A Hyperconcentrator Switch for Routing Bit-Serial Messages

Thomas H. Cormen*

Charles E. Leiserson†

Laboratory for Computer Science

Massachusetts Institute of Technology

Cambridge, Massachusetts 02139

*Supported in part by the Defense Advanced Research Projects Agency under Contracts N00014–80–C–0622 and N00014–87–K–0825 and by a National Science Foundation Fellowship.

†Supported in part by the Defense Advanced Research Projects Agency under Contracts N00014–80–C–0622 and N00014–87–K–0825 and by an NSF Presidential Young Investigator Award with matching funds provided by AT&T Bell Laboratories, IBM Corporation, and Xerox Corporation.

Running head: *A hyperconcentrator switch*

Send proofs to

Thomas H. Cormen
545 Technology Square, Room 369
Cambridge, MA 02139
(617) 253-1499
`thc@theory.lcs.mit.edu`

Abstract

In highly parallel message routing networks, it is sometimes desirable to concentrate relatively few messages on many wires onto fewer wires. We have designed a VLSI chip for this purpose which is capable of concentrating bit-serial messages quickly. This hyperconcentrator switch has a regular layout using ratioed nMOS and takes advantage of the relatively fast performance of large fan-in NOR gates in this technology. A signal incurs exactly $2 \lg n$ gate delays through the switch, where n is the number of inputs to the circuit. The architecture generalizes to domino CMOS and BiCMOS as well.

The hyperconcentrator design has applications other than message concentration. It can be used in a superconcentrator switch to provide fault tolerance when interconnections are nonfunctional. Multiple hyperconcentrator switches can be configured into a large partial concentrator switch. The hyperconcentrator design can also be used in a processor datapath to allow various manipulations on bits, including barrel shifting, APL-style compression and expansion, and bit interleaving.

Mathematical Symbols

Symbol	*Comments*
lg	typeset in Roman
$\leq$	
$>$	
$<$	
$\ldots$	
$\lceil \quad \rceil$	
$+$	
$/$	
$=$	
log	typeset in Roman
$\vee$	appears in two different sizes within the text
$\wedge$	
$\overline{A}_i$	line over A does not extend out over subscript
O	italic uppercase "oh"
o	italic lowercase "oh"
Θ	uppercase Greek letter theta
ϕ	lowercase Greek letter phi
Ω	uppercase Greek letter omega
α	lowercase Greek letter alpha
β	lowercase Greek letter beta
μ	lowercase Greek letter mu
ϵ	lowercase Greek letter epsilon
$\sqrt{\quad}$	square root
$\mathrm{E}[\quad]$	Roman uppercase E with argument in square brackets
$\mathrm{Var}[\quad]$	Roman "Var" with argument in square brackets

1 Introduction

The problem of concentrating relatively few communications on many input lines onto a lesser number of output lines must be solved in communication networks of all kinds. In many parallel computing systems, communications are packaged into messages which are routed among the processors. This paper presents a VLSI implementation of a fast concentrator switch suitable for routing bit-serial messages in a parallel supercomputer. (See Figure 1.)

An *n-by-m concentrator switch* has n input wires $X_1, X_2, \ldots, X_n$ and $m < n$ output wires $Y_1, Y_2, \ldots, Y_m$. The switch can establish m disjoint electrical paths from any set of m input wires to the m output wires. A concentrator switch always routes as many messages as possible. Specifically, whenever k out of the n input wires of an n-by-m concentrator switch carry messages, one of the following is true:

- If $k \leq m$, then an electrical path is established from each input wire that contains a message to an output wire.

- If $k > m$, then each output wire has an electrical path established from an input wire that contains a message.

When $k > m$, some messages cannot be successfully routed, in which case we say the switch is *congested.* Typical ways of handling unsuccessfully routed messages in a routing network are to buffer them, to misroute them, or to simply drop them and rely on a higher-level acknowledgment protocol to detect this situation and resend them. The switch design in this paper is compatible with any of these congestion control methods.

One way to create a concentrator switch is with a hyperconcentrator switch. An *n-by-n hyperconcentrator switch* (the terminology is drawn from Valiant [17]) has n input wires $X_1, X_2, \ldots, X_n$ and n output wires $Y_1, Y_2, \ldots, Y_n$. The switch can establish disjoint electrical paths from any set of k input wires, for any $1 \leq k \leq n$, to the first k output wires $Y_1, Y_2, \ldots, Y_k$. In other words, we route the k messages to the first k output wires. We can make any n-by-m concentrator switch from an n-by-n hyperconcentrator switch by simply choosing the first m output wires of the hyperconcentrator switch, $Y_1, Y_2, \ldots, Y_m$, as the m output wires of the concentrator switch.

A hyperconcentrator switch can be implemented using a sorting network, as described by Knuth [8, pp. 220–246]. The inputs to the sorting network are 1's and 0's, representing the presence or absence of messages on the input wires to the switch. The sorting of the 1's and 0's, with 1's before 0's, causes the k input messages to occupy the first k outputs.

Many sorting networks, such as Batcher's bitonic sort [8, pp. 232–233], employ the technique of recursive merging. A problem of size n is divided into two problems of size $n/2$, which are recursively solved in parallel. The two sorted sets are then merged to produce the solution to the original problem. The recursion requires $\lceil \lg n \rceil$ levels (we use the notation $\lg n$ to denote $\log_2 n$), and since each merge step can be performed in $O(\lg n)$ time in parallel, the total time to sort n values is $O(\lg^2 n)$. Sorting networks of depth $O(\lg n)$ are known [1], but they are impractical to use as hyperconcentrator switches because of the large associated constant.

The n-by-n hyperconcentrator switch presented in this paper also uses recursive merging, but by taking advantage of the relatively fast performance of high fan-in NOR gates in nMOS technology, each merge requires only 2 gate delays. A signal therefore incurs exactly $2 \lceil \lg n \rceil$ gate delays in passing through the switch. The switch has a simple design and a regular layout in both ratioed nMOS and domino CMOS technologies. Unlike many concentrator switches in the literature [12, 13, 14], our switch sets itself up "on-line" when messages are presented to it.

