IN-61-CR

Using data tagging to improve the performance of Kanerva's sparse distributed memory

David Rogers

January 1, 1988

RIACS Technical Report TR-88.1

NASA Cooperative Agreement Number NCC 2-408

RIACS

Research Institute for Advanced Computer Science

Using data tagging to improve the performance of Kanerva's sparse distributed memory

David Rogers

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS Technical Report TR-88.1
January 1, 1988

Abstract: The standard formulation of Kanerva's sparse distributed memory (SDM) involves the selection of a large number of data storage locations, followed by averaging the data contained in those locations to reconstruct the stored data. In this work I discuss a variant of this model, in which the *predominant pattern* is the focus of reconstruction. First, I will propose one architecture that returns the predominant pattern rather than the average pattern. However, this model will require too much storage for most uses. Next, a *hybrid* model will be proposed, called *tagged SDM*, which approximates the results of the predominant pattern machine, but is nearly as efficient as Kanerva's original formulation. Finally, we will show some experimental results which confirm that significant improvements in the recall capability of SDM can be achieved using the tagged architecture.

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

Using data tagging to improve the performance of Kanerva's sparse distributed memory

David Rogers

Research Institute for Advanced Computer Science

January 1, 1988

1. Introduction

Kanerva's sparse distributed memory (SDM) is a simple and mathematically elegant formulation for an associative memory (Kanerva, 1988). It is related to the cerebellar model of Marr (1969) and the Cerebellar Model Arithmetic Computer (CMAC) of Albus (1971, 1981). In all of these models, the memory is *sparse* -- that is, the number of physical storage locations is small relative to the number of possible addresses -- and *distributed* -- that is, each datum is stored in a large number of physical locations. These properties can be used to build a memory that is *associative* and *fault-tolerant*, two qualities that characterize human memory.

In distributed storage, writing is done by storing a data pattern in multiple locations within some radius of a reference address. Reading is more difficult. First, we collect all the data patterns stored within some radius of the reference address. Second, we need to construct a single data pattern that will represent the *contents* of the memory from the multiple patterns collected. How is this construction accomplished? Kanerva's model proposes one solution: that the contents of the memory should be represented by the bitwise average of the multiple data patterns. However, other solutions are possible. In particular, we will discuss representing the contents as the *predominant pattern* in the patterns collected.

This variant has a number of interesting features: for example, it always returns a data pattern that was previously stored. However, it is difficult to implement. In Kanerva's words, "The method is sound in principle, but the task of deciding which word is the most frequent . . . seems insurmountable" (Kanerva, 1984, p. 93).

For cases where many data patterns are stored, the proposed model will indeed be impractical. However, the proposed architecture will lead to another, *hybrid*, architecture called *tagged SDM*. Tagged SDM will offer a better approximation of the predominant pattern than does the standard model.

Finally, we will show some experimental comparisons between Kanerva's SDM and the tagged SDM. These

results will illustrate the ability of tagged SDM to correct bitwise errors that are generated in the standard model. By increasing the fidelity of the stored data, we significantly improve the capacity of a sparse distributed memory implementation.

2. Distributed Storage

In the general model for a sparse distributed memory, the data pattern is stored in a set of locations that are within a given radius of some address. For example, in figure 1, writing α at reference address **A** stores copies of data pattern α in the six locations within the shown radius. Similarly, writing β at reference address **B** stores copies of data pattern β in the five locations within the circle around point **B**.

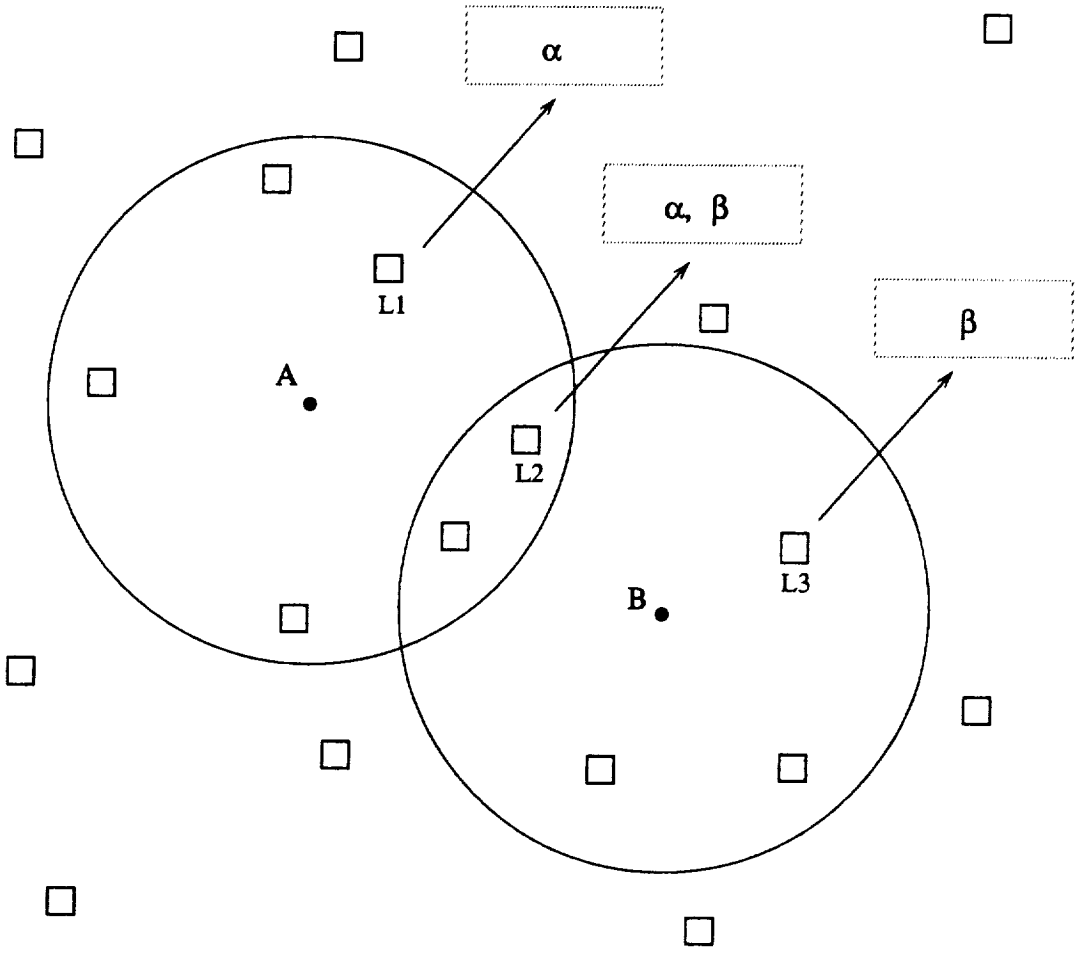

Figure 1: Distributed storage after writing α at **A** and β at **B**

What is the 'content' of a given physical location, such as L1? If a location contains only one pattern, we would feel comfortable saying that it contained that item (for L1, the data pattern α). This is not as clear for locations that contain more that one pattern, such as L2. Would the 'content' be α, β, or some mixture of the two? In these cases, it is uncertain what the 'right' answer should be.

This uncertainty is compounded when we try to read at a reference address. For example, let's read at point A: if we pool the contents of the locations within the given radius around A, we have a set of five α's and two β's. What is the answer we should return?

Essentially, the problem involves how we take a group of multiple patterns and generate a single pattern that represents the contents of that set. Some kind of voting seems in order; two kinds of voting come to mind.

First, we could have each pattern in the pool participate in *bitwise voting* towards the value of each bit in the 'answer'. In this case, each pattern instance votes for a zero in a bit position if it has a zero in that position, and for a one in a bit position if it has a one in that position. The value of a position in the answer is determined by whether it receives more votes for a zero or for a one. Essentially, we are taking a *predominant vote bitwise* to get the answer. This approach is used in Kanerva's formulation for SDM, and we will refer to it as the *standard* model for SDM.

Second, we could select the *most frequently appearing* data pattern as our 'answer'. All equivalent data pattern instances vote as a block. The block with the most votes wins, and that data pattern is the answer. In this case, we are taking *predominant vote patternwise* to get the answer. This approach will be referred to as the *predominant pattern* model.

Each approach is of interest in itself. Depending on the data patterns we are storing and the desired use of the memory, one or the other technique may be preferable. The averaging process of the standard model may aid in using common features among different data patterns, while the error-resistance of the predominant pattern model may be useful in removing errors from patterns as the memory fills. In some cases, the fact that the predominant pattern model always returns a previously stored data pattern may be useful.

3. Implementation of Kanerva's SDM

Figure 2 illustrates the standard formulation for Kanerva's sparse distributed memory. (This figure will show a memory with twelve locations, ten-bit addresses and ten-bit data.) Initially, the *location addresses* are filled with random values, and the *data counters* are zeroed. All operations begin with *addressing* the memo-

