GRANT / AMES

31 pages

IN-28444

Notes on Implementation of Sparsely Distributed Memory

James D. Keeler
Peter J. Denning

August 7, 1986

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS Technical Report 86.15

```
(NASA-TM-89254)  NOTES ON IMPLEMENTATION OF          N86-33031
SPARSELY DISTRIBUTED MEMORY (NASA)  31 p
                                   CSCL 09B
                                                      Unclas
                                             G3/60   44589
```

RIACS

Research Institute for Advanced Computer Science

Notes on Implementation of Sparsely Distributed Memory

James D. Keeler
Peter J. Denning

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS Technical Report 86.15
August 7, 1986

The Sparsely Distributed Memory (SDM) developed by Kanerva is an unconventional memory design with very interesting and desirable properties. The memory works in a manner that is closely related to modern theories of human memory. In the following, the SDM model is discussed in terms of its implementation in hardware. Two appendices discuss unconventional approaches to components of the SDM: Appendix A treats a resistive circuit for fast, parallel address decoding; Appendix B treats a systolic array for high-throughput read and write operations.

Work reported herein was supported in part by Cooperative Agreement NCC 2-387 between the National Aeronautics and Space Administration (NASA) and the Universities Space Research Association (USRA).

James Keeler's permanent address is Department of Physics and Institute for Nonlinear Dynamics B-019, U.C. San Diego, La Jolla, CA 92093.

Notes on Implementation of Sparsely Distributed Memory

James D. Keeler
Peter J. Denning

Research Institute for Advanced Computer Science

August 7, 1986

1. Introduction

The Sparsely Distributed Memory (SDM) is exactly what the name implies: It is a set of storage locations whose addresses are distributed sparsely over the space of possible addresses. The advantages of this memory design are many. It can perform pattern recognition with automatic error correction. It can perform brain-like functions. It is immune to failure of particular elements; the SDM would be able to function reliably if 5 to 10% of its elements failed. It is versatile: It can act as a content-addressable memory, a random-access memory (RAM), or a sequential-access memory. In this report, we discuss briefly the SDM model and give a schematic description of how the SDM could be realized in hardware, both digital and analog.

A few concepts should be explained before we discuss the hardware implementation of the SDM. The SDM starts with an address space, the set of 2^n

distinguishable n-bit addresses. For n moderately large, the number of possible memory locations becomes astronomical.. Indeed, for $n = 1000$, 2^n is larger than the number of particles in the known universe. Obviously, there is no way of associating all, or even a relatively small fraction, of these addresses with physical storage locations. Hence, we pick at random m addresses to be associated with physical storage locations; we are assuming m on the order of a million to a billion. Because m is small compared to 2^n, these randomly chosen addresses point to a set of memory locations that are sparsely distributed in the address space.

To function as a memory, this system should be able to write and read data. To write, we need as input both the address and the data word itself. In the SDM, the address size and the data-word size are the same. This size equivalence is not a limitation since there need be no requirement that the address and the data-word size be different.

2. Selection

Given an address, where are the corresponding data written? The given address is quite unlikely to point to any one of the m randomly chosen memory locations. However, some of the memory locations are closer to the input address than others. The selection rule is: select all locations whose addresses are within a Hamming distance D of the given address. If we view n-bit addresses as points in n-dimensional address space, the selected locations will lie within a (hyper)sphere of Hamming radius D centered at the input address (see Figure 1). The data are written into every memory location within this sphere. This is why we say that the information is *distributed* over all the selected memory locations.

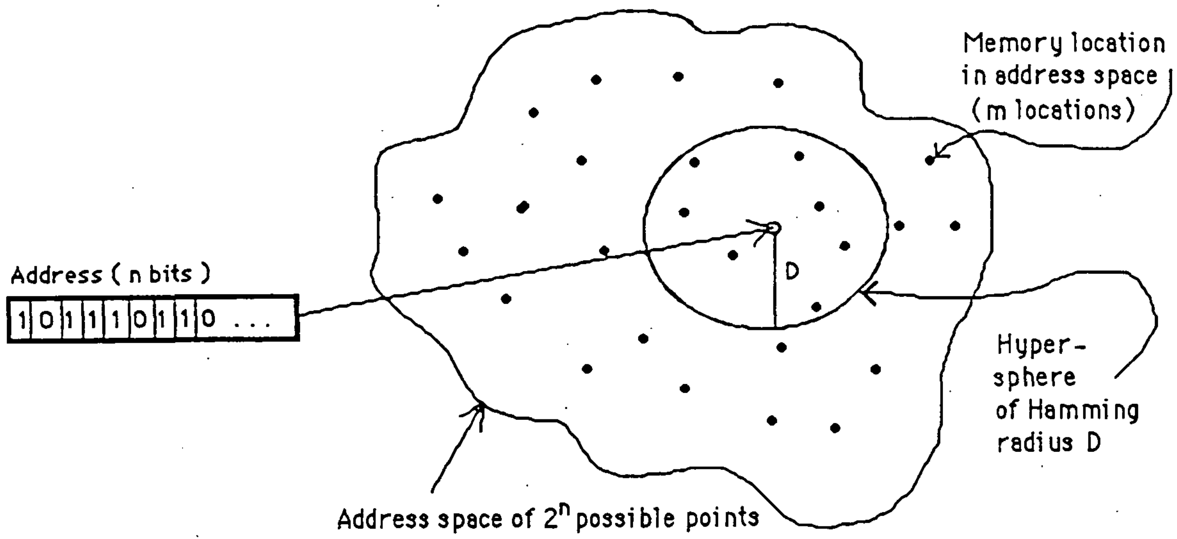
Memory location in address space (m locations)
Address (n bits)
1 0 1 1 1 0 1 1 0 . . .
D
Hyper-sphere of Hamming radius D
Address space of 2^n possible points

Figure 1. This qualitative picture of address space shows memory locations in the address space, an input address to read from or write into memory, and the Hamming (hyper)sphere containing all selected locations.

