have lengths $\frac{S}{P}(2 \cdot 2^{k \bmod p} - 1)$, $k = \{0, 1, \ldots, i\}$. The intercycle channel lengths in the CCC vary similarly; $\frac{S}{P}(2 \cdot 2^{k \bmod p'} - 1)$, $k = \{0, 1, \ldots, i\}$. For FFT like communication patterns the performance of Gray code derived layouts may be at most a factor of two lower than the normal layout. Mesh-like communication may be a factor of $P/2$ higher.

2.4 Channel width and cycle time comparison

We have $S = Ps + 2Pw_g \approx Ps + 2P^2 w_h \approx Ps + P^2 w_c/p$. Thus, $w_h = w_g/P$ and $w_c = 2pw_g/P$. With the normal layout the maximum signal propagation distance is $S/2$ for the hypercube and the CCC and S/P for the mesh. With the Graycode derived layouts for the hypercube and the CCC, the maximum distance is S. For the constant, capacitive and resistive delay models, we define t_1,t_2 and t_3 resp. as the time taken for signal propagation over a distance S/P i.e. in general signal propagation over distance l requires time t_1, $c_2 \log l$ and $t_3 l/(\frac{S}{P})$ resp. with $t_2 = c_2 \log \frac{S}{P}$. This gives us table 1, ignoring some lower order terms.

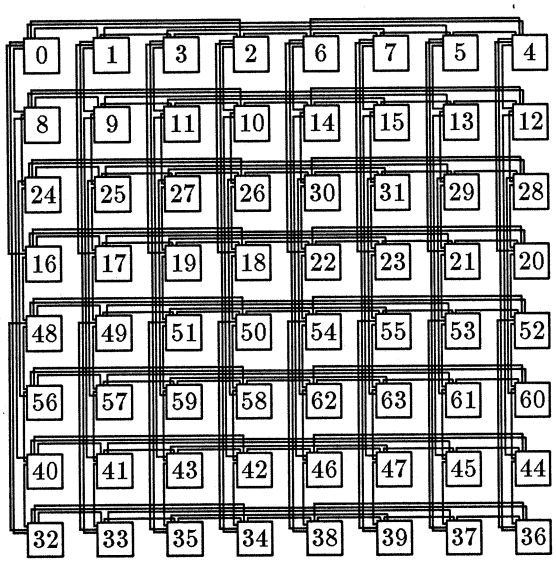

Figure 2: Graycode derived layout for a hypercube

	Channel width	Max. length	Delay along longest channel		
			Const.	Cap.	Res.
Mesh	w	S/P	t_1	t_2	t_3
Hypercube	w/P	$\frac{1}{2}S$	t_1	$t_2 + c_2 p$	$\frac{1}{2}Pt_3$
Hypercube$_{Gray}$	w/P	S	t_1	$t_2 + c_2 p$	Pt_3
CCC	$2wp/P$	$\frac{1}{2}S$	t_1	$t_2 + c_2 p$	$\frac{1}{2}Pt_3$
CCC$_{Gray}$	$2wp/P$	S	t_1	$t_2 + c_2 p$	Pt_3

Table 1: Channel Delay and Width

3 Emulation of two-dimensional meshes

We consider the naive means of emulating meshes in the different host architectures. An edge of the mesh is mapped to a single edge/path in the Boolean cube. There exist p distinct paths between an arbitrary pair of nodes in the Boolean cube, but the paths are, in general, not edge-disjoint, and data cannot be transmitted concurrently along all p paths for all pairs without contention. The case of the CCC is similar. Consider an $N \times N$ two-dimensional guest mesh with $N = 2^n = KP$. When the host architecture is also a mesh, guest processor $x|_p v|_{n-p} y|_p w|_{n-p}$ is mapped onto host processor $x|y$. We prefer this *consecutive* [3] mapping in that it minimizes the interprocessor communication. For a Boolean cube host, guest processor $x|_p v|_{n-p} y|_p w|_{n-p}$ is mapped onto host processor $\text{Gray}_p(x)|\,\text{Gray}_p(y)$, and for a CCC host, submeshes of size $\sqrt{2p'} \times \sqrt{2p'}$ are mapped onto each cycle. Specifically, processor $x|_{p'} v|_{p-p'} a|_{n-p} y|_{p'} w|_{p-p'} b|_{n-p}$ is mapped onto processor $x|y|\,\text{path}(v, w)$ of the CCC. In each case, host processors are assigned square submeshes of size K^2.