The remainder of this paper is organized as follows. Section 2 discusses the merge box, the basic building block of our switch. Section 3 shows how to use merge boxes to implement the hyperconcentrator switch and describes an nMOS implementation. Section 4 addresses issues that arise when implementing the hyperconcentrator architecture in domino CMOS, other precharged MOS disciplines, and BiCMOS. Section 5 covers some applications that benefit from the switch. Finally, Section 6 contains further remarks about the switch.

We shall adopt some notational conventions to ease the exposition in the following sections. Bit and boolean values are denoted by "1" and "0", or by "high" and "low", for TRUE and FALSE respectively. Uppercase symbols denote wire names and lowercase symbols denote integer values. We shall also use uppercase symbols to denote bit values on the wires they name when the usage is unambiguous. Wire names will usually have subscripts.

We assume that the hyperconcentrator switch routes *bit-serial messages*. Each message is formed by a stream of bits arriving at a wire at the rate of one bit per clock cycle. The first bit of each message that arrives at an input wire is the *valid bit*, indicating whether subsequent bits arriving on that wire form a valid message or an invalid message. The bit sequence following a valid bit of 1 forms a *valid message*, which we would like to be routed from an input wire to an output wire of the switch. From there it may pass through the remainder of the routing network. A valid bit of 0 indicates an *invalid message*, which does not need to be routed to an output wire. We assume that in an invalid message, not only is the valid bit 0, but so are all the remaining bits in the message. (This assumption is easy to enforce—just AND the valid bit into each subsequent bit of the message.)

The valid bits all arrive at the input wires of the hyperconcentrator switch during the same clock cycle, which we call *setup*. An external control line signals setup. Message bits entering through input wires at cycles after setup follow the electrical paths in the switch that are established during setup.

2 The Merge Box

This section presents the design of the merge box, the key portion of the hyperconcentrator switch architecture. The hyperconcentrator switch consists of many merge boxes, of various sizes, connected as shown in the next section. The design exploits the fast performance of large fan-in NOR gates in nMOS technology, as a PLA does, to merge two sets of messages of any size in only two gate delays. The merge box design presented in this section uses ratioed nMOS technology and no pass transistors.

A merge box merges two sets of messages, each set sorted by their valid bits, into one sorted set of messages. A *merge box of size* $2m$, where m is a power of 2, has two sets of input wires $A_1, A_2, \ldots, A_m$ and $B_1, B_2, \ldots, B_m$ and one set of output wires $C_1, C_2, \ldots, C_{2m}$. We assume that the lower-numbered wires of both the A and B input sets carry valid messages and that the higher-numbered wires of both the A and B input sets carry invalid messages. That is, if we let p and q be the number of valid messages entering the A and B wire sets respectively, where $0 \leq p, q \leq m$, we require that the valid bits appear on the input wires during setup as follows:

$$\begin{aligned} A_1, A_2, \ldots, A_p &= 1\ , \\ A_{p+1}, A_{p+2}, \ldots, A_m &= 0\ , \\ B_1, B_2, \ldots, B_q &= 1\ , \\ B_{q+1}, B_{q+2}, \ldots, B_m &= 0\ . \end{aligned}$$

During setup, the merge box establishes disjoint electrical connections between the $p+q$ input wires with valid messages and the $p+q$ lower-numbered output wires $C_1, C_2, \ldots, C_{p+q}$ in a combinational fashion, as shown in Figure 2. The connections $C_1 = A_1, C_2 = A_2, \ldots, C_p = A_p, C_{p+1} = B_1, C_{p+2} = B_2, \ldots, C_{p+q} = B_q$ are established, and valid bits appear on the output wires as follows:

$$\begin{aligned} C_1, C_2, \ldots, C_{p+q} &= 1\ , \\ C_{p+q+1}, C_{p+q+2}, \ldots, C_{2m} &= 0\ . \end{aligned}$$

These connections are maintained during subsequent cycles for the remaining bits in the message streams to follow.

Figure 3 is a schematic diagram of a merge box for which $m = 4$. This merge box includes eight NOR gates, with diagonal output wires labeled $\overline{C}_1, \overline{C}_2, \ldots, \overline{C}_8$. Each of these NOR gate outputs is inverted to produce the merge box outputs $C_1, C_2, \ldots, C_8$, so we may view the pulling down of a diagonal wire $\overline{C}_i$ to be equivalent to the corresponding output C_i being 1. The NOR gates have fan-ins ranging from just one pulldown circuit (e.g., the gate with output $\overline{C}_8$) to 5 pulldown circuits (e.g., the gate with output $\overline{C}_4$). In general, the NOR gates have fan-ins of up to $m+1$ pulldown circuits. Each pulldown circuit consists of just one or two transistors, regardless of the size of the merge box, making for fast NOR gates and low-area pulldowns, even with minimum-sized pullups. As can be verified from Figure 3, a merge box of size $2m$ implements the following function:

$$C_i = \begin{cases} A_i \vee \left(\bigvee_{j=1}^{i} (B_j \wedge S_{i+1-j}) \right) & \text{if } 1 \leq i \leq m \ , \\ \bigvee_{j=1}^{2m+1-i} (B_{m+1-j} \wedge S_{i+j-m}) & \text{if } m < i \leq 2m \ . \end{cases}$$

The *switch settings* S_1, S_2, S_3, S_4, S_5 are computed and stored in registers during setup, based on the valid bits appearing at the A and B input wires. In general, a merge box of size $2m$ has switch settings $S_1, S_2, \ldots, S_{m+1}$. These stored settings continue to be used during subsequent cycles. These switch settings establish the electrical connections throughout the entire hyperconcentrator switch. Other than the storing of the switch settings, the operation of the merge box is purely combinational.

Let us look at the operation of the merge box during setup. The lower-numbered A and B input wires have valid bit values of 1, and the higher-numbered A and B input wires have valid bit values of 0. If input A_i is 1, then the NOR gate output $\overline{C}_i$ is pulled down by the single transistor whose gate is A_i. The inverter causes output C_i to be 1. Having input values

$A_1, A_2, \ldots, A_p = 1$ thus causes the outputs $C_1, C_2, \ldots, C_p$ to be 1.