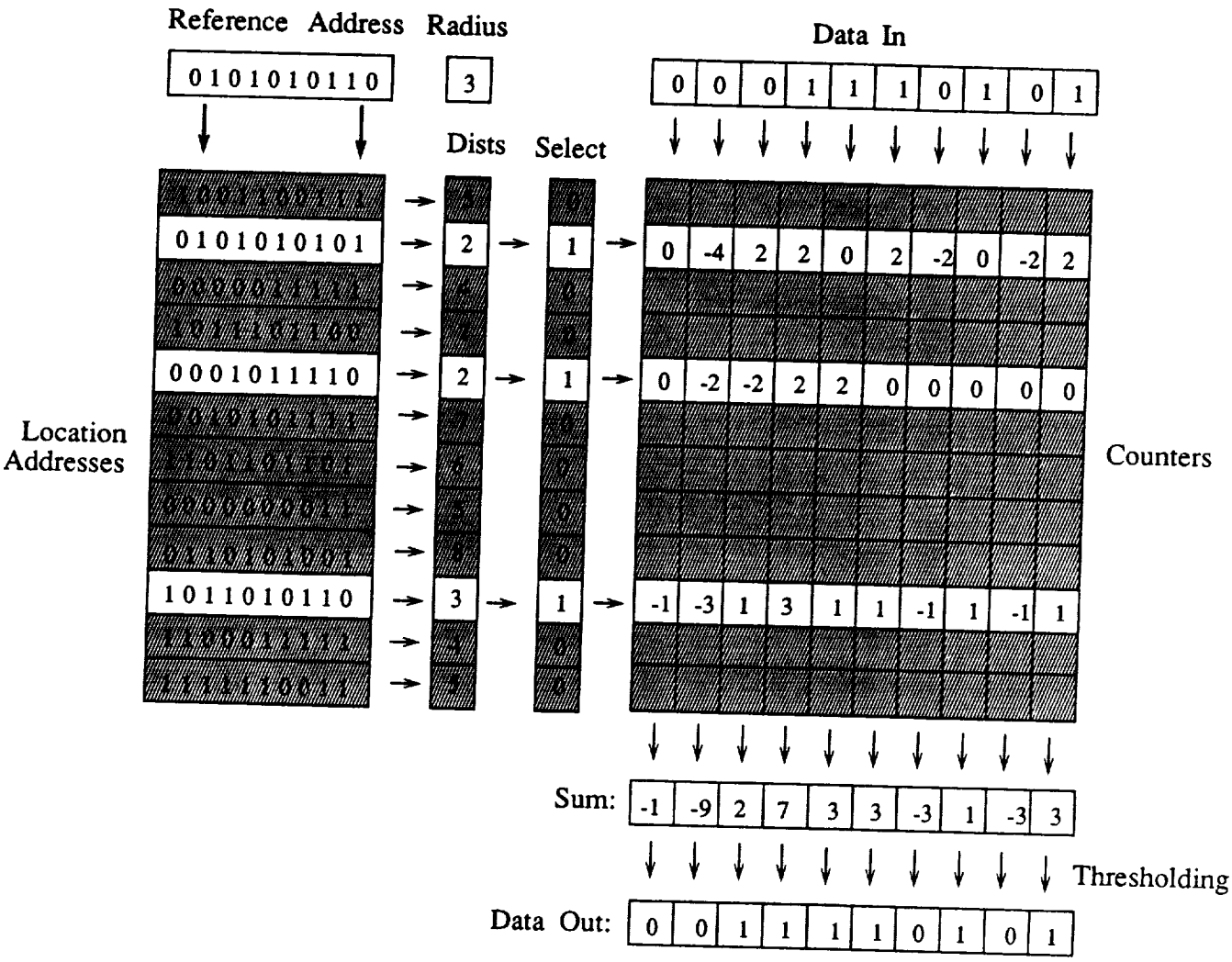

Figure 2: Standard formulation for sparse distributed memory (SDM)

ry; this entails finding the hamming distance between the *reference address* and each of the location addresses. If this distance is less than or equal to the *hamming radius*, the weight for that locations is set to 1, and that location is termed *selected*. Selection is noted in the figure as non-gray rows.

If we are *writing* to memory, all selected counters beneath elements of *Data In* equal to 1 are incremented and all selected counters beneath elements of *Data In* equal to 0 are decremented. This completes the write operation.

If we are *reading* from memory, we collect the sum of the selected data counters into the register *Sum*. If the value of a counter sum is greater than or equal to zero, we set the corresponding bit in *Data Out* to 1; other-

wise, we set the bit in *Data Out* to 0. (When reading, the contents of *Data In* are ignored.)

This example makes clear that the data is *distributed* during writing over the data registers when writing, and is reconstructed during reading by *averaging* the sums of these counters. However, depending on what additional patterns was written into some of these locations, the reconstruction may have one or more bitwise errors.

4. Implementation of the predominant pattern SDM

Figure 3 illustrates one possible implementation for a predominant pattern sparse distributed memory. As with the standard formulation for SDM, the *location addresses* are filled with random values, the *tag counters* are zeroed, and the *pattern bank* is zeroed. All operations begin with *addressing* the memory. This entails finding the hamming distance between the *reference address* and each of the location addresses. If this distance is less than or equal to the *hamming radius*, that location is selected. Selection is noted in the figure as non-gray rows.

Writing to memory consists of *getting a tag* for our new pattern followed by *setting the tags* in the tag counters. *Getting a tag* can be thought of as a symbol-table lookup. First, we check the table for an occurrence of the pattern. If the pattern is already in the table, we use the assigned index; otherwise, we add the pattern to the end of the table, and use the newly-assigned index. *Setting the tags* involves incrementing the tag counters corresponding to this tag number in all selected locations.

This process is illustrated in figure 3. We are writing the data pattern **1111101111** at address **0101010110**. The pattern is stored in pattern bank item 3, since it is not currently in the pattern bank, and thus gets tag 3. All selected locations have their third tag counter incremented.

If we are *reading* from the memory, we add the tag counters from the selected locations into the *Tag Sums*. The entry with the largest value is the index of the answer. The data pattern associated with that index is returned as the answer.

This implementation of predominant pattern SDM is suitable for a small number of data patterns, but as the number of data patterns increases, it becomes very expensive to allocate the storage needed. In this regard, it is far less practical than Kanerva's formulation. For example, start with a SDM with 2^{10} (1024) bit addresses and data, and L physical locations. Kanerva has shown that this memory can store about $L/2^3$ patterns, and the

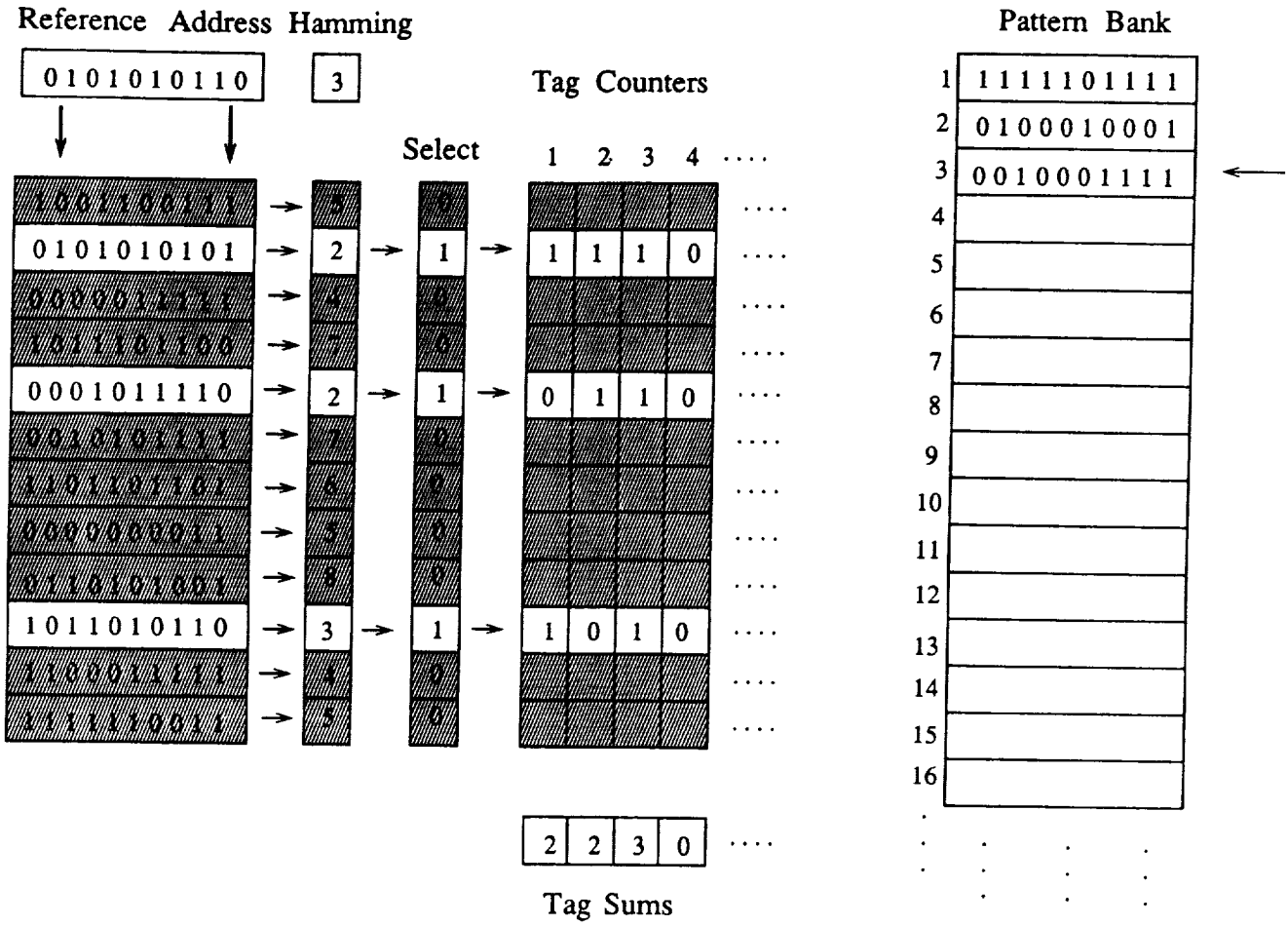

Figure 3: One design for a predominant pattern sparse distributed memory

data locations require (if we assume a counter is 2^3-bits wide) $L * 2^3 * 2^{10}$ bits of storage. To store $L/2^3$ patterns in the predominant pattern SDM, we will need $L/2^3 * 2^{10}$ bits to construct the pattern bank, and $L/2^3 * 2^3 * L$ bits to construct the tag counters. Solving for L, we can show that predominant pattern SDM will require more storage than standard SDM when $L \geq 2^{13} + 2^7$, which is about 8,000 locations (and about 1,000 patterns).

For applications that greatly exceed this amount, the required storage makes the predominant pattern model prohibitive. However, the predominant pattern model is still of interest; perhaps a *hybrid* model would have a cost close to that of Kanerva's SDM but a recovery property close to our predominant pattern model. Given