Suppose each node of the guest mesh communicates L bits with its neighbors. Then for the mesh host, each of its 4 channels handles KL bits. Similarly, each utilized channel of the hypercube transfers KL bits. The total time required for the commuincation depends upon the widths of the data paths, and the rate at which signals can travel along the paths, Table 2. On the CCC, there is an extra level in the hierarchy. The submeshes held by the $2p'$ processors in each cycle together form a larger submesh of size $\sqrt{2p'}K \times \sqrt{2p'}K$. Neighboring submeshes of the larger size lie in cycles directly connected by an intercycle channel. Thus, the neighbors of the submesh held by each processor may lie within the cycle or in a neighboring cycle. The data movement has three phases. In the first, the processors communicate with their neighbors within the cycle. Processors whose neighbors lie outside the cycle also send the required data to the processor that can communicate with the appropriate cycle. In the second phase, the data collected in the first phase for intercycle communication is actually sent over to the other cycle. Finally, in the last phase the data received by each cycle is forwarded to the appropriate processor.

The first phase communication can be accomplished by a simple scheme in which the data associated with all boundaries of the smaller meshes is circulated around the cycle. This communication requires $4KL \cdot 2p'/2w_c \approx 2PKL/w$ transmissions utilizing the bidirectional channels, $w_c = 2wp/P$, and assuming $p \approx p'$. Each transmission is over a distance of at most $2S/P$. The final phase can easily be accomplished in the same time as the first phase. Thus the combined time for these phases for the three timing models can be estimated to be $4PKLt_1/w$, $4PKL(t_2 + c_2)/w$ and $8PKLt_3/w$. In the second phase all four intercycle channels can transmit concurrently. The number of transmissions required is $KL\sqrt{2p'}/w_c = PKL\sqrt{2p'}/2wp$. The Graycode layout guarantees that the length of each channel is at most $2S/P' = 2\sqrt{2p'}S/P$. Thus, the time required for this phase is respectively $PKLt_1/(\sqrt{2p}w)$, $(t_2 + c_2 \log 2\sqrt{2p'})PKL/(\sqrt{2p}w)$, and $2PKLt_3/w$ for the three timing models. The time for the first and last phases dominates, except for the resistive model. For the normal layout, the longest channel has a length S/2. The results of this analysis are summarized in table 2.

4 Butterfly based algorithms

A butterfly network with $N = 2^n$ inputs has $n + 1$ stages numbered 0 through n, each stage having 2^n nodes. The nodes in the last stage also form the outputs of the network. Node i in stage j is numbered (i, j). Node $(a|_{n-j-1} b|_1 c|_1 d|_{j-1}, j)$ receives L bit arguments for its computation from nodes $(a|b|c|d, j-1)$ and $(a|b|\overline{c}|d, j-1)$ for $0 < j \leq n$, and sends the the L bit results of its computation to nodes $(a|b|c|d, j+1)$ and $(a|\overline{b}|c|d, j+1)$ for $0 \leq j < n$. The algorithm has n steps, 1 through n. At the beginning of step j, nodes in stage j receive data from nodes in stage $j-1$, which they use to compute during that step. Thus at the jth step only the nodes in stage j are active. The $N = K^2P^2$ nodes are mapped onto the P^2 processors so that the initial $\log K^2$ steps are executed independently by each processor (*consecutive* mapping). The time required for the first $2k = logK^2$

	Delay Model		
	Constant	Capacitive	Resistive
Mesh	KLt_1/w	KLt_2/w	KLt_3/w
Hypercube	$PKLt_1/w$	$(t_2 + c_2 p)$ $\cdot PKL/w$	$P^2KLt_3/2w$
Hypercube$_{Gray}$	$PKLt_1/w$	$PKLt_2/w$	$PKLt_3/w$
CCC	$4PKLt_1/w$	$(4t_2 + c_2\sqrt{\frac{1}{2}p})$ $\cdot PKL/w$	$(P/\sqrt{8p} + 8)$ $\cdot PKLt_3/w$
CCC$_{Gray}$	$4PKLt_1/w$	$4PKLt_2/w$	$10PKLt_3/w$

Table 2: Time for emulating one mesh like communication

steps is $2kK^2$. The remaining steps are similar to a butterfly algorithm with P^2 inputs.