The switch settings S act as steering signals, sending the B values B_1, B_2, $\ldots$, B_m to the output wires $C_{p+1}, C_{p+2}, \ldots, C_{p+m}$. The S values are computed and stored during setup so that only the setting S_{p+1} is 1, corresponding to input A_{p+1} being the lowest-numbered A with a valid bit of 0. (If no input wire A_i is 0, then $p = m$, and only switch S_{m+1} is set to 1.) The S values are defined by the valid bits on the A wires as follows:

$$S_i = \begin{cases} \overline{A}_1 & \text{if } i = 1 \ , \\ A_{i-1} \wedge \overline{A}_i & \text{if } 1 < i \leq m \ , \\ A_m & \text{if } i = m+1 \ . \end{cases}$$

Of the two-transistor pulldown circuits, only column $p+1$ may possibly pull a diagonal wire down to 0, since only switch setting S_{p+1} is high. Similarly, a diagonal wire $\overline{C}_i$ may be pulled down only by input wire A_i or the conjunction $B_{i-p} \wedge S_{p+1}$. The only NOR gate that may be pulled down by input B_1 has output wire $\overline{C}_{p+1}$, and in general the only NOR gate that may be pulled down by input B_i has output wire $\overline{C}_{p+i}$.

For example, suppose that, as in Figure 3, the input wires have the following valid bits during setup:

$$\begin{aligned} A_1, A_2 &= 1 \ , \\ A_3, A_4 &= 0 \ , \\ B_1, B_2, B_3 &= 1 \ , \\ B_4 &= 0 \ . \end{aligned}$$

Then, $p = 2$, $q = 3$, $S_3 = 1$, and all other S_i are equal to 0. There are five valid messages passing through the merge box, and there are exactly five conducting paths to ground, shaded in Figure 3, one for each of the first five diagonal wires, $\overline{C}_1, \overline{C}_2, \overline{C}_3, \overline{C}_4, \overline{C}_5$. These paths to ground cause output values of 1 on the corresponding output wires C_1, C_2, C_3, C_4, C_5. The remaining three diagonal wires, $\overline{C}_6, \overline{C}_7, \overline{C}_8$, are not pulled down to ground by these input values, and thus the output wires C_6, C_7, C_8 all have the value 0.

Now we look at the message bits that arrive after setup. The switch settings S were computed and stored during setup, and they remain unchanged in their registers. Just as during setup, a bit with the value 1 that enters through input wire A_i directly pulls down the diagonal wire $\overline{C}_i$, regardless of the S values. A bit with the value 1 that enters through input wire B_i may pull down only the diagonal wire $\overline{C}_{p+i}$ because the only switch setting with value 1 is S_{p+1}. The only difference in the merge box operation between setup and later cycles is that the S values, which are always used to steer the B values to the appropriate C outputs, are computed and stored only during setup. In the cycles following setup, the merge box is a combinational circuit, reading the registers holding the switch settings S.

Recall that in Section 1, we required that all bits in an invalid message must be 0. We now can see the reason for this restriction. Suppose that in our above example, in which we had $A_3 = 0$ and $S_3 = 1$ during setup, we had $A_3 = 1$ and $B_1 = 0$ at some cycle following setup. We would expect C_3 to be 0 in this case, since B_1, which is routed to C_3, is 0. Since A_3 is 1, however, $\overline{C}_3$ is pulled down, and C_3 becomes 1. The requirement that A_3 and A_4 be 0 after setup eliminates spurious pulldowns.

3 The Hyperconcentrator Switch

In this section, we give the recursive construction for assembling merge boxes into a hyperconcentrator switch. We also show that a signal incurs exactly $2\lceil \lg n \rceil$ gate delays through the switch.

A hyperconcentrator switch with input wires $X_1, X_2, \ldots, X_n$, of which k contain messages, and output wires $Y_1, Y_2, \ldots, Y_n$ routes the k valid messages to the first k output wires $Y_1, Y_2, \ldots, Y_k$. Since valid messages are identified by a valid bit of 1 during setup, a hyperconcentrator switch may be viewed as a network that sorts 1's and 0's, with 1's before 0's in the output. The switch is set during setup, with subsequent bits following these established electrical paths.

We use recursive merging to sort the messages, solving the subproblems at each level of the recursion in parallel. By knowing in advance the size of the problem, we know in advance exactly how sets will be divided and merged. We can thus build the dividing process into the hardware and successively merge larger sets of bits through cascades of parallel merge boxes.

Figure 4 shows the organization of a 16-by-16 hyperconcentrator switch. There are four stages through which the bits cascade, from bottom to top in the figure. By this construction, an n-by-n hyperconcentrator switch, composed of $\lceil \lg n \rceil$ stages of combinational merge boxes, is itself a combinational circuit. Signals incur exactly $2\lceil \lg n \rceil$ gate delays in an n-by-n hyperconcentrator switch. During setup, the S switches in each merge box are computed and stored in registers. These switches establish electrical paths for messages in each merge box. Since there are no other switches between merge boxes, the S switches actually establish the paths through the entire hyperconcentrator switch. As in the individual merge boxes, message bits that enter after the valid bit follow the established paths through the hyperconcentrator switch.

The area of this n-by-n hyperconcentrator switch is $\Theta(n^2)$, which we show as follows. The area of a merge box of size m is $\Theta(m^2)$, since it contains $m(m+1)$ constant-size pulldown circuits and $m+1$ constant-size registers. The area of an n-by-n hyperconcentrator switch is then given by the recurrence

$$A(n) = \begin{cases} \Theta(1) & \text{if } n \leq 2 \text{ ,} \\ 2A(n/2) + \Theta(n^2) & \text{if } n > 2 \text{ .} \end{cases}$$

The $\Theta(n^2)$ term dominates the recurrence, so the solution is is $A(n) = \Theta(n^2)$.

Since the minimum clock period for the hyperconcentrator switch increases with the size of the switch, the clock period of a really large hyperconcentrator switch may be sufficiently long that other hardware using the same clock cannot operate at maximum speed. The clock period of the hyperconcentrator switch can be bounded by placing pipelining registers after every sth stage, for some constant s, letting messages propagate through s stages per clock cycle. A message then requires $(\lg n)/s$ clock cycles to pass through an n-by-n hyperconcentrator switch. The architecture of the hyperconcentrator switch makes the inclusion of pipelining registers a straightforward modification.

