IN-60
DATE OVERRIDE

43033
P-47

# AN ALTERNATIVE DESIGN FOR A SPARSE DISTRIBUTED MEMORY

*Louis A. Jaeckel*

July 1989

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS Technical Report 89.28

NASA Cooperative Agreement Number NCC 2-387




**Research Institute for Advanced Computer Science**
An Institute of the Universities Space Research Association

# AN ALTERNATIVE DESIGN FOR A SPARSE DISTRIBUTED MEMORY

*Louis A. Jaeckel*

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS Technical Report 89.28
July 1989

**Abstract:** This report describes a new design for a Sparse Distributed Memory, called the *selected-coordinate design*. As in Kanerva's original design, there are a large number of memory locations, each of which may be activated by many different addresses (binary vectors) in a very large address space. In the new design, each memory location is defined by specifying ten *selected coordinates* (bit positions in the address vectors) and a set of corresponding *assigned values*, consisting of one bit for each selected coordinate. A memory location is activated by an address if, for all ten of the location's selected coordinates, the corresponding bits in the address vector match the respective assigned value bits, regardless of the other bits in the address vector. Some comparative memory capacity and signal-to-noise ratio estimates for the two designs are given. A few possible hardware embodiments of the new design are described. The new design is the subject of a patent application filed in 1988.

---

Work reported herein was supported in part by Cooperative Agreements NCC 2-408 and NCC 2-387 between the National Aeronautics and Space Administration (NASA) and the Universities Space Research Association (USRA).

# AN ALTERNATIVE DESIGN FOR A SPARSE DISTRIBUTED MEMORY

## INTRODUCTION

A Sparse Distributed Memory, as proposed by Kanerva (1988), is a memory system that uses addresses that are very long bit strings, or binary vectors, and is able to retrieve stored data quickly if the retrieval information (the read address) is known only approximately.

This report describes an alternative design for a Sparse Distributed Memory. This new design, called the *selected-coordinate design*, is like Kanerva's design in that it consists of a large number of memory locations, called *hard locations*, each of which may be activated by many different addresses in a very large address space. The difference between the two designs is in how the set of addresses that activate a particular hard location is defined. In Kanerva's design, a hard location is activated by any read or write address that is within a fixed Hamming distance of the address of the hard location. The set of addresses that would activate a memory location in the new design will be defined below. Other aspects of the new design, such as the counters at the hard locations and the method of storing and retrieving data in the memory, will remain the same as in Kanerva's design.

Note that in either design a memory location can be

described or represented by giving the set of read or write addresses that would activate it. Conversely, for each possible read or write address, there is a corresponding subset of memory locations, consisting of those locations that would be activated by that address. These relationships will be useful in understanding the designs.

By considering alternative designs for a Sparse Distributed Memory, we can gain greater insight into the ideas underlying the memory system, and when we apply the memory we will be able to choose the design that best fits the particular problem. A later report will describe a more general class of alternative designs, which includes both Kanerva's design and the selected-coordinate design.

I will begin with a review of Kanerva's design and a brief description of the *Stanford prototype*, a small-scale hardware prototype of Kanerva's design. (See Flynn et al., 1988.)

Then I describe the selected-coordinate design, in which each memory location is defined, not by a point in the address space as in Kanerva's design, but by specifying ten *selected coordinates* (bit positions in the address vectors) and a set of corresponding *assigned values*, consisting of one bit for each selected coordinate. A memory location is activated by an address if, for all ten of the location's selected coordinates, the corresponding bits in the address vector match the respective assigned value bits, regardless of the other bits in the address vector.

The new design can be compared to Kanerva's design by

computing the expected number of hard locations in the *access overlap*, that is, the set of memory locations activated by both of two addresses. The ability of the memory to recover a stored data word when reading from the memory at an address near the address at which the data word was written depends on the size of the access overlap as a function of the Hamming distance between two given addresses. The access overlap should be large for two addresses near each other, and small for two addresses that are far apart. I will compute some estimates of comparative memory capacity and some approximate signal-to-noise ratios for the two designs, under some assumptions as to the randomness of the data words written to the memory. These estimates indicate that the performance of the new design is somewhat better than that of Kanerva's design.

One way to implement either design is to have an address decoder for each hard location. During a read or a write operation, each of these address decoders would determine whether its location is to be activated. I will also describe two possible hardware embodiments of the new design, based on some of the design features of the Stanford prototype. The first embodiment uses 256-bit addresses, like the Stanford prototype, but it is somewhat simpler and faster. The second embodiment can use addresses consisting of up to 32,768 bits, but it is slower and more complex than the first embodiment.

The selected-coordinate design is the subject of a patent application, submitted to the U. S. Patent Office in 1988. The descriptions of the design and its hardware embodiments in this

report are based on material in the patent application.

To make the exposition more concrete, and to compare the new design with some examples in Kanerva (1988), I will generally assume that n, the length of the address vectors, is 1000, that there are one million hard locations chosen at random, and that approximately 1/1000 of them are activated by a given read or write address. These numbers could of course be varied.

## APPLICATIONS

A Sparse Distributed Memory system has many potential applications in fields such as signal detection, computer vision, speech recognition and transcription, and robotics, and in other such fields involving adaptive learning and control. The memory system would be the key component in an adaptive system that must deal with complex, noisy real-world information. Data from the environment would enter through sensory systems that would preprocess and encode the data as binary words to be transmitted to the memory. The system would learn, that is, adaptively adjust itself based on a set of "training data", by storing the training data in its memory.

The system would perform the tasks required of it by receiving data patterns from its sensors and then reading from the memory, using the encoded incoming data as the read address. The result of the read operation would be a binary word which in some applications would represent a pattern which the system has been trained to recognize. The patterns to be recognized could be encodings of visual images or symbols, written text, spoken

words, or other such patterns or sequences of data. Since the system can respond to patterns that are similar to those with which it has been trained, it could recognize patterns from input data that are noisy or incomplete.

In other applications, such as robotics or control systems, the result of a read operation would be a word representing command or control information for driving a motor system such as a robot arm. After the memory system has been trained by storing in it the appropriate responses to a set of given situations, the system can quickly respond to a new situation by producing a response similar to the responses it has been taught for similar situations.

## KANERVA'S SPARSE DISTRIBUTED MEMORY

A conventional computer memory system can retrieve stored data only if the retrieval information, such as the read address, is known exactly. The reason for this is that in a conventional computer, a string of binary digits (a data word) is stored at a specific location in the computer's memory, and nothing else is stored at that location at the same time.

P. Kanerva (1988) has proposed a memory system called a Sparse Distributed Memory. The memory is *distributed* in the sense that each data word is stored at many of the memory locations simultaneously, and each memory location contains a linear combination of a subset of the stored data words rather than a single word. A brief description of his system follows.

Let the address space S be the set of all n-bit binary

words. That is, S is the set of all n-dimensional binary vectors, or vectors in which each component is either 0 or 1. S will represent both the set of all possible read or write addresses, and also the set of all possible addresses for memory locations. This distinction will become clear as we proceed. If n is large, say between 100 and 10,000, then the number of possible addresses, $2^n$, is so large that we cannot build a memory with that many memory locations. Therefore, instead of implementing a memory location for each possible address, we choose a large random sample of the addresses, say one million of them, and implement a memory location for each of these addresses. This is why the memory system is called *sparse*. These implemented memory locations will be called *hard locations*.

During a read or a write operation, many of the hard locations are activated, instead of just one, as in a conventional computer. Which hard locations are activated depends on the read or write address; the rule for activating the hard locations is given below. Data will be read from or written to those hard locations that are activated by the read or write address.

