NOTE: This paper appeared as:
Murdocca, M. J., and V. Gupta, "Architectural Implications of Reconfigurable Optical Interconnects," *Journal of Parallel and Distributed Computing*, vol 17, no. 3, pp. 200-211, (Mar. 1993).

Architectural Implications of Reconfigurable Optical Interconnects

Miles Murdocca and Vipul Gupta

Department of Computer Science

Rutgers University, Hill Center

New Brunswick, NJ 08903

(908) 932-2654

murdocca@cs.rutgers.edu

ABSTRACT

We consider the influence of reconfigurable optical interconnects on computer architecture. A computer system is viewed as a sea of interconnected logic gates, in which the interconnects are dynamically reconfigured during operation to suit the computation being performed. We explore reconfiguration at the gate level, at the processing element (PE) level, and at the system level. A significant finding is that the conventional von Neumann model of a digital computer, may benefit from the use of reconfigurable interconnects at the gate level by implementing only the most recently used instructions in a *function cache*. By implementing only a subset of the instructions in a PE at any time, the physical size of the PE can be reduced and performance can be improved. The runtime behavior of programs executing on a uniprocessor is used as evidence that a function cache can be effective. Case studies are also provided for reconfiguration at the gate level and at the system level. The conclusion is made that reconfiguration can improve performance at all levels of the computer hierarchy, providing that the improvement in performance offsets the cost of reconfiguration.

1. INTRODUCTION

Optical computing technology has advanced to the extent that optical logic gates based on multiple quantum wells (MQWs) are commercially available, such as the S-SEEDs [1] through AT&T, and new surfaCe-Emitting Laser Logic devices (CELLs) [2] under development at Sandia National Laboratories and Photonics Research Incorporated (PRI), which are based on Heterojunction PhotoTransistors (HPTs) and Vertical-Cavity Surface-Emitting Lasers (VCSELs) [3]. An all-optical digital processor with feedback has been created with S-SEED devices at AT&T [4] and other S-SEED processors have been constructed at AT&T for photonic switching applications [5,6]. In general, optical logic gates are relatively large (a few microns in diameter) and are placed on a pitch (center to center spacing) on the order of tens of microns in order to facilitate cooling and to prevent coupling due to diffraction. This results in a low density of logic when compared with a conventional silicon based digital electronic technology. Thus, in order to achieve a comparable gate count to an electronic technology, the physical chips will be less dense for optical logic gates than for electronic logic gates. The volume consumed by free-space interconnects is another contributor to cost that can be ignored in initial calculations since device size and spacing are the dominant costs. In order to create optical computers of reasonable complexity then, it may be necessary to consider reconfiguring the gate-level interconnects to implement different functions as a computation proceeds.

The ability to change the interconnects as a computation proceeds is not a new concept. Reconfiguration takes many forms in conventional electronics, such as in multistage interconnection networks (MINs) and in modifying the microcode of a writable control store (WCS) for processors that use microcode. The use of free-space optics, however, offers two distinct advantages: (1) parallel access to the crosspoints, and (2) the ability to physically redirect a channel (through a beam-steering element, for example). The parallel access capability allows an entire interconnection pattern to be changed at the bit rate for some optical technologies, which is typically managed by sending signals to several channels simultaneously, and then masking out the unwanted channels. Since fan-out is naturally limited, this form of reconfiguration may work best when the circuits are small. The ability to redirect channels by steering beams of light offers a powerful new capability, but beam-steering methods usually introduce losses or reconfigure much slower than the bit rate,

and are therefore more appropriate for interconnects that can operate at high powers and change less frequently, as between PE's. Although these capabilities for reconfiguration exist in various forms, it is not intuitively clear that the cost of supporting reconfiguration is offset by a corresponding gain in performance. Here, we characterize the degree of improvement that is required in order to realize a payoff.

One potential improvement that reconfiguration offers that affects a broad range of processors is to tailor the size of the PE to match the nature of the computation being performed. Observations that most of the instructions that are executed in general purpose computers are simple (such as MOVEs) and involve few stack manipulations for subroutine linkage [7] motivated the development of reduced instruction set computers (RISCs) such as the SPARC and RISC II. A related observation is that approximately 90% of the execution time of a program is spent in just 10% of the code (the locality principle), and further, that only a fraction of the 10% is active for a given interval of time [8]. This observation led to the development of cache memories, in which the small fraction of code that is active is kept in a small fast memory that is local to the processor. The result is a significant improvement in performance for a small increase in cost.

In addition to locality of reference which applies to accesses to main memory, there is also *functional locality* which applies to instructions executing on a processor. The concept behind functional locality is that only a fraction of the hardware in a processor is used for a given interval of time, and that there is repetition within that interval. In support of this idea, we have taken measurements of instruction execution on running programs. What we have found is that even when a RISC processor with a small instruction set is used, that just a fraction of the available instructions are needed. For example, only 60 out of 180 available instructions might be used in a typical program. This is a previously known result. What we report here is that if we consider implementing just a few instructions at a time, say for example, 50% of the total number of different instructions needed in a program, then we can often achieve a hit ratio that is much greater than 50%, ranging between 85% and 97% for the samples we have taken. We take our measurements by generating a trace of instructions in the order that they execute, and then sliding a window of size

N over the trace, which represents the *N* instructions that are simultaneously implemented in the processor at a given time, and then recording how many times an instruction is needed that is not currently implemented. The interconnection pattern for the instruction that generates the miss is then brought into the processor, at the expense of the interconnection pattern for some other instruction. As a result of our findings, we propose the development of a *function cache* which keeps the most recently used hardware in the form of interconnection patterns in a high speed section of a processor.

In order for a function cache to be effective, there is a need for a technology that is capable of reconfiguring the interconnects, and for this reason we turn to free space optics. For the study reported here, we abstract away the physical design of the processor, and focus on those aspects of design that pose fundamental limitations, such as fan-out constraints and propagation limits due to diffraction.

2. RECONFIGURATION AT THE GATE LEVEL

The basic optical computing model considered here is shown in Figure 1, which consists of arrays of optical logic gates interconnected in free space. Binary 1's and 0's are represented as intensities of light beams. The gate level interconnects are assumed to have a regular pattern such as a perfect shuffle or a crossover [9]. This type of interconnect is used experimentally in a number of AT&T optical processor testbeds [5, 6]. The reason for using regular interconnects at the gate level is to allow the beams to share the same field of a single lens. If we choose to use irregular interconnects instead, then each channel requires a separate imaging system. This can be achieved through holography or through micro-optic fabrication techniques, but the small diameters of the resulting lenses limit propagation to 1mm or less [10] due to diffractive coupling between neighboring channels. The resulting interconnection problem is similar to limiting chip-to-chip connections in the electronic domain to only 1mm. The geometries at these dimensions are too difficult to work with using current optical technology, and so we make use of regular gate-level interconnects to relax the constraint on propagation distance.

The model shown in Figure 1 is composed of alternating arrays of optical logic gates and free-space regular interconnects. Masks in the image planes block light at selected locations which customize the interconnects to perform specific logic functions. The system is fed back onto itself and an input channel and an output channel are provided. Feedback is imaged with a single row vertical shift so that data spirals through the system, allowing a different section of each mask to be used on each pass. Optical signals travel orthogonal to the device substrates.

An optical logic gate

A number of optical logic gates have been proposed and constructed [1-3, 11, 12]. Here, we consider an optical logic gate that matches the model shown in Figure 1 particularly well. The CELL device [2] is composed of vertical-cavity surface-emitting lasers (VCSELs) that are controlled by integrated heterojunction phototransistors (HPTs). VCSELs have been fabricated and demonstrated in two-dimensional arrays, and CELLs are in the process of being fabricated at Sandia National Laboratories and at Photonics Research Inc. (PRI). Figure 2 shows an array of VCSELs. The devices vary from 1.5μm to 5μm in diameter and are about 6μm in height. Depending on their construction, the lasers can emit light at different wavelengths. Emission at 960nm is a desirable property since the GaAs substrate is transparent at this wavelength, which allows for detector/laser pairs which comprise the CELL to be monolithically integrated.

The microlaser operates by electrically energizing atoms in the amplifying portion. A small amount of light is generated through spontaneous emission, and the light wave traveling through the amplifying medium interacts with an energized atom. Stimulated emission occurs, and the atom converts its energy to light at the same wavelength as the traveling wave. Partially reflective mirrors at the ends of the microlaser allow only a fraction of the light to pass, while the remainder of the light remains within the cavity to continue the process of amplification.

The structure of the CELL device is shown in Figure 3. The CELL operates by allowing a low intensity signal on the HPT to create a photocurrent which is electrically amplified to a level large

enough to drive the microlaser above threshold. Thus an optical-in/optical-out device is created which combines desirable attributes of both transistors and lasers. Fully developed CELLs are expected to operate at speeds in the range of several hundred MHz to a few GHz, although this is not yet fully demonstrated.

A gate-level optical interconnect

Figure 4 shows a schematic diagram of the optical crossover interconnect [9] which is used in a number of the AT&T testbeds [5, 6]. An array of input beams is split into two identical copies. One copy is imaged onto a mirror and is reflected back through the system to the output plane, while the other copy is permuted according to the period of the prism array. The combined copies are displaced slightly with respect to each other so that each copy is independently masked in the output plane. The gate-level interconnection pattern that this interconnect achieves is shown for varying periods of the prism arrays in the right side of the figure.

The crossover, perfect shuffle, and banyan interconnects share a property called *isomorphism*, which means that they can be mapped between each other as shown in Figure 5. One network can be mapped onto another by replacing corresponding connection patterns and relabeling nodes. Labeling of nodes is indicated in the figure by numbers in the range 0-7, in which the perfect shuffle serves as the reference network. One way to think about mapping one network onto another is to move nodes while keeping the connections attached, in effect, rubber banding the connections. More specifically, if node 0 in the perfect shuffle receives inputs from nodes 0 and 4 in the previous stage, then node 0 in the banyan and crossover also receive inputs from nodes 0 and 4 in their previous stages. The operations inside of the nodes remain unchanged when one network is mapped onto another. For the discussion here, the nodes represent logic gates with fan-ins and fan-outs of two.