Each memory location is actually a set of n counters. The reason is that we may wish to write two (or more) sets of data whose addresses point to overlapping spheres; by allowing memory locations to be written into more than once, we allow retrieval of multiple sets of data. The data-word is written in the selected locations by adding it bitwise to the location's counters; the rule is, a 1-bit of data causes 1 to be added to the corresponding bit counter, while 0 causes

1 to be subtracted. The details of how this scheme works can be found elsewhere.[1-3]

3. Reading and Writing

Figures 2 and 3 are schematic representations of the SDM hardware; Figure 2 describes writing into the memory and Figure 3 describes reading. The system consists of two basic functional parts: First there is the address-decoding stage, in which the input address is broadcast on an address bus and is compared to each and every of the m (fixed) addresses of the m memory locations. This comparison is done in the device labeled "Address comparator" in the figures. The address comparator is a very simple device that performs bitwise XORs on the two addresses and computes the number of bits in which the two addresses differ. If the Hamming distance is less than D, select is set *true* and the data are written into that memory location by adding the n bits of data individually to the location's n counters.

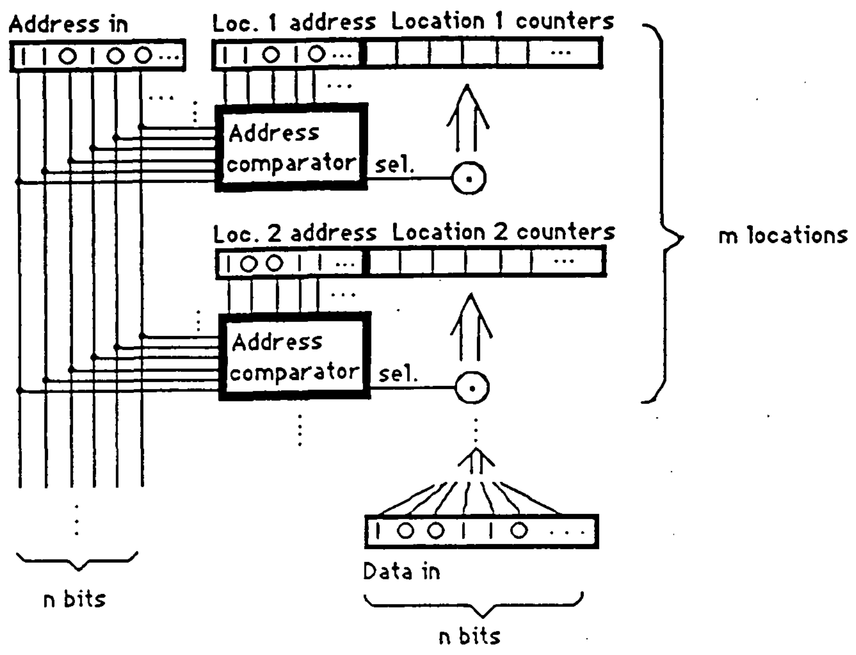

Figure 2. Writing into the SDM. An input address is given along with its input data. The input address is compared to each and every address in the list of memory locations. If the address is less than D Hamming units away, then *sel.* is set true, and the data are written into that memory location. Each location is actually a group of n counters, where n is the data-word size. Upon writing, each selected location will increment or decrement each of its n counters according to the data in. The address comparison is done for all m locations, and the writing is done only for each of the selected locations.

Reading is done by giving the address for the data to be read. As with writing, the address is compared to each of the m location-addresses, selecting those within D of the read address. Then, the contents of the counters for each of the selected memory locations are added together to yield n separate sums. Each of these n sums is then passed through a threshold. If a sum exceeds the threshold, the corresponding output-data bit is set to 1; otherwise it is set to 0.

In this manner the data that were written for a given address are retrieved with a very high probability.

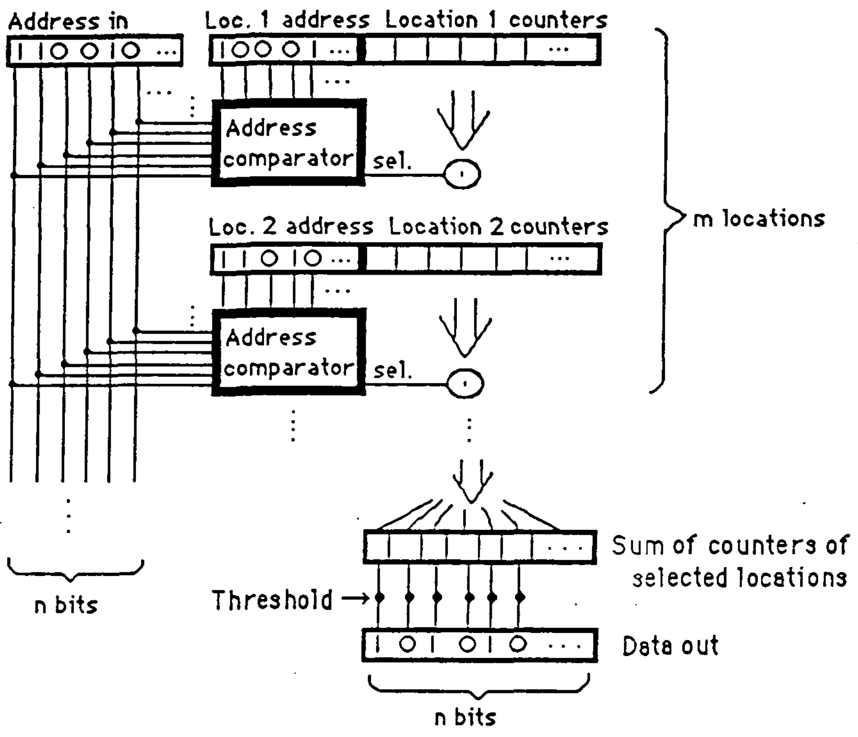

Figure 3. Reading from the SDM. Reading is simply done by giving the address of the data to be read. This address is compared to the address of each and every location in the memory. If the address is less than D Hamming units away, *sel.* is set true. All locations with their *sel.* = *true* have the contents of their counters added together yielding n sums. The sums are passed through a threshold device that yields the data out.

The discussion above omits important details on how to implement the address decoders and how to perform the read and write operations. These functions can be implemented in straightforward (but tedious) ways in conventional logic. Two slightly unconventional alternatives are discussed in the appendices.

Appendix A analyzes a resistive network capable of determining the cells in the selected hypersphere in parallel, in time proportional to $n + m$. Appendix B presents a systolic array that can perform read and write operations of many requests in parallel.

4. References

[1] Pentti Kanerva, *Self-Propagating Search: A Unified Theory of Memory.* MIT Press Bradford Books (1986), in press.

[2] Pentti Kanerva, "Parallel structures in human and computer memory," RIACS Technical Report TR-86.2 (1986).