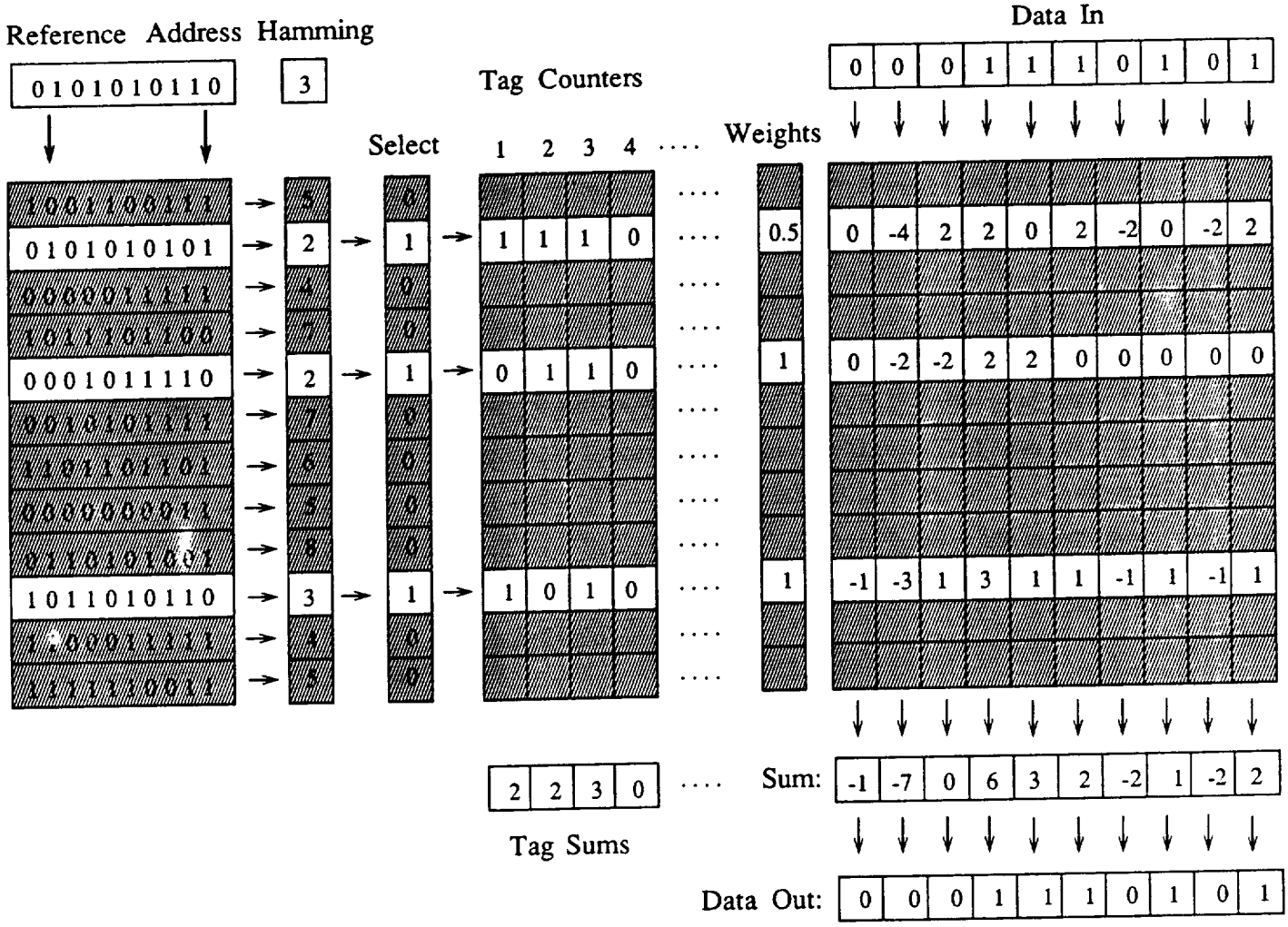

Figure 4: One design for a hybrid sparse distributed memory

the general formulation for Kanerva's SDM, can we construct an answer that *approximates* the pattern that would be generated by the predominant pattern model?

5. Prototype of hybrid SDM

Figure 4 shows a hybrid SDM that contains some of the features of a standard SDM, and some of the features of our predominant pattern SDM. We have eliminated the *Pattern Bank*. Instead, we use data counters to store the data patterns exactly as done with the standard formulation for Kanerva's SDM. However, we still increment the tag associated with the data as we did in the predominant pattern model, with each data pattern

assigned a sequential number. (Of course, without the *Pattern Bank*, we can no longer recognize duplicate patterns. This problem will be dealt with in our next design.)

Because of the tags, we can still use the *Tag Sums* to calculate which pattern number would have won the predominant pattern competition. The question is how we can use this information since we can no longer 'look up' the desired pattern. The key is to notice that some locations are tagged as having lots of the desired data pattern, while others have little. This suggests that a *weighted combination* of the locations may be possible to maximize the desired pattern.

These weights can be calculated if we assume that the data patterns are uncorrelated. Appendix A shows how this assumption leads to a formula for the weighting that maximizes the contribution of the desired data pattern in the output. This weight is simply S/N, where S is the contents of the counter for the desired tag, and N is the sum of all the other tag counters in a location. Each location's data counters are multiplied by their weight before adding into the sum.

There are two problems with this design: first, we still need a set of tag counters for each data pattern, and second, the weights are not integer valued, and so may increase the computational demand by forcing the use of floating point numbers. While improved, this design is not yet very practical.

6. Model for tagged SDM

The final model to be discussed here is called *tagged SDM*. It circumvents the problems in our first prototype. Figure 5 shows the architecture of this model. The largest change is that we have severely limited the number of tags. Indeed, the example shown has only has four different tags. Instead of assigning tags through a sequential counter, we use a *tag function* to calculate the tag from the data pattern. This tag function plays the role of a *hash function* in that it partitions the data patterns into equivalence classes; this distribution usually assigns non-equivalent data patterns different tag numbers. This will cause errors whenever we have overlap between two data patterns with the same tag value. For practical implementations, around 32 tags makes such overlaps rare.

Second, we note that a *scale* has been added for the weights. This is an integer that is multiplied by the S/N ratio to give the weight. Proper selection of the scale can eliminate the fractional values that plagued the intermediate prototype.

This model *approximates* the behavior of the predominant pattern design. However, as it is just an approxima-

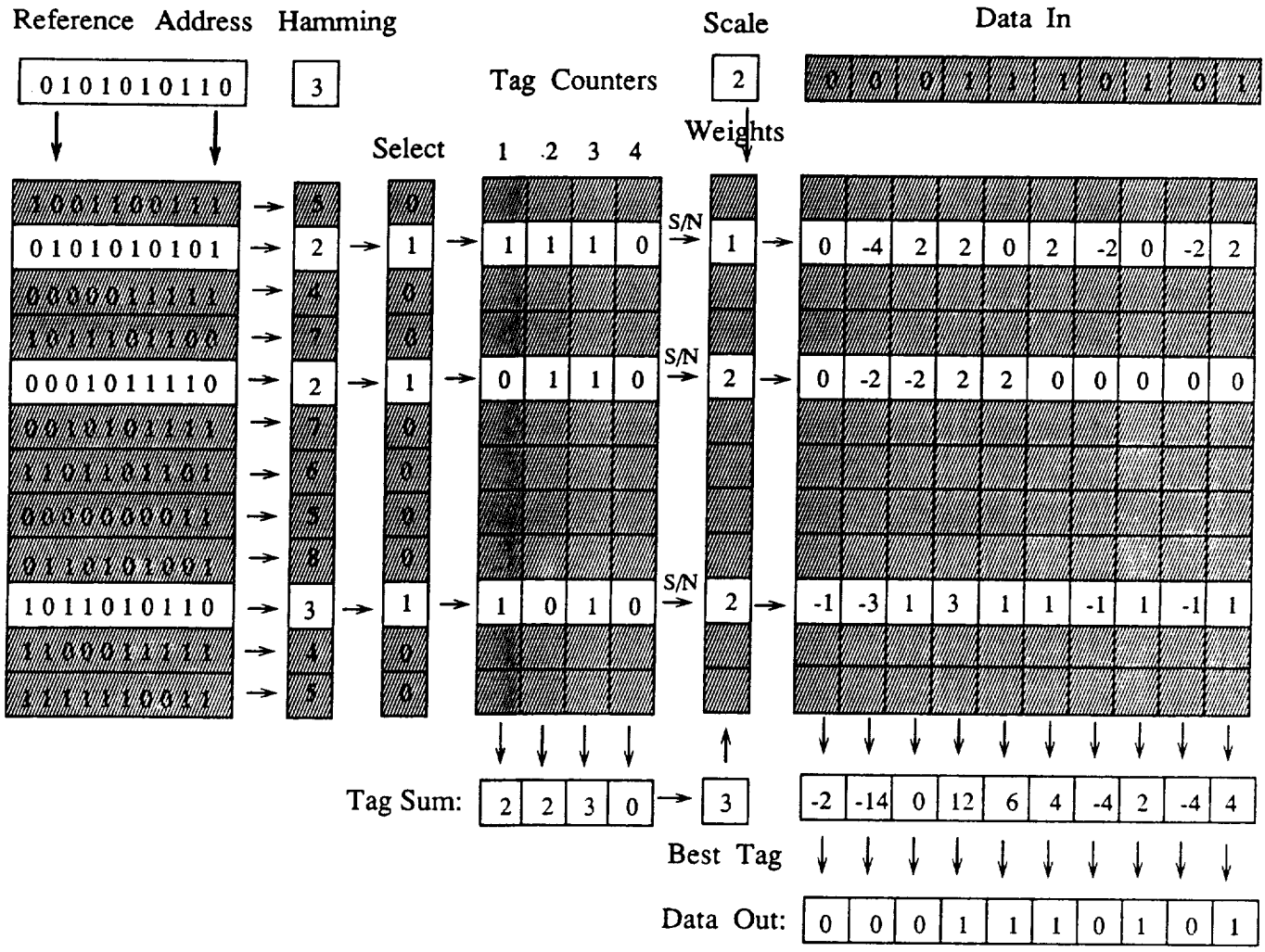

Figure 5: Design for the tagged sparse distributed memory

tion, it does not always return a stored pattern. The interest in this model is in the error-resistance of the predominant pattern design: transfer of that property to an efficient relative of Kanerva's SDM might offer a valuable increase in robustness as we fill the memory.

7. Computational costs of tagging

Assessing the costs of data tagging is not a simple process, for it will depend crucially upon the implementation. For these comment, we assume that *cost* is caused by adding steps to the memory cycle that cannot be performed in parallel with other steps. In this light, data tagging adds cost to the *read* cycle but not to the *write* cycle.

The tag counters are quite similar to the data counters. The only difference is that *all* data counters of the selected locations are either incremented or decremented during a write operation, while only the tag counter associated with the tag of our data is incremented, and the other tag counters remain untouched. (Sensitive souls who find this asymmetry distasteful can be assured that one can formulate the tag counters to behave identically to the data counters. For this work, such an alternate procedure was not felt to make the issues clearer to the reader.) Since the tag counters can be incremented in parallel to the operations on the data counters, there is no additional time cost to data tagging during the write cycle.

During the read cycle, however, data tagging increases the cost an amount roughly equivalent to the cost of summing the data counters. This is because we cannot sum the tag counters and the data counters in parallel. Information that we obtain from the tag counters is needed *before* we can sum the data counters. Once the tag counters are summed, there are additional steps that must be taken. First, we must scan each tag to find the tag with the maximum value, called the *signal tag*. Second, all selected locations (in parallel) use the information about the signal tag to calculate the amount of signal versus the amount of noise in their location. Finally, all selected locations (in parallel) use the amounts of signal and noise to calculate the *weight* that should be given to their location.

Once the weights are assigned, the summing of the data locations can proceed. This process costs slightly more than it did with the standard SDM model, for before we sum the data counter, we must multiply their values by the weight associated with the location. However, since all locations can do this operation in parallel, this should not result in a significant additional cost.

The major cost of tagging involves summing the tag counters, since this cannot be performed in parallel with summing the data counters. At worst (that is, if summing of counters is the major cost), tagging doubles the number of cycles it takes to read from the memory. However, this cost only becomes apparent in massively parallelized versions of SDM (Keeler, 1986). For non-parallel versions of SDM, the additional cost of data tagging will not be noticed due to the massive cost of the addressing phase.