Optical logic circuits

Unlike conventional digital design for electronic circuits in which the placement of wires is abstracted away during the design phase, the object of digital design for the model shown in Figure

1 is to map arbitrarily complex digital circuits onto the regular structure. The positions of the logic gates are fixed, all gates have fan-in and fan-out of two, all devices on the same array perform the same function such as OR or NOR, and the interconnects between arrays have a regular structure such as the crossover. The only choices available to the circuit designer are the positions of the inputs, the positions of the outputs, and the configurations of the masks that block unwanted connections. The general approach to digital design in this domain is to implement programmable logic arrays (PLA's) by first generating minterms, and then selecting and combining the minterms that are needed to implement specific functions [13, 14]. A dual-rail logic system is adopted since the strict regularity does not support a relative inversion, because every signal travels through the same number of logic gates of the same type. Various optical schemes can eliminate the need for this added cost, such as exchanging pairs of output beams for complementary optical logic devices.

An example of a simple 3-to-8 dual-rail decoder is shown in Figure 6. The inputs are *x*, *y*, and *s*, and the outputs are shown as the eight minterms at the bottom. The circuit is implemented in one pass through the model shown in Figure 1. If the circuit was any deeper, then a second pass through the system would be needed. Lightly shaded connections are disabled by configuring fixed masks in the output planes of the crossover interconnects. Circuit breadth and depth are relatively small when compared with more irregular interconnection schemes using fan-in/fan-out of two as argued in Ref. [13], so that although gate count is somewhat high, the gate count is not so high as to offset the practical gains in simplifying the optical interconnects.

Given that we can manage the complexity of designing digital circuits while maintaining regular interconnects at the gate level, we now consider taking the design further by reconfiguring the gate-level interconnects during operation. The model shown in Figure 1 is customized by selectively blocking beams with etched patterns on a glass substrate. There are various methods, however, in which the pattern can be changed dynamically. An example is the use of ferro-electric liquid crystal (FLC) devices [12], in which the mask patterns can be written similar to the way that an image is produced on a liquid crystal television screen. The reconfiguration time is relatively long (on the order of microseconds) in comparison with the speed of light propagation through the masks, and

so reconfiguration should be a relatively infrequent operation. Another example of a reconfigurable device is the matrix addressable microlaser array which is being considered for development at PRI, in which selected sets of microlasers are enabled through electrical signals at the edges. The configuration of the PRI device array is shown in Figure 7. Devices that are located at the crosspoints of active rows and columns are enabled. The advantage of this configuration is that for an N^2 increase in the size of an array, the bonding pad complexity increases by only $2N$, which allows for a simplified electronic interface. A disadvantage is that the computer designer loses a degree of freedom in selecting combinations of logic gates to enable or disable. For example, in Figure 7, there is no combination of active rows and columns that will generate a checkerboard pattern. Despite the limited number of possible on/off combinations for a matrix addressable array, the available complexity is considered sufficient for general purpose computing. Evidence that this may be the case is provided by Murdocca's previous work on using regular interconnects for optical logic circuits [14] in which interconnection topologies are severely restricted when compared with an electronic approach. Although design flexibility is sacrificed, the complexity of the optics and the complexity of the electronic addressing are simplified, which are currently more important considerations.

Given that technologies exist, or are under development for dynamically modifying the interconnects, we consider two examples of how reconfiguration at the gate level can improve performance.

2.1 Reconfiguration within an arithmetic logic unit

Although most of a computer is idle most of the time in a von Neumann style architecture, this is due almost entirely to the large transistor count devoted to random access memory (RAM). RAM is very dense, and consumes a small fraction of computer volume even though it accounts for the bulk of the active components. It is typical that when a computer adds, it does not subtract, nor does it perform a division at the same time. Thus the argument might be made that most of the central processing unit (CPU) and not just the memory is idle for most of the time. This, however, is not true in general. Consider the truth table and logic diagram for the 16-function 74181

arithmetic logic unit (ALU) [15] shown in Figures 8 and 9. Although only one function is selected at a time, there is a great deal of sharing of logic among the 16 functions. Nearly every logic gate is used for the ADD operation, as it is for the remaining functions. Thus, most of a CPU may not be wasted as an observer might initially conclude. However, although most of the logic gates in an ALU may be used most of the time, most of their functionality is underutilized most of the time. That is, a four-input AND gate may need only two of its inputs at any time, so that although the gate is still being used, it introduces a cost factor of two in circuit area since it is underutilized. Consider the shaded portion of Figure 9, which indicates the underutilized logic for the ADD operation. There are 63 logic gates, of which only 15.78 are used for the ADD operation as indicated in the figure, so 15.78/63 = 25% of the 63 logic gates are needed. The first two of the six gate levels (counting from the left, and ignoring inverters and the rightmost AND gate which is not used for the ADD) are not needed since they are devoted to function decoding which can be eliminated, since a reconfigurable approach can simply project the proper control patterns into the system. This reduces circuit latency by 1/3 in addition to reducing gate count.

2.2 Reconfiguration within a memory

A potential opportunity for reconfigurable interconnects is the application of Gaussian elimination to the solution of linear equations [16]. The process is data-independent if the problem of pivoting, which involves rearranging rows so that the top left element is relatively large, is ignored. However, in the real world, zeros or very small numbers do in fact appear on the diagonal, so that the pivoting problem must be addressed through interchanging rows.

A reconfigurable interconnection technology can offer a solution without compromising performance severely by simply reconfiguring the decoder section of the memory which stores the rows of the matrix. For example, consider the augmented coefficient matrix shown in Figure 10 for three linear equations in three unknowns. The indices in the upper left corner of the 12 cells indicate the addresses of the memory locations that store the corresponding coefficients.

Figure 11 shows two decoder circuits for the memory that map four-bit addresses into spatial locations. The circuit on the left is a conventional decoder that maps addresses into locations according to the matrix layout shown in Figure 10. The circuit on the right shows the configuration of a decoder that swaps the top and bottom rows of the matrix by changing the crosspoint settings through some reconfigurable interconnection approach. Notice that the actual data has not moved. Only 1/6 of the crosspoints are changed between the two forms, even though 3/4 of the elements are interchanged. Further, for a modest word size of 32 bits, the total number of bits that are effectively interchanged are $3/4 \cdot 12 \cdot 32$ bits = 288 bits even though only 16 crosspoints are changed in the decoder. An important property of this approach is that the modified decoder is simply projected into the system without regard for the actual data being interchanged, so that explicit data paths between all possible pairs of rows that might be interchanged do not need to be provided. This effect is more greatly pronounced for complex interchange operations such as a transpose, in which every element is affected. A single pattern that is imaged into the decoder in a single step is all that is required to implement the transpose. Thus, the performance advantage of using pointer addressing which is normally used can be achieved without paying the increased delay penalty that is associated with a fixed interconnect approach in an electronic implementation.

3. RECONFIGURATION AT THE PROCESSING ELEMENT LEVEL

We introduce the notion of *functional locality* and claim that a typical computer program displays locality in terms of the kinds of instructions it executes, *i.e.* a program is likely to execute an instruction from the small set of instructions that it executed most recently.

Just as spatial and temporal locality exhibited by a program are utilized to speed up effective memory reference delays using memory caches, we believe that functional locality makes a strong case for the introduction of *function caches*. Reconfigurable processors that execute only a small set of machine instructions at any given instant but at rates faster than non-reconfigurable processors can exploit functional locality to achieve higher performance. Our belief that such a reconfigurable processor will execute its instructions faster than a comparable non-reconfigurable

microprocessor is based on the design guideline that *smaller hardware is faster* [7]. Evidence that functional locality exists is given in the remainder of this section in the form of a study of instruction usage in sample programs.

Measurements of instruction set usage

Hennessy and Patterson [7] report on the instruction set usage for a number of application programs running on different architectures, and one of their conclusions is that programs use only a small part of the total instruction set provided by the architecture, and that an even smaller set of instructions (about twelve or so) account for as much as eighty percent of the total number of instructions executed. This observation motivates us to look for functional locality in programs. Our study is carried out in two parts. We look first at the extent of functional locality that arises solely from the fact that some instructions are executed more often than others. In this part of the study, using the run-time frequency count information collected in the instruction usage study reported in Ref. [7], we synthesized random runs with uniform frequency distributions of machine instructions matching those reported, and studied the hit-ratio that a reconfigurable processor would achieve for different sizes of the function cache using a first-in-first-out (FIFO) instruction replacement strategy. The reader is reminded here that it is the actual hardware that is being replaced, and not simply the codewords that represent instructions. This part of the study uses two architectures – the DEC VAX, and the DLX, which is a generic Load/Store architecture described in Ref. [7]. Three different programs are used on each machine: **gcc** (a C compiler), **spice** (a circuit simulator), and **tex** (a text formatter).

Figure 12 shows a plot of hit-ratio against code size for the DLX running **gcc** for different function cache sizes. Plots for **spice** and **tex** for the DLX, and also for these three programs on the VAX are nearly identical in form to Figure 12. For our purposes, the hit-ratio is the percentage of the total instructions executed for which reconfiguration is required assuming that at any given instant, the processor implements only as many instructions as the size of the function cache allows and reconfigures itself to implement an instruction that is not in the function cache. In each case the hit-ratio increases as the function cache size increases but is almost completely insensitive to code size.

For this reason, we use a code size of no more than 1,000,000 machine instructions for the measurements that follow.