4.1 Naive butterfly emulations

For the hypercube, node $(x|_{2p}y|_{2k}, j)$ is mapped onto processor x. In step $2k+i$, $1 \leq i \leq 2p$ of the algorithm processor $a|_{2p-i}b|_1 c|_{i-1}$ communicates with processor $a|\overline{b}|c$, i.e., over a distance of $2^{i \bmod p}\frac{S}{P}$ for the normal layout with delays t_1, $t_2 + c_2(i-1) \bmod p$, and $t_3 2^{i-1 \bmod p}$. The total time, computed by summing over i, is $2pt_1K^2L/w_h$, and $2pt_2K^2L/w_h + c_2p(p-1)K^2L/w_h$, and $2Pt_3K^2L/w_h$. The time required for the Graycode derived layout is computed in a similar manner.

For the mesh the communication in step $2k+i$ is between processors that are at a distance of $2^{i \bmod p}\frac{S}{P}$. The number of transmissions required is $2^{i \bmod p}K^2L/w$. The total number of transmissions over the course of the algorithm is $\sum_{i=1}^{i=2p} 2^{i-1 \bmod p}K^2L/w \leq 2PK^2L/w$.

For the CCC the first $2k$ local steps are followed by $\log 2p'$ steps requiring communication within cycles. This communication can be done in $2p'$ iterations [5], each iteration requiring K^2L elements to be communicated between nearest neighbors in the cycle. The maximum communication distance in this phase is $2\frac{S}{P}$, and the number of transmissions in this phase is $2p'K^2L/w_c$. The computation takes $\log 2p'$ iterations, with each processor computing the results for K^2L nodes in each iteration. The final phase consists of $2p'$ steps and requires intercycle communication in $4p'$ iterations [5]. In each iteration, the maximum number of data transmitted is K^2L, and the maximum distance of transmission is $S/2$ with the normal layout, and S with the Graycode layout. Each iteration requires processors to compute results for K^2L nodes. These results are summarized in table 3.

4.2 Butterfly emulations fully utilizing the Hypercube bandwidth

The naive butterfly emulation is unable to efficiently utilize the channels in the hypercube. There are two ways in which this can be improved. Suppose the computation at each step is negligible, as in barrel shifts, packing, bit reversal etc. Since these algorithms only involve data movement, it is possible to start data transmission along dimension i as soon as data begins to arrive along dimension $i-1$. This pipelining allows all the communication channels to become active very fast. For the constant delay model and $K^2L/w_h > 2p$, pipelining improves the channel utilization by a factor $2p$. Similar results are obtained for the other models. Such improvements are also possible by using a different technique when the computation involved in each step is not negligible. This is best described in the context of an FFT. If the number of elements at each processor is K^2, it is possible to implement a radix $2k = logK^2$ FFT. For $2k \leq 2p$, this results in increasing communication channel utilization by a factor $2k$.

	Communication Time			Computation Time
	Constant Delay	Capacitive Delay	Resistive Delay	
Mesh	$2Pt_1 \lceil K^2L/w \rceil$	$2Pt_2 \lceil K^2L/w \rceil$	$2Pt_3 \lceil K^2L/w \rceil$	$K^2L(2m+2p)$
Hypercube	$2pt_1 \lceil PK^2L/w \rceil$	$p(2t_2 + c_2(p-1)) \cdot \lceil PK^2L/w \rceil$	$2Pt_3 \cdot \lceil PK^2L/w \rceil$	$K^2L(2m+2p)$
Hypercube_{Gray}	$2pt_1 \lceil PK^2L/w \rceil$	$p(2t_2 + c_2(p+1)) \cdot \lceil PK^2L/w \rceil$	$4Pt_3 \cdot \lceil PK^2L/w \rceil$	$K^2L(2m+2p)$
CCC	$6pt_1 \lceil PK^2L/2wp \rceil$	$2p(3t_2 + c_(2p-1)) \cdot \lceil PK^2L/2wp \rceil$	$2p(P+2)t_3 \cdot \lceil PK^2L/2wp \rceil$	$K^2L(2m+4p)$
CCC_{Gray}	$6pt_1 \lceil PK^2L/2wp \rceil$	$2p(3t_2 + c_2(2p+1)) \cdot \lceil PK^2L/2wp \rceil$	$4p(P+1)t_3 \cdot \lceil PK^2L/2wp \rceil$	$K^2L(2m+4p)$

Table 3: Time for FFT

These techniques can improve the performance of the hypercube by a factor of at most $2p$, under proper conditions, thus making its performance competitive with that of the mesh and the CCC for the constant delay model.

5 Summary

Meshes are superior in emulating meshes if the data volume is large, $(KL > w/P)$ for all three timing models. The hypercube layout performs comparable to the mesh layout if the data volume is low and if the constant delay model is applicable, and also under the capacitive and resistive model if the Gray code layout is used. The performance of the CCC layout is inferior to the hypercube layout by a constant factor.