For two binary vectors $x = (x_1,\cdots,x_n)$ and $y = (y_1,\cdots,y_n)$ in S, let

$$d(x,y) = \sum_{i=1}^{n} |x_i - y_i| \ .$$

This is known as the *Hamming distance* between x and y. It is the number of coordinates for which $x_i \neq y_i$.

The rule for activating the hard locations is as follows: When a read or a write operation is performed at an address $x$, all of the hard locations whose addresses are within a fixed Hamming distance $r$ of $x$ are activated. This region of activation may be viewed geometrically as a sphere in $S$ with center at $x$ and radius $r$. Suppose, for example, that $n = 1000$ and $r = 451$. Kanerva (1988), p. 62, showed that in that case the sphere contains about $1/1000$ of the points in $S$. Therefore, since the addresses of the hard locations are randomly distributed throughout $S$, the number of hard locations in this sphere is approximately $1/1000$ of the total number of hard locations. If there are one million hard locations, approximately one thousand of them are activated by a given read or write address. The actual number of hard locations activated will vary somewhat, due to the random choice of the hard location addresses.

Note that there are actually two kinds of spheres here: As stated above, the hard locations activated by a given read or write address $x$ are those whose addresses lie in a sphere with center at $x$ and radius $r$. Conversely, for a particular hard location, the set of read or write addresses that would activate it constitutes a sphere in $S$ of radius $r$ centered at the address of that hard location. Thus, if $n = 1000$ and $r = 451$, a hard location would be activated by about $1/1000$ of the possible read or write addresses.

The memory system may be constructed as follows: For each hard location, we could have an address decoder to determine

whether to activate that location during a read or a write operation. The address decoder would compute the Hamming distance between the given read or write address and the address of the hard location, and if that distance is less than or equal to r, it would activate the location.

Assume that the data words to be stored in the memory are m-dimensional binary vectors, for some fixed m. In some applications the data words will be n-dimensional vectors like the address vectors, so that a data word may be thought of as an address, or as a pointer to the memory. Each hard location has m counters for storing the data, one counter for each of the m bit positions, or coordinates, in the data vectors. The method of storing and retrieving the data is as follows:

When a data word (a binary vector) is written to the memory at address x, the word is added to the counters at each of the activated hard locations (those within distance r of x) according to the following rule: For each i, if the value of the $i^{th}$ bit of the data word is 1, the $i^{th}$ counter is incremented; if the value of that bit is 0, the counter is decremented. However, since each counter has finite capacity, it must have an upper and a lower limit. If a counter's limit has been reached, and the system then tries to add another bit that would take the counter beyond its limit, the counter simply remains at its current value. On the other hand, if adding the new bit will keep the counter within its range, the counter is updated. Eight-bit counters, having a range of ±127, should be sufficient for many applications. Thus, if the limits of the

counters are not exceeded, the counters at each hard location will contain a linear combination of those data words for which the location was activated when the data words were written to the memory.

When a read operation is performed at an address y, then, separately for each bit position i in the data words, the values stored in the $i^{th}$ counters of all of the activated hard locations are sent to an accumulator and added. Each of these sums is then compared to a threshold value, and if a sum is greater than the threshold, a 1 is recorded for that coordinate. Otherwise, a 0 is recorded. These 1's and 0's form an m-dimensional binary vector which is the result of the read operation.

Kanerva (1988) showed that if a word is written at address x, and if a read operation is later done at address y near to x in Hamming distance, then many of the hard locations which had been activated by the write operation at x will also be activated by the read at y. Conversely, for a data word stored at an address more distant from y, few or none of the hard locations to which it was written will be activated by the read at y. As a result, the vector of sums computed during the read operation (at address y near x) will contain many copies of the data word written at x, one copy for each of the hard locations activated by both x and y, along with "random noise" due to small numbers of copies of other data words written at more distant addresses. (This will be expressed mathematically in a later section.) Consequently, if x is the only write

address near y, then the binary vector resulting from the read operation at y will be close to the data word originally stored at x. Because of the random noise, some of the bits may not be recovered correctly; however, since the memory system is designed to work with approximate information, its goal in many applications will be achieved if it can recover most of the bits in the stored data word.

Kanerva (1988) computed the expected number of hard locations activated by both a write at x and a read at y, as a function of n, r, and d(x,y). Since the region of activation for x is a sphere of radius r centered at x, and the region of activation for y is a similar sphere about y, the hard locations activated by both x and y are those whose addresses fall in the intersection of the two spheres. This region will be called the *access overlap*. Kanerva (1988) derived a formula for the volume of the intersection of two such spheres, that is, the number of points of S lying in the intersection. Since the hard locations are randomly distributed, the expected number of hard locations in the intersection is proportional to the volume of the intersection. Some representative values of this expected number are given in Table 1 below.

The performance of the Sparse Distributed Memory may be judged by its ability to recover a stored word with some degree of accuracy when we read from the memory at an address near to the address at which the word was written, under the assumptions that a certain number of other words have been written to the memory at other addresses, and that the memory has a given number

of hard locations. Thus it is clear that for the system to perform well, the access overlap must be large if d(x,y) is small, and small if d(x,y) is large. (Which distances are "small" and which are "large" may depend on the particular application.) Some estimates of memory capacity and signal-to-noise ratios will be derived in a later section.

A limitation on the performance of Kanerva's design is imposed by the fact that if the read address is more than a small distance from the write address of the stored data word that is to be recovered, there is a substantial decrease in the size of the access overlap, as can be seen from Table 1. Consequently, it may be difficult to recover the data word if the address is not accurately known, and if many other data words have been written to the memory. It would be better to have an even greater access overlap for small d(x,y) and a smaller access overlap for large d(x,y), so that the signal-to-noise ratio would be improved. We will see that the new design has these properties. Another disadvantage of Kanerva's design is that computing the Hamming distance for each hard location involves summing a large number of bits, an operation that requires specially designed hardware if it is not to be very time-consuming.

The Computer Systems Laboratory at Stanford University has constructed a small-scale prototype of Kanerva's Sparse Distributed Memory, referred to below as the *Stanford prototype*. It is described in Flynn et al. (1988). Two possible hardware embodiments of the selected-coordinate design will be described

below. Since their architecture is similar in some respects to that of the Stanford prototype, I will give a brief description of it here.

The Stanford prototype uses 256-bit addresses and 256-bit data words. It will hold up to 8192 hard memory locations, with 256 eight-bit counters for each hard location. The addresses of the hard locations may be set by the user. The address decoding, that is, determining which hard locations to activate, is done by a custom-designed *address module*. During a read or a write operation, it computes the 256-bit Hamming distance between the *reference address* — that is, the read or write address — and the address of each hard location, one at a time, and compares that distance to a given radius. There is a specially designed set of adders to compute the Hamming distance sum quickly. If the Hamming distance is less than or equal to the radius, which means that the hard location is to be activated, a 13-bit *tag* identifying the hard location is sent to a *tag cache*, a buffer in the address module that holds the tags of the activated hard locations until the data in their counters can be processed.

The process of updating the counters for the activated hard locations during a write, or accumulating the data in those counters during a read, is done by a stack module, which contains the counters, consisting of 256 bytes of memory for each hard location, and a processor to do the additions. Since the stack module receives tags from the tag cache, it can begin working while the address module is continuing to determine which locations should be activated. If the tag cache becomes full (an

unlikely event), the address module must pause until the stack module can catch up.

There is a control module that sends commands and data to the other modules, and there is also an executive module, which functions as a user interface. The overall arrangement of the modules is shown in Figure 1.

The Stanford prototype is designed to perform a read or a write operation in about 1/50 of a second.