Figures 13 and 14 show plots of hit ratios as functions of cache sizes for the DLX and the VAX, respectively, for synthesized runs based on the instruction mixes found in **gcc**, **spice**, and **tex**. As shown in the plots, high hit ratios are obtained for small cache sizes. Motivated by these results, we developed software tools for the second part of the study, which allowed us to gather entire runs of some sample programs. The architecture used for this part of the study is the SPARC based Sun-4 and the programs studied are **latex** and the **gcc** components: **gcc-cpp**, **gcc-cc1**, **as** and **ld**. Collecting a program trace in this fashion slows down the program being traced by a large factor. For example, one trace of seven million instructions required nine hours of actual time. For this reason, the programs were executed using small sample files. Figure 15 shows the effect of changing the size of the function cache on the hit-ratio for different programs. For the programs shown in Figure 15, we also gathered data on the run-time frequency distribution, generated runs with matching frequency distributions and studied the effect of function cache size on hit-ratio. Our observations are shown in Figure 16. Note that the hit-ratio values we see in Figure 15 are higher than corresponding values seen in Figure 14 for the same function cache size. This indicates that the programs in our study exhibit functional locality to a higher degree than would be exhibited simply because programs execute some instructions more often than others.

Discussion

The data we have collected provides evidence for the existence of functional locality in programs. Based on the evidence, we claim that a reconfigurable processor that modifies its hardware to execute a slowly changing set of machine instructions can exploit functional locality to achieve higher performance than a non-reconfigurable processor. In order to quantify the performance gain, we define α (> 1) as the ratio by which the execution of an instruction which is not in the function cache is slowed down compared to the execution of an instruction in the cache. The factor by which a reconfiguring multiprocessor is slowed down because of misses is then given by $slowdown = h + \alpha (1 - h)$ where h is the hit-ratio. If α (< 1) is the ratio of the speed with which

the reconfigurable processor executes an instruction that it finds in its cache to the speed at which a non-reconfigurable processor executes an instruction, then in order for the reconfigurable processor to be faster than the non-reconfigurable processor, α should be less than $\frac{1}{slowdown}$. From Figure 17, it is clear that the higher the hit-ratio, the lesser the sensitivity of the slowdown factor to the cost of reconfiguration. We choose a sample point $\alpha = 5$, based on the expected reconfiguration time of the PRI matrix addressable devices as compared to the bit rate, and choose h near 0.8 which is typical for execution runs we have observed. This sample point gives us a slowdown of 2, which means that the processor runs twice as slow as a result of misses than it would run if there were no misses at all. A reconfigurable processor is assumed to be faster than a non-reconfigurable processor as a result of its reduced size, however, and so the speedup must compensate for the slowdown. The operating region in which a reconfigurable approach breaks even is shown for this sample point in Figure 18. In order for the reconfigurable processor to break even, it must execute instructions at twice the rate of a non-reconfigurable processor.

4. RECONFIGURATION AT THE SYSTEM LEVEL

In the previous sections, we explored reconfiguration at the gate level and at the instruction level. In this section, we explore reconfiguration at the system level for one particular example. A case study is provided for a matrix-vector multiplication architecture, in which the precision of computation and the size of the matrix are varied while time and space constraints are held fixed.

Consider the general form for the matrix-vector computation $A\mathbf{x} = \mathbf{b}$:

$$\begin{bmatrix} A_{0,0} & \cdots & A_{0,N-1} \\ \vdots & & \vdots \\ A_{N-1,0} & \cdots & A_{N-1,N-1} \end{bmatrix} \begin{bmatrix} x_0 \\ \vdots \\ x_{N-1} \end{bmatrix} = \begin{bmatrix} b_0 \\ \vdots \\ b_{N-1} \end{bmatrix}$$

A solution for **b** can be obtained by evaluating the expressions:

$$\begin{aligned} b_0 &= A_{0,0}x_0 + A_{0,1}x_1 + \cdots + A_{0,N-1}x_{N-1} \\ &\vdots \\ b_{N-1} &= A_{N-1,0}x_0 + A_{N-1,1}x_1 + \cdots + A_{N-1,N-1}x_{N-1} \end{aligned}$$

Each b_i requires N multiplications and N-1 additions. There are no dependencies among the equations, and there are no dependencies among multiplications within an equation, so that all multiplications can execute in parallel. Thus, the time to complete all of the multiplications is the same as the time to complete a single multiplication, but the additions require $\lceil \log_2(N) \rceil$ steps for each row if a simple adder tree is used in summing each b. The space complexity is: N^2 □ (size of multiplier) + N □ (N - 1) □ (size of adder) since there are N -1 adders in an adder tree of depth $\lceil \log_2(N) \rceil$.

The sizes of the multipliers and adders will vary depending on the design choices that we make. For this example, consider using a 32-bit parallel pipelined multiplier that has the form shown in Figure 19 [17]. A 32-bit multiplicand enters at the top and a 32-bit multiplier enters at the right. The 64-bit output is produced in systolic fashion at the bottom and to the right. An array of 32 wide by 33 deep partial product (PP) elements implement the multiplication. Only the low order 32 bits of the 64 bit result (p_0 - p_{31}) are used here, and the high order 32 bits are discarded, which is indicated in the highlighted region. Thus, only (32·32)/2 = 512 partial product elements need to be implemented. Since the upper 32 bits are discarded, there is no need to perform sign extension of the multiplicand or multiplier, so that the multiplier works correctly for both signed and unsigned arithmetic. Each partial product element has depth = 3 and area = 5 (these are sample figures taken from an unspecified design). The precision is P = 32 bits, and so the latency for one multiplication is P □ 3 and the space for the entire multiplier is (P^2/2) □ 5.

In order to maintain the pipelined nature of the architecture, the adder unit must also be pipelined. For the purpose of simplicity, consider the form for a 32-bit pipelined ripple-carry adder shown in Figure 20. Although it is a somewhat wasteful approach in terms of space, our goal here is to show how precision and matrix size can be traded for each other through reconfigurable interconnects, rather than focus on specific methods of design. The depth for one of the adder PLA's is 1 and the area is 3 (again, these are sample figures taken from an unspecified design). The latency for one

addition then is $P \times 1 = P$ and the space for the entire adder is $(P^2 \times 3)$. The general architecture now looks as shown in Figure 21.

We can now define the total parallel time PT and the total space S of the architecture, in terms of N and P:

Total parallel time PT = (time for one multiplication) + (time for $\log_2 N$ additions)

$$= P \times 3 + \lceil \log_2(N) \rceil \times P.$$

Total space $S = N^2 \times$ (size of multiplier) $+ N \times (N - 1) \times$ (size of adder)

$$= N^2 \times P^2/2 \times 5 + N \times (N - 1) \times P^2 \times 3.$$

Next, we choose values for N and P and compute PT_0 and S_0. For this example, we use $N_0 = 8$ and $P_0 = 32$.

$$PT_0 = P_0 \times 3 + \lceil \log_2(N_0) \rceil \times P_0$$
$$= 32 \times 3 + \lceil \log_2(8) \rceil \times 32$$
$$= 192.$$

$$S_0 = (N_0)^2 \times P^2/2 \times 5 + N_0 \times (N_0 - 1) \times (P_0)^2 \times 3$$
$$= 8^2 \times 32^2/2 \times 5 + 8 \times (8 - 1) \times (32)^2 \times 3$$
$$= 335{,}872.$$

Now suppose that we need to increase the matrix size from 8 to 16, and we are willing to sacrifice precision in order to maintain the same space requirements and the same parallel time. We need to find a value for P_1 when $N_1 = 16$ and $S_1 \leq S_0$, while maintaining $PT_1 \leq PT_0$. The calculation is shown below.

$$S_1 = (N_1)^2 \times (P_1)^2/2 \times 5 + N_1 \times (N_1 - 1) \times (P_1)^2 \times 3 \leq 335{,}872$$
$$= (16)^2 \times (P_1)^2/2 \times 5 + 16 \times (16 - 1) \times (P_1)^2 \times 3 \leq 335{,}872$$

$= 1360(P_1)^2 \leq 335{,}872$

thus $\lceil P_1 \rceil = 15$.

Now check that $PT_1 \leq PT_0$.

$$PT_1 = P_1 \times 3 + \lceil \log_2(N_1) \rceil \times P$$
$$= 8 \times 3 + \lceil \log_2(8) \rceil \times 8$$
$$= 48 < PT_0.$$

So when we double N we approximately halve the precision. The parallel time for the new matrix-vector multiplier is 1/4 less than the budgeted parallel time of 192. Thus, we might choose a different multiplier and adder in order to increase the precision at the expense of parallel time if we choose to squeeze the most performance out of the available budget. We do not take the example further here, but simply draw attention to the fact that this capability is made possible when interconnects are reconfigured at all levels of the computer hierarchy.

5. CONCLUSION

We reported on the influence of reconfigurable optical interconnects at various levels in the computer hierarchy. We identified the principle of functional locality and suggest that by using a reconfigurable interconnection technology to implement a function cache, we may potentially be able to achieve the functional complexity of a large instruction set processor such as a 68000 or a VAX, with the small size and high speed characteristics of a RISC.

A remaining problem is in how to implement the reconfiguration technology. We can imagine using electronic programmable logic arrays (PLAs) that can be reconfigured on demand, but we are then limited by the interconnection patterns that connect the PLAs since fabricated wires cannot be

moved on demand. An optical approach, possibly using reconfigurable masks under programmed control, may provide the needed mechanism.

The work reported here is jointly supported by the Air Force Office of Scientific Research and the Office of Naval Research under grant N00014-90-J-4018. The initial exploration that led to the function cache work was supported by the Strategic Defense Initiative Office under contract F49620-91-C-0055, which was administered through the Air Force Office of Scientific Research.

6. REFERENCES

[1] Lentine, A. L., H. S. Hinton, D. A. B. Miller, J. E. Henry, J. E. Cunningham and L. M. F. Chirovsky, "The symmetric self electro-optic effect device," in *Conference on Lasers and Electro-optics*, Technical Digest Series 1987, vol. 14, (Optical Society of America, Washington, D.C., 1987, 249), postdeadline paper.

[2] Olbright, G. R., R. P. Bryan, K. L. Lear, T. M. Brennan, Y. H. Lee, and J. L. Jewell, "Cascadable laser logic devices: discrete integration of phototransistors and surface-emitting lasers," *Electron. Lett.,* (Feb. 1991).