One suggestion to avoid these costs would be to use the standard SDM formulation and iterate, using the natural error-correcting abilities of SDM. This is not sufficient. For example, in the case where we are reading data at the exact address we once wrote at, standard SDM will start generating errors as we fill the memory with other patterns. *These errors are not correctable by iteration, as the address we wrote at is no longer a stable point.* However, data tagging can still correct many of these errors.

8. Storage costs of tagging

Data tagging also has a cost relative to the storage requirements of the memory. This is due to the additional counters that are needed to keep track of the tags. For example, in a 32-tag model, we would need 32 additional counters for each physical location. For 256-bit wide data counters, this is a requirement for roughly 12% more counters than in the standard formulation for SDM.

While 12% may seem small, it leaves open the question as to whether the improvements we have seen could have been obtained in a much simpler manner, that is, by simply using these counters to add on 12% more physical locations to the memory. The claim is that it is significantly better to use the counters as tag counters, rather than to use them to construct additional counters. We will not offer a formal proof, but instead offer the following 'back of the envelope' argument.

The capacity of the memory is roughly linear with respect to the number of physical locations. Thus, doubling the number of physical locations roughly doubles the number of patterns that can be stored. To consider the example given above, a 12% increase in the number of physical locations results in approximately a 12% increase in the capacity of the memory.

We have not yet presented the results from data tagging experiments. Indeed, the results are not simple to generalize about, since they depend crucially on the correlatedness of the addresses and the amount of memory currently in use. However, in the *worst* cases, an average correction of more than 50% of the bitwise errors was observed. If we assume that the occurrence of bitwise errors is linear with respect to the number of writes, then we would need to *double* the number of physical locations to get similar error behavior.

In conclusion, while there are storage costs to data tagging, the improvement in performance of the memory is far more considerable than it would be had we simply used the storage to construct additional counters. The arguments given above were conservative. Most experimental results suggested even more substantial improvements than the case presented here.

9. Experiments

Finally, we present results from a simulation of tagged SDM, and compare those results to a standard SDM model. The comparisons will demonstrate that the tagged SDM has a significantly increased resistance to errors as we reach the capacity of the memory. All of the experiments we conducted by writing one word of

data to the memory at a given address, then writing random data at random addresses to the memory, while watching the effect of this on recall. The effect measured was the *number of bits in error after reading once at the initial address*. A rough algorithm describing the experimental process is given in figure 6.

```
{ Initialize the addresses and data patterns to random bits }
FOR i = 1 TO 300 DO
    a_i := RANDOM_BITS;
    d_i := RANDOM_BITS;
END;

{ Write the first word; loop and read it over and over, writing
  an additional address/data pair each time }
WRITE d_1 AT a_1;
FOR i = 0 TO 300 DO
    offset_address := RANDOMIZE offset BITS OF a_1;
    data := READ AT offset_address;
    error := HAMMING_DISTANCE(d_1, data);
    PLOT (i, error);
    IF i ≠ THEN
        WRITE d_{i+1} AT a_{i+1};
END;
```

Figure 6: Pseudo-algorithm for conduct of first experiment with a given offset

The only part of this algorithm that needs additional comment is the idea of an *offset*. An offset is a number; before we read at the initial address a_1, we invert *offset* number of bits in the address. One of the valuable features of SDM is its ability to gracefully handle errors in the reference address. The experiments were designed to discover whether tagging is also useful when we have errors in the reference address.

Of greatest interest will be the graphs in Figure 9, that show the *percentage correction* we obtain by using the tagged version of sparse distributed memory. These graphs show the percentage of the errors generated by the standard SDM model that were corrected by the tagged SDM model. (For simplicity, if neither the tagged

nor the standard model showed errors, the correction is graphed as 100%. The case did not arise where the tagged SDM model generated errors where the standard SDM model did not.)

This process will show the value of tags when confronted with random filling of the memory. We will then show that they are equally useful when dealing with *correlated* writing, that is, writing to memory at addresses that are not roughly random in distribution. Given the increasingly poor behavior of the standard SDM model as the addresses become more and more correlated, this offers a useful method to improve the performance of Kanerva's SDM without sacrificing the associativity of the memory, the distributed nature of data storage, or the simplicity of the model.

The following sections describe the experiments. For a more precise presentation of the conduct of the experiments, the reader is referred to the **Experimental** section at the end of this report.

9.1 Performance of standard SDM

Figure 7 shows the error behavior of a standard sparse distributed memory as we write random data. For this experiment, a 1000-location memory was used with 256-bit addresses. The data were generated by averaging the results of eight independent trail runs. The value displayed is the average number of bitwise errors in the data obtained by reading at a given address.

Two features in these graphs deserve comment. First, the degradation of the memory is gradual as the memory fills, and the degradation is nearly linear. Second, the greater the number of errors in the address, the greater the rate of degradation as we fill the memory. The former effect was predicted in the theoretical formulation of Kanerva's sparse distributed memory; the latter effect has not been previously commented upon.

9.2 Performance of tagged SDM

Figure 8 shows the error behavior of a sparse distributed memory with the addition of tags as we write random data. As with the previous experiment, we used a 1000-location memory with 256-bit addresses. This data was, as in the previous experiment, generated by averaging 8 independent trial runs.

If one compares these results to the results of the standard SDM, the improvements in error behavior when tags are used becomes apparent. We will explicitly show the percentage correction in following section, so

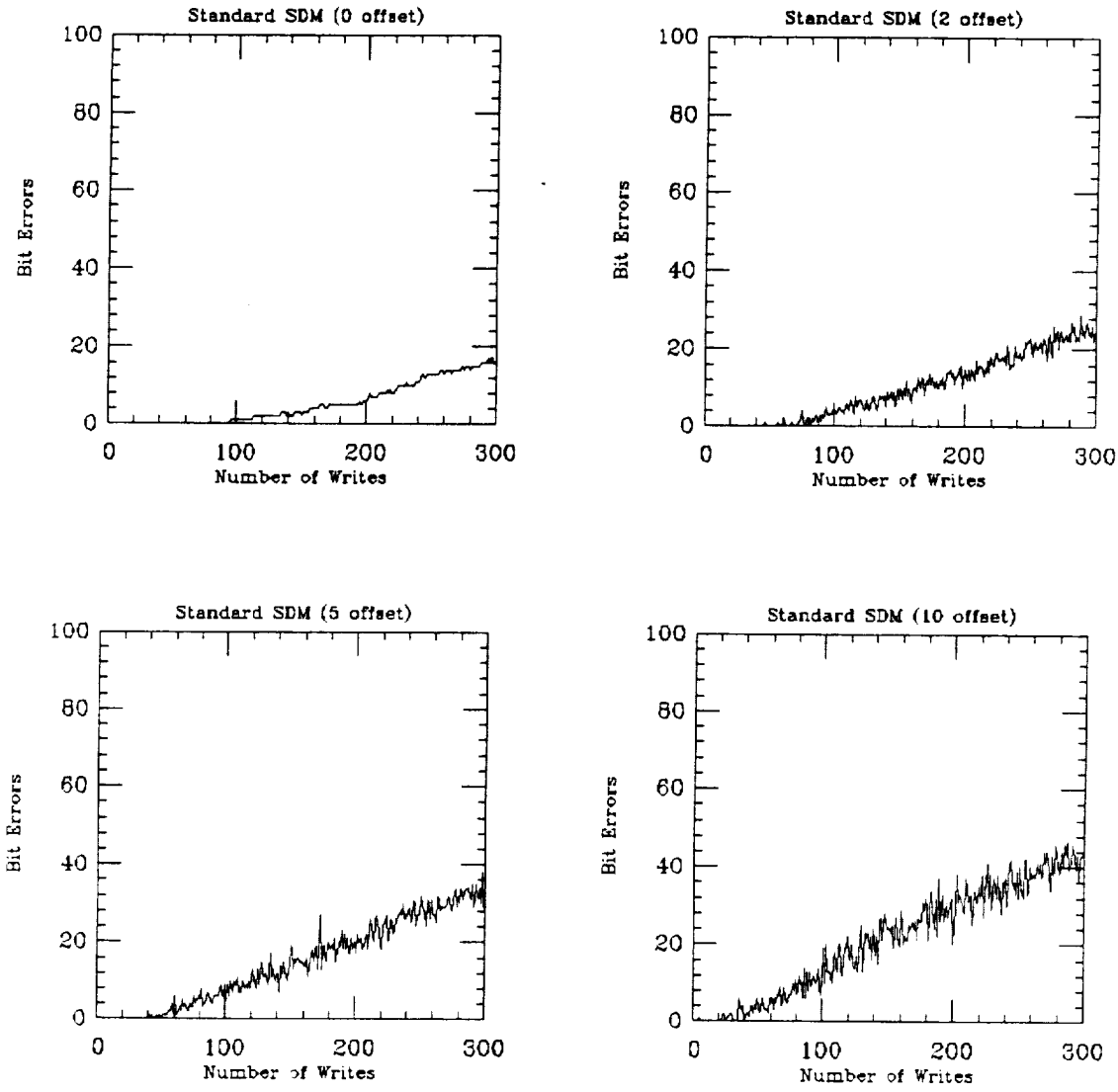

Figure 7: Number of bitwise errors reading a standard SDM vs. number of random writes

you don't need to flip back and forth between figure 7 and 8. However, it should be noted that the point at which the average number of errors becomes non-zero moves towards larger numbers of writes. This means we have not only *reduced the percentage of errors* for a memory filled to a given capacity, but have also *delayed the onset of errors* as we fill the memory.