## THE SELECTED-COORDINATE DESIGN

I will now describe a different design for a Sparse Distributed Memory system, which I call the *selected-coordinate design*. Compared to Kanerva's design above, the new design can be built with simpler address decoders that operate faster and require fewer connections with other parts of the system. In addition, the new design has better performance than Kanerva's design, due to the fact that when x is near to y, the expected number of hard locations activated by both x and y is greater than in Kanerva's design, while for x and y farther apart the expected number of activated locations is about the same, or somewhat less. (See Table 1 below.) Thus, we will see that the new design is better able to recover a stored word when reading at an address near to where the word was written to the memory.

In a Sparse Distributed Memory, a memory location may be activated by any of a number of different read or write addresses (points in S). Hence, any memory location may be described, or represented, by specifying the subset of S consisting of those

addresses that would activate the location. We saw above that in Kanerva's design these subsets are spheres centered at points in S. The new design differs from Kanerva's design in the way in which the subsets of S representing memory locations are defined. These subsets will not be centered at points in S, as they are in Kanerva's design. Other features of the new design, such as the counters at the hard locations and the way in which they are written to and read from, will remain the same as in Kanerva's design.

To describe the new design, I will designate a large collection of subsets of the space S, each containing a certain proportion of the points in S, say about 1/1000 of them. Each of these subsets will represent a potential memory location, in the sense that the subset is the set of all addresses in S that would activate that location. A certain number of these potential memory locations, say, one million, are chosen at random and implemented as the hard locations. I will also assume that n, the dimension of S, is 1000, so that the number of points in S is $2^{1000}$. Each of the above numbers could of course be varied. Since Kanerva (1988) used these numbers in some of his examples, their use here will facilitate comparison of the two designs.

I will define a subset of S as follows: Select any ten of the n = 1000 coordinates, for example, the $37^{th}$, the $152^{nd}$, and so on. Assign a value of 0 or 1 to each of the ten selected coordinates, say, 1 for Bit 37, 0 for Bit 152, and so on. The subset defined is then the set of all points (binary vectors) in

S such that, for all ten of the selected coordinates, the corresponding components of the vector agree with the respective assigned values for the selected coordinates; the values of the bits for the other 990 coordinates are free to be either 0 or 1. In the example above, the subset is the set of all binary vectors in S whose $37^{th}$ bit is 1 and whose $152^{nd}$ bit is 0, etc. This subset represents a potential memory location, in the sense that the memory location would be activated by any address in the subset.

Thus a subset of this kind is defined by giving its ten *selected coordinates* and the ten corresponding *assigned values*. The number of such subsets is

$$\binom{1000}{10} \times 2^{10}$$

(approximately $3 \times 10^{26}$), where the first term is the number of ways of selecting ten coordinates out of 1000, and the second term is the number of ways of assigning values of 0 or 1 to each of the ten selected coordinates. These subsets are isomorphic to one another in the sense that they can be mapped onto one another by permuting the order of the coordinates and interchanging the values 0 and 1 on the individual coordinates. Each of these subsets represents a potential memory location in the new design. A certain number of them are chosen at random to be implemented as hard locations. (Each of these randomly chosen subsets will most likely be based on a different set of ten selected coordinates.)

Each such subset contains $2^{990}$ points, which is 1/1024

of the entire space S. Note that the number of selected coordinates is ten, so that each memory location would be activated by 1/1024 of the addresses in S. If we wanted each subset to contain some other proportion of S, such as $1/2^q$ of S, we would define the memory locations by selecting q coordinates to be given assigned values, instead of ten.

For a given read or write address, that is, a point in S, the class of potential memory locations activated by it (when reading or writing at that address) is represented by the class of subsets of S of the kind defined above that contain the point. A subset of this kind contains the given point if and only if, for all ten of the selected coordinates which define that subset, the coordinates of the point agree with the assigned values. In other words, a memory location is activated by this address if the assigned values for all ten of its selected coordinates match the corresponding components of the given address. The number of potential memory locations activated by a given address is therefore

$$\binom{1000}{10},$$

since this is the number of ways of selecting ten coordinates with which to define a subset. Once the ten coordinates have been selected, there is only one way to assign values to them so that they all agree with the given address. This number is 1/1024 of the total number of potential memory locations. If there are one million hard locations selected at random, the expected or average number of hard locations activated by an

address is 1,000,000/1024 = 976.56. The actual number will vary, but will be between 945 and 1008 for the majority of the addresses in S.

We have a kind of duality here: Each memory location corresponds to a subset of S (those addresses that activate the location), and each address in S corresponds to a subset of memory locations (those locations that are activated by the address). In Kanerva's design, a subset of the first kind is a sphere in S with Hamming radius r centered at the address of a memory location, while a subset of the second kind is a sphere with the same radius, but centered at a read or write address. In the new design, these two kinds of subsets are very different from each other because they are subsets of different sets. In either design, however, the access overlap for two addresses is the intersection of two subsets of the second kind.

In Kanerva's design it is obvious that the number of addresses that activate a given memory location is the same as the number of potential memory locations that are activated by a given address. In the new design these numbers cannot be compared directly. But we can compare proportions: We saw above that in the new design each subset of the first kind contains 1/1024 of the points in S, and that each subset of the second kind contains that same proportion of the set of potential memory locations. This equivalence is true more generally, as I will now show.

In any design for a Sparse Distributed Memory, we have a set S of addresses, indexed by x, and a set M of potential memory

locations, indexed by $\mu$. As above, each memory location $\mu$ corresponds to a subset of S (the addresses that activate $\mu$), and each address x corresponds to a subset of M (the memory locations activated by x). Define the following:

Let $A(x,\mu) = 1$ if the address x activates the memory location $\mu$; if it does not, let $A(x,\mu) = 0$.

Let $P_x = \frac{1}{|M|} \sum_{\mu \epsilon M} A(x,\mu)$ be the proportion of memory locations that are activated by the address x.

Let $Q_\mu = \frac{1}{|S|} \sum_{x \epsilon S} A(x,\mu)$ be the proportion of addresses that activate the memory location $\mu$.

In both Kanerva's design and the new design, $P_x$ has the same value P for all x, and $Q_\mu$ has the same value Q for all $\mu$. This may not be true for other designs.

The overall average of $A(x,\mu)$ is

$$\frac{1}{|S| \cdot |M|} \sum_x \sum_\mu A(x,\mu) = \frac{1}{|S|} \sum_x P_x = \underset{x}{\text{ave}} \{P_x\}$$

$$= \frac{1}{|M|} \sum_\mu Q_\mu = \underset{\mu}{\text{ave}} \{Q_\mu\} .$$

That is, the average of the $P_x$ equals the average of the $Q_\mu$. In other words, if in a "typical" read or write operation the address activates, say, 1/1000 of the memory locations, then a "typical" memory location will be activated about once in every thousand read or write operations. If all $P_x = P$ and all $Q_\mu = Q$, as in the two designs considered here, then $P = Q$.

Given two addresses x and y with Hamming distance d(x,y) between them, we need to know the size of the access

overlap, that is, the number of potential memory locations activated by both addresses. The expected number of hard locations activated by both addresses is proportional to this number. If x is a write address and y is a read address, then, as in Kanerva's design, the number of copies of the word written at x that will be contained in the sums computed by a read at y is the number of hard locations activated by both x and y.

A memory location will be activated by both x and y if and only if both x and y agree with the assigned values for all ten of the selected coordinates which define that memory location. But this can happen only if x and y agree with each other on each of those ten coordinates. In other words, all of the location's selected coordinates must be among the coordinates on which x and y agree, and for each of those selected coordinates the assigned value must agree with the common value of x and y. For example, with the numbers used above, if $x_{37} = y_{37} = 1$, $x_{152} = y_{152} = 0$, etc., then the memory location defined by: Bit 37 = 1, Bit 152 = 0, etc. would be activated by both x and y. The number of potential memory locations activated by both x and y may be found as follows: If $d(x,y) = d$, then x and y differ on d coordinates and agree on $1000 - d$ coordinates. If a potential memory location is activated by both x and y, all ten of the selected coordinates defining it must be among the $1000 - d$ coordinates on which x and y agree. The number of ways in which ten coordinates can be selected so that x and y agree on all of

them is therefore

$$\binom{1000-d}{10},$$

the number of subsets of size ten that can be chosen from a set of size 1000 - d. As for the assigned values, if a memory location is to be activated by both x and y, the values assigned to its selected coordinates must agree with the corresponding components of x and y. Therefore, one and only one potential memory location defined by a given set of ten selected coordinates on which x and y agree can be activated by both x and y, so the number above is the number of potential memory locations activated by both x and y.