[3] Jewell, J. L, A. Scherer, S. L. McCall, Y. H. Lee, S. J. Walker, J. P. Harbison and L. T. Florez, "Low threshold electrically-pumped vertical cavity surface-emitting microlasers," *Electron. Lett.*, **25**, 1123-1124, (1989).

[4] Prise, M. E., N. C. Craft, M. M. Downs, R. E. LaMarche, L. A. D'Asaro, L. M. F. Chirovsky, and M. J. Murdocca, "An Optical Digital Processor Using Arrays of Symmetric Self-Electrooptic Effect Devices," *Applied Optics*, **30**, no. 11, (Apr. 10, 1991).

[5] F. A. P. Tooley, T. J. Cloonan, and F. B. McCormick, "Use of retroreflector arrays to implement crossover interconnections between arrays of S-SEED logic gates", *Optical Engineering*, **30**, pp. 1969-1975, (Dec. 1991).

[6] F. B. McCormick, F. A. P. Tooley, J. L. Brubaker, J. M. Sasian, T. J. Cloonan, A. L. Lentine, R. L. Morrison, R. J. Crisci, S. L. Walker, S. J. Hinterlong, and M. J. Herron, "Optomechanics of a free-space photonic switching fabric," *Proc. SPIE*, **1533**, paper 12, (1991).

[7] Hennessy, J. L. and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, (1990).

[8] Przybylski, S. A., *Cache and Memory Hierarchy Design*, Morgan Kaufmann Publishers, (1990).

[9] Jahns, J. and M. J. Murdocca, "Crossover networks and their optical implementation," *Appl. Opt.*, **28**, 182, (Aug. 1, 1988).

[10] Smith, D., M. Murdocca, and T. Stone, "Free-Space Optical Interconnection," book chapter in *Optical Computing Hardware*, edited by J. Jahns and S. Lee, Academic Press, (1992, in press).

[11] A. McAulay, *Optical Computer Architectures*, John Wiley & Sons, pp. 86-89, (1991).

[12] Lang, R., K. Kasahara, and M. Sakaguchi, "VSTEP and its applications," *Proc. 1990 International Topical Meeting on Optical Computing,* April 8-12, Kobe, Japan, 365, (1990).

[13] Murdocca, M. J., A. Huang, J. Jahns, and N. Streibl, "Optical Design of Programmable Logic Arrays," *Applied Optics*, **27**, pp. 1651-1660, (May 1, 1988).

[14] Murdocca, M., *A Digital Design Methodology for Optical Computing*, The MIT Press, (1990).

[15] Texas Instruments Incorporated, *The TTL Data Book*, 2nd ed., (1976).

[16] Murdocca, M. J. and S. Levy, "Design of a Gaussian Elimination Architecture for the DOC II Processor," *Proc. of the 1991 O-E/Lase Symposium*, (Jul. 1991).

[17] Hamacher, V. C., Z. G. Vranesic and S. G. Zaky, *Computer Organization*, McGraw-Hill, (1978).

FIGURE CAPTIONS

Figure 1: *Arrays of optical logic gates are interconnected with optical crossovers [9]. Masks in the image planes block light at selected locations which customize the system for specific functions.*

Figure 2: *Scanning electron micrograph of a small portion of an array of vertical-cavity surface-emitting lasers [3]. (Photograph made by Axel Scherer, Bellcore, and provided by the courtesy of Jack Jewell, Photonics Research Inc.)*

Figure 3: *Schematic diagram showing the structure of the CELL. (Illustration provided by the courtesy of Jack Jewell, Photonics Research Inc.)*

Figure 4: *Optical crossover interconnect. A two-dimensional array of input beams is split into two identical copies. One copy is imaged onto a mirror and is reflected back through the system to the output, while the other copy is imaged onto a prism array that permutes the beams according to its period. Connection paths achieved with different prism array periods are shown in the right panel.*

Figure 5: *Isomorphism among the perfect shuffle, banyan, and crossover interconnects.*

Figure 6: *A 3-to-8 decoder is implemented in an OR-NOR / OR-NOR network. Lightly shaded connections are blocked by fixed masks in the output planes of the crossover interconnects. Flow of signals is from the top to the bottom.*

Figure 7: *A matrix addressable array of devices requires only a linear growth in the number of bonding pads for a power of two growth in the number of devices. The indicated pattern is selected by enabling rows 2, 3, and 5 and enabling columns 3, 4, and 7.*

Figure 8: *Truth table for the 74181 4-bit ALU.*

Figure 9: *Modified version of the 74181, showing the amount of underutilized logic (indicated by shading). Fractions indicate the percentage of gate inputs that are used for the ADD operation.*

Figure 10: *An augmented matrix is shown for a system of linear equations in three unknowns. Indices in the upper left corners of cells indicate the storage locations in a random access memory.*

Figure 11: *Two forms are shown for a four-variable address decoder. The configuration on the left corresponds to the cell numbering shown in Figure 10. The configuration on the right corresponds to a cell numbering in which the top and bottom rows are interchanged. Only 1/6 of the crosspoints are changed between the two forms, even though 3/4 of the elements are interchanged.*

Figure 12: *Hit ratio versus code size for different sizes of the function cache for synthesized runs of gcc on the DLX.*

Figure 13: *Hit ratio as a function of the size of the function cache for synthesized runs on the DLX.*

Figure 14: *Hit ratio as a function of the size of the function cache for synthesized runs on the VAX.*

Figure 15: *Hit ratio as a function of the size of the function cache for actual runs on the SPARC. The solid line that starts higher is for* ***ld****.*

Figure 16: *Hit ratio as a function of the size of the function cache for synthesized runs on the SPARC. The solid line that starts lower is for* ***ld****. These synthesized runs have the same frequency distribution of machine instructions as the actual runs corresponding to Figure 15.*

Figure 17: *Slowdown as a function of the reconfiguration cost and hit-ratio.*

Figure 18: *Break-even operating region in which the slowdown that results from cache misses is compensated by the speedup due to the smaller size of the processor.*

Figure 19: *A 32-bit parallel pipelined multiplier produces a 64-bit result. A partial product element is shown at the bottom. The bottom row consists of partial product elements operating as full adders (FAs) [17].*

Figure 20: *A parallel pipelined adder tree for 3-bit numbers.*

Figure 21: *Block diagram of the matrix-vector multiplier architecture.*

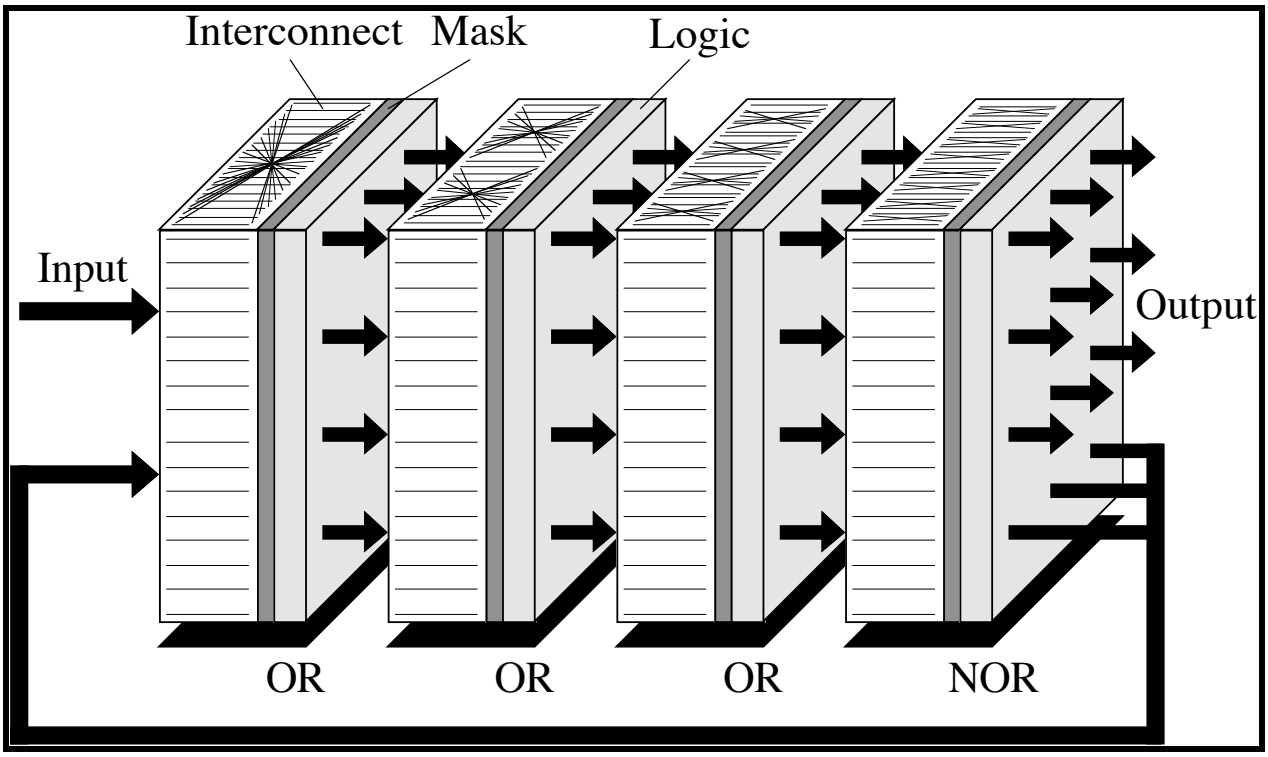


Figure 1: *Arrays of optical logic gates are interconnected with optical crossovers [9]. Masks in the image planes block light at selected locations which customize the system for specific functions.*