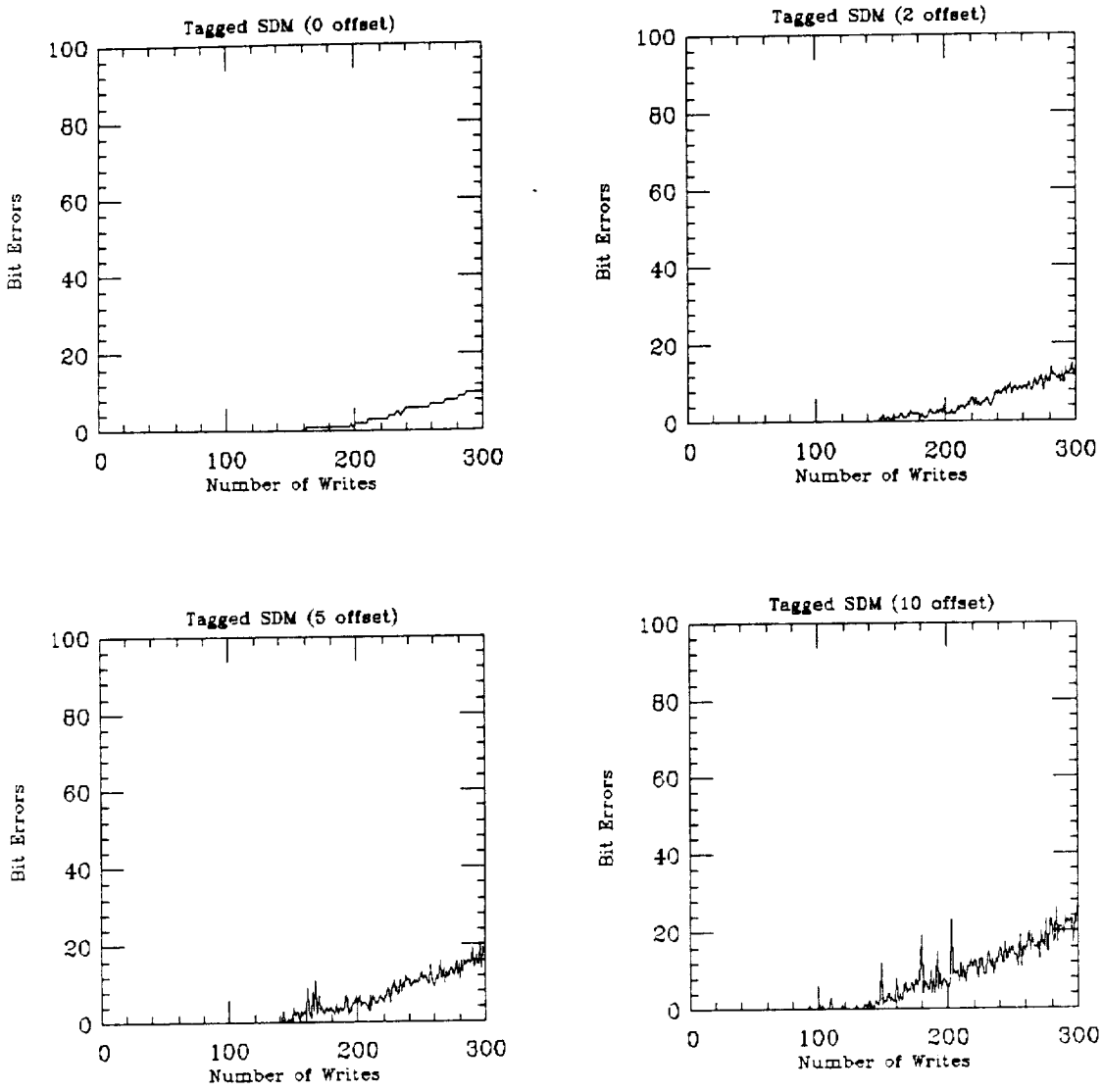

Figure 8: Number of bitwise errors reading a tagged SDM vs. number of random writes

9.3 Percentage correction by tagged SDM

Figure 9 shows the percentage of errors generated by a standard SDM that are corrected using a tagged SDM as we write random data. These graphs were generated from the previous two pages of graphs, by taking the ratio of the difference between the bit errors of standard and tagged SDM over the number of bit errors in the standard SDM.

Massive correction of errors is seen in all cases. Specifically, around 70% of the errors generated by a standard

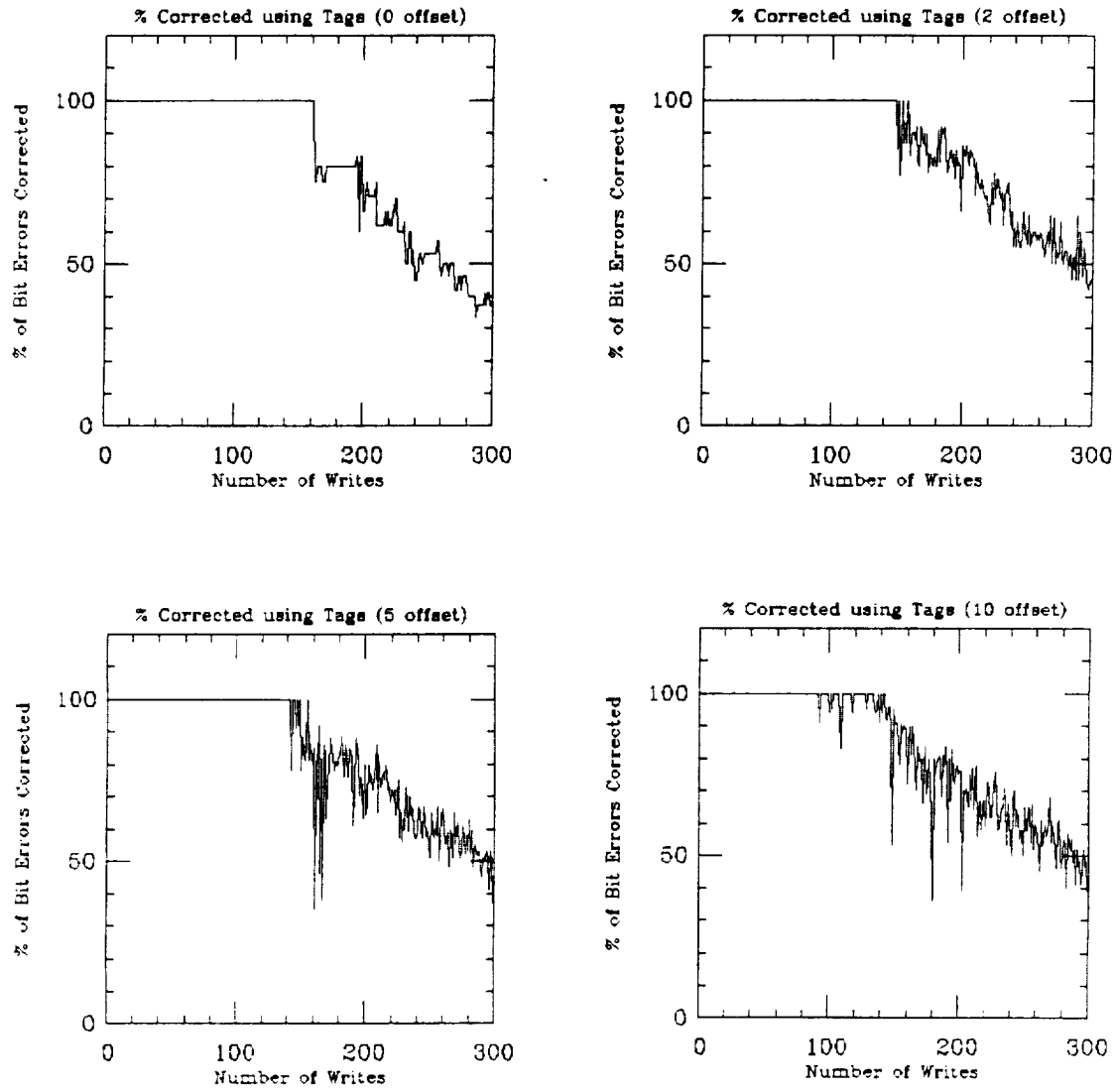

Figure 9: Percentage correction of errors using tags vs. number of random writes

SDM were corrected by using tags. Put another way, tagged SDM produced approximately 30% of the errors that a standard SDM produced. This represents a significant improvement of recall performance at a given capacity. Equivalently, this represents the ability to obtain a given level of recall performance at much greater capacities.

One feature worth noting is that the percentage correction correlates well with the number of writes, but is relatively independent of small initial error in the address. This implies that convergence behavior during a series of reads should be uniformly improved by using tags.

10. Experiments with Correlated Addresses

Data tags have been shown to significantly improve the performance of Kanerva's sparse distributed memory when writing is done at uncorrelated addresses. It remains to be shown whether data tags are as useful when we write using correlated addresses. Most real-life uses for SDM will have some correlation in their addresses; thus, the performance of SDM in these uses is of the highest priority.

Before we continue, we need to explain how the generation of correlated addresses was done. Random addresses have (on the average) 128 bits out of 256 bits that have value 1. We generated correlation by selecting addresses that were on a sphere of radius N away from address "000...000". The points on this sphere are simply addresses with exactly N bits having the value 1. Each of the graphs on the following pages shows this information above the graph.

The difficulty with correlated addresses is that the physical locations in our memory are not evenly filled. The correlation introduces a bias towards overuse of the subset of the locations. Thus, the capacity of the memory is reduced when correlated addresses are used for writing. (Keeler (unpublished) demonstrates that the proper selection of addresses for the physical locations is one method for improving performance with correlated addresses. It is possible that data tags can be used in cooperation with such techniques for additional performance improvement.) Data tagging will be useful here if the variations in writing density are such that they are felt over the area surrounding an address we wish to read from. It is unclear whether the concentration of the memory caused by correlating data will improve or degrade the improvement we see using data tagging. The experimental results will demonstrate that not only do we keep the high levels of correction seen in the uncorrelated case, we surpass them. Thus, *correlating addresses seems to increase the variation between the contents of the selected locations. This allows data tagging to work even more efficiently.*

10.1 Performance of standard SDM with correlated addresses

Figure 10 contains four graphs, showing the number of bit errors that occurred when writes used random addresses, 96-bit addresses, 64-bit addresses, and 32-bit addresses. The major feature of note is the massive degradation of capacity that occurs as we introduce increasing correlation into the addresses. This is due to the increased dependence on a smaller and smaller subset of the physical locations as we increasingly correlate the addresses we write at.