Figure 1 shows a chip containing a 16-by-16 hyperconcentrator switch, which was fabricated in 1985 using 4μm nMOS MOSIS design rules. In order to provide enough drive for the pulldown transistors of the next stage, the inverters following the NOR gates in each merge box are actually inverting superbuffers. The propagation delay through a 32-by-32 version of this circuit is under 70 nanoseconds in the worst case, an impressive figure especially in light of the conservative technology.

4 Domino CMOS Design

In this section, we examine issues that arise when we design the hyperconcentrator chip with a precharged methodology instead of the level-sensitive approach used in ratioed nMOS design. We will be concerned exclusively with domino CMOS, which serves as a good example of a precharged discipline.

We must take care with the inputs to domino CMOS gates. In domino CMOS design, the outputs of some gates are precharged high during a precharge phase $\overline{\phi}$. The pulldown circuit of such a gate must be open during the precharge phase to prevent the immediate discharging of the gate's output node. During an evaluate phase ϕ, the pulldown may optionally close, allowing the discharge of the gate's output. In a level-sensitive design methodology, such as ratioed nMOS, the pulldown circuit may close and then open again during a single clock phase, as long as the circuit is in its final state long enough for the logic to settle. In a domino CMOS design, however, if the pulldown circuit closes at any time during the evaluate phase, the output node may discharge. Even if the pulldown circuit later settles open during the same evaluate phase, the gate's output node incorrectly remains low. To avoid this phenomenon of premature discharging, domino CMOS circuits are designed with all precharged gate inputs *monotonically increasing*—having no 1-to-0 transitions—during the evaluate phase. (Readers desiring more information about domino CMOS design are referred to [5] and [18].)

The circuit resulting from the straightforward modification of the ratioed nMOS design to domino CMOS—adding n-channel evaluate transistors to each pulldown circuit and replacing the depletion mode pullups by p-channel precharge transistors—is not a well-behaved domino CMOS circuit during setup. It is well behaved during cycles following setup, however. The simple inverters at the NOR gate outputs cause each merge box output wire to implement a monotonically increasing function, since the outputs are each the OR of AND's of input values. Since the result of composing monotonically increasing functions is a monotonically increasing function, the entire hyperconcentrator switch is therefore a well-behaved domino CMOS circuit after setup.

Unfortunately, during setup not all the inputs to the merge boxes are computed by monotonically increasing functions. In particular, the switch settings S, defined by $S_i = A_{i-1} \wedge \overline{A}_i$, are not monotonically increasing. If we start with A_{i-1} and A_i at 0 and raise them each to 1, the value of S_i can go from 0 to 1 and then back to 0:

A_{i-1}	A_i	S_i
0	0	0
1	0	1
1	1	0

The problem is how to set the S values in the face of their apparent nonmonotonicity.

Referring to the schematic diagram in Figure 5, our solution is as follows. We of course include the p-channel precharge and n-channel evaluate transistors (shown with input ϕ). But the major design change from ratioed nMOS to domino CMOS is in the values assigned to the S wires during setup. Suppose once again that during setup we have the following valid bit values on the input wires:

$$\begin{aligned} A_1, A_2, \ldots, A_p &= 1 \ , \\ A_{p+1}, A_{p+2}, \ldots, A_m &= 0 \ , \\ B_1, B_2, \ldots, B_q &= 1 \ , \\ B_{q+1}, B_{q+2}, \ldots, B_m &= 0 \ . \end{aligned}$$

Then during setup, instead of setting only S_{p+1} to 1 (as in the ratioed nMOS design), we set

$$\begin{aligned} S_1, S_2, \ldots, S_{p+1} &= 1 \ , \\ S_{p+2}, S_{p+3}, \ldots, S_{m+1} &= 0 \ . \end{aligned}$$

We still load the registers, which we now name $R_1, R_2, \ldots, R_{m+1}$, only during setup, so that only R_{p+1} is 1, as in the ratioed nMOS version:

$$R_i = \begin{cases} \overline{A}_1 & \text{if } i = 1 \ , \\ A_{i-1} \wedge \overline{A}_i & \text{if } 1 < i \leq m \ , \\ A_m & \text{if } i = m + 1 \ . \end{cases}$$

After setup, we set the values $S_i = R_i$ for $1 \leq i \leq m+1$, so the S wires then take on the values stored in the registers during setup. Only S_{p+1} is 1, again just as in the ratioed nMOS design.

We shall now see why this construction works. We first look at setup. In the precharge phase $\overline{\phi}$ of setup, the diagonal wires $\overline{C}_1, \overline{C}_2, \ldots, \overline{C}_{2m}$ are charged high, so the output wires $C_1, C_2, \ldots, C_{2m}$ are all low. During the evaluate phase ϕ of setup, the wires $\overline{C}_1, \overline{C}_2, \ldots, \overline{C}_p$ are pulled down by $A_1, A_2, \ldots, A_p$, as in the ratioed nMOS design. Unlike the ratioed nMOS case, however, $S_1, S_2, \ldots, S_{p+1} = 1$ and $B_1, B_2, \ldots, B_q = 1$. If $q > 0$, then the wires $\overline{C}_1, \overline{C}_2, \ldots, \overline{C}_{p+q}$ (not just $\overline{C}_{p+1}, \overline{C}_{p+2}, \ldots, \overline{C}_{p+q}$) are pulled down by the S and B wires during the evaluate phase ϕ of setup. If instead $q = 0$, then no $\overline{C}$ wires are pulled down by S and B wires. The result is that during the evaluate phase of setup, the output wires take on the values

$$\begin{aligned} C_1, C_2, \ldots, C_{p+q} &= 1 \ , \\ C_{p+q+1}, C_{p+q+2}, \ldots, C_{2m} &= 0 \ , \end{aligned}$$

as desired. During evaluate phases after setup, the A, B, and S wires take on the same values as they do in the ratioed nMOS circuit, so the circuit again works correctly.