[3] Peter Denning, "A view of Kanerva's sparse distributed memory," RIACS Technical Report TR-86.14 (1986).

APPENDIX A: Analog Parallel Address Comparator

The SDM is based on massively parallel processing. Some of this processing could done in analog. There are two stages in the SDM where a linear-threshold circuit is required. Linear-threshold circuits are difficult to build with digital circuitry; the linear threshold is usually achieved with a number of XORs, adders and a comparators. However, analog circuits can perform linear thresholds all at once in a resistive bridge (among other things). The following is a description of a simple analog circuit that could be used for the address-comparison stage of the SDM, and would perform the address selections in parallel simultaneously.

The main idea behind the address comparator is to compare one address to another address and to set a select signal to *true* if the addresses are within D Hamming units of each other. Although the input address is variable, the addresses of the cells (memory locations) are fixed (and randomly chosen). Thus, it is possible to encode the addresses of the cells in a fixed resistive network. In the analog address comparator, the input address is given in parallel on an input-address bus. The input address is represented as a string of 0s (ground) and 1s (V_s , the source voltage). The addresses of the cells are also represented as strings of 0s and 1s, but these addresses are encoded by the placing of resistors at proper intersections between address lines and cell lines in the resistive network. The principle behind the operation of the circuit is that the current flowing through the wires of each cell line will be proportional to the Hamming distance between the input address and the address of that particular memory location. This current is converted into a voltage that can be compared to the threshold to give a select signal.

Suppose that the address of one of the cells is given by the string (1,1,0,1,0,0,...). This address is encoded in the resistive network in the following fashion: For each cell, there are two wires (running vertically in Figure A1), the "0s" wire and the "1s" wire; the 0s wire is attached to ground, and the 1s wire to V_s. The address is simply encoded by placing resistors at intersections between the input-address wires and the cell wires according to the address bits. For example, if the address of the first cell were (1,1,0,1,0,0,...), a resistor would be placed between the first line of the input address and the 1s wire of the first cell (this is marked with a black dot labeled R at the intersection of the first horizontal and vertical wires in Figure A1), another resistor would be placed at the crossing of the second input-address wire and the 1s wire of the first cell, a third resistor on the third input line and the 0s wire of the first cell, etc. In all, there would be n resistors for each address in the memory location, and if there are m cells, there would be m pairs of wires on which to place resistors. Thus, the number of wires is $2m+n$ in the resistive bridge and the number of resistors is nm. Buffers can be used on the input-address line after so many cells to boost the signal on the input-address bus. These buffers are shown at the top right in Figure A1.

How does this circuit work? First, look only at the 0s line of the first address from the top of the circuit down to ground in Figure 1. If R_a is zero

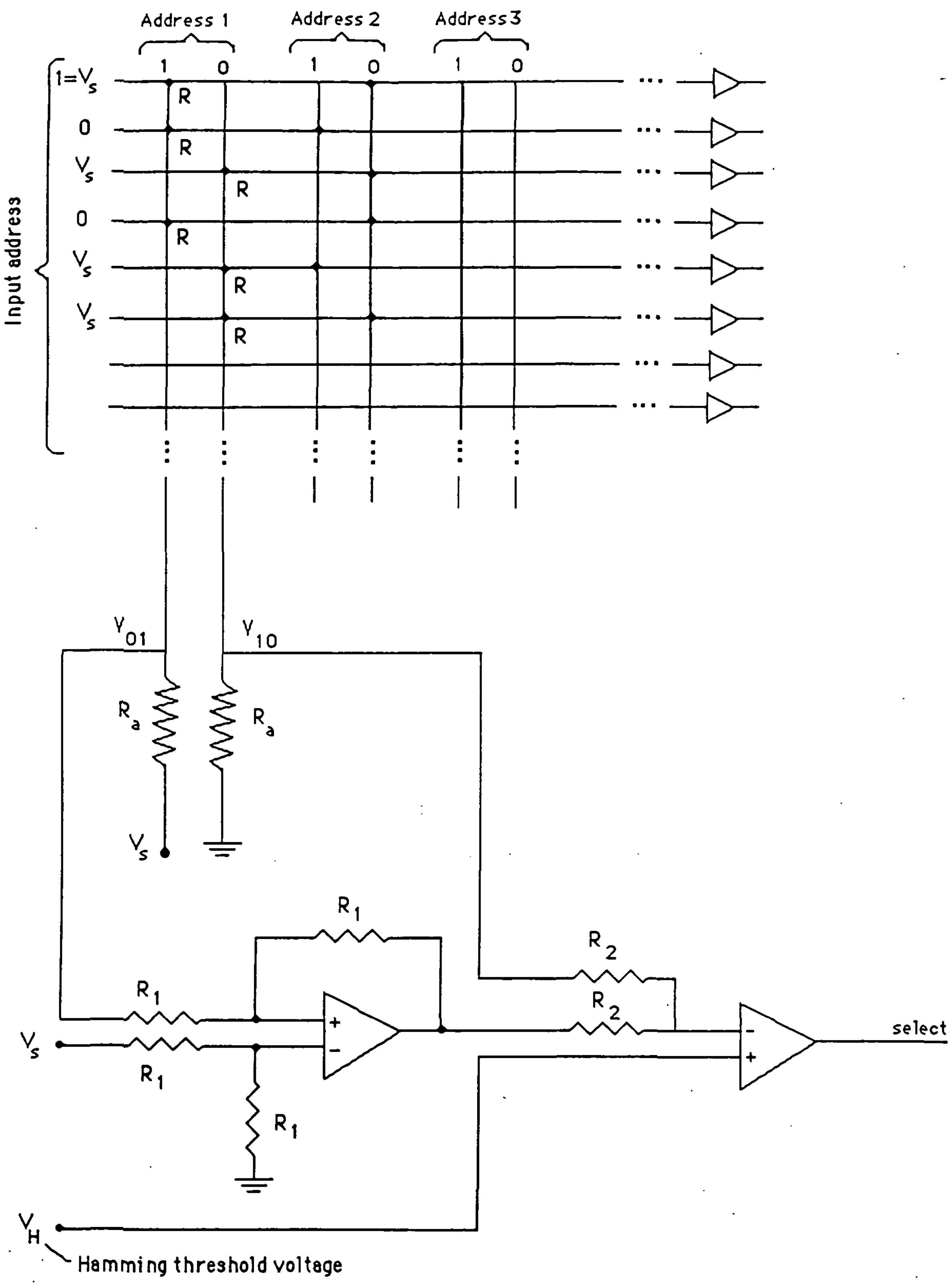

Figure A1. Schematic diagram for the analog address comparator. The first six bits of the Input address are 101011 and of Address 1 they are 110100. The black dots marked R represent resistors at the intersection of address lines and cell lines. The current running through the cell lines is proportional to the Hamming distance. The op-amps at the bottom are used to sum the currents as voltages and threshold this voltage with the input Hamming voltage. The op-amps could be realized with 2-4 transistor circuits not shown here explicitly.