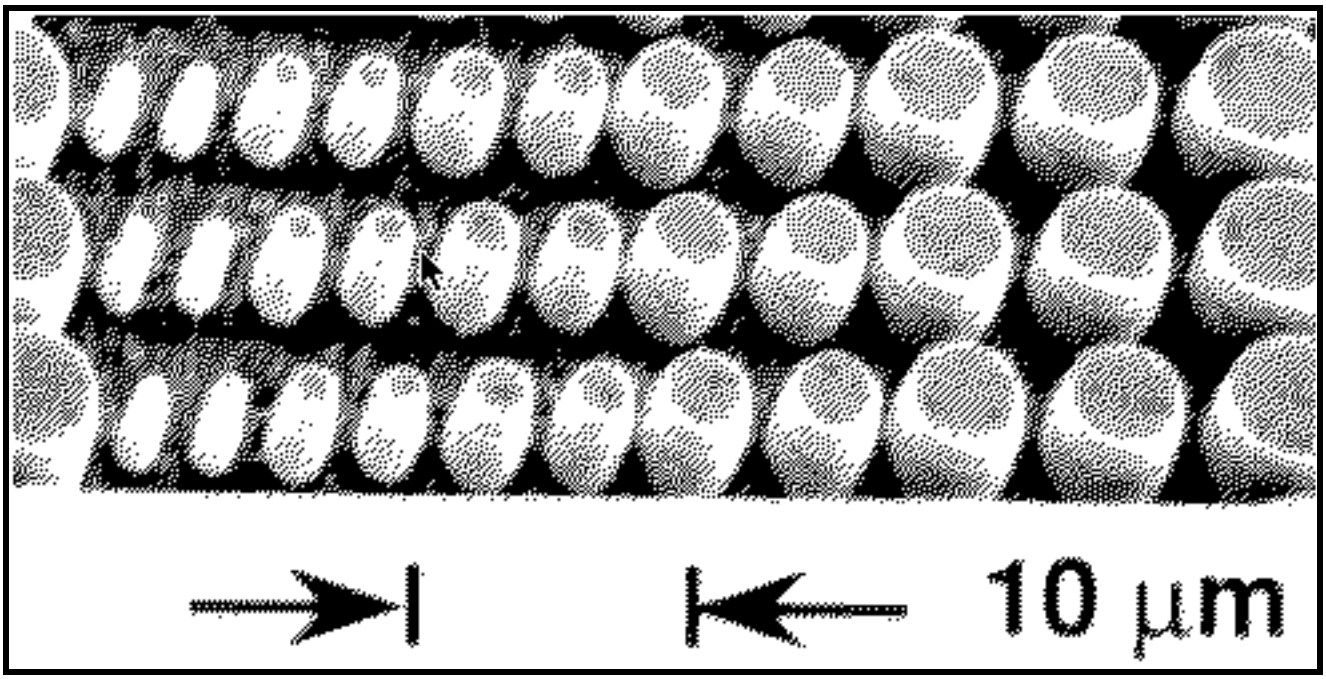


Figure 2: *Scanning electron micrograph of a small portion of an array of vertical-cavity surface-emitting lasers [3]. (Photograph provided by the courtesy of Jack Jewell, Photonics Research Inc.)*

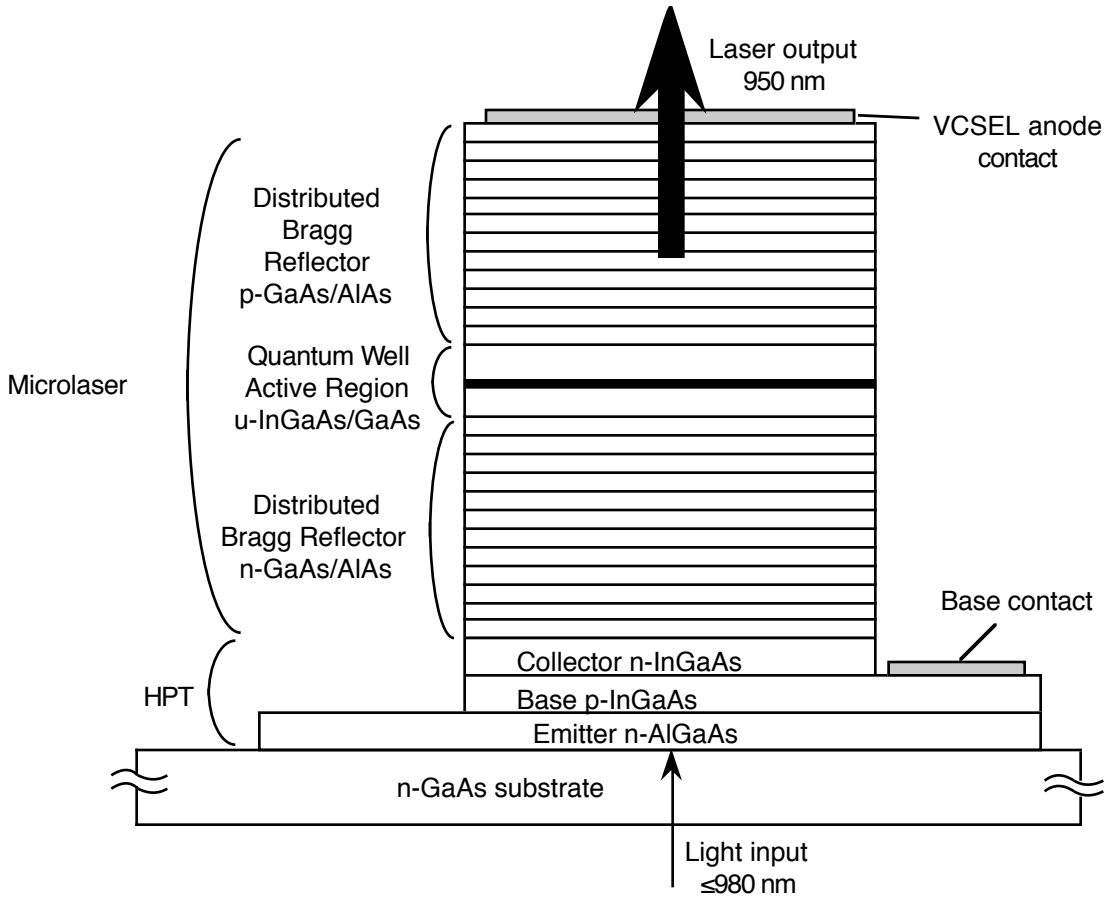


Figure 3: *Schematic diagram showing the structure of the CELL. (Illustration provided by the courtesy of Jack Jewell, Photonics Research Inc.)*

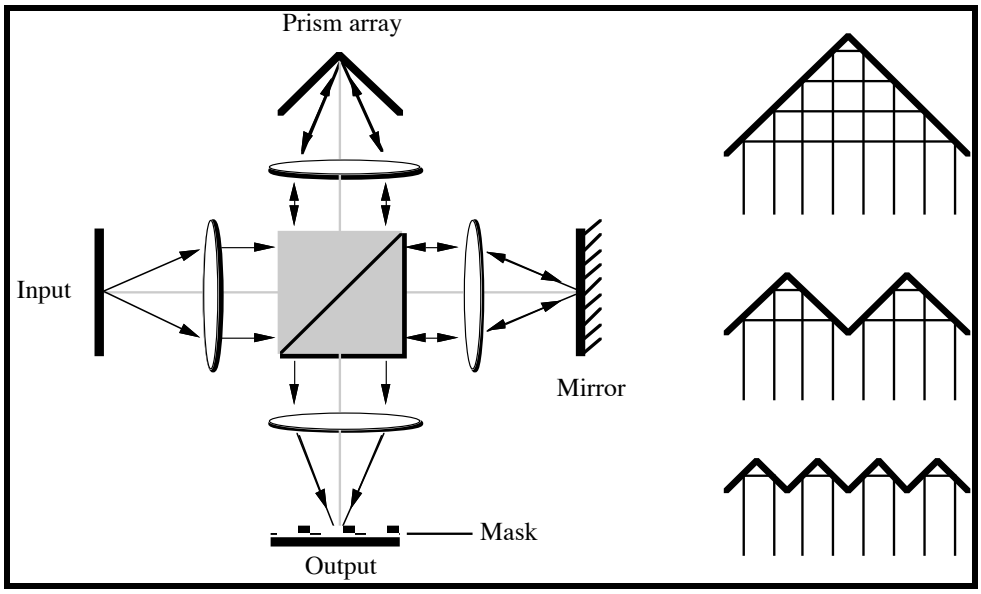


Figure 4: *Optical crossover interconnect. A two-dimensional array of input beams is split into two identical copies. One copy is imaged onto a mirror and is reflected back through the system to the output, while the other copy is imaged onto a prism array that permutes the beams according to its period. Connection paths achieved with different prism array periods are shown in the right panel.*

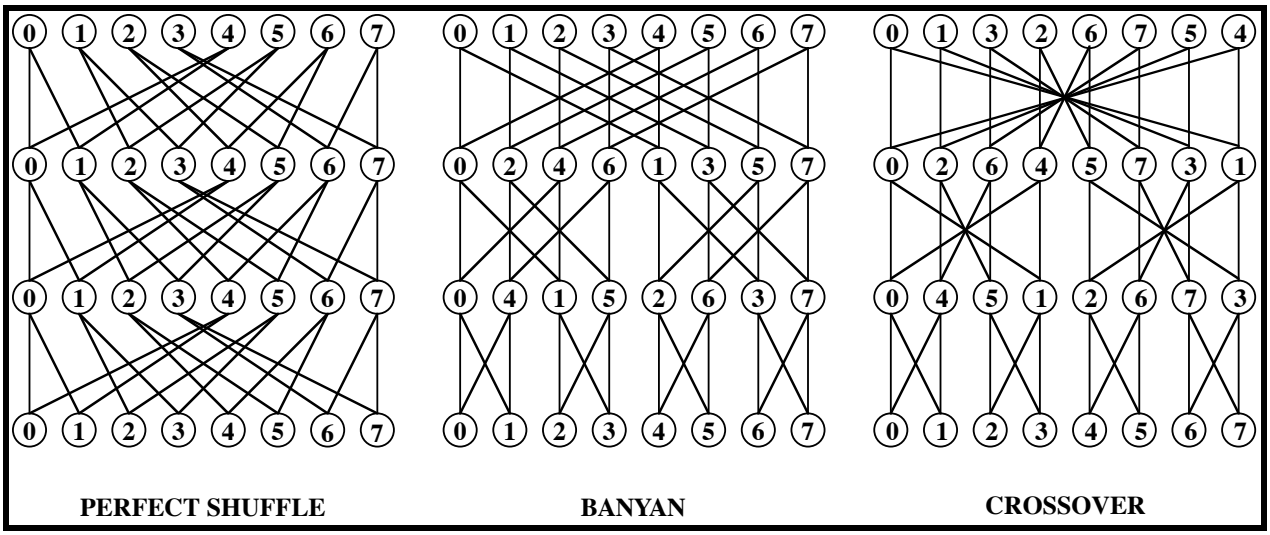


Figure 5: *Isomorphism among the perfect shuffle, banyan, and crossover interconnects.*