With the A and B inputs monotonically increasing during the evaluate phase, and with the R registers designed so that their outputs undergo no 1-to-0 transitions during the evaluate phase, each merge box is a well-behaved domino CMOS circuit. Since the output of each merge box is monotonically increasing during the evaluate phase and is the input of a merge box in the next stage, all we need for the entire hyperconcentrator switch to be a well-behaved domino CMOS circuit is for the A and B inputs to the first stage to be monotonically increasing during the evaluate phase.

Since bipolar technology supports a fast "wired-OR" capability, the emerging BiCMOS technology should also provide a suitable implementation medium for the hyperconcentrator design.

5 Applications

This section discusses applications of the hyperconcentrator switch. One application, the one for which the switch was designed, allows us to use the available clock period more efficiently in bit-serial routing networks, thus improving their performance. Another application is its use in building a superconcentrator switch. We also show how the hyperconcentrator switch can be used as a subcircuit in building larger switches that span multiple chips. Finally, we show how the switch can be used in a processor datapath to manipulate bits efficiently.

Improving Network Performance

We can replace small, simple switches in a bit-serial routing network by concentrator switches to successfully route more messages in a single clock cycle, thus using the available clock period more efficiently. The routing network switches we shall consider route valid messages either left or right, based on an address bit immediately following the valid bit. Such a routing scheme is used, for example, in a butterfly network. An address bit of 0 indicates that the valid message should be routed to a left output of a switch, and an address bit of 1 indicates that the valid message should be routed to the right.

Consider the 2-input, 2-output butterfly node shown in Figure 6. A single level of a routing network such as a butterfly would typically have several such nodes side-by-side. The node contains two simple 2-by-1 concentrator switches, depicted as trapezoids, one with outputs going left and one with outputs going right. Each simple concentrator switch is preceded by a selector circuit that, given an input valid bit and an address bit, produces a new valid bit that is 1 if and only if the input valid bit is 1 and the address bit matches the output direction of the concentrator switch. If two valid messages with equal address bits enter a butterfly node, only one is successfully routed.

The problem with this scheme is that it does not use the available clock period efficiently. The simple node uses only a few levels of logic, so the delay through it is only a few nanoseconds. But because of the large amount of time required to get signals on and off chips in current technologies, we might be unable to distribute a clock with a frequency high enough to match the short delay of this node. In fact, the clock period we can distribute is typically at least an order of magnitude greater than the delay through this node. This node therefore performs no useful work in at least 90 percent of each clock cycle.

Now consider the generalized n-input, n-output butterfly node shown for $n = 8$ in Figure 7. Like four simple butterfly nodes of Figure 6 laid side-by-side, it has a total of 8 input wires and 8 output wires, with 4 outputs going left and 4 outputs going right. But here we use two n-by-$n/2$ concentrator switches, one with outputs only going left and one with outputs only going right.

The advantage held by the larger node is that at the same clock speed as the simple nodes, it can successfully route more valid messages in each clock cycle. The clock speed remains the same because the additional delay introduced by the larger concentrator switches is just soaked up by the unused portion of the clock period. These nodes use a larger portion of the available clock period. Since the simple nodes leave so much of the available clock period unused, we can even scale these concentrator switches up considerably before the delay introduced exceeds the original clock period.

We shall now show that more valid messages are routed in a single clock cycle by the larger nodes. We assume that a valid message arrives at each input wire of each node and that the address bit is 0 with probability 1/2, independent of the address bits of other messages. We shall show that on average, the small node successfully routes only a constant fraction of the valid messages, but on average, the larger nodes, with n input wires, successfully route $n - O(\sqrt{n})$ valid messages. Intuitively, the larger nodes successfully route more valid messages because they have more freedom in mapping inputs to outputs.

First consider the 2-input, 2-output node of Figure 6. If the valid messages have unequal address bits, which occurs with probability 1/2, no valid messages are lost. If the address bits are equal, which also occurs half the time, one of the valid messages is lost. Since the switches act independently, the probability that a valid message is lost is 1/4, so we expect that 3/4 of the valid messages are successfully routed.

Now consider a routing network node like that of Figure 7, but with n inputs and n outputs. Suppose we have k valid messages with address bits of 0 (0-messages) and $n-k$ with address bits of 1 (1-messages). If $k > n/2$, then

$k - n/2$ of the 0-messages are lost and no 1-messages are lost. Conversely, if $k < n/2$, then $n/2 - k$ of the 1-messages are lost and no 0-messages are lost. Thus, when there are k 0-messages, there are $|k - n/2|$ valid messages lost. The number k of 0-messages is binomially distributed, with the probability of a valid message being a 0-message equal to $1/2$ and the expected number of 0-messages equal to $\mathrm{E}[k] = n/2$. The expected number of valid messages lost is $\mathrm{E}[|k - n/2|]$. To determine an upper bound on this value, we note that

$$\begin{aligned}
\mathrm{E}\left[|k - n/2|^2\right] &= \mathrm{E}\left[(k - n/2)^2\right] \\
&= \mathrm{E}\left[(k - \mathrm{E}[k])^2\right] \\
&= \mathrm{Var}[k] \\
&= n\left(\frac{1}{2}\right)\left(1 - \frac{1}{2}\right) \\
&= n/4 \ .
\end{aligned}$$

For any random variable X, we have the identity

$$\mathrm{Var}[X] = \mathrm{E}[X^2] - (\mathrm{E}[X])^2$$

which, combined with

$$\mathrm{Var}[X] \geq 0 \ ,$$

gives us

$$(\mathrm{E}[X])^2 \leq \mathrm{E}[X^2]$$

or

$$\mathrm{E}[X] \leq \sqrt{\mathrm{E}[X^2]} \ .$$

Thus,

$$\begin{aligned}
\mathrm{E}\left[|k-n/2|\right] &\le \sqrt{\mathrm{E}\left[|k-n/2|^2\right]} \\
&= \sqrt{n}/2 \ .
\end{aligned}$$

The expected number of valid messages lost is therefore $O(\sqrt{n})$ and thus the expected number of successfully routed valid messages is $n - O(\sqrt{n})$.