resistance, then it is clear that the current flowing through the 0s wire is proportional to the number of places at which the input address has 1s and location 1 address has 0s. The resistor R_a is just put in the circuit to convert this current to a voltage (i.e., this is just a very simple ammeter). Next, look only at the 1s line of the first address. If R_a were absent, the current flowing through this wire would be proportional to the number of places at which the input address has 0s and the location address has 1s. The sum of the two currents would then be proportional to the total number of places at which the input address and the location address differ (that is, the current is proportional to the Hamming distance). The rest of the circuit converts these currents to voltages and sums them together to get a voltage that can be compared with a programmable Hamming-distance voltage to give the final select signal.

Detailed Circuit Analysis

The above paragraph is qualitative. Now we will give a quantitative analysis to demonstrate the truth of the claims. In this section, we assume that all circuit elements are ideal. The nonideal case will be discussed in the next section. The resistive network for the first two cell lines can be reduced to the equivalent circuits shown in Figures A2 and A3. Both circuits are just simple voltage dividers. The key point is being able to produce a good ammeter, which means choosing R_a very small compared to the parallel sum of all the other R s. This is feasible since we want $R/n \ll R_a$; for $n = 1000$, and $R_a = 100\Omega$, R can

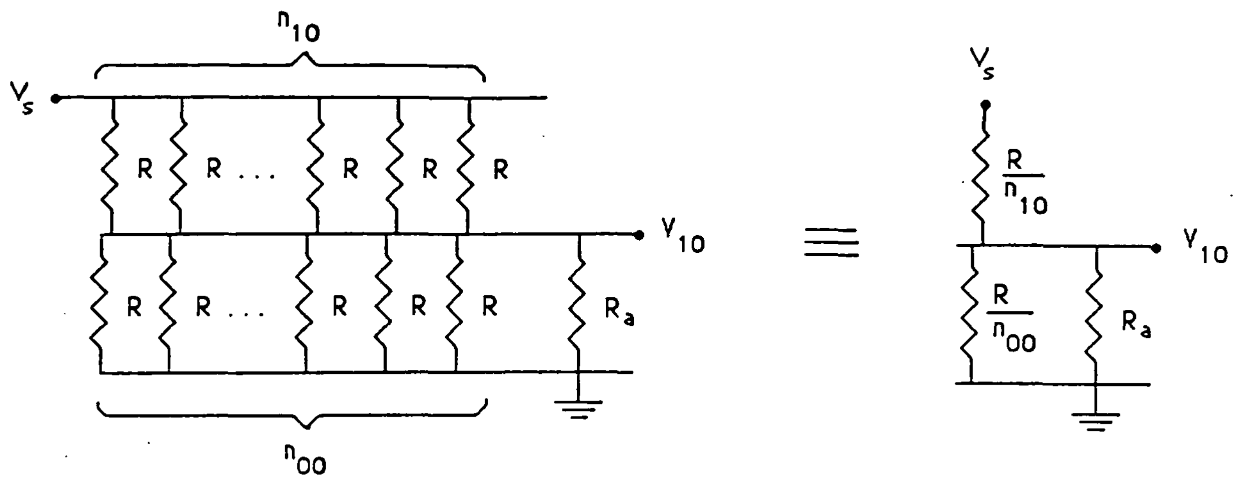

Figure A2. The circuit equivalent for the second vertical wire (the 0s line) and the input-address lines in the address decoder of Figure A1.

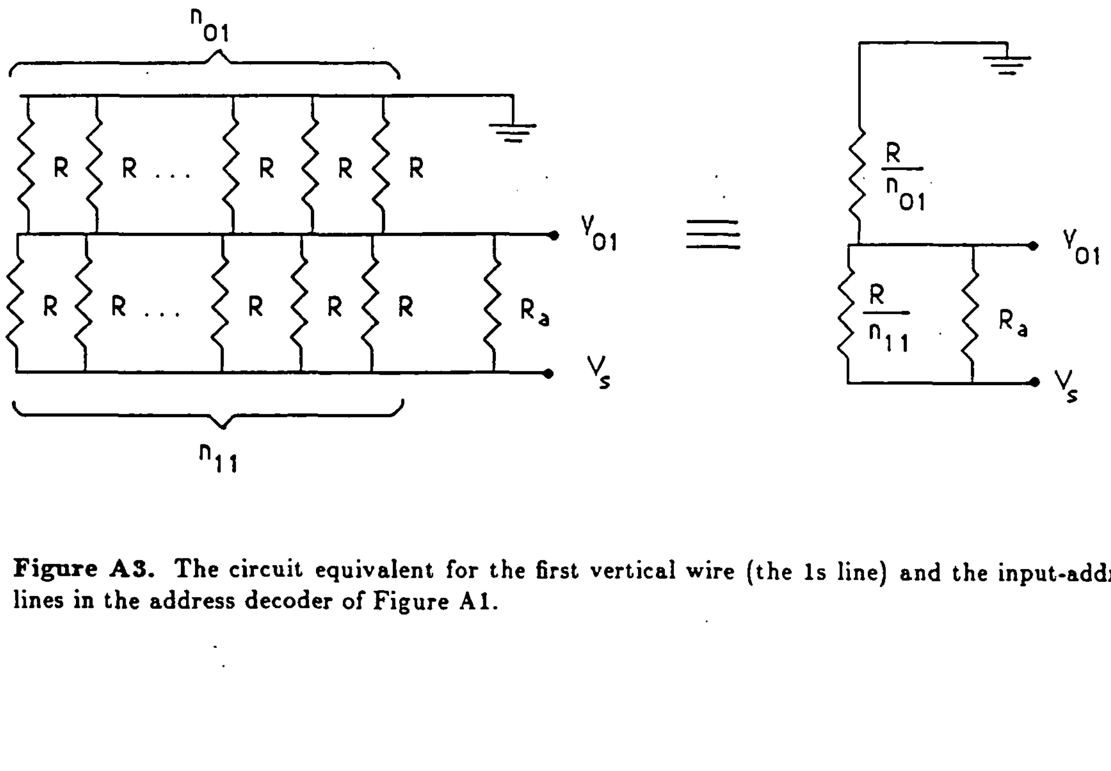

Figure A3. The circuit equivalent for the first vertical wire (the 1s line) and the input-address lines in the address decoder of Figure A1.

be 100 MΩ.