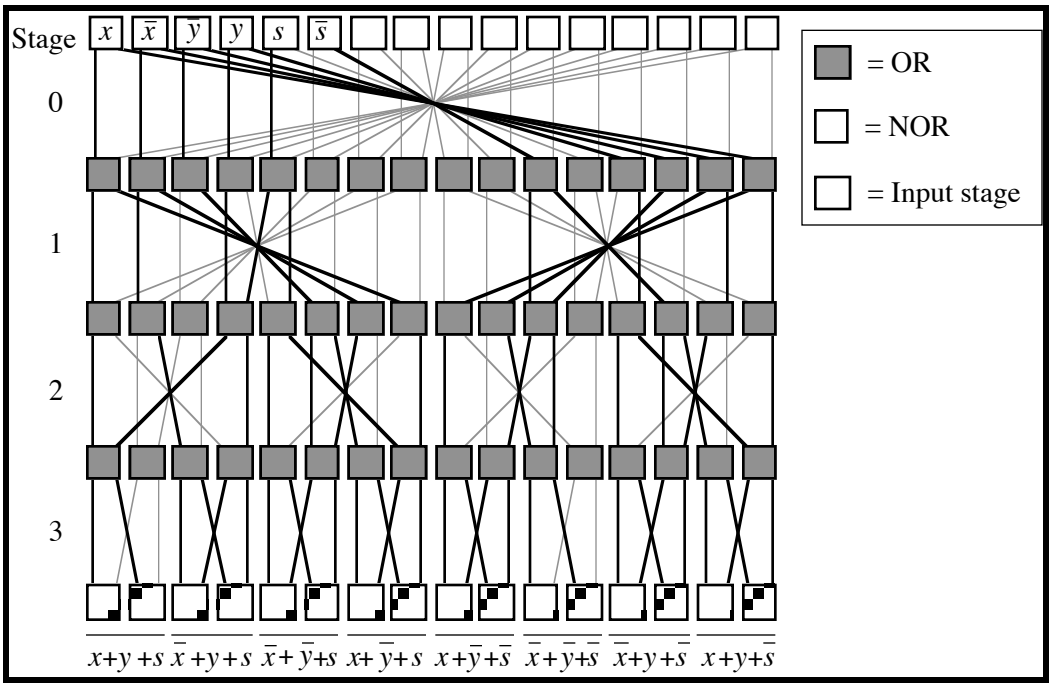


Figure 6: *A 3-to-8 decoder is implemented in an OR-NOR / OR-NOR network. Lightly shaded connections are blocked by fixed masks in the output planes of the crossover interconnects. Flow of signals is from the top to the bottom.*

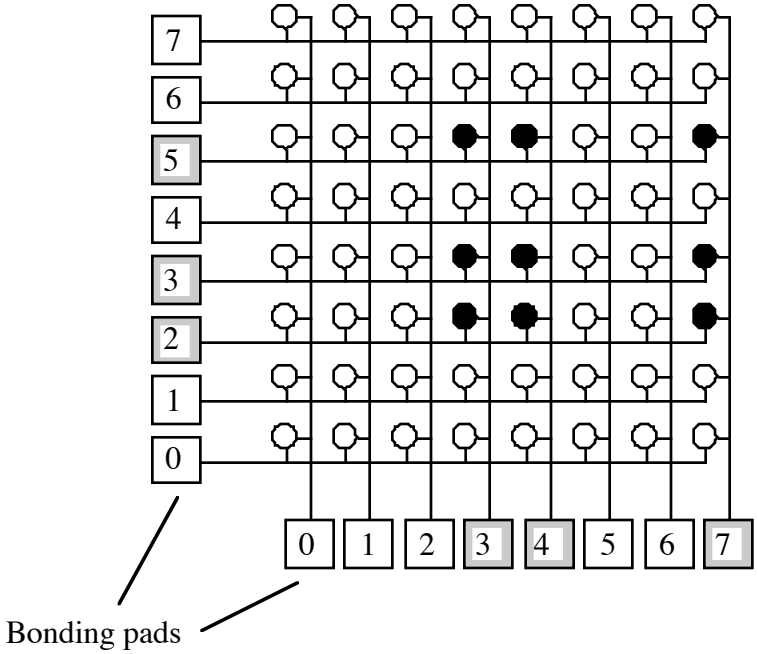


Figure 7: *A matrix addressable array of devices requires only a linear growth in the number of bonding pads for a power of two growth in the number of devices. The indicated pattern is selected by enabling rows 2, 3, and 5 and enabling columns 3, 4, and 7.*

Selection				Arithmetic operations
S3	S2	S1	S0	M=0, Cn = carry-in
0	0	0	0	F = A PLUS 1
0	0	0	1	F = (A + B) PLUS 1
0	0	1	0	F = (A + B') PLUS 1
0	0	1	1	F = ZERO
0	1	0	0	F = A PLUS AB' PLUS 1
0	1	0	1	F = (A + B) PLUS AB' PLUS 1
0	1	1	0	F = A MINUS B
0	1	1	1	F = AB'
1	0	0	0	F = A PLUS AB PLUS 1
1	0	0	1	F = A PLUS B PLUS 1
1	0	1	0	F = (A + B') PLUS AB PLUS 1
1	0	1	1	F = AB
1	1	0	0	F = A PLUS A PLUS 1
1	1	0	1	F = (A + B) PLUS A PLUS 1
1	1	1	0	F = (A + B') PLUS A PLUS 1
1	1	1	1	F = A

Figure 8: *Truth table for the 74181 4-bit ALU.*

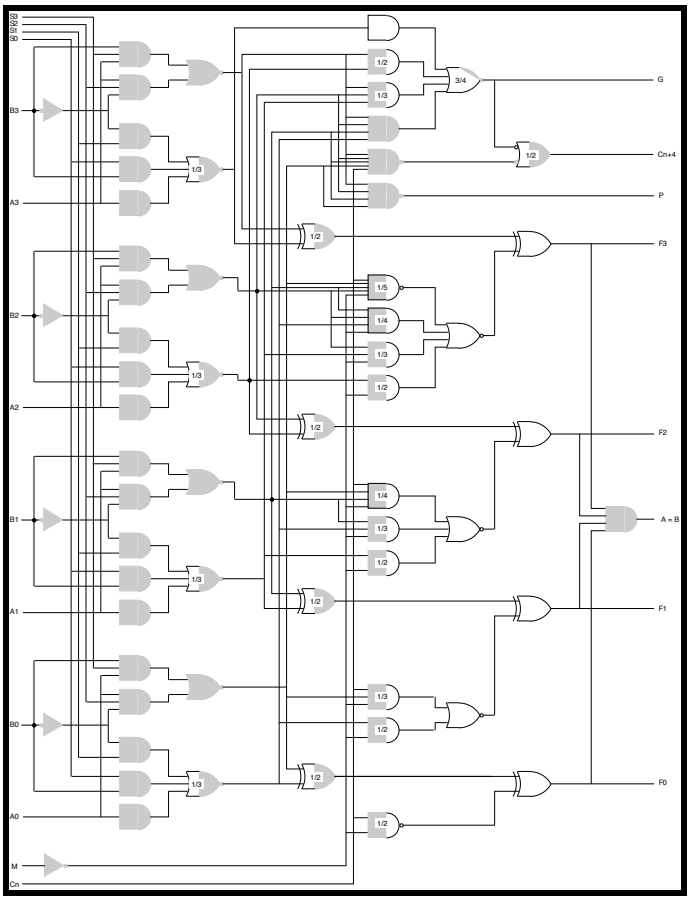
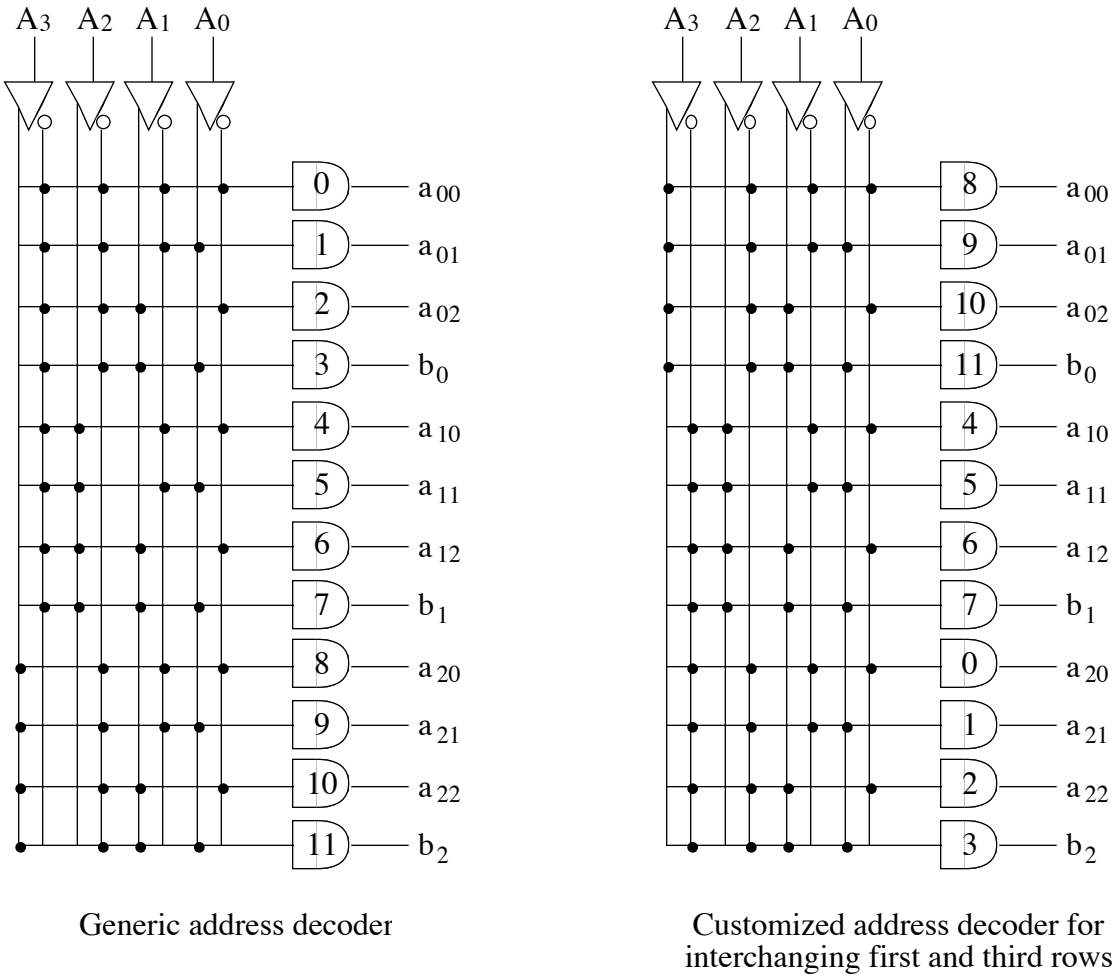