Superconcentrator Switches

Another application of hyperconcentrator switches is in building superconcentrator switches. An *n-by-n superconcentrator switch* has n input wires $X_1, X_2, \ldots, X_n$ and n output wires $Y_1, Y_2, \ldots, Y_n$. For any $1 \le k \le n$, disjoint electrical paths may be established from any set of k input wires to any arbitrarily chosen set of k output wires. Superconcentrator switches are useful in fault-tolerant systems. If some of the output wires of a concentrator switch may be faulty, we can use a superconcentrator switch that routes signals to only the good output wires.

Following the construction by Valiant [17], we can build a superconcentrator switch out of two full-duplex hyperconcentrator switches H_F and H_R, as shown in Figure 8. (After setup in a full-duplex hyperconcentrator switch, signals can travel along the established paths simultaneously in both forward and reverse directions. Extending the design of the hyperconcentrator switch to make it full-duplex is straightforward.) The output wires of the switch H_F (a "forward" hyperconcentrator switch) feed directly into the reverse input wires of the full-duplex hyperconcentrator switch H_R (a "reverse" hyperconcentrator switch). Suppose there are l good output wires of the superconcentrator switch. Before setup of the superconcentrator switch, the switch H_R sets up electrical paths from its first l reverse input wires $Z_1, Z_2, \ldots, Z_l$ to the l good reverse output wires. These paths are established by assigning a 1 to each forward input wire of the switch H_R that corresponds to a good output wire, assigning a 0 to the forward input wires corresponding to faulty output wires, and running a setup cycle of the switch H_R.

Setup of the superconcentrator switch is then just setup of the hyperconcentrator switch H_F. The k valid messages are routed through the switch H_F to the wires $Z_1, Z_2, \ldots, Z_k$ and then along the reverse paths through the switch H_R to the first k good output wires.

Building Large Switches

The hyperconcentrator switch can also be used as a building block in large concentrators. For example, replacing the comparators in an arbitrary sorting network by n-by-n hyperconcentrator switches yields a large hyperconcentrator. (Actually, only the first level of comparators must be replaced by hyperconcentrator switches; merge boxes suffice at all subsequent levels.)

We have also found that efficient multichip partial concentrator switches can be built from hyperconcentrator chips. An (n, m, α) *partial concentrator switch* has n inputs, m outputs, and a fraction α such that if there are k valid messages entering the switch, then

- If $k \leq \alpha m$, each valid message is routed to an output.
- If $k > \alpha m$, at least αm valid messages are routed to outputs.

A lightly loaded (n, m, α) partial concentrator switch is similar to an n-by-m concentrator switch, in that if the number of valid messages to route is at most αm, then *all* the valid messages are successfully routed to outputs.

One multichip partial concentrator switch construction [2, 3] is based on the Revsort two-dimensional mesh sorting algorithm of Schnorr and Shamir [15] and uses $3\sqrt{n}$ hyperconcentrator chips with with $\sqrt{n}$ inputs each. This construction yields an $(n, m, 1 - O(n^{3/4}/m))$ partial concentrator switch in three-dimensional volume $\Theta(n^{3/2})$. A signal incurs $3 \lg n + O(1)$ gate delays in passing through this switch.

Another such construction [3], based on Leighton's Columnsort algorithm [9], uses $\Theta(n^{1-\beta})$ hyperconcentrator chips with $\Theta(n^{\beta})$ inputs each, for any $1/2 \leq \beta \leq 1$. This construction produces an $(n, m, 1 - O(n^{2-2\beta}))$ partial concentrator switch in volume $\Theta(n^{1+\beta})$. A signal incurs $4\beta \lg n + O(1)$ gate delays in passing through this switch.

One might wonder how to build multichip hyperconcentrator switches. Partitioning the n-by-n hyperconcentrator switch presented in this paper among multiple chips with p pins each requires $\Omega((n/p)^2)$ chips, since each p-pin chip has area $O(p^2)$ and there are $\Theta(n^2)$ components to partition. We may want to partition a hyperconcentrator switch for two reasons:

1. The $\Theta(n^2)$ area may exceed the available chip area.

2. If the switch is to be packaged by itself on a chip, it may require more input and output pins than are provided by the packaging technology.

A different n-by-n hyperconcentrator switch design, consisting of a parallel prefix circuit and a butterfly network [2], can be built in volume $\Theta(n^{3/2})$

with $O(n \lg n)$ chips and as few as four data pins per chip, but this switch is not combinational. Although its sequential control is not very complex, it is not as simple as that of a combinational circuit.

We can build multichip hyperconcentrator switches by extending either of the above multichip partial concentrator switch designs [3]. By extending the Revsort-based design, we can build a multichip n-by-n hyperconcentrator switch that uses $\Theta(\sqrt{n} \lg \lg n)$ chips with $\Theta(\sqrt{n})$ pins each in volume $\Theta(n^{3/2} \lg \lg n)$, inducing $4 \lg n \lg \lg n + 8 \lg n + O(\lg \lg n)$ gate delays. An extension of the Columnsort-based design yields a multichip n-by-n hyperconcentrator switch that uses $\Theta(n^{1-\beta})$ chips with $\Theta(n^{\beta})$ pins each in volume $\Theta(n^{1+\beta})$ for any $1/2 \leq \beta \leq 1$. A signal incurs $8\beta \lg n + O(1)$ gate delays in passing through this switch.

Processor datapath

The hyperconcentrator design can also be used in a processor datapath to allow various manipulations on bits, including barrel shifting, APL-style compression and expansion, and bit interleaving. It requires at most a constant factor more area than a barrel shifter—which microprocessor datapaths typically include—and the hyperconcentrator provides more functionality. We now sketch how it can be used in a processor that uses n-bit words.

To accomplish a left shift of a word by k bit positions, we feed the hyperconcentrator switch with 2-bit "messages." The "valid bits" consist of k 0's followed by $n - k$ 1's. The "data" portion of the message is the word to be shifted. The bits that emerge from the switch in the cycle following setup form the desired output. A right shift can be accomplished by setting up a full-duplex switch similarly, but feeding the data bits through in the reverse direction. (Of course, we can modify the circuit to make it be entirely combinational if we wish, rather than having it require two cycles for each of these operations. Also, if the value k is provided in binary, we can use a decoder to produce the valid bits.)