The equivalent circuit for the input address and the 1s line is shown in Figure A2. From this circuit, elementary circuit analysis shows that

$$V_{10} = \frac{R_y}{R_x + R_y} V_s , \tag{A1}$$

where $R_x = R / n_{10}$, $R_y = R_a R / (n_{00} R_a + R)$, where n_{10} is the number of places where the input address has 1s and the location address has 0s, and n_{00} is the number of places where the input address and the location address both have 0s. If $n_{00} R_a \ll R$ and $n_{10} R_a \ll R$, we get

$$V_{10} = n_{10} \frac{R_a}{R} V_s . \tag{A2}$$

By a similar analysis, we get

$$V_{01} = V_s - n_{01} \frac{R_a}{R} V_s . \tag{A3}$$

Hence,

$$V_{10} + (V_s - V_{01}) = (n_{01} + n_{10}) \frac{R_a}{R} = D \frac{R_a}{R} , \tag{A4}$$

where n_{01} is the number of places where the input address has 0s and the location address has 1s, and n_{11} is the number of places where the input address and the location address both have 1s. Thus, we have found a quantity proportional to the Hamming distance, D . The rest of the circuitry is just to perform the summation of V_{10} and $(V_s - V_{01})$ and to compare the result to the Hamming voltage V_H , which can be changed to give different Hamming distances.

The above analysis holds for every location address (as shown in the next section). Thus, the total circuit would have m copies of these two lines and the summing-threshold circuit at the bottom. These m copies could certainly be placed on different circuit chips and connected together by buffering the input-address bus as described earlier.

The analysis above considers currents on a cell's two lines. Another argument is needed to show that there is no leakage current -- i.e. no additional currents from other cell lines. We will show that no leakage currents exist for the ideal case of perfect grounds and perfect source voltages.

Consider a circuit for two cells. To help visualize this circuit, the address lines and the input-address bus are shown in 3-dimensional perspective in Figure A4. The equivalent circuit is shown in Figure A5. From this equivalent circuit, it is easily seen that no current can flow through R_a on any cell line starting from a different cell line.

For implementing a real circuit, one could choose n =1000, $R = 100M\Omega$, and $R_a = 100\Omega$. For the worst case, $n_{10}=n$, these values would yield a maximum error of 0.1% for the simple ammeter we have constructed here. The number of addresses could be anything if we remember to place buffers on the address bus to boost the signal power. With conventional off-the-shelf op-amps, we would have to place buffers every 50 to 100 address lines to meet the power requirements. The input address could be multiplexed onto the chip to reduce

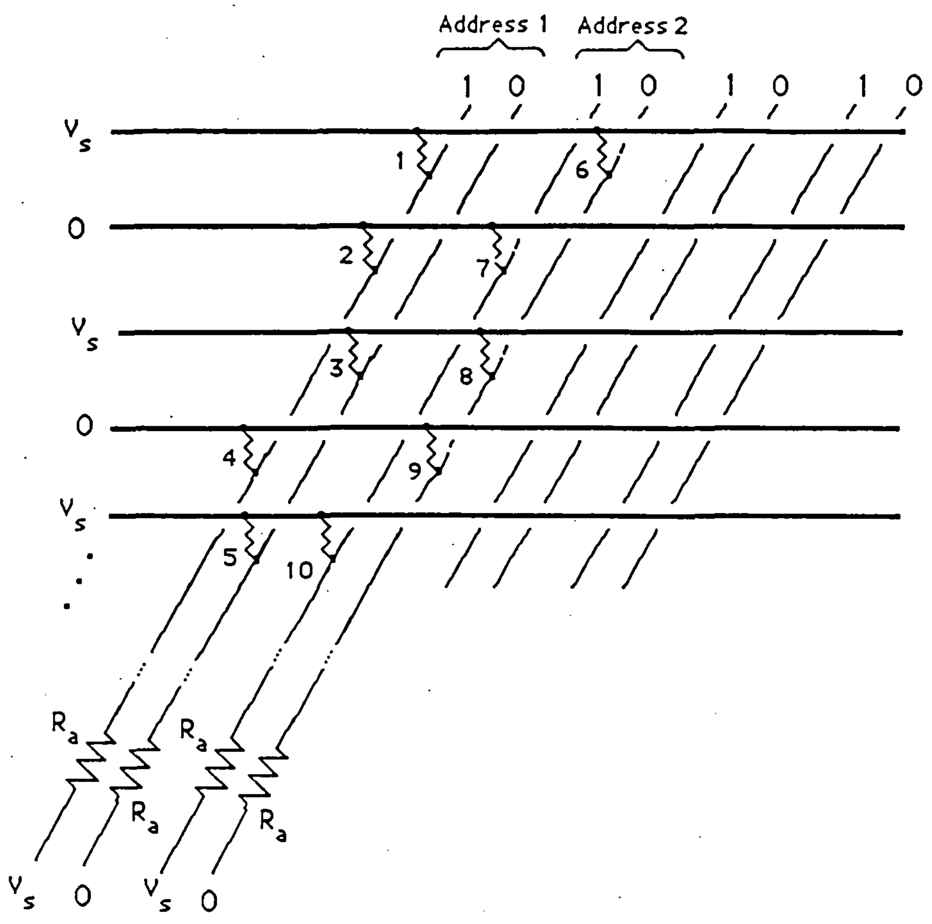

Figure A4. A three-dimensional perspective drawing of a circuit for two address lines connected to the address bus.