Figure 9: *Modified version of the 74181, showing the amount of underutilized logic (indicated by shading). Fractions indicate the percentage of gate inputs that are used for the ADD operation.*

$$
\begin{aligned}
a_{00}x_0 + a_{01}x_1 + a_{02}x_2 &= b_0 \\
a_{10}x_0 + a_{11}x_1 + a_{12}x_2 &= b_1 \\
a_{20}x_0 + a_{21}x_1 + a_{22}x_2 &= b_2
\end{aligned}
$$

0 a_{00}	1 a_{01}	2 a_{02}	3 b_0
4 a_{10}	5 a_{11}	6 a_{12}	7 b_1
8 a_{20}	9 a_{21}	10 a_{22}	11 b_2

Figure 10: *An augmented matrix is shown for a system of linear equations in three unknowns. Indices in the upper left corners of cells indicate the storage locations in a random access memory.*



Generic address decoder

Customized address decoder for interchanging first and third rows

Figure 11: *Two forms are shown for a four-variable address decoder. The configuration on the left corresponds to the cell numbering shown in Figure 10. The configuration on the right corresponds to a cell numbering in which the top and bottom rows are interchanged. Only 1/6 of the crosspoints are changed between the two forms, even though 3/4 of the elements are interchanged.*

Figure 12: *Hit ratio versus code size for different sizes of the function cache for synthesized runs of gcc on DLX.*

Figure 13: *Hit ratio as a function of the size of the function cache for synthesized runs on the DLX.*

Figure 14: *Hit ratio as a function of the size of the function cache for synthesized runs on the VAX.*

Figure 15: *Hit ratio as a function of the size of the function cache for actual runs on the SPARC.*

Figure 16: *Hit ratio as a function of the size of the function cache for synthesized runs on the SPARC.*

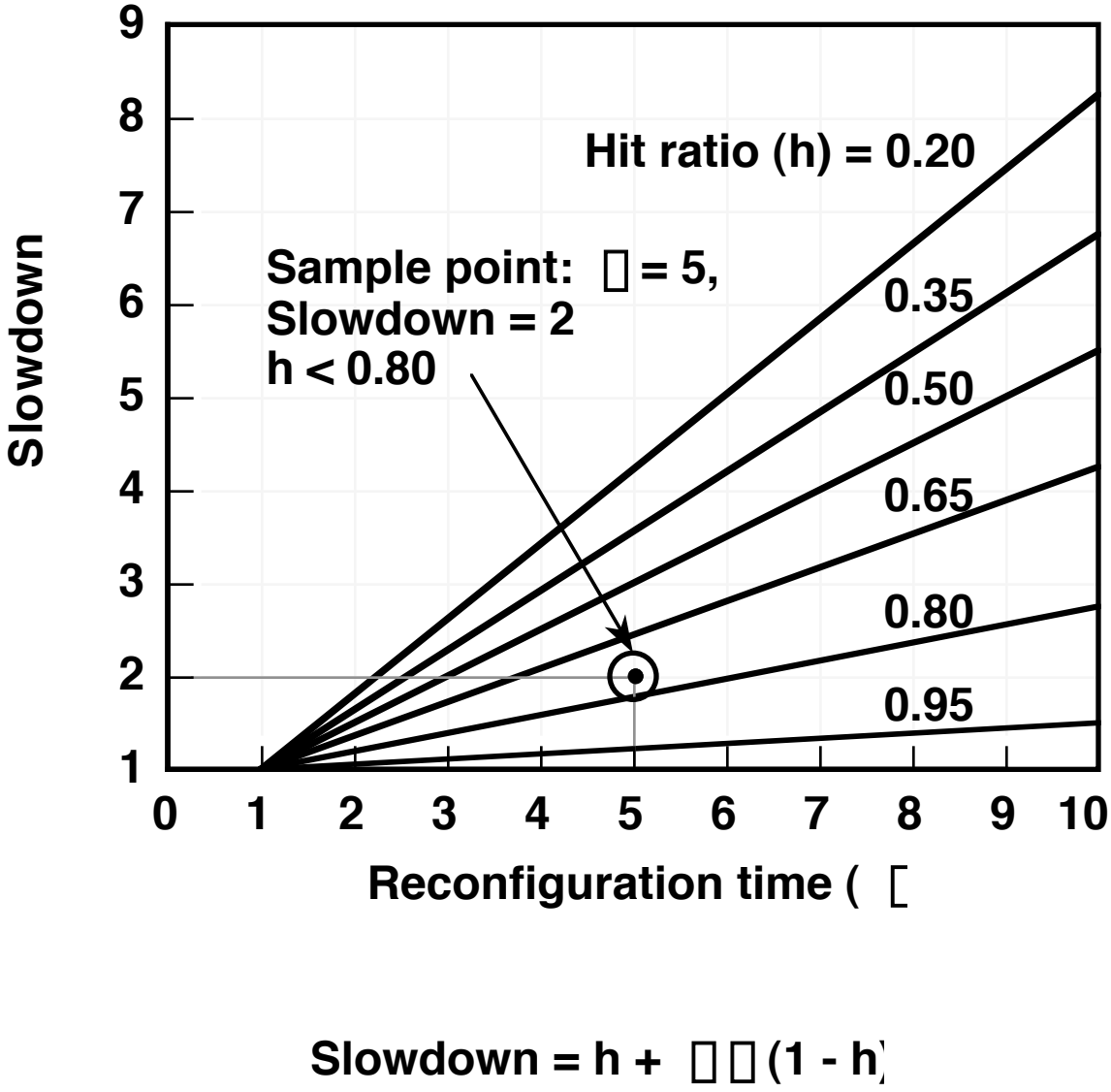


Figure 17: *Slowdown factor as a function of the reconfiguration cost and hit-ratio.*

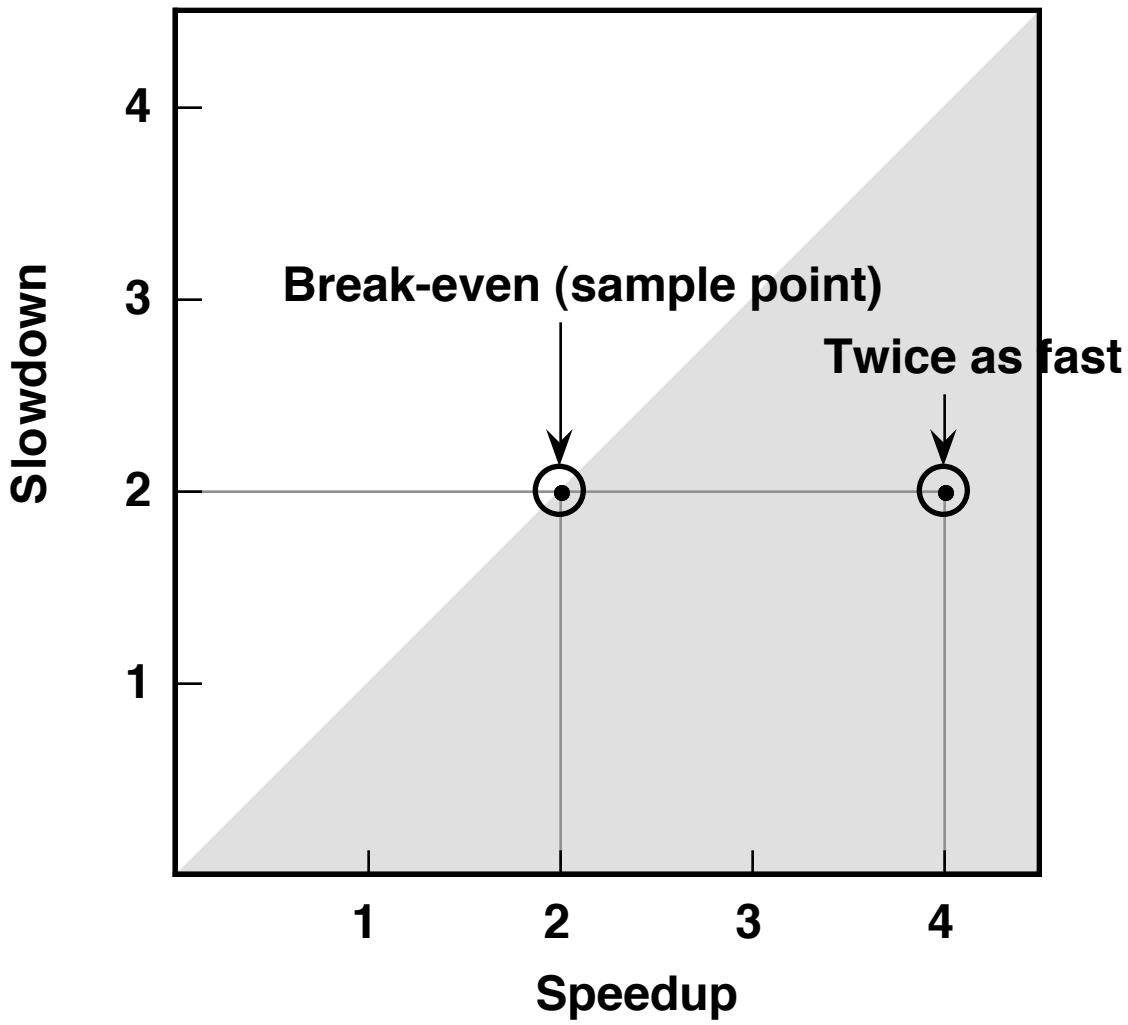


Figure 18: *Break-even operating region in which the slowdown that results from cache misses is compensated by the speedup due to the smaller size of the processor.*