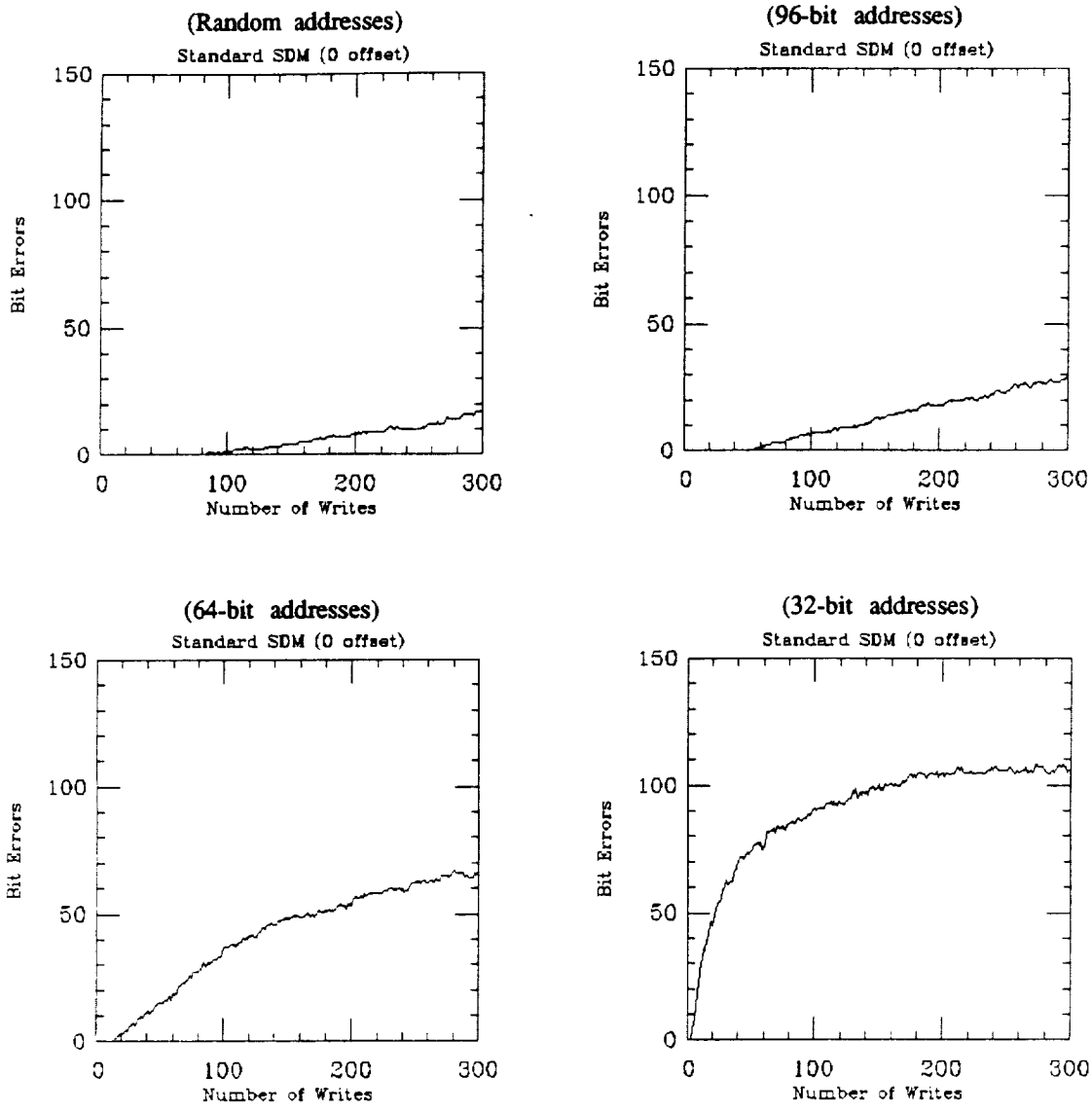

Figure 10: Number of bitwise errors in a standard SDM vs. number of writes, for random writes, 96-bit addresses, 64-bit addresses, and 32-bit addresses

10.2 Performance of tagged SDM with correlated addresses

Figure 11 contains four graphs, showing the number of bit errors that occurred when writes used random addresses, 96-bit addresses, 64-bit addresses, and 32-bit addresses. In all cases, significant improvement in the number of errors is noted relative to the standard SDM showing in figure 10.

However, what we wish to see is whether tagging offers greater benefit, less benefit, or about the same benefit as we correlate the data. For this, we need to compare the percentage correction graphs, which are shown in the next figure.

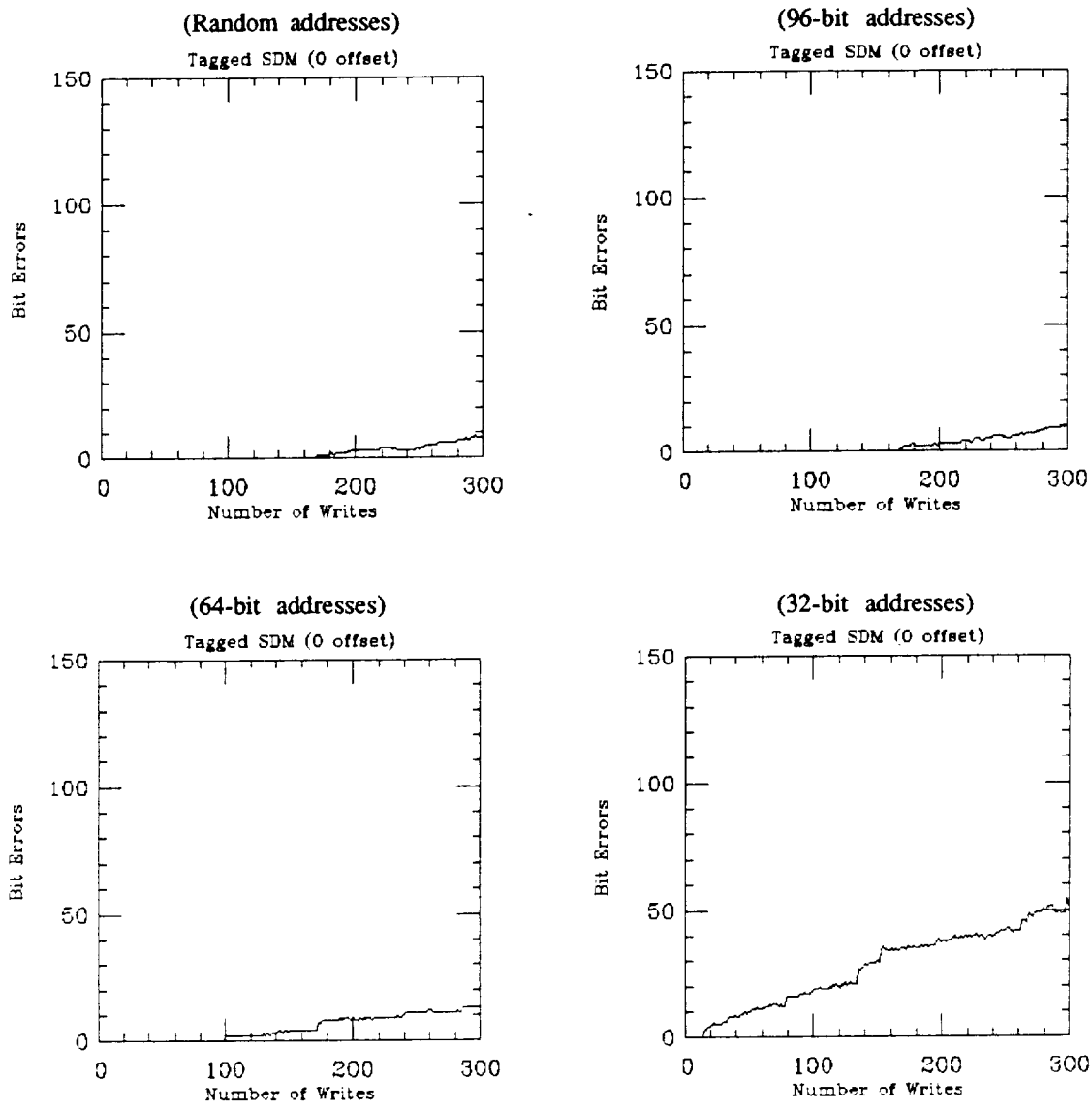

Figure 11: Number of bitwise errors in a tagged SDM vs. number of writes, for random writes, 96-bit addresses, 64-bit addresses, and 32-bit addresses

10.3 Percentage correction by tagged SDM with correlated addresses

Figure 12 contains four graphs, showing the percentage correction that occurred when writes used random addresses, 96-bit addresses, 64-bit addresses, and 32-bit addresses. For mildly correlated data, the tagged architecture improved the performance of the memory even more than it did for the uncorrelated case. This is especially impressive in the 64-bit address case, where the average percentage correction had a *minimum* of 80% correction.

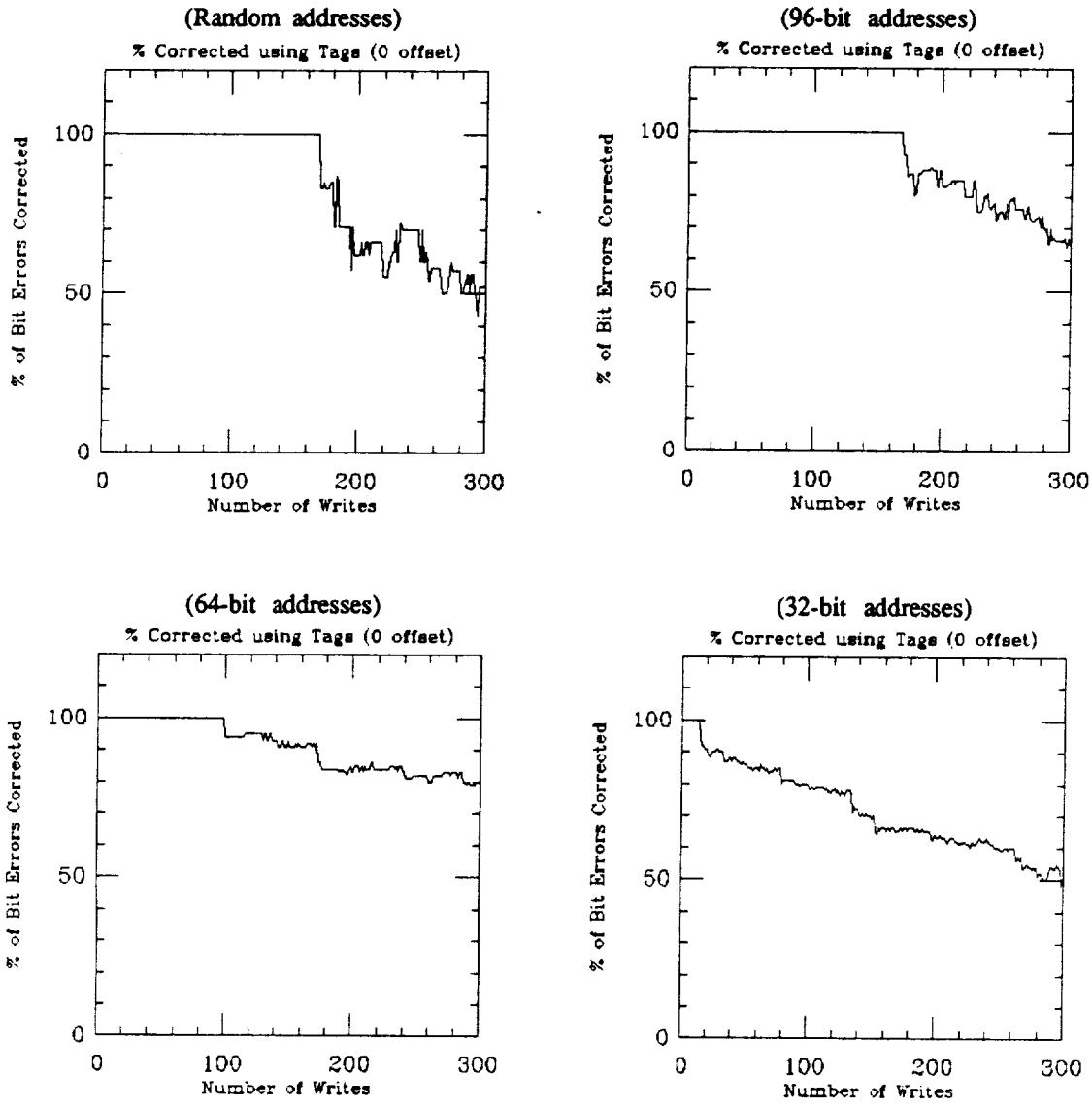

Figure 12: Percentage correction of errors using tags vs. number of writes, for random writes, 96-bit addresses, 64-bit addresses, and 32-bit addresses

The more highly correlated 32-bit addresses case still shows highly significant correction, even when the memory was highly loaded. However, it had less success in delaying the *onset* of errors, possibly due to the rapid onset of extremely large errors in the standard case shown in figure 10.