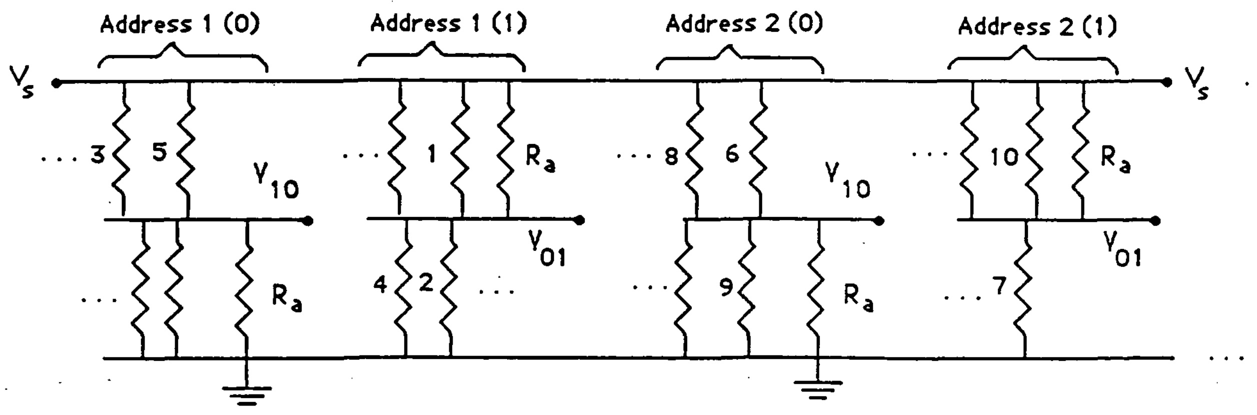

Figure A5. The equivalent circuit for the address decoder shown in Figure A4. Note that each of the address lines decouple because the current flowing through each R_a is dependent only on the resistors in that address line, not on the resistors in the entire network.

the number of wires going into the chip.

Nonideal Circuit Analysis

The preceding arguments assumed ideal resistors and voltage sources. In real circuits, the resistors and voltage sources are likely to differ from their nominal values. It is reasonable to assume that the source voltages and grounds for the cells have no errors. In this case, the current in each cell depends only on the cell's address-line voltages and resistors; thus the error in the current in R_a depends only on the errors in these voltages and resistances.

Next, we assume that an erroneous address-line voltage can be simulated by a nominal voltage applied to an erroneous resistor. It is, therefore, sufficient to consider the effects of resistance errors on the voltages across R_a . In Figure A2, let each resistor now be $R + \delta$ where δ is a random variable representing the random errors in the resistances. For an approximation, we will suppose δ is normally distributed with mean 0 and standard deviation σ. What is a good value of σ? Assume that the combined errors are equivalent to 10% tolerance resistors; then 99.99% of the resistors will be in the range $[0.9R, 1.1R]$ if $4\sigma = 0.1R$, or $\sigma = 0.025R$.

We are interested here only in the error incurred in the main network, not in the ammeter portion. Hence, assume for the moment that the R_a have no errors. Keeping track of the errors in the cell's resistors, the equivalent circuit now yields

$$V_{10} = \frac{R_y}{R_z + R_y} V_s , \tag{A5}$$

where

$$R_z = \frac{R}{n_{10}\left(1 - \frac{1}{n_{10}R} \sum_{i=0}^{n_{10}} \delta_i \right)}, \qquad R_y = \frac{R_a \; R_z}{R_a + R_z} \tag{A6}$$

and where

$$R_z = \frac{R}{n_{00}\left(1 - \frac{1}{n_{00}R} \sum_{i=0}^{n_{00}} \delta_i \right)}, \tag{A7}$$

If we assume $R_a \ll \min(R_z , R_z)$ the equation for V_{10} reduces to

$$V_{10} = \frac{R_a}{R_z} = \frac{n_{10}R_a}{R}\left(1 - \frac{1}{n_{10}R} \sum_{i=1}^{n_{10}} \delta_i \right) V_s . \tag{A8}$$

The sum in this expression has expected value 0. Thus, the expected voltage is to be proportional to n_{10}:

$$\mathrm{E}[V_{10}] = \frac{n_{10}R_a}{R} V_s . \tag{A9}$$

Similarly, the expectation value of V_{01} is

$$\mathrm{E}[V_{01}] = \frac{n_{01}R_a}{R} V_s . \tag{A10}$$

Let's examine the sum in equation (A8) more closely. The sum is

$$S = \frac{1}{n_{10} R} \sum_{i=1}^{n_{10}} \delta_i , \qquad \text{(A11)}$$

and a similar expression for the sum in V_{01}. By looking at this sum, we gain a measure of the relative error of the voltage across R_a. The expectation of S is 0, and the variance is $\sigma^2 / n_{10} R^2$. To achieve a 99% confidence level that the relative error incurred by this sum is less than 1%, and with $\sigma \approx 0.025R$, we must have $n_{10} > 42$. Since the expectation of n_{10} is $0.25n$, this would require $n > 168$. Thus, for $n \approx 200$, ~~the relative~~ less than 1% relative error in the voltage across R_a is caused by the cell's erroneous resistors.

Equation A9 showed that the voltage is expected to be proportional to n_{10}. The proportionality constant is R_a. If R_a has a 10% error, the final signal will be off by 10%. This emphasizes the need for fairly good ammeters (a 10% error does not constitute a good ammeter). Fortunately, this error does not present a problem. The structure of the SDM itself allows sloppy components. The only requirements are that the cells selected by an address for writing are the same as those selected by the same address for reading and that addresses close to this address will select almost the same set of cells. This will happen with very high probability in the above circuit even when it has imprecise elements. If R_a has a small error, this effectively increases or decreases n_{10} or n_{01}. Hence, the selected addresses lie within a hyperellipsoid instead of a hypersphere. Since there is a range of Hamming distance of about 10% around an optimal distance that works quite well for the SDM, this ellipsoid will have the same characteristics as the sphere (as long as it is not too elliptical and the mean of the principle axes is the mean value of the range of proper Hamming distances). This range can be achieved by adjusting the Hamming-threshold voltage.

Finally, we comment on a few other features of this circuit, and speculate on its performance. First, the Hamming voltage can be used to control the number of selected addresses. This voltage could be adjusted through a feedback loop to insure that the number of selected states is approximately constant. This adjustment may improve stability of the SDM. Second, the Hamming voltage could also serve as a "concentration level." For very small voltages, nothing is written or read, hence the "concentration" of the system is very small. For a higher voltage, many locations are written to or read from, indicating a higher concentration level. This could be used to tell the system to pay attention only to important data. Third, present technology allows the placing of resistors in a network such as this using wires of 1 micron width and spacing of 2 microns between the resistors. Hence, one could achieve a 1000-bit address comparator with 1000 cells on a chip of approximately 3 mm × 6 mm. The other components, such as the op-amps, could be placed on the outer part of the chip.