The expected number of hard locations in the access overlap, that is, the expected number activated by both x and y, may be found by multiplying the number above by the ratio of the number of hard locations to the total number of potential memory locations. If there are one million hard locations, this expected number is:

$$\frac{1{,}000{,}000}{2^{10} \times \binom{1000}{10}} \times \binom{1000-d}{10} = 976.56 \times \frac{(1000-d)\cdots(991-d)}{1000 \cdot \cdots \cdot 991}.$$

Some examples are given in Table 1 below.

## COMPARISON OF THE SELECTED-COORDINATE DESIGN TO KANERVA'S DESIGN

The new design can be constructed with simpler hardware than can Kanerva's design. One way to implement either design is to have an address decoder for each hard location, as was described above for Kanerva's design. These address decoders would

simultaneously determine whether each location is to be activated. For the new design, the address decoder for each hard location would have ten inputs from the area where the current read or write address is stored, one for each of the selected coordinates defining that location. The decoder would compare the value of each of the ten inputs with the respective assigned value for that selected coordinate; if all ten match, the location would be activated. Thus the address decoder is simply a device to compare the ten address bits with the assigned value bits, followed by an "AND" gate whose inputs are the results of the ten comparisons. Since the complexity and speed of these address decoders do not depend on the value of n, the same address decoders could be used in a memory system with any value of n, no matter how large. In Kanerva's design, however, assuming n = 1000, each address decoder would have 1000 inputs, from which it must compute a Hamming distance, an operation that involves comparing and adding 1000 bits. The address decoders in the new design, therefore, are simpler, work faster, and have fewer input connections. Also, we will see that the possible hardware embodiments of the new design, described below, are simpler in some respects than the Stanford prototype.

The following table compares the size of the access overlap for the new design to that for Kanerva's design, for selected values of d(x,y). I assume that n = 1000, that there are one million hard locations in each design, and that in Kanerva's design the radius of the spheres is 451, since the volume of a sphere with this radius is approximately 1/1000 of S. The

numbers in the middle column of the table are taken from Table 7.1 in Kanerva (1988), p. 63. The numbers in the third column are computed from the formula above.

TABLE 1

Expected number of hard locations activated by both x and y

| d(x,y) | Kanerva's design | New design |
|---|---|---|
| 0 | 1000 | 977 |
| 1 | 894 | 967 |
| 10 | 743 | 883 |
| 50 | 445 | 583 |
| 100 | 267 | 339 |
| 150 | 162 | 191 |
| 200 | 97 | 104 |
| 300 | 30 | 27 |
| 400 | 7 | 6 |
| 500 | 1 | 0.91 |

To compare the performance of the two designs, I will make some assumptions so that I can compute approximate memory capacities and signal-to-noise ratios for each design. Although these assumptions may be simplistic, a comparison of the numbers obtained for the two designs under the same conditions will give us an idea of their relative performance. I will assume that both designs are constructed using the parameter values on which Table 1 is based.

Assume that the data in a "training set" have been written

to the memory. The training set consists of $t$ write addresses $x_1, x_2, \cdots, x_t$ and an m-bit data word for each address. These data words are written to the memory by the method described above: The counters in the activated hard locations are incremented or decremented according to whether the corresponding data bit is 1 or 0. No "retraining" of the memory to improve its response is done. The number of words written to the memory might be in the thousands or tens of thousands. Kanerva (1988), Keeler (1988), and Chou (1988) have studied the memory capacity of Kanerva's design, assuming large $n$ and a large number of hard locations. They give formulas for the approximate number of data words that can be stored in the memory, under some assumptions of randomness.

Suppose that we are reading at address $y$, and that there is one and only one address, say $x_1$, near $y$, at which a data word was written. The goal is to recover the data word written at $x_1$. I will also assume that all of the other write addresses are randomly scattered about $S$, and that the data words written to the memory are random. These assumptions will be made more precise below. If a point in $S$ is chosen at random, its expected distance from $y$ is $n/2 = 500$, and with very high probability its distance from $y$ will be in the range 450 to 550. Thus I will assume that for all of the write addresses except $x_1$, their distance from $y$ is in that range.

We can see from Table 1 that when we read at $y$, each of the write addresses, other than $x_1$, has a very small access overlap with $y$. Thus, when the contents of the counters of the

activated hard locations are summed, the contributions to the sums of the counters due to the data words written at those other addresses will be only a few copies, if any at all, of each of the words written at those addresses. The word written at $x_1$ will of course be included in these sums many times, once for each hard location in the access overlap of $x_1$ and $y$. Thus, when we compute these sums and compare them with appropriate threshold values, we should recover most of the individual bits of the word written at $x_1$. We may not recover all of the bits correctly, because it may happen that for a small proportion of the coordinates, the "random noise" will overwhelm the multiple copies of the correct value, and we will lose those bits. But, in many applications, it is sufficient to recover most of the bits correctly, if we are given only approximate retrieval information, that is, the address $y$.

If $y$ is very close to $x_1$, that is, if $d(x_1,y)$ is considerably less than 50, then in either design the access overlap is so large that we should be able to recover the stored word accurately, even in the presence of a large amount of noise. On the other hand, if $d(x_1,y) > 200$, the access overlap is small, and about the same, in both designs, so if $t$ is large it will be difficult to recover the stored word accurately. Kanerva (1988) showed that for his design, under some assumptions similar to those made here, a distance of 200 between $x_1$ and $y$ is nearly the outer limit for recovering the word stored at $x_1$ by reading at $y$. Since his argument is based on the size of the access overlap as a function of the distance between $x$ and $y$,

Table 1 shows that a similar argument holds for the new design. So I will compare the designs for $d(x_1,y) \leq 200$.

To compare the designs, it will simplify matters if I assume that the write addresses $x_i$ and the read address $y$ are given, with $y$ near $x_1$ and approximately 500 away from the other $x_i$. There are two remaining sources of random fluctuation: the bits in the data words, which I will assume are random, and the random sample of potential memory locations to be implemented as hard locations.

It should be noted that there are many possible ways to construct a probability model for this process. A more realistic approach might be to think of the read address $y$ as a noisy approximation of the "target" address $x_1$, and therefore to treat $y$ as a random vector, rather than as fixed. Thus, in order to analyze the situation where $y$ is a given distance $d$ from $x_1$, we could assume that $y$ has a uniform probability distribution over the set of points in $S$ that are a distance $d$ from $x_1$. Computing the variance of the noise in such a model would be more complex than under the assumptions made here. In constructing a probability model, there is the general question of which factors to consider fixed and which to consider random. For example, we might consider the choice of hard locations to be fixed, since in practice they would be fixed in advance. The same could be said for the data words, since they would already have been written to the memory. The assumptions made above will allow the computations to be relatively simple, and should give us a valid comparison of the two designs, even if the actual numbers derived

for each design are only rough approximations.

