KANERVA'S SPARSE DISTRIBUTED MEMORY: AN ASSOCIATIVE MEMORY ALGORITHM WELL-SUITED TO THE CONNECTION MACHINE

David Rogers

November, 1988

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS Technical Report 88.32

NASA Cooperative Agreement Number NCC 2-387

Research Institute for Advanced Computer Science

KANERVA'S SPARSE DISTRIBUTED MEMORY: AN ASSOCIATIVE MEMORY ALGORITHM WELL-SUITED TO THE CONNECTION MACHINE

David Rogers

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS Technical Report 88.32
26 November 1988

ABSTRACT

The advent of the Connection Machine profoundly changes the world of supercomputers. Its highly nontraditional architecture makes possible the exploration of algorithms that were impractical for standard Von Neumann architectures. Kanerva's sparse distributed memory (SDM) is an example of such an algorithm.

Sparse distributed memory is a particularly simple and elegant formulation for an associative memory. In this paper I describe the foundations for sparse distributed memory, and give some simple examples of using the memory. I continue by showing the relationship of sparse distributed memory to three important computational systems: random-access memory, neural networks, and the cerebellum of the brain. Finally, I discuss the implementation of the algorithm for sparse distributed memory on the Connection Machine.

Keywords: Connection Machine, neural networks, cerebellum, associative memory

---

Work reported herein was supported in part by Cooperative Agreements NCC 2-408 and NCC 2-387 from the National Aeronautics and Space Administration (NASA) to the Universities Space Research Association (USRA). Funding related to the Connection Machine was jointly provided by NASA and the Defense Advanced Research Projects Agency (DARPA).

KANERVA'S SPARSE DISTRIBUTED MEMORY: AN ASSOCIATIVE MEMORY ALGORITHM WELL-SUITED TO THE CONNECTION MACHINE

DAVID ROGERS
Research Institute for Advanced Computer Science
NASA Ames Research Center, Mail Stop 230-5
Moffett Field, California 94035, USA

ABSTRACT

The advent of the Connection Machine profoundly changes the world of supercomputers. Its highly nontraditional architecture makes possible the exploration of algorithms that were impractical for standard Von Neumann architectures. Kanerva's sparse distributed memory (SDM) is an example of such an algorithm.

Sparse distributed memory is a particularly simple and elegant formulation for an associative memory. In this paper I describe the foundations for sparse distributed memory, and give some simple examples of using the memory. I continue by showing the relationship of sparse distributed memory to three important computational systems: random-access memory, neural networks, and the cerebellum of the brain. Finally, I discuss the implementation of the algorithm for sparse distributed memory on the Connection Machine.

Keywords: Connection Machine, neural networks, cerebellum, associative memory

1. Introduction

History suggests that the first conceived use for new technologies is nearly always an improvement on existing processes. It takes the passage of time before new, and often more valuable, applications are conceived. For example, the first movies were usually of people acting on a stage; it took later insight to see that the true power of the motion-picture camera lay in its ability to free the viewer from the theater seat. Another example is the first general-purpose computer, which was built to calculate ballistic information for artillery gunners. It took years before

the confining image of the computer as 'number cruncher' gave way to the more powerful image of *information processor*.

A similar situation exists for the community doing research with supercomputers. A typical viewpoint is that supercomputers are simply faster versions of standard 'mainframe' computers. This led to a culture where speed was the primary measure of success, and most effort was concentrated on porting 'dusty decks' containing classical algorithms on the new supercomputers. In many ways, this viewpoint was justified, as the first generation of supercomputers often *were* just faster versions of their mainframe counterparts.

The introduction of the Connection Machine, a massively parallel supercomputer, no longer allows this viewpoint. Its highly nontraditional architecture makes conversion of existing algorithms challenging or impossible. But more importantly, the architecture makes possible the development of algorithms that were impractical for standard Von Neumann architectures. Supercomputers with nontraditional architectures, such as the Connection Machine, will not be used primarily to run standard algorithms faster. They will be used to run new algorithms that would be *essentially impossible* to run or to develop on standard machines.