11. Experiments with text

I chose not to use text for the major part of this study due to the amount of variance possible. Depending on the format and the source of the text, it may be either highly correlated or nearly random. However, it is still of interest to see whether data tagging would be of help for some given sample of text.

The text was translated into addresses and data by simply taking 32-byte strings and treating the ASCII representation of the text as an 8-bit chunk. Thus, a 32-character text string becomes a 256-bit address or data string. For the addresses, the names and institutions were randomly chosen from a local address file. If they were longer than 32 characters, the string was truncated, else they were padded with spaces to 32 characters. For data, phone numbers (as text) were randomly selected from the file, and padded with spaces to 32-characters. Instead of writing random data patterns at random addresses, addresses and data pattern generated using the above technique. A sample of the generated text is shown in figure 14.

Addresses	Data
Jim Bigham; University of Waterl	471-3434
Erol Gelenbe; Carleton Universit	213/825-2929
J. M. Cotton; NRL	612/681-3352
Rebecca Gering; NASA	213/822-1511
David Gries; University of Illin	614/422-2571
Steve Bruell; Lunar and Planetar	202/357-9776 (w)
J. R. Einstein; Rensselaer Polyt	605/677-5295
Ellen C. Giles; University of Mi	713/483-7060
Daniel F. Flores, Jr.; NASA-Ames	415/723-2419
Jan Cuny; University of Missouri	814/863-1469

Figure 14: Some examples of generated text

The results of this experiments are shown on the following page as figure 13. First, we notice that the asymptote is low, due to the high correlation among the data (nearly 2/3 of the average data pattern were the trailing spaces). The major improvement seen by using tags is the delay in the onset of errors. For the standard SDM, errors began (on the average) after only 10 writes. For the tagged SDM, errors began after about 80 writes. After the onset of errors, tagging still provides significant improvement in data recall, though the percentage correction degrades to less than 25% by the time 300 writes were performed.

Attentive readers will have noticed that as the number of writes grows, the number of bitwise errors no longer goes to 128. This is due to the massive amount of correlation in the data. First, most items are only 12 characters long, with 20 trailing spaces. Second, most of the characters in the data are numbers, which can

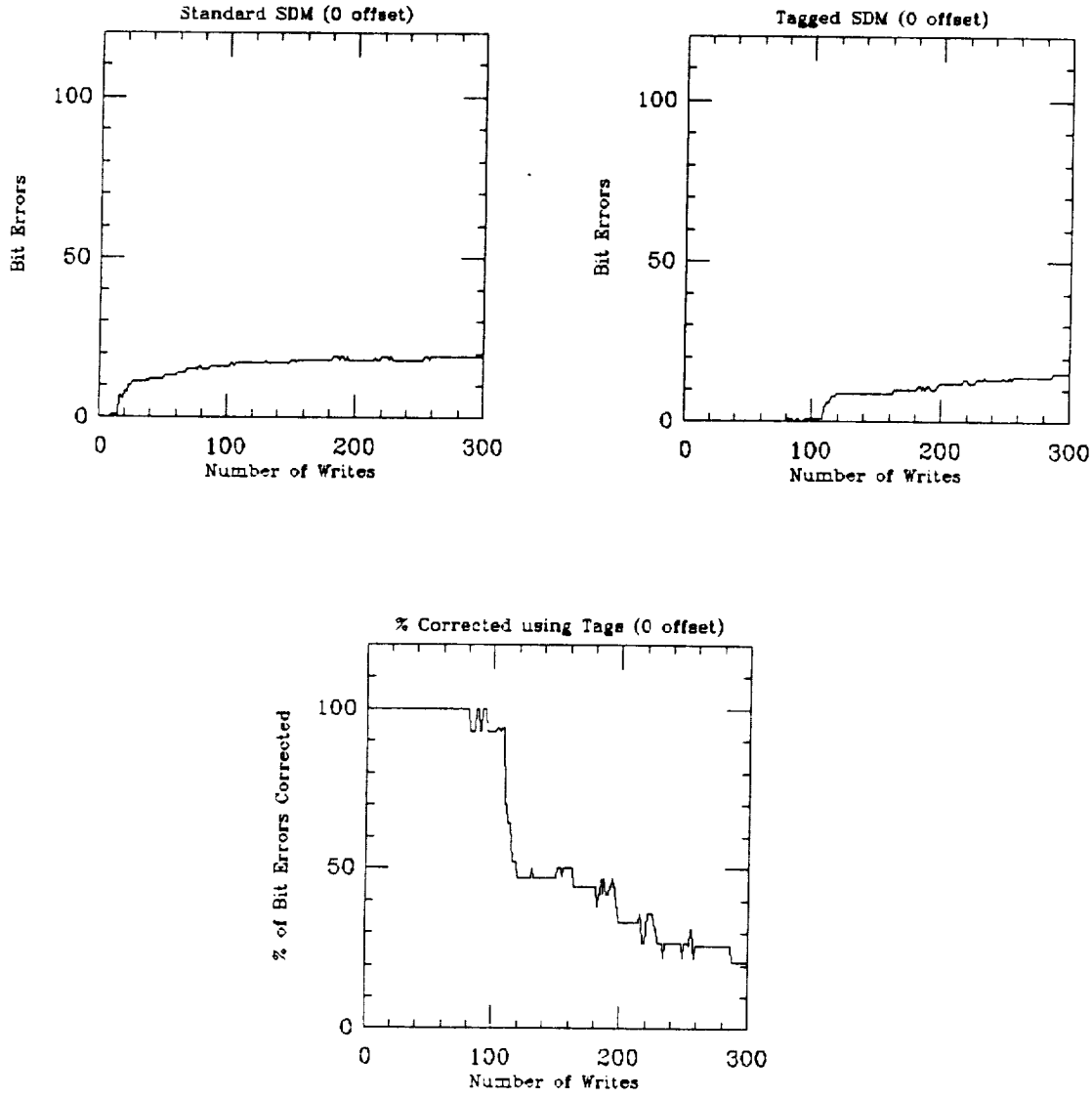

Figure 13: Number of bitwise errors reading a standard SDM and a tagged SDM using text as address and data, and percentage correction of errors using tags, vs. number of writes

disagree on only 4 of the 8 bits in the character byte.

These results are suggestive, though many more studies would be needed to determine the improvement that would be expected working with text from real-world cases. The effects of correlated data, in particular, deserve a more detailed study. It is encouraging that, with the crudeness of this trial, consistent and significant correction of errors was still seen.

12. Number of tags needed

A final question of interest is the number of tags we need to obtain maximum benefit from data tagging. This is a complex question, and one that will not be formally handled here. However, a few comments seem to be in order, given the importance of selecting enough tags to get improvements in performance, but not so many that we waste resources to little additional benefit.

Tags are useful due to the variations in the amount of noise contained in different physical locations within a given distance of some address. This amount of variation can be judged by using the *variance* expected. Following Kanerva (1988), the variance with respect to the number of writes to a location will be approximately np^2, where n is the number of writes, and p is the probability that a given write will add data to that location. It is common to use a hamming radius that results in the selection of approximately $N^{1/2}$ locations, where N is the total number of physical locations. Thus, the probability p is $N^{1/2}/N$, or $N^{-1/2}$. The variance is thus n/N.

If we are studying the memory with a given percentage of writes made, for example, $n = .01*N$, the variance is constant. *Since the tags depend on the variance in the number of writes to different locations for correction, and the variance does not degrade as we add more physical locations, we do NOT need to increase the number of tags as we increase the number of physical locations. Also, the variance does not depend on the number of dimensions in the address or data, and so changes in the number of dimensions should not affect the number of tags needed.*

This result is admittedly informal, but is suggestive of the power of data tagging. Once we have determined a useful number of tags at some value of N, we can use the same number of tags for a much increased value of N. Since the experimental results show that a value of N=32 is suitable for the memory, we are assured of the existence of a reasonable number of tags for any memory size.

13. Conclusions

This work proves that there exists a mathematical foundation for an improved method of averaging selected locations in a sparse distributed memory, that data tags can be used to estimate the needed signal to noise ratio in each location, and that experiments confirm a significant and consistent improvement in the recall capability of a sparse distributed memory equipped with data tags. The high quality of the results supports the view that data tags should be strongly suggested for inclusion into future SDM implementations.

Especially interesting was the behavior of tagging with correlated data. The correction abilities of tagging were amplified further when moderately correlated data was used to fill the memory. Since most real-life data contains correlations that would be reflected in most representations, this would help protect the memory from premature breakdown in real applications. This again argues for the inclusion of data tags in future implementations.

Data tags are but one possible way of implementing an ability to estimate the signal to noise ratio of a location. However, they are not necessarily the only method, and are not completely accurate in their estimates. It is quite possible that other methods may be found to give better estimates of the signal to noise ratio.

14. Future directions

A number of experiments and variations were not performed, and it may be interesting to study them at some future date. For example, the effects of *correlated data* were not studied, nor were the effects of tagging on the ability of the memory to *generalize* from a given collection of data. The tag function is also a candidate for change: perhaps tags should be selected by choosing the *least represented tag* in the hamming radius to minimize collisions. Another tag function might optimize the selection of the same tag for close but nonequivalent data patterns. The *optimum* number of tags was not explored, nor its relation to the number of physical locations or the dimensions of an address. The fast computation of the predominant tag is an area of interest; perhaps a procedure could developed where this is performed in parallel with the summation of the data counters (such as a systolic array; see (Keeler and Denning, 1986)). All of these areas offer potentially fruitful areas of study for future researchers.