In a parallel computer, a processor address may be split into physical and virtual bits, where the physical bits correspond to actual processors and virtual bits correspond to logical processors that share a single physical processor. When a two-dimensional array is mapped onto the address space, it is desirable to preserve locality by sharing the virtual bits equally between the x- and y-coordinates. Thus, we need to extract and deposit bit fields that are not necessarily contiguous. A hyperconcentrator-based datapath can facilitate such operations.

The idea is to use a hyperconcentrator switch to perform what in APL are called the compression and expansion operations. In compression, we are given a bit mask that tells which bits of a data word we are interested in. The compress operation extracts those bits from the data word and makes them contiguous in the leftmost bits of the output. For example, suppose that the bits representing the x-coordinate of an address are stored in the odd-indexed bit positions, so that the virtual part of the address is split evenly between x- and y-coordinates. To extract the x-coordinate from an address, we compress the address with a mask that contains 1's in the odd-indexed positions and 0's in the even-indexed positions. To extract the y-coordinate, we compress with the complement of this mask. A hyperconcentrator performs compression in much the same way as it shifts. The mask is provided as the valid bits, and the data bits are fed through in the next cycle.

Expansion, the opposite of compression, can be performed by using the reverse direction of a full-duplex switch. For example, to interleave the x- and y-coordinates in an address, we expand the x-coordinate into the odd-indexed positions, expand the y-coordinate into the even-indexed positions, and then OR the two results.

6 Conclusion

In this section, we describe a circuit containing the hyperconcentrator switch which has been fabricated. We also briefly discuss other applications of the switch. Finally, we pose some open questions.

We have implemented a 4μm nMOS 16-by-16 hyperconcentrator switch, which was fabricated by MOSIS in 1985. The chip, shown in Figure 1, contains programmable selector circuitry preceding the hyperconcentrator switch so that an independent routing decision can be made for each input, as in Figures 6 and 7. Each of the 16 selectors includes a UV write-enabled PROM cell, described by Glasser [4]. The bit value stored in each PROM cell is compared with an address bit in the input message to determine whether the message is going in the correct direction.

The approach of replacing many small routing nodes by fewer nodes with larger concentrator switches is used by the cross-omega network [7]. Part of the cross-omega network is based on a truncated butterfly network. Single wires of the butterfly network are replaced by bundles of 32 wires, and the simple butterfly network nodes are replaced by nodes like that of Figure 7, but with 32 inputs, 32 outputs, and two 32-by-16 concentrator switches. Fat-trees serve as another example of a class of routing networks that makes use of concentrator switches. The interested reader is referred to [6] and [11] for details. More recently, "multibutterfly" networks have been proposed [10, 16] that can effectively use variants on the hyperconcentrator switch.

An open question is for what functions $f(p)$ can we build an $(\Omega(f(p)), m, 1-o(p/m))$ partial concentrator switch, given chips with p pins and using only two stages of chips. The Columnsort-based partial concentrator construction, for example, gives us $f(p) = p^{2-\epsilon}$ for any $0 < \epsilon \leq 1$. Can we achieve $f(p) = \Omega(p^2)$? In general, how large a function $f(p)$ can we achieve with k stages?

It is natural to ask whether a simple design for a concentrator switch exists when we relax the constraint that all the valid messages arrive at the same time. A crossbar switch has the capability of allowing valid messages to come and go at any time, but switch setup can be expensive. It may be that a concentrator switch can be designed that allows new messages to be routed in batches while preserving old connections.

Acknowledgements

Thanks to Eric Tenenbaum and Forrest Thiessen of M.I.T. for their help in the nMOS implementation of the hyperconcentrator switch and to Chris Terman for his help in performing the timing analysis. We also thank Johan Hastad of M.I.T. for his help in simplifying the proof that the expected

number of messages lost by an n-input, n-output butterfly node is $O(\sqrt{n})$.

References

[1] Ajtai, M., Komlós, J., and Szermerédi, E. Sorting in $c \log n$ parallel steps. *Combinatorica* **3** (1983), 1–19.

[2] Cormen, T. H. Concentrator switches for routing messages in parallel computers. Masters thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1986.

[3] Cormen, T. H. Efficient multichip partial concentrator switches. Laboratory for Computer Science Tech. Memo. MIT-LCS-TM-322, Massachusetts Institute of Technology, Cambridge, MA, Feb. 1987.

[4] Glasser, L. A. A UV write-enabled PROM. *Proc. 1985 Chapel Hill Conference on VLSI.* Computer Science Press, Rockville, MD, 1985, pp. 61–65.

[5] Glasser. L. A., and Dobberpuhl, D. W. *The Design and Analysis of VLSI Circuits.* Addison-Wesley, Reading, MA, 1985.

[6] Greenberg, R. I., and Leiserson, C. E. Randomized routing on fat-trees. *Proc. 26th Annual IEEE Symposium on Foundations of Computer Science.* IEEE Computer Society, Washington, D.C., 1985, pp. 241–249.

[7] Knight, T. F. Unpublished manuscript.

[8] Knuth, D. E. *The Art of Computer Programming, Volume 3: Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.

[9] Leighton, F. T. Tight bounds on the complexity of parallel sorting. *IEEE Trans. Comput.* **C-34** 4 (Apr. 1985), 344–354.

[10] Leighton, F. T., and Maggs, B. M. Expanders might be practical: fast algorithms for routing around faults on multibutterflies. *Proc. 30th Annual Symposium on Foundations of Computer Science*, IEEE Computer Society, Los Alamitos, CA, 1989, pp. 384–389.

[11] Leiserson, C. E. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE Trans. Comput.* **C-34** 10 (Oct. 1985), 892–901.

[12] Masson, G. M. Binomial switching networks for concentration and distribution. *IEEE Trans. Comm.* , **COM-25** 9 (Sept. 1977), 873–883.

[13] Pinsker, M. S. On the complexity of a concentrator. *Proc. 7th International Teletraffic Conference*, 1973, pp. 318/1–318/4.

[14] Pippenger, N. Superconcentrators. *SIAM J. Comput.* **6** 2 (June 1977), 298–304.