This change of focus towards new algorithms is not just driven by the new supercomputers; it is also driven by the needs of application developers. For example, the slow pace of research in computer-related disciplines such as artificial intelligence suggests that what is required are indeed new algorithms, not just computers that run the old algorithms better. The emergence of *neural networks* is a reflection of this search for new algorithms. Many of these new algorithms are poorly suited to standard architectures; the advent of nontraditional architectures will prove vital for continued research in such areas.

One such new algorithm is Kanerva's sparse distributed memory.[1,2] Sparse distributed memory (SDM) is a particularly simple and elegant formulation for an associative memory. The massive amount of computation and memory needed by the SDM algorithm made it impractical for full-scale implementation on standard sequential machines. However, the massive parallelism inherent in this algorithm is well-suited to the architecture of the Connection Machine-2 (CM-2).[3] For the first time, nearly full-scale versions of SDM can be built; this will likely drive the development of both novel applications and a generation of hardware for implementing the memory model.

The importance of developing new algorithms lies not only in their intrinsic value, but in the insights they can give us relative to other, known, algorithms. SDM occupies a particularly valuable position, as it is related to three important computational systems: random-access memory (RAM), neural networks, and the cerebellum of the brain. Studies on the behavior of SDM have implications for our understanding of these systems.

In this paper I begin by giving two simple examples of using a sparse distributed memory. I then describe the algorithm for SDM in detail as a variant of random-access memory. Next, I show the relationship of SDM to neural networks and the cerebellum of the brain. Finally, I show how this algorithm can be implemented on the Connection Machine efficiently and elegantly.

2. Using a Sparse Distributed Memory

Before detailing the algorithm for a sparse distributed memory, it is useful to show some simple examples of the processing done by the memory. The first example is of SDM acting as an *associative memory*, that is, a memory that can recall data when addressed 'close to' the address where the data are initially stored. The second example shows the natural noise-correcting behavior of SDM, where random noise in the data patterns is reduced.

2.1 Sparse Distributed Memory as an Associative Memory

For retrieval of a stored data pattern, a standard random-access memory requires the exact address at which the data were previously stored. An associative memory, however, only requires an address that is *sufficiently close* to the address at which the data were stored. This property makes an associative memory a powerful tool if the addresses may have random noise, outright errors, or may be only partially specified.

Figure 1 shows an example of writing to a sparse distributed memory with a 256-bit address and data size. Each 256-bit pattern is shown as a 16x16 grid of points, which can be seen as a series of bit maps for Roman numerals. The arrow is the direction of the sequence; "A --> B" means pattern B was written at address A. Reading at address A will now recall pattern B.

Such a sequence could also be stored in a standard random-access memory, if one could be built to accommodate the large address and data sizes. However, random-access memory cannot be used if for some reason the exact 256 bits of the reference address are not available. This is one way in which associative memories have an advantage over RAM.

Figure 2 is an example of reading from a sequence when starting at an address containing noise. In this case, reading is begun with an instance of the Roman numeral three containing 35% noise. If an address is sufficiently close to an address where data are stored, an associative memory should return data with less noise than the noise in the original address. This is confirmed, as the memory returns the data pattern for the Roman numeral four containing about 16% noise. Subsequently, reading with this data pattern successfully recalls the Roman numeral five with no noise. The remainder of the sequence can be recalled without any problem.

This example demonstrates that associative memories in general, and sparse distributed memories in particular, are tolerant of errors in the address. The following section shows that sparse distributed memories have another property that is not shared by all associative memories: they are not only tolerant of noise in the address but they are tolerant of noise in the data as well. Associative memories that use a nearest-neighbor rule, such as the memory of Baum, Moody, and Wilczek,[4] are not tolerant of noise in this way.