Acknowledgments

A number of people contributed to the development of this work and the creation of an environment where it was possible to do it. Foremost, Pentti Kanerva deserves mention for this ground-breaking work into SDM and assistance in developing the mathematics needed for the signal to noise calculations. Peter Denning is responsible for a series of challenging and thought-provoking discussions about these issues, and was a primary figure in formulating the essence of the work. Mike Raugh is and has been a strong supporter of this project and its members. Jim Keeler is an enthusiastic member of the SDM group (one day a week!) and inspired the formal drawings for the workings of the memory. Umesh Joglekar originated the idea of using the sphere to generate correlated data. Bruno Olshausen did a number of proofreadings with great enthusiasm. Thanks is extended to all of RIACS, which has shown the foresight to support the development of this work, and offered a friendly environment that made it easy to enjoy the time spent working on this.

Funding agencies have been outstanding in their support of this work. Funds for this work primarily from NASA Cooperative Agreement NCC 2-408. One can not help but feel proud of working with agencies and organizations that made this development a priority.

Appendix A. Signal to noise ratio calculation

We are given a collection of locations L_i that contain a mixture of signal and noise; we write each location in the form $L_i = (s_i S + n_i N)$, where s_i and n_i are real numbers greater than or equal to zero. We wish to combine these locations so as to maximize the signal and minimize noise. We can state this task as finding a set of constants $(w_1 \cdots w_m)$ that maximize the signal to noise ratio when we sum the locations. The formula for our sum is $L_{sum} = (\Sigma w_i L_i)$.

The signal terms simply add. For example, one unit of signal plus one unit of signal equals two units of signal. The noise terms are more complex, since the noise value N in each location is uncorrelated with a noise value from another location. If we assume that N is a normally distributed random variable, then the terms no longer add. Instead, they combine as the square root of the sum of the squares of the multipliers. (For example, adding N to N does not give twice as much noise, but rather $2^{1/2}$ as much noise. This is because random noise can sometimes cancel out rather than augment independent noise. Signal, on the other hand, always augments when added to another signal. For further detail, the reader is encouraged to consult a good text on probability, for example, (Ross, 1984).)

Equation (1) shows the calculation of the signal-to-noise ratio for the combined location L_{sum}.

$$\begin{aligned} \text{Signal}(w_1, w_2, \ldots, w_m) &= w_1 s_1 + w_2 s_2 + \cdots + w_m s_m \\ &= \sum_i w_i s_i \end{aligned}$$

$$\begin{aligned} \text{Noise}(w_1, w_2, \cdots, w_m) &= (w_1^2 n_1^2 + w_2^2 n_2^2 + \cdots + w_m^2 n_m^2)^{1/2} \\ &= (\sum_i w_i^2 n_i^2)^{1/2} \end{aligned}$$

$$\frac{\text{Signal}}{\text{Noise}}(w_1, w_2, \cdots, w_m) = \frac{\sum_i w_i s_i}{(\sum_i w_i^2 n_i^2)^{1/2}} \qquad (1)$$

To find the maximum of the signal-noise ratio, we take the partial derivatives of the signal-noise function with respect to each of the variables $(w_1 \cdots w_m)$. Extrema will exist where all of these derivatives equal 0. The partial derivative of equation (1) with respect to the weight w_i is given in equation (2):

$$\frac{\partial}{\partial w_i}\frac{\text{Signal}}{\text{Noise}}(w_1, w_2, \cdots, w_m) = \frac{s_i \ (\sum_j w_j^2 n_j^2)^{1/2} - w_i n_i^2 \ (\sum_j w_j^2 n_j^2)^{-1/2} \ \sum_j w_j s_j}{(\sum_j w_j^2 n_j^2)} \qquad (2)$$

What we are interested in is where these partials equal zero. Setting each partial to zero and simplifying, we

get the following equations:

$$\text{for } i = 1 \text{ to m}, \qquad w_i = \frac{s_i}{n_i^2} \frac{\sum_j w_j^2 n_j^2}{\sum_j w_j s_j} \tag{3}$$

We still cannot solve for w_i. However, we can solve for the ratio of w_i to w_j:

$$\frac{w_i}{w_j} = \frac{s_i \, n_j^2}{s_j \, n_i^2} \tag{4}$$

Equivalently, we can write the equation for w_j with respect to w_i:

$$w_j = \frac{s_j \, n_i^2}{s_i \, n_j^2} \, w_i \tag{5}$$

These equations give a simple linear interrelationship between the different weights. If we are not interested in having the relative amplitude of the sum correct, we can just select any value for some weight, say w_1, and solve for the remaining weights.

Appendix B. Example of using signal to noise ratio weighting

We can use equation (5) to give an example of combining locations. In this example, we will have three locations, each with one unit of signal, and with one, two, and three units of noise respectively. We will combine the signals both with an unweighted average and with the suggested signal to noise determined weighting to give the reader an idea of the effectiveness of this process. We will begin with three locations, called L_1, L_2, and L_3.

$$L_1 = S + 3N \qquad L_2 = S + 2N \qquad L_3 = S + N \tag{6}$$

We wish to combine these locations so as to maximize the signal to noise ratio. Setting the weight used for location L_1 to 1, we can calculate the weights to use for the remaining locations:

$$w_1 = 1$$

$$w_2 = \frac{s_2 n_1^2}{s_1 n_2^2} w_1 = \frac{3^2}{2^2} = 9/4 \tag{7}$$

$$w_3 = \frac{s_3 n_1^2}{s_1 n_3^2} w_1 = \frac{3^2}{1^2} = 9$$

Equation (7) tells us that the best way to combine these locations is to take one measure of L_1, two and one-quarter measures of location L_2, and nine measures of location L_3. (It is interesting that even in this simple case, the weightings suggested are greatly different from simple averaging.) How much does this change the signal to noise ratio relative to simple averaging of the three locations? We can answer that by calculating the S/N ratio for each case:

Given $\frac{\text{Signal}}{\text{Noise}}(w_1, w_2, w_3) = \frac{\sum_i w_i s_i}{(\sum_i w_i^2 n_i^2)^{1/2}}$ **then**

$$\frac{\text{Signal}}{\text{Noise}}(1, 9/4, 9) = \frac{1 + 9/4 + 9}{(3^2 1^2 + 2^2(9/4)^2 + 1^2 9^2)^{1/2}}$$

$$= \frac{49}{2\ (441)^{1/2}}$$

$$\cong \mathbf{1.16}$$

$$\frac{\text{Signal}}{\text{Noise}}(1, 1, 1) = \frac{1 + 1 + 1}{(3^2 1^2 + 2^2 1^2 + 1^2 1^2)^{1/2}}$$

$$= \frac{49}{2\ (441)^{1/2}}$$

$$\cong .802$$

We can see from these results that the signal to noise ratio in the combined location can be significantly improved by using a *weighted average* rather than an *unweighted average*.

This work suggests that if we can correctly calculate the signal to noise ratio in a location, we can assign a weight to that location so as to maximize the signal-to-noise ratio in the combined location.

Appendix C. Experimental

The experiments were conducted with a SDM program written in C, and running on a SUN-3/60. The size of the memory was 1000 physical locations, with 256-bit addresses and 256-bit data. The addresses of the locations were generated randomly. When data tagging was included, the tags were in the range [1,32], which required 32 tag counters per location. All counters were 16-bits words, and no overflow of the counters occurred during any of the trials.

A scale of 128 was used. All weights were truncated to integers. Signal values greater than 1 were treated as equal to 1 (in a process called *signal equalization*, to be describe more fully elsewhere), since we did not do any writes more than once. If a location claimed to contain only signal and no noise, it was given a signal/noise ration of 5, rather than infinity. This gives such locations a weight 5 times greater than that of a location with one unit of signal and one of noise.

The results used to generate figures 7, 8 and 9 were generated in the following way. A random address and random data pattern were created. The data pattern was written into a cleared SDM at the given address, and a counter for the number of writes was set to zero. This begins the testing cycle: we read from the memory at the given address, and compare the result with the expected data pattern. The number of bits at which they differ is recorded along with the counter. We then write some newly generated random data pattern at some newly generated address, increment the counter, and go to the start of the testing cycle. (This process is identical for graphs where we read with an offset, expect that we flip that number of bits in the given address before reading.) This generates information that can be plotted as *Number of Writes* versus *Bit Errors*.

To avoid effects that were apparent only in a specific run, the number of bit errors was averaged over 8 independent runs. All results shown throughout the paper are from such 8-run averages.

The information for standard SDM and tagged SDM were obtained from the same runs by turning the tagging off and on. This means that the comparisons of tagged versus untagged reading were from memories with identical structure and write histories.

The *% Corrected Using Tags* graphs were computed by graphing the ratio of the number of errors using standard SDM minus the number of errors using tagged SDM, over the number of errors using standard SDM. If both standard SDM and tagged SDM gave no errors, correction was recorded as 100%. In no case in this study using the average of 8 runs did tagged SDM give errors when standard SDM did not.

The results used to generate figures 10, 11, and 12 were generated in an identical fashion, except that all addresses, rather than being randomly generated, were generated by first zeroing and then randomly turning on the required number of bits.

The text strings for the text experiments were generated by randomly pulling names, institutions, and phone numbers from the RIACS addresses file. 500 addresses were created by appending an institution to a name, and either truncating to 32 characters or padding with spaces to 32 characters. 500 data patterns were created by treating phone numbers as text, and then padding to 32 characters with spaces. For the runs, whenever a random address and data pattern were needed for writing, an address string and data string were chosen from the set of 500. After selection, these strings were eliminated from future selection. Other than this difference, the runs were conducted identically to the runs described above.

References

Albus, James S., *Brains, Behavior, Robotics*, Peterborough, NH: BYTE Books, 1981.

Albus, James S., "A theory of cerebellar functions," *Mathematical Biosciences* **10**, pp. 25-61, 1971.

Kanerva, Pentti, *Sparse Distributed Memory*, Cambridge, Mass: MIT Press, 1988.

Kanerva, Pentti, "Self-propagating Search: A Unified Theory of Memory," Center for the Study of Language and Information Technical Report No. CSLI-84-7, March, 1984.

Keeler, J. D., Unpublished results

Keeler, J. D., "Comparison between sparsely distributed memory and Hopfield-type neural network models," RIACS Technical Report 86.31, 1987.

Keeler, J. D. and Denning, P. J., "Notes on Implementation of Sparsely Distributed Memory," RIACS Technical Report 86.15, 1986.

Marr, D., "The cortex of the cerebellum," *Journal of Physiology*, **202**, pp. 437-470, 1969.

Ross, Sheldon, *A First Course in Probability*, New York, New York: Macmillan, 1984.

RIACS

Mail Stop 230-5
NASA Ames Research Center
Moffett Field, CA 94035
(415) 694-6363

The Research Institute for Advanced Computer Science
is operated by
Universities Space Research Association
The American City Building
Suite 311
Columbia, MD 21044
(301) 730-2656