Consider one of the $m$ bit positions in the data words. Since the data in each of these bit positions is processed separately, the same analysis would apply to each bit position. For each data word, define a random variable $B_i$ corresponding to the bit in the $i^{th}$ data word in the bit position under consideration as follows: If the bit in the $i^{th}$ data word is 1, let $B_i = 1$, and if the bit is 0, let $B_i = -1$. These values correspond to incrementing or decrementing the counters when the word is written to the memory. I will assume that $P(B_i = 1) = P(B_i = -1) = 0.5$, so that $E(B_i) = 0$. I also assume that each $B_i$ is independent of the other $B_j$ $(j \neq i)$, that the values of the $B_i$ are unrelated to the write addresses or to the read address, and that the $B_i$ are independent of the random choice of hard locations. Since the goal in reading at $y$ is to recover the data bit represented by $B_1$, we can assume that $B_1$ is fixed.

For each write address $x_i$, let the random variable $L_i$ be the number of hard locations in the access overlap of $x_i$ and $y$. The expected value $\lambda_i$ of $L_i$ is a function of $d(x_i,y)$, depending on the design; the formulas derived below apply to both designs, except that the values of the $\lambda_i$ are different for the two designs. (Table 1 gives values of $\lambda_i$ for each design.) Since the hard locations are chosen at random, and $\lambda_i$ is small compared to the total number of hard locations, the distribution of $L_i$ is very close to a Poisson distribution. We can therefore approximate the variance of $L_i$ by $Var(L_i) = E(L_i) =$

$\lambda_i$, from which it follows that

$$E(L_i^2) = Var(L_i) + [E(L_i)]^2 = \lambda_i + \lambda_i^2 \ .$$

Since the read and write addresses are assumed to be fixed, it follows from the assumptions above that each of the $B_i$ is independent of all of the $L_i$.

When we read at y, the contents of the counters at the hard locations activated by y are summed. Let $\Sigma$ be the sum of the counters at the activated locations for the bit position under consideration. Since the counter at each of those locations contains the sum of the $B_i$ corresponding to the $x_i$ that activated that location, it follows that for each i, the value $B_i$ occurs in the sum $\Sigma$ once for each hard location activated by both $x_i$ and y. Therefore,

$$\Sigma = \sum_{i=1}^{t} L_i B_i \ .$$

See Kanerva (1988), p. 67.

Since the goal is to recover $B_1$, which is assumed to be fixed, we can rewrite the sum as

$$\Sigma = L_1 B_1 + \sum_{i=2}^{t} L_i B_i$$

$$= \lambda_1 B_1 + (L_1 - \lambda_1) B_1 + \sum_{i=2}^{t} L_i B_i \ .$$

In the last expression above, I will regard $\lambda_1 B_1$ as the *signal* ($\lambda_1$ is the expected number of copies of $B_1$ in $\Sigma$), and the other terms, which contain the random variables, as the *noise*.

The first noise term is due to the uncertainty in the number of copies of $B_1$ in $\Sigma$, and the other noise terms are due to the other data words.

We can now compute the expected value and the variance of the noise. The derivation is similar to that given by Chou (1988). Since $E(L_1) = \lambda_1$ and $B_1$ is fixed, we have $E[(L_1 - \lambda_1)B_1] = 0$; since for $i > 1$, $B_i$ is independent of $L_i$ and $E(B_i) = 0$, we have $E(L_iB_i) = E(L_i)\cdot E(B_i) = 0$. So the expected value of the noise is 0.

The variance of a sum is the sum of the variances and the covariances of the summands. First I will show that under the above assumptions all of the covariances are 0. Since $B_1$ is fixed and the other $B_i$ are independent of each other and of all of the $L_i$, we have, for $2 \le i \le t$,

$$\begin{aligned}
&\mathrm{Cov}[(L_1 - \lambda_1)B_1,\ L_iB_i] \\
&\quad = E[(L_1 - \lambda_1)B_1L_iB_i] - E[(L_1 - \lambda_1)B_1]\cdot E(L_iB_i) \\
&\quad = E[(L_1 - \lambda_1)B_1L_i]\cdot E(B_i) - 0\cdot 0 \\
&\quad = 0\ .
\end{aligned}$$

And for $2 \le i < j \le t$,

$$\begin{aligned}
&\mathrm{Cov}(L_iB_i, L_jB_j) \\
&\quad = E(L_iB_iL_jB_j) - E(L_iB_i)\cdot E(L_jB_j) \\
&\quad = E(L_iL_j)\cdot E(B_i)\cdot E(B_j) - 0\cdot 0 \\
&\quad = 0\ .
\end{aligned}$$

So the variance of the noise is the sum of the variances of the noise terms. First, since $B_1 = \pm 1$, $\mathrm{Var}[(L_1 - \lambda_1)B_1] = \lambda_1$. And for $2 \le i \le t$,

$$\begin{aligned} \mathrm{Var}(L_i B_i) &= E(L_i^2 B_i^2) - [E(L_i B_i)]^2 \\ &= E(L_i^2 \cdot 1) - 0 \\ &= \lambda_i + \lambda_i^2 \ , \end{aligned}$$

since $B_i^2$ is always 1. Therefore, the variance of the noise is

$$\lambda_1 + \sum_{i=2}^{t} (\lambda_i + \lambda_i^2) \ .$$

If $d(x_i,y) = 500$, then, using the values in Table 1, we see that for Kanerva's design, $\lambda_i + \lambda_i^2 = 2$, and for the selected-coordinate design it is 1.74. Since I am assuming that for $2 \leq i \leq t$, $d(x_i,y)$ is near 500, and since for both designs $\lambda_i$ in this range is a gradually decreasing function of the distance, I will approximate $\lambda_i + \lambda_i^2$ by 2 for Kanerva's design and by 1.74 for the new design. This will result in a slight underestimate of the variance of the noise for both designs. (See Keeler, 1988, and Chou, 1988, for more detailed treatments of the variance of the noise.) The approximate variance of the noise for Kanerva's design is then

$$\lambda_{1,k} + 2(t - 1) \ ,$$

and for the new design it is

$$\lambda_{1,s} + 1.74(t - 1) \ .$$

The subscripts $k$ for Kanerva's design and $s$ for the selected-coordinate design indicate that the value of $\lambda_1$ is different for the two designs.

After $\Sigma$ is computed, it is compared to a threshold value. If $\Sigma$ is above the threshold, the data bit recovered by the read

operation is a 1; otherwise it is a 0. Since $\Sigma$ is the sum of a constant, $\lambda_1 B_1$, plus a sum of uncorrelated noise terms, the distribution of $\Sigma$ is approximately normal with mean $\lambda_1 B_1$ and variance as given above. Therefore, for a given threshold value, we can use the normal distribution to find the approximate probability of correctly recovering the data bit represented by $B_1$. I will use 0 as the threshold value so that the probability of recovering the data bit correctly is approximately the same, whether $B_1$ is 1 or -1. For example, if $d(x_1,y) = 100$ and if we assume that $B_1 = 1$, then, for Kanerva's design, $\Sigma$ is approximately normally distributed with mean 267 and variance $267 + 2(t - 1)$. Therefore,

$$Z = \frac{\Sigma - 267}{\sqrt{267 + 2(t - 1)}}$$

is approximately a standard normal random variable (mean 0 and variance 1), and $\Sigma > 0$ is equivalent to

$$Z > \frac{-267}{\sqrt{267 + 2(t - 1)}} .$$

Thus, for a given number, $t$, of stored data words, the probability of correctly recovering $B_1$ under the conditions above is the probability that $Z$ satisfies this inequality.

Conversely, if we want the probability of recovering $B_1$ to be, say, 99% when $d(x_1,y) = 100$, we can compute the maximum value of $t$ for which, under the conditions above, this probability will be at least 99%. Since $Z$ is approximately normal, we have $P(Z > -2.33) \cong 99\%$. Therefore, we need to find the value of $t$ for which the right side of the inequality above

is -2.33. Solving for $t$ in the equation

$$\frac{-267}{\sqrt{267 + 2(t - 1)}} = -2.33 ,$$

we find $t_k \cong 6{,}433$ data words.