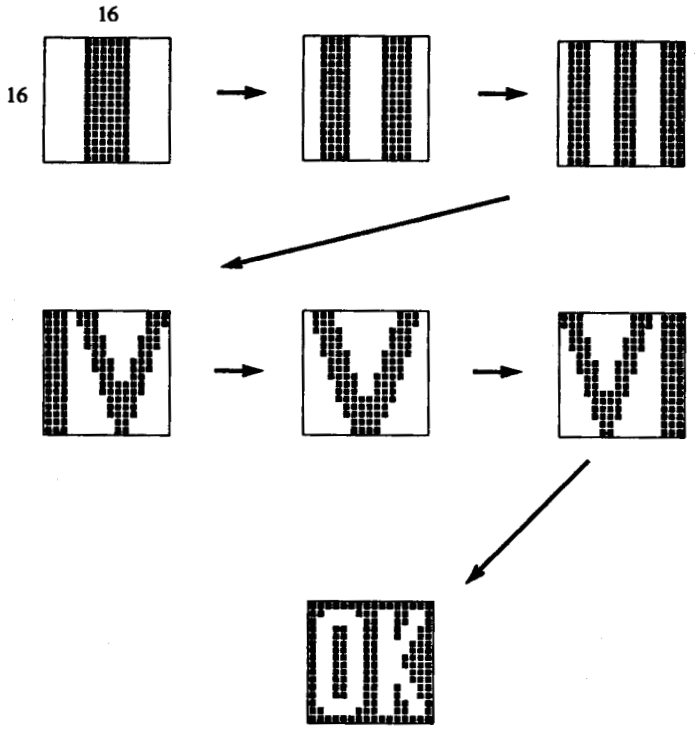

Fig. 1. A series of seven 256-bit patterns that were written into a sparse distributed memory. "A --> B" means the pattern B was written at address A, and that reading at address A will now give pattern B.

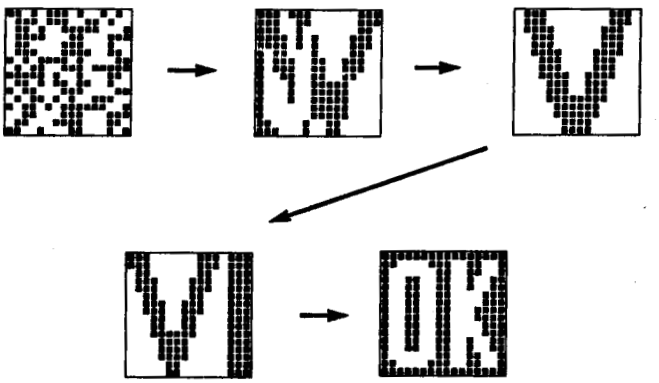

Fig. 2. Reading out a sequence starting with a noisy instance of the Roman numeral three.

2.2 *Sparse Distributed Memory as a Data-Correcting Memory*

Figure 3 shows an example of sparse distributed memory's ability to correct random noise in the data patterns.

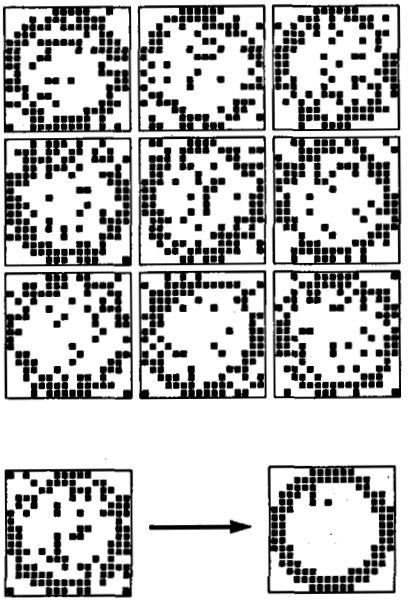

Fig. 3. Example of sparse distributed memory correcting 20% random noise in data patterns.