APPENDIX B: Systolic Implementations

The large fan-ins and fan-outs of data during write and read operations can be implemented in various ways. We will discuss one based on systolic arrays, and organization that can permit many queries to be processed in parallel.

Figure B1 shows the memory structure in the form of a matrix. The following notations are used:

$a = (a_1, \ldots, a_n)$	-- the address bits
$d = (d_1, \ldots, d_n)$	-- data bits
$m_i = (m_{i1}, \ldots, m_{in})$	-- address of cell i
$c_i = (c_{i1}, \ldots, c_{in})$	-- contents of cell i
$s = (s_1, \ldots, s_m)$	-- selection bits

Note that $s_i = 1$ if and only if $H(a, m_i) \leqslant D$, where H denotes Hamming distance between two vectors. A write operation follows the rule:

$$c_{ij} \leftarrow s_i \, f(c_{ij}, d_j),$$

where $f(x,1) = x+1$ and $f(x,0) = x-1$. A read operation follows the rule:

$$d_i \leftarrow g\left(\sum_{i=1}^{m} s_i c_{ij}\right),$$

where $g(x)=1$ if $x>0$ and 0 otherwise.

Figure B2 shows the set of counters of the memory array arranged in a two dimensional network. Position 0 of the memory array serves as a port to write in address and data bits, and to read out data bits. The n vertical lines denote

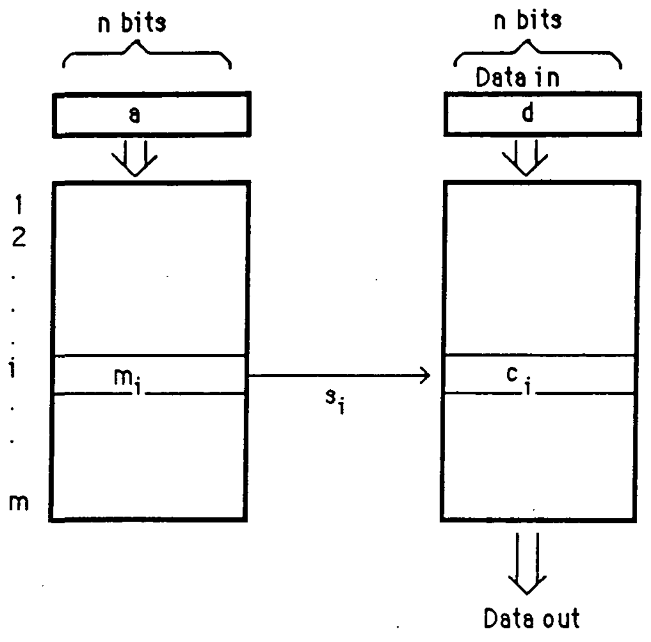

Figure B1. The SDM can be visualized as two $m \times n$ matrices. The rows of the address matrix (left) compared in parallel against the input address a, to produce an *m-bit* selection vector s. Data are written into selected cells by adding (subtracting) one to (or from) counters in the memory array corresponding to 1s (0s) in the data; they are read out by generating 1s whenever a column sum of counters is positive. The address array need not be implemented explicitly; it can be encoded into the logic that determines whatever $s_i = 1$ for a given address.

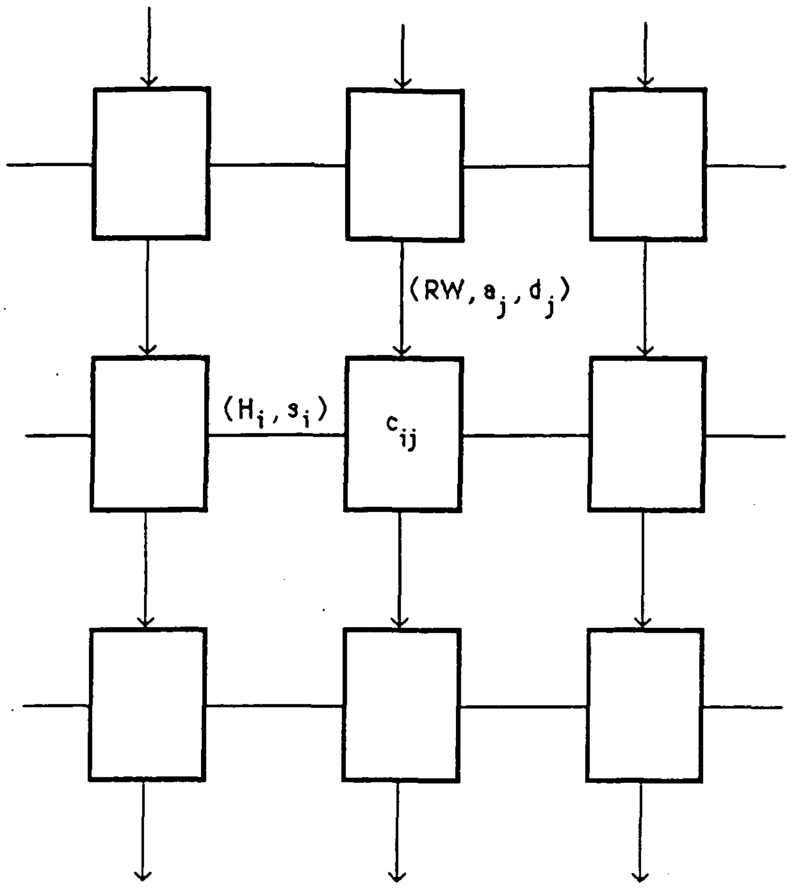

Figure B2. Portion of the memory array. At each vertical time step, a row of query packets is shifted down one position; at each horizontal time step, a column of bookkeeping packets is shifted right one position. The rows and columns are implemented as rings. Row 0 acts as an input/output port; column 0 acts as a bookkeeping-packet initializer.

rings carrying address bits and data; there is wraparound from row m to row 0. The horizontal lines denote rings carrying information used to make the selections of address decoding; these lines also form a ring, with wraparound from column n to column 0.

Information transmitted along the horizontal and vertical rings is contained in packets. At each vertical time step, all vertical packets are shifted one position downward along their rings. Similarly, at each horizontal time step, all horizontal packets are shifted one position rightward along their rings.

It is convenient to describe the operation of the array in two stages. In stage 1, called vertical pipelining, we will assume that the vertical time step is small compared to the horizontal time step: the horizontal ring can be used to determine completely whether the given cell is selected before vertical packets are moved. In stage 2, called full pipelining, we will replace this with the assumption that both the horizontal and vertical time steps are equal.