If we do the same computations for the new design, we have

$$Z = \frac{\Sigma - 339}{\sqrt{339 + 1.74(t - 1)}} ,$$

and if we solve for $t$ in the equation

$$\frac{-339}{\sqrt{339 + 1.74(t - 1)}} = -2.33 ,$$

we find $t_s \cong 11{,}972$ data words. This is 86% more than the value found above for Kanerva's design. In other words, under these conditions, if the new design has 86% more stored data words than Kanerva's design, the two designs will have the same probability of recovering a data bit.

We can define a *signal-to-noise ratio* as the size of the signal, $\lambda_1$, divided by the standard deviation of the noise. If $t$ is large, we can simplify the formulas by omitting the first term in the variance expressions above; although doing this will somewhat underestimate the noise for both designs, we will still have a fair comparison of the two designs. The approximate signal-to-noise ratio for Kanerva's design is then $\lambda_{1,k}/\sqrt{2t}$, and for the new design it is $\lambda_{1,s}/\sqrt{1.74t}$.

As stated earlier, I want to compare the designs when $d(x_1,y) \leq 200$. For example, if $d(x_1,y) = 50$, the approximate signal-to-noise ratio is $445/\sqrt{2t} = 315/\sqrt{t}$ for Kanerva's design and $583/\sqrt{1.74t} = 442/\sqrt{t}$ for the new design, an improvement of

40% for a given $t$. Another way to express this comparison is to compute the relative number of data words that can be stored in the memory so that the signal-to-noise ratio is the same for both designs. If we store $t_k$ words in Kanerva's design and $t_s$ words in the new design, and set their signal-to-noise ratios equal to each other, we have

$$\frac{315}{\sqrt{t_k}} = \frac{442}{\sqrt{t_s}} ,$$

and we find that $t_s = 1.97t_k$. That is, the new design can store 97% more data words than can Kanerva's design, with the same signal-to-noise ratio when $d(x_1,y) = 50$.

Repeating the above computations for $d(x_1,y) = 100$, we find signal-to-noise ratios of $189/\sqrt{t}$ and $257/\sqrt{t}$ for the two designs, an improvement of 36% for a given $t$. Setting these ratios equal to each other, we find that $t_s = 1.85t_k$, so the new design can store 85% more data words than can Kanerva's design and achieve the same signal-to-noise ratio. (This is very close to the 86% figure obtained above, which is a more accurate figure for the particular assumptions under which it was computed.)

For $d(x_1,y) = 150$, the signal-to-noise ratios are $115/\sqrt{t}$ and $145/\sqrt{t}$, an improvement of 26%, from which we find that $t_s = 1.59t_k$, so the new design can store 59% more data words with the same signal-to-noise ratio.

Finally, for $d(x_1,y) = 200$, the signal-to-noise ratios are $69/\sqrt{t}$ and $79/\sqrt{t}$, an improvement of 14%, from which we find that $t_s = 1.31t_k$, so the new design can store 31% more data words with the same signal-to-noise ratio. For distances beyond 200 and for

large t, it will be difficult for either design to recover stored data words accurately.

## POSSIBLE HARDWARE EMBODIMENTS

I will now describe two possible hardware embodiments of the selected-coordinate design. Certain parts of their architecture, such as the stack module and the tag cache, are similar to the Stanford prototype, described above, that has been built to implement Kanerva's design.

As we saw earlier, either of the Sparse Distributed Memory designs above could be implemented by having an address decoder for each hard location. During a read or a write operation, these address decoders would function simultaneously, each determining whether its location is to be activated. The address decoding would thus be done very quickly. In practice, however, it would be expensive to include so many such units in the system, and for many applications adequate speed could be attained by having one or more specially constructed units to perform the address decoding for the hard locations one at a time. Each of the embodiments described below has one such unit, called an *address module*, for this purpose. However, either of these embodiments could include several identical address modules, each working in parallel on a different subclass of the hard locations. The address decoding could then be done faster, or more hard locations could be handled in the same amount of time.

The new design cannot be run on the Stanford prototype

because the new design is based on a different method for determining which hard locations are to be activated. In the new design, a hard location is indifferent to the values of most of the bits in the read or write address. More importantly, each hard location is indifferent to a different subset of the address bits. The Stanford prototype does not provide for computing the Hamming distance based on a different subset of the address bits for each hard location. (The Stanford prototype does allow for a "global" mask, so that a given subset of the 256 address bits can be used in a read or a write operation, but that subset must remain the same throughout the operation. The new design requires that a different subset be used for each hard location.)

The overall arrangement of the modules in the two embodiments of the new design is similar to that in the Stanford prototype, and is shown in Figure 1. The executive module is the user interface to the Sparse Distributed Memory. It consists of a computer workstation, with software allowing the user to define the selected coordinates and assigned values for the hard locations, write to and read from the memory, and perform various debugging operations.

The memory system itself contains three modules. The address module, the key element in each of the two embodiments herein, is designed differently in each case; these designs will be described in detail below. The stack module in each embodiment, consisting of a processor and a large amount of memory for the counters, is like the one in the Stanford prototype, and would function in the same way. The control

module receives commands and data from the executive module, sends data to it, passes tags from the tag cache to the stack module, and generally controls the functioning of the address and stack modules.

The address module for the *first embodiment* of the new design is shown in Figure 2. As in the Stanford prototype, a clock unit contains a clock generator, various registers, and a tag counter, which acts as a pointer to the hard locations. During a read or a write operation, the tag counter is successively incremented so that it points to each of the hard locations in turn. A hard address unit stores the addressing information — the selected coordinates and their assigned values — which defines each of the hard locations. A logic unit compares the *reference address* (the read or write address) to the information defining each hard location, one location at a time, and determines whether that location is to be activated. As explained earlier, a location is activated if the assigned values for all of its selected coordinates agree with the corresponding bits in the reference address. If a location is to be activated, a *tag* identifying the hard location (the tag is the current value in the tag counter) is stored in a *tag cache*, which is a buffer like the one in the Stanford prototype. These tags are then sent to the stack module, where the data in the counters is processed. The tag cache is not an essential part of the design, but it enables the system to function more efficiently.

I will assume that this embodiment uses 256-bit addresses and has 8192 hard locations, like the Stanford prototype,

although these numbers could be varied.

The information defining each hard location is stored in the hard address unit as two 256-bit words, one in each half of that unit. The first word for each hard location contains the assigned values for the selected coordinates, in the positions corresponding to those selected coordinates; the other bits in this word may have any values. The second word acts as a mask: It has a 1 in the position corresponding to each selected coordinate, and all of the other bits are 0's. Hence the hard address unit must have 64 bytes of memory (512 bits) for each hard location, twice the amount used in the Stanford prototype. Although this method of storing the information is a somewhat inefficient use of memory space, it allows us to have a relatively simple logic unit. The tag counter has a line to each half of the hard address unit, so that it can point simultaneously to both parts of the information defining a hard location.

The logic unit works as follows: At the beginning of a read or a write operation, a 256-bit reference address register (which could be constructed as in the Stanford prototype) receives the reference address from the control module. Then, for each hard location, the following steps are performed: An array of logic elements performs a 256-bit exclusive-or (XOR) of the reference address and the word from the first half of the hard address unit, which contains the assigned values for the selected coordinates of the hard location pointed to by the tag counter. Then another logic array performs a 256-bit logical AND of the

result of the XOR and the word from the second half of the hard address unit, the mask containing 1's at the positions of the selected coordinates for that hard location.