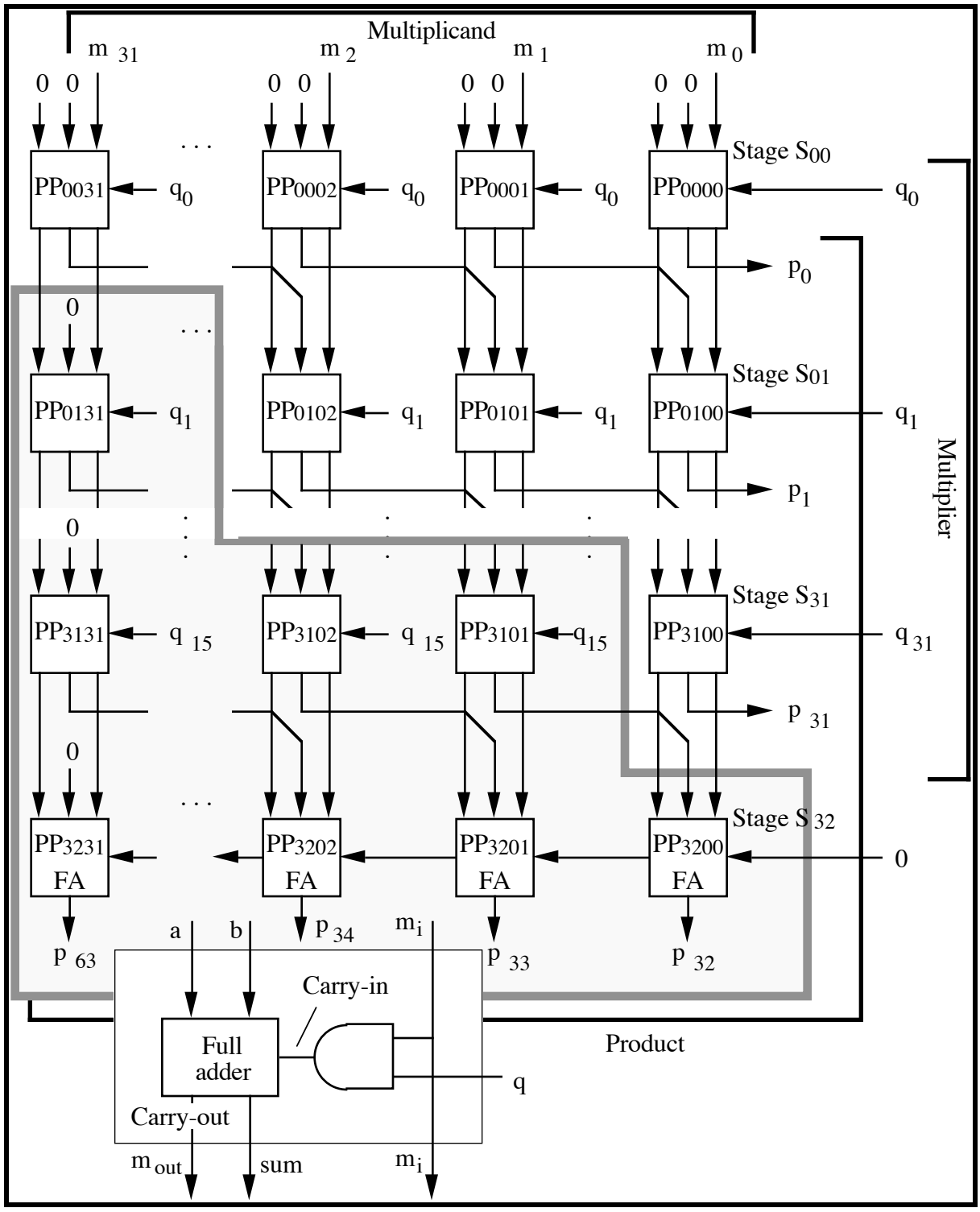


Figure 19: *A 32-bit parallel pipelined multiplier produces a 64-bit result. A partial product element is shown at the bottom. The bottom row consists of partial product elements operating as full adders (FAs) [17].*

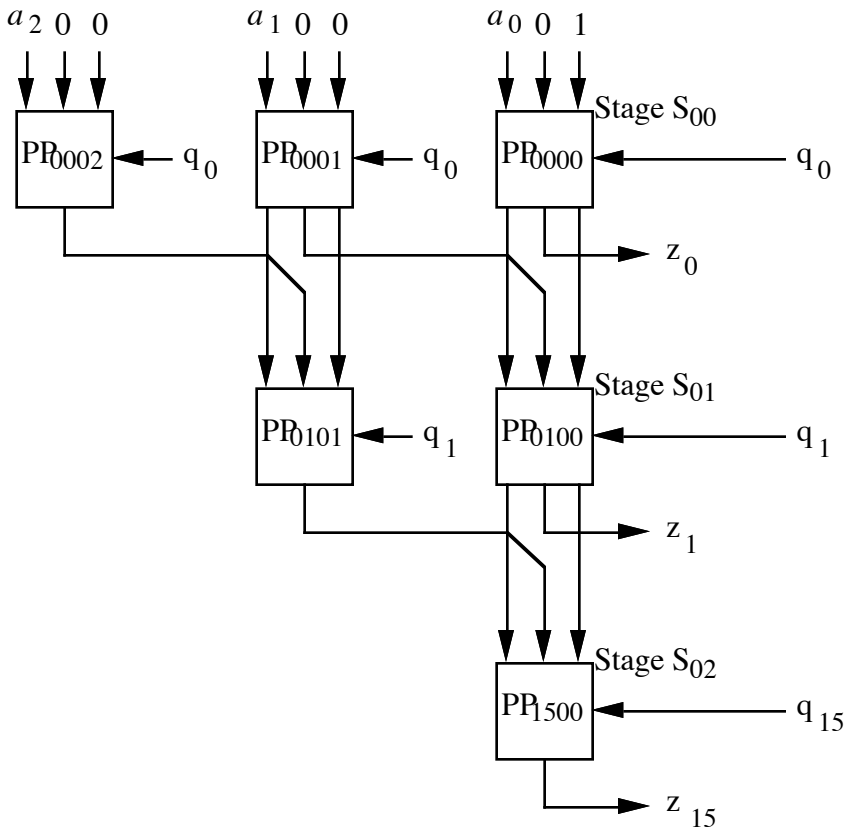


Figure 20: *A parallel pipelined adder tree for 3-bit numbers.*

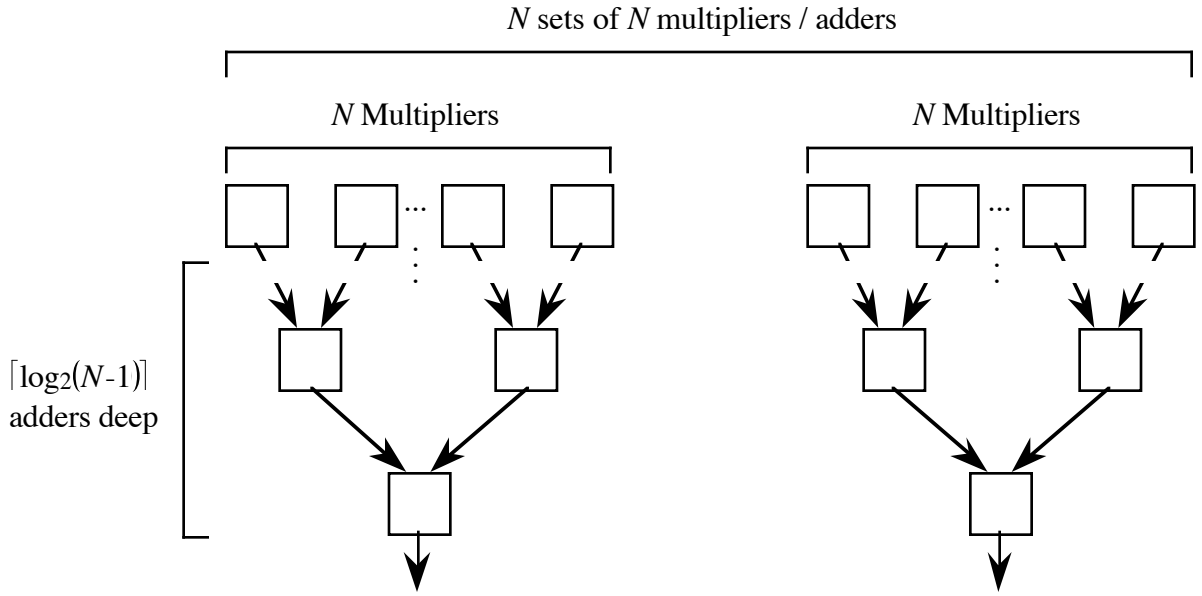


Figure 21: *Block diagram of the matrix-vector multiplier architecture.*