The emulation of butterfly networks on a mesh layout yields a higher performance than on the CCC and the hypercube for all three timing models if the data volume is high $(K^2L > w)$. The performance of the CCC is only off by a small constant factor (1.5) in the constant delay case. The Gray code layout is almost as effective as the normal layout in the capacitive case, and inferior by a factor of 2 in the resistive case. For small data volume, the wide channels in the mesh cannot be utilized fully. Under the constant (capacitive) delay model, the narrow channels in the CCC can be kept busy, and there is no (low) penalty for having long channels. In these cases the CCC performs better than the mesh. The CCC is in general superior to the hypercube for butterfly emulation.

Table 4 summarizes the performance on broadcasting L bits using spanning trees. The conclusions are similar to those for butterfly emulation. Note that for the resistive delay model and hypercubes it is preferable not to use a spanning tree of minimum height, but to form a spanning tree of the wires forming a mesh embedded in the cube. However, the mesh layout is faster due to its wider data paths. If latency is important, then the mesh is superior by a constant factor only for the resistive model.

We conclude that with high data volume the mesh layout yields the best performance for all the emulations under all timing models. For small data volumes, the mesh is comparable to the hypercube and CCC if the Gray code layout is used for emulating meshes. It is inferior to both the hypercube and the CCC for emulating butterfly networks and spanning trees if the constant or capacitive delay models apply. Even for small data volumes, meshes are superior if the resistive model applies.

For the emulations considered, the benefit of the Gray code layouts in emulating meshes is higher than the drawback in emulating butterfly networks or trees.

	Constant Delay	Capacitive Delay	Resistive Delay
Mesh	$(\lceil L/w \rceil - 1 + 2P)t_1$	$(\lceil L/w \rceil - 1 + 2P)t_2$	$(\lceil L/w \rceil - 1 + 2P)t_3$
Hypercube	$(\lceil PL/w \rceil - 1 + 2p)t_1$	$(\lceil PL/w \rceil - 1)(t_2 + c_2(p-1))$ $+2pt_2 + c_2p(p-1)$	$(\frac{1}{2}(\lceil PL/w \rceil - 1) + 2)Pt_3$
Hypercube (Gray)	$(\lceil PL/w \rceil - 1 + 2p)t_1$	$(\lceil PL/w \rceil - 1)(t_2 + c_2p)$ $+2pt_2 + c_2p(p+1)$	$(\lceil PL/w \rceil - 1 + 4)Pt_3$
CCC	$(\lceil PL/wp \rceil - 1 + 4p)t_1$	$(\lceil PL/wp \rceil - 1)(t_2 + c_2(p-1))$ $+4pt_2 + c_2p(p+1)$	$(\frac{1}{2}(\lceil PL/wp \rceil - 1) + 2)Pt_3$
CCC (Gray)	$(\lceil PL/wp \rceil - 1 + 4p)t_1$	$(\lceil PL/wp \rceil - 1)(t_2 + c_2p)$ $+4pt_2 + c_2p(p+3)$	$(\lceil PL/wp \rceil - 1 + 4)Pt_3$

Table 4: Time for broadcast using naive spanning trees of minimum height

References

[1] C. L. Chiang and S. L. Johnsson. *Distributed RC Delay Line Model and MOS PLA Timing Estimation.* Technical Report, Computer Science, California Institute of Technology, September 1983.

[2] William Dally. *A VLSI architecture for concurrent data structures.* Technical Report 5209:TR:86, California Institute of Technology, 1986.

[3] S. L. Johnsson. Communication efficient basic linear algebra computations on hypercube architectures. *Journal of Parallel and Distributed Computing*, :, 1986. Report YALEU/CSD/RR-361, January 1985, Dept. of Computer Science, Yale University.

[4] C. A. Mead and L. Conway. *Introduction to VLSI Systems.* Addison-Wesley, 1980.

[5] Franco P. Preparata and Jean Vuillemin. The cube-connected cycles: a versatile network for parallel computation. *CACM*, 24:5:300–309, May 1981.

[6] A. L. Rosenberg. Preserving proximity in arrays. *SIAM J. Computing*, 4(4):443–460, 1975.

[7] C. L. Seitz. Self-timed VLSI systems. In *Proc. of the Caltech Conference on Very Large Scale Integration*, pages 345–355, Computer Science, California Institute of Technology, 1979.