The resulting 256 bits have the following values: If a bit position is not a selected coordinate, the bit is a 0 (due to the AND). If a bit position is a selected coordinate, then the bit is a 0 if the corresponding bit in the reference address agrees with the assigned value, and the bit is a 1 if they disagree (due to the XOR). It follows that the hard location is to be activated if and only if all 256 of the resulting bits are 0's. One way to test the 256 bits for all 0's is to send them to a two-stage array of logic elements. The first stage consists of 16 elements, each having 16 inputs. Each element sends a 0 to the second stage if and only if all of its inputs are 0's; otherwise it outputs a 1. The second stage consists of one element just like the elements in the first stage; it outputs a 0 if all 16 of its inputs are 0's. If the output of the second stage is a 0, indicating that the hard location is to be activated, the location's tag is stored in the tag cache. Then, whether or not the hard location is activated, the tag counter is incremented and the logic unit proceeds with the next hard location.

This address module can run somewhat faster than the one in the Stanford prototype, because the latter must compute the sum of the 256 bits resulting from the XOR of the reference address and each hard location address, whereas the module above simply performs a logical AND, and tests the 256 bits for all zeros.

For clarity, a few counters, logic elements, and data lines are not shown in Figures 2 or 3. For example, there must be a line from the control module to the hard address unit so that the information defining the hard locations can be sent to that unit; counters and logic elements to control the tag cache; and a line to signal that the tag cache is full.

Many applications may require addresses that are longer than 256 bits. The architecture of the embodiment described above could be modified so that longer addresses can be used, but the hard address unit and the logic unit would require proportionately more hardware. For example, to allow for 512-bit addresses, about twice as many components would be needed for those units.

The *second embodiment* of the new design, whose address module is shown in Figure 3, can be used in applications where the addresses are very long binary words. Addresses consisting of many thousands of bits might be required, for example, in cases where the addresses represent complex visual or auditory information. In the description given below, it is assumed that the reference addresses can consist of as many as $2^{15} = 32,768$ bits, although this number could be varied. (Note that the number of counters per hard location in the stack module need not be the same as the number of bits in the addresses; although in principle we can have any number of counters per hard location, to implement 32,768 counters per hard location would require a huge amount of memory.)

The underlying principle of the second embodiment is the way

in which the information defining the hard locations is stored in its hard address unit. A hard location is defined by giving its selected coordinates and their assigned values. If the addresses are 32,768-bit words, a coordinate (a bit position) may be indicated by a number between 0 and 32,767, which may itself be represented by a 15-bit binary number. Therefore, a selected coordinate, together with one bit for its assigned value, may be represented by a 16-bit word, which can be stored in two bytes of memory. Thus, assuming ten selected coordinates per hard location, each hard location requires 20 bytes of memory in the hard address unit, compared to 64 in the first embodiment. Packing the assigned value bit with the number of the selected coordinate in this way is of course not necessary, but it is convenient.

As in the first embodiment (and the Stanford prototype), a tag counter steps through the hard locations, and a logic unit determines whether each location is to be activated, one location at a time. Components of the address module not described here are assumed to be like those in the first embodiment.

The logic unit is a processor that does the following: At the beginning of a read or a write operation, it receives from the control module the length n of the reference address (up to 32,768 bits), the number of selected coordinates per hard location (up to ten in this example, although the number could be greater), and the reference address. The logic unit stores the reference address in a set of internal memory locations, one bit per location, so that it will have direct access to each bit.

(These internal locations could be wired directly to the control module.)

The logic unit then performs the following steps for each hard location: For each selected coordinate, it receives the two bytes from the hard address unit giving the number of that selected coordinate and its assigned value. It shifts this two-byte word one bit, to separate the bit for the assigned value from the number of the selected coordinate. The latter is used as an internal address to retrieve the corresponding bit in the reference address. This reference address bit is then compared to the assigned value bit. If these bits are equal, the logic unit repeats the above steps with the next selected coordinate for that hard location, and then the next, and so on, as long as the bit in the reference address corresponding to the selected coordinate is equal to the assigned value bit. If the bits match for all of the selected coordinates, then the hard location is to be activated; in that case its tag is stored in a tag cache, as in the first embodiment. However, if for any selected coordinate the bits are unequal, that hard location will not be activated, so the logic unit immediately proceeds to the next hard location.

We can now make a rough estimate of the relative speed of the first and the second embodiments of the new design. Assuming that the selected coordinates and their assigned values are chosen at random, we can see that the logic unit in the second embodiment will eliminate most of the hard locations after checking only a few of the selected coordinates: Half of the hard locations will be eliminated on the first selected

coordinate, half of the remaining ones on the second, and so on. Only a very small proportion of the hard locations must have all ten of their selected coordinates checked. In fact, the average number of selected coordinates that must be checked per hard location is slightly less than two.

In the first embodiment the address module performs approximately seven basic steps for each hard location, while in the second embodiment the address module must perform about the same number of steps for each selected coordinate for each hard location. Therefore, since the logic unit in the second embodiment checks an average of two selected coordinates per hard location, the second embodiment will take about twice as long as the first to perform a read or a write operation. (If greater speed is desired in either embodiment, it could have two or more logic units, operating in parallel on different subclasses of the hard locations.) On the other hand, the second embodiment can handle much longer addresses than can the first, and it requires less memory space in its hard address unit.

Compared to the Stanford prototype, the first embodiment of the new design requires a larger hard address unit, but it would operate faster. The second embodiment allows for much longer addresses, but without a proportionate increase in the hardware needed. It should operate at a speed comparable to that of the Stanford prototype.

REFERENCES

Chou, P. A. (1988). The Capacity of the Kanerva Associative Memory. Submitted to *IEEE Transactions on Information Theory*.

Flynn, M. J., Kanerva, P., Ahanin, B., Bhadkamkar, N., Flaherty, P., & Hickey, P. (1988). Sparse Distributed Memory Prototype: Principles of Operation. Technical Report CSL-TR-87-338, Computer Systems Laboratory, Stanford University.

Kanerva, P. (1988). *Sparse Distributed Memory*. MIT Press, Cambridge, Mass.

Keeler, J. D. (1988). Capacity for patterns and sequences in Kanerva's SDM as compared to other associative memory models. In D. Anderson (ed.), *Neural Information Processing Systems*. American Institute of Physics, New York.

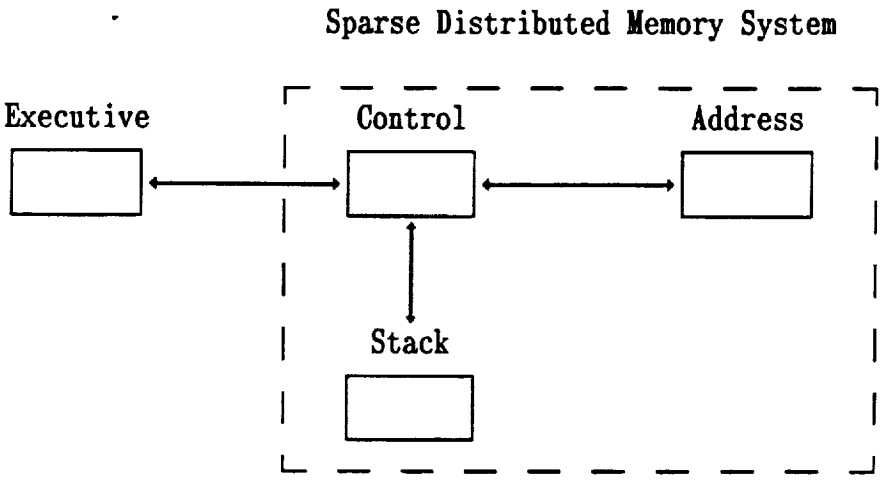



**Figure 1: Arrangement of Modules**

| | |
|---|---|
| Executive module: | A workstation for user interface. |
| Control module: | Communicates with other modules and controls operation of address and stack modules. |
| Address module: | Determines which hard locations are to be activated for a given reference address. |
| Stack module: | Contains the counters for the hard locations and a processor to read data from and write data to the counters. |