[15] Schnorr, C. P., and Shamir, A. An optimal sorting algorithm for mesh connected computers. *Proc. of the 18th Annual ACM Symposium on Theory of Computing*, Association for Computing Machinery, New York, 1986, pp. 255–263.

[16] Upfal, E. An $O(\log N)$ deterministic packet routing scheme. *Proc. of the 21st Annual ACM Symposium on Theory of Computing*, Association for Computing Machinery, New York, 1989, pp. 241–250.

[17] Valiant, L. G. Graph-theoretic properties in computational complexity. *J. Comput. System Sci.* **13** 3 (Dec. 1976), pp. 278–285.

[18] Weste, N., and Eshraghian, K. *Principles of CMOS VLSI Design, A Systems Perspective*, Addison-Wesley, Reading, MA, 1985.

Figure captions

Figure 1. A 4μm nMOS chip containing a 16-by-16 hyperconcentrator switch. The recursive nature of the switch can be seen. The switch is preceded by programmable selector circuitry that allows an independent routing decision to be made for each input.

Figure 2. The paths taken by valid messages in a merge box. Valid bits are shown as they enter and leave the merge box. The p valid messages arriving at input wires $A_1, A_2, \ldots, A_p$ are routed to output wires $C_1, C_2, \ldots, C_p$ respectively. Here, the only A wires with valid messages are A_1 and A_2. These valid messages are routed to C_1 and C_2 respectively. The q valid messages arriving at input wires $B_1, B_2, \ldots, B_q$ head toward $C_1, C_2, \ldots, C_q$ but are steered to $C_{p+1}, C_{p+2}, \ldots, C_{p+q}$. Here, the valid messages entering through B_1, B_2, and B_3 are steered to output wires C_3, C_4, and C_5 respectively.

Figure 3. A merge box of size 8. The input wires are A_1, A_2, A_3, A_4 and B_1, B_2, B_3, B_4. The output wires are $C_1, C_2, \ldots, C_8$. The switch settings are stored during setup in registers S_1, S_2, S_3, S_4, S_5. Here, $p = 2$ and $q = 3$ during setup. The valid bit values on each A, B, and C wire are shown, as are the S switch settings. All conducting paths to ground are shaded.

Figure 4. A 16-by-16 hyperconcentrator switch with four stages of merge boxes. Individual merge boxes are oriented as in Figure 2, with input wires A entering the bottom left, input wires B entering the bottom right, and output wires C leaving the top left and top right. Messages flow from bottom to top. The output wires $C_1, C_2, \ldots, C_m$ of a merge box of size m are the input

wires of a merge box of size $2m$, either $A_1, A_2, \ldots, A_m$ or $B_1, B_2, \ldots, B_m$. Valid bit values are shown entering the first stage and leaving the last stage of the cascade. The electrical paths established within the merge boxes and the switch during setup are shown in heavy lines.

Figure 5. A domino CMOS merge box of size 8. During setup, the S wires have the values $S_1, S_2, \ldots, S_{p+1} = 1$, but the registers are set as in the ratioed nMOS design, with only register R_{p+1} receiving the value 1. After setup, the S wires get their values from the registers, so only $S_{p+1} = 1$, as in the ratioed nMOS case. The conducting paths to ground during the evaluate phase of setup are shaded for the case of $p = 2$ and $q = 3$.

Figure 6. A 2-input, 2-output butterfly node. The selectors and 2-by-1 concentrator switches ensure that both input messages reach output wires if their address bits specify that they are going in different directions. If the messages contend for the same output wire, the concentrator switches ensure that one makes it through. With randomly chosen address bits, we expect $3n/4$ of the n messages to be successfully routed through this node.

Figure 7. A generalized butterfly node with n inputs and n outputs, shown here for $n = 8$. There are two n-by-$n/2$ concentrator switches. With randomly chosen address bits, we expect $n - O(\sqrt{n})$ messages to be successfully routed through this node.

Figure 8. A superconcentrator switch built out of two hyperconcentrator switches. The hyperconcentrator switch H_R is set up the connect the first l reverse input wires $Z_1, Z_2, \ldots, Z_l$ to the l good reverse output wires, which

also serve as the output wires of the superconcentrator switch. The k valid messages are then routed by the hyperconcentrator switch H_F to the wires $Z_1, Z_2, \ldots, Z_k$ and through the switch H_R to the first k good output wires.

Author Biographies

Thomas H. Cormen received the B.S.E. degree in Electrical Engineering and Computer Science from Princeton University in 1978 and the S.M. degree in Electrical Engineering and Computer Science from M.I.T. in 1986. He is currently a Ph.D. candidate in EECS at M.I.T. He is coauthor of the textbook *Introduction to Algorithms*, which he wrote with Charles E. Leiserson and Ronald L. Rivest. His research interests focus on sequential and parallel algorithms and data structures.

Charles E. Leiserson received the B.S. degree in computer science and mathematics from Yale University, New Haven, CT, in 1975 and the Ph.D. degree in computer science from Carnegie Mellon University, Pittsburgh, PA, in 1981.

He is currently Associate Professor of Computer Science and Engineering at the Massachusetts Institute of Technology, Cambridge, MA. As a graduate student at Carnegie Mellon, he wrote the first paper on systolic architectures with his advisor H. T. Kung. His Ph.D. dissertation "Area-Efficient VLSI Computation," which deals with the design of systolic systems and with the problem of determining the VLSI area of a graph, won the first ACM Doctoral Dissertation Award. In 1981, he joined the faculty of the Theory of Computation Group in the Laboratory for Computer Science, Massachusetts Institute of Technology. With Ronald L. Rivest and Thomas H. Cormen, he authored the textbook *Introduction to Algorithms*, published by the MIT Press and McGraw-Hill. He has also authored over 25 papers on analysis of algorithms, VLSI theory, computer-aided design, and parallel algorithms and architectures. His principal interest is in the theoretical foundation of parallel computation, especially as it relates to engineering reality.

Prof. Leiserson is a member of the ACM, IEEE, and SIAM. In 1985 he received a Presidential Young Investigator Award from the National Science Foundation.