Let us focus first on a row of vertical packets. Each packet will be of the form (RW,A,B), where RW is a bit indicating read (0) or write (1), A is an address bit, and B is a data field. For a write operation, the row of vertical packets generated at the port at time 0 has the form

$$(1,a_1,d_1)\ ,\cdots,\ (1,a_n,d_n)$$

At time $t=i$, these packets arrive at the row of counters for cell i. The address bits in them can be used to determine whether or not that cell is selected -- i.e., whether $s_i = 1$. (This can be done quickly using a highly parallel circuit such as the one described in Appendix A, or more slowly using a horizontal pipeline technique described below.) After the selection bit is determined, the write rule is performed, and the row of packets is passed downward at time $t = i + 1$.

For a read operation, the row of vertical packets generated at the port at time 0 has the form

$$(0, a_1, 0) \ , \cdots , (0, a_n, 0)$$

At time $t = i$, these packets arrive at the row of counters for cell i. The address bits in them can be used to determine whether or not that cell is selected, and the read rule performed, using the data field as a running sum. At time $t = m$, these packets return to the port; their data fields are then thresholded to obtain the output data bits.

With this organization, a query (consisting of a read/write request, an address, and data) can be entered into the vertical ring at each vertical time step. After the pipeline is filled, the memory array will return one response each time step. The delay for a response to a particular query will be m time steps. For example, if the vertical time step is 1 microsecond and $m = 10^6$, the memory will be capable of processing 1 million queries in parallel and the response time to any one query will be 1 second.

Let's turn now to full pipelining. Consider what must happen in one vertical time step in the stage-1 description: the address decoder must compute

$$H(a, m_i) = \sum_{j=1}^{n} h(a_j, m_{ij}),$$

where $h(x, y) = 1$ if $x \neq y$ and $h(x, x) = 0$. (Note that XORs are not necessary: because $h(x, 1) = \bar{x}$ and $h(x, 0) = x$, the address of a cell, m_i, can be encoded by hard-wiring in a random choice of $\overline{a_j}$ or a_j. at each counter of the memory array.) This sum could be computed by passing a running sum, H_i, along each horizontal row. After n horizontal time steps, the values in the H_i emerges from the rightmost columns; then the selection bits are computed from $s_i = g(H_i)$.

How do we synchronize this computation with the passage of the vertical packets? A simple approach is to stagger the insertion times of the set of packets corresponding to a given query: packet (RW,A_1,B_1) is inserted at time 1, packet (RW,A_2,B_2) at time 2, and so forth. Then, the arrival of packet (RW,A_j,B_j) at counter c_{ij} will be synchronized with the arrival of the horizontal packet containing H_i.

Staggering the insertion of address bits complicates the read/write operations. n time units after the first address bit reaches the first counter of a row, the selection bit s_i of that row will be available; that bit will require an additional n time units to be passed around the horizontal ring a second time. This means that the packets corresponding to a query must be split in two: the address bits are sent out in one wave of packets and, n time units later, the (RW,B) bits are sent out in a second wave. (See Figure B3.) As before, this

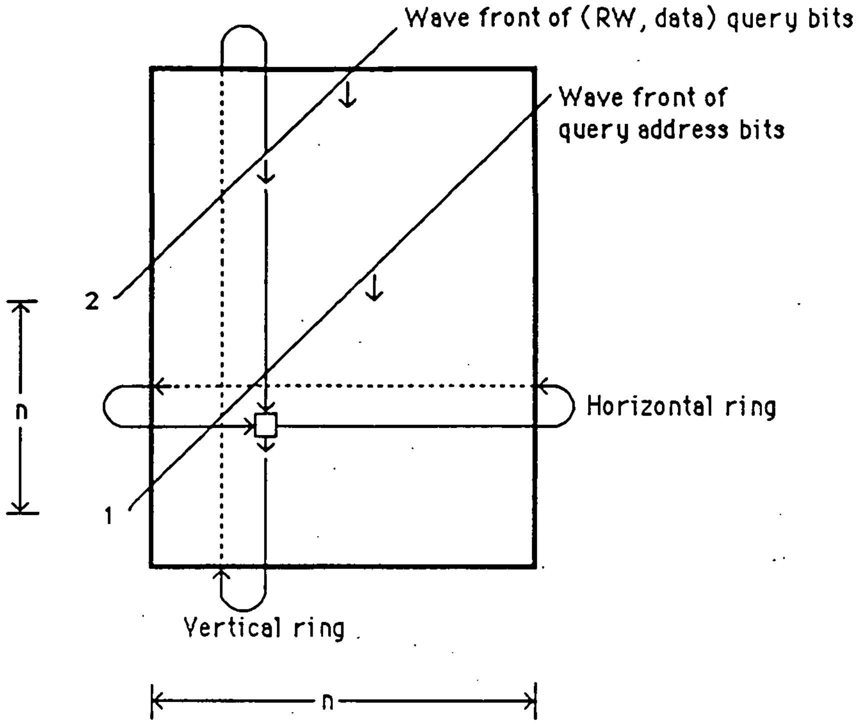
Wave front of (RW, data) query bits
Wave front of query address bits
2
n
1
Horizontal ring
Vertical ring
n

Figure B3. With vertical and horizontal pipelining, horizontal packets carry partial sums for Hamming distances between addresses on Front 1 and memory cell addresses. When Front 2 arrives at cell i the selection bit s_i has been computed and passed along the same ring in synchronization with data-part of the query.

circuit will yield one new response each time step (with smaller time steps than in stage 1), but the delay for a response is $m+2n$ time steps.

Note how fields of queries are distributed among the actual packets traveling along the rings. Each vertical packet contains the address bits for one query and the data bits for the query made n time steps earlier. Each horizontal packet contains the Hamming partial sum (H_i) for one query and the selection bit (s_i) for the query made n time steps earlier.

The systolic organization has high throughput but relatively long response time to any given query. The original model of SDM assumes that the Focus (processing element) generates one query and then awaits one response. Is this organization compatible with the original model? The answer is, may be. Large patterns representing sensory input may well be decomposable into smaller chunks, each representing the local output of a part of the sensor; each chunk can be checked separately for similarity to a pattern stored in the memory, and the result formed by joining the queries. Moreover, one systolic SDM could simultaneously serve many Foci working in parallel.