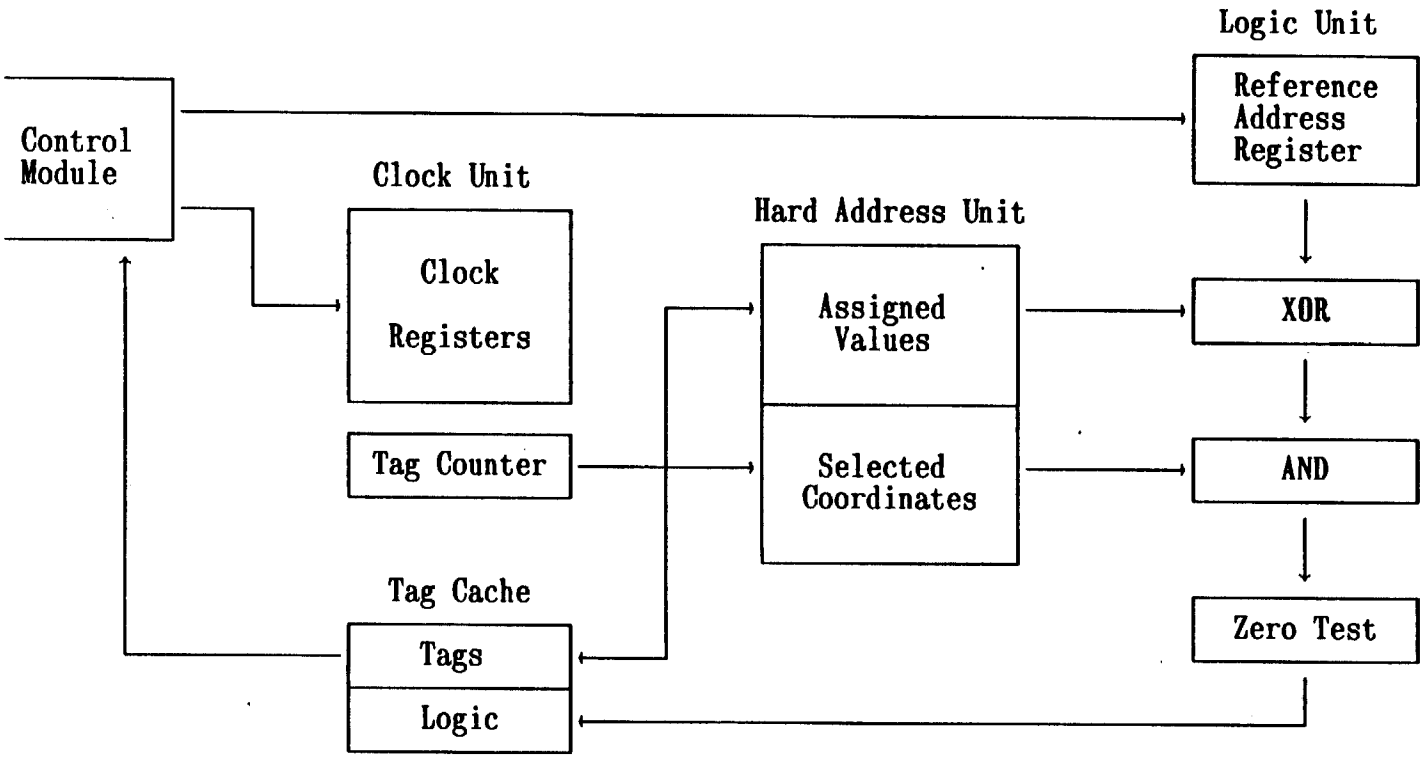
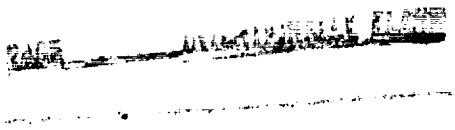



**Figure 2: Address Module for First Embodiment**



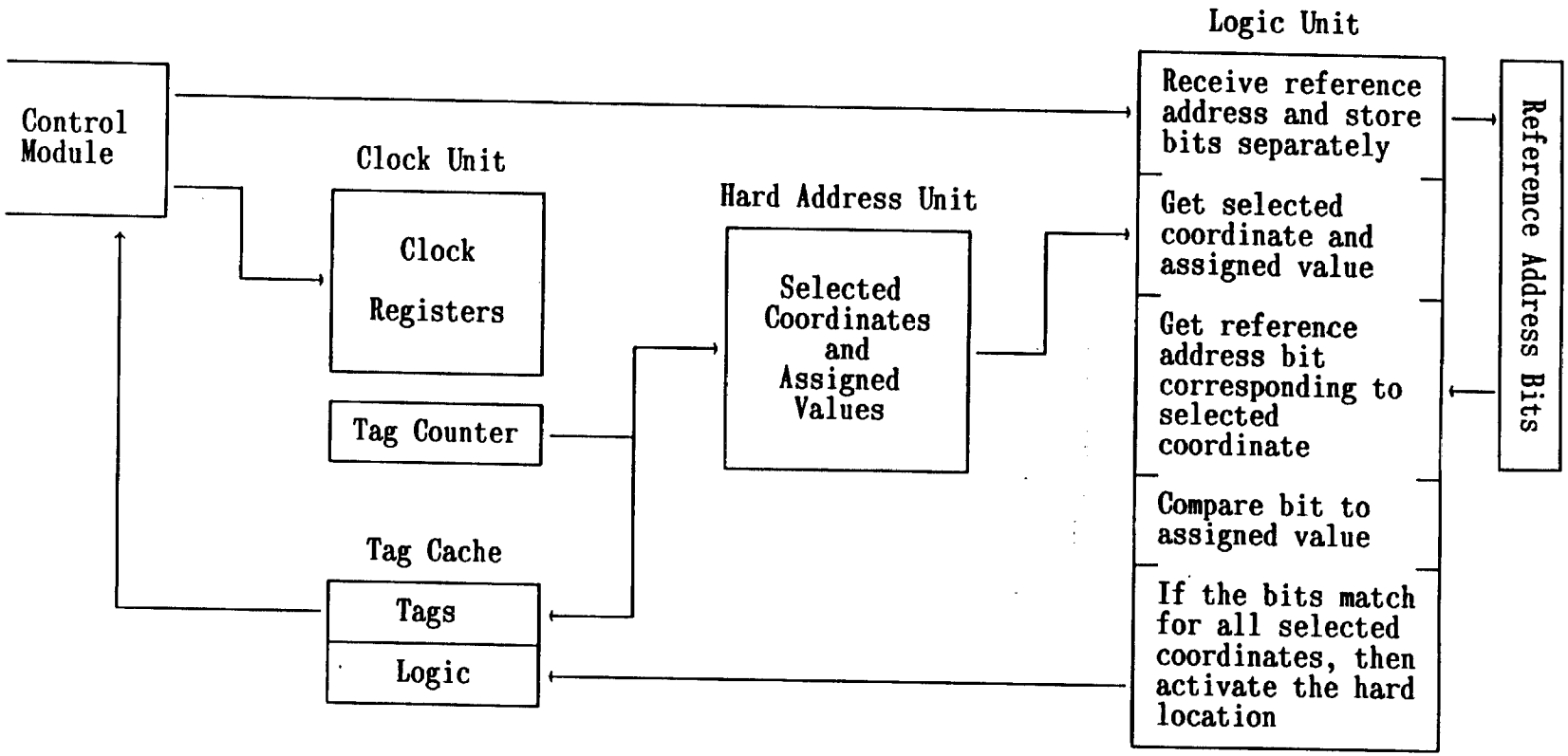



**Figure 3: Address Module for Second Embodiment**

PAGE _____ UNAVAILABLE