In this example, each of the top nine patterns is stored in the memory, using its own 256-bit value as both the address and the data pattern. These nine patterns were generated by adding 20% random noise to a bitmap of the letter "O". Reading from the memory at an address nearby the nine written addresses retrieves a bitgraph of the letter "O" which has much less noise than any of the stored data patterns.

Processes such as these are increasingly important as computers are used in real-world applications, where some deviation from perfection must be expected in the input sensors of a system. Algorithms that implement these processes could play an important role in the next generation of computers.

3. Sparse Distributed Memory as a Variant of Random-Access Memory

I have shown some simple examples of the processing performed by a sparse distributed memory and will now explain how a sparse distributed memory works. It is perhaps easiest to explain the sparse-distributed-memory algorithm as a variant of an algorithm commonly used to implement random-access memory. The structure of such a random-access memory is shown in Figure 4. (The example given is for a RAM with 10-bit addresses and data.)

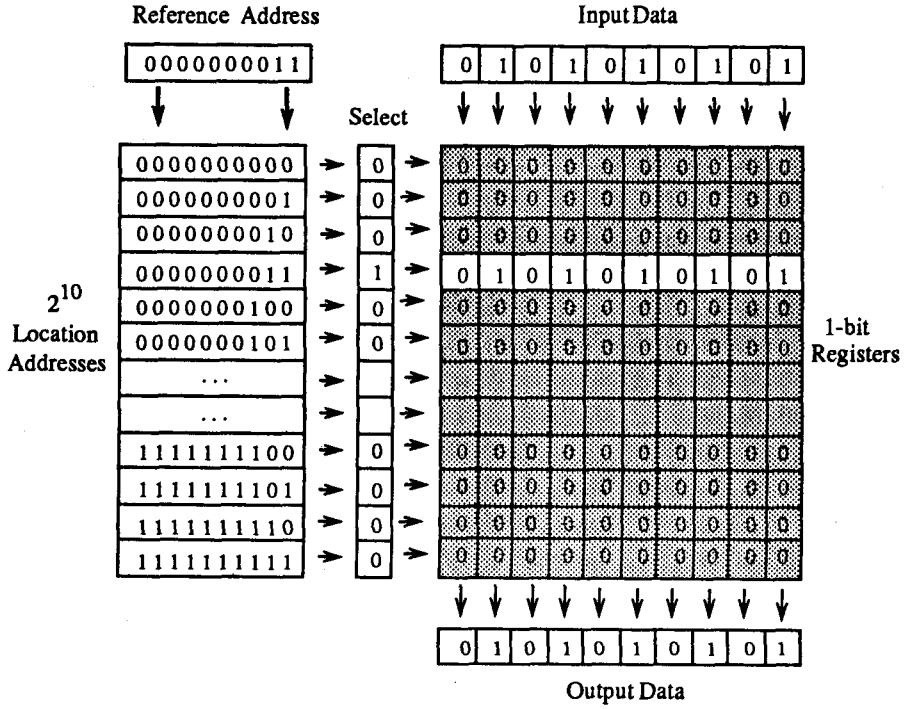

Fig. 4. Structure of a simple random-access memory.

3.1 Structure of Random Access Memory

The address at which reading or writing will be requested is called the *reference address.* The memory compares that address against the address of each of the memory locations. The location that matches the reference address is selected, which is denoted by a 1 in the select vector.

If writing to the memory, the *input data* is supplied. The input data is stored in the ten 1-bit data storage registers of the selected location.

If reading from the memory, the contents of the selected data registers are broadcast on the data bus and made available as the *output data.*

3.2 Structure of Sparse Distributed Memory

Sparse distributed memory can be considered an extension of random-access memory. The structure of a sparse distributed memory is shown in Figure 5. (The reader should note that a typical SDM often has 256-bits of address and data, and can have more than a thousand bits; the example is shown with only 10 bits.)

In each of the three computations done by RAM (addressing, reading, and writing) there exists a major alteration to the RAM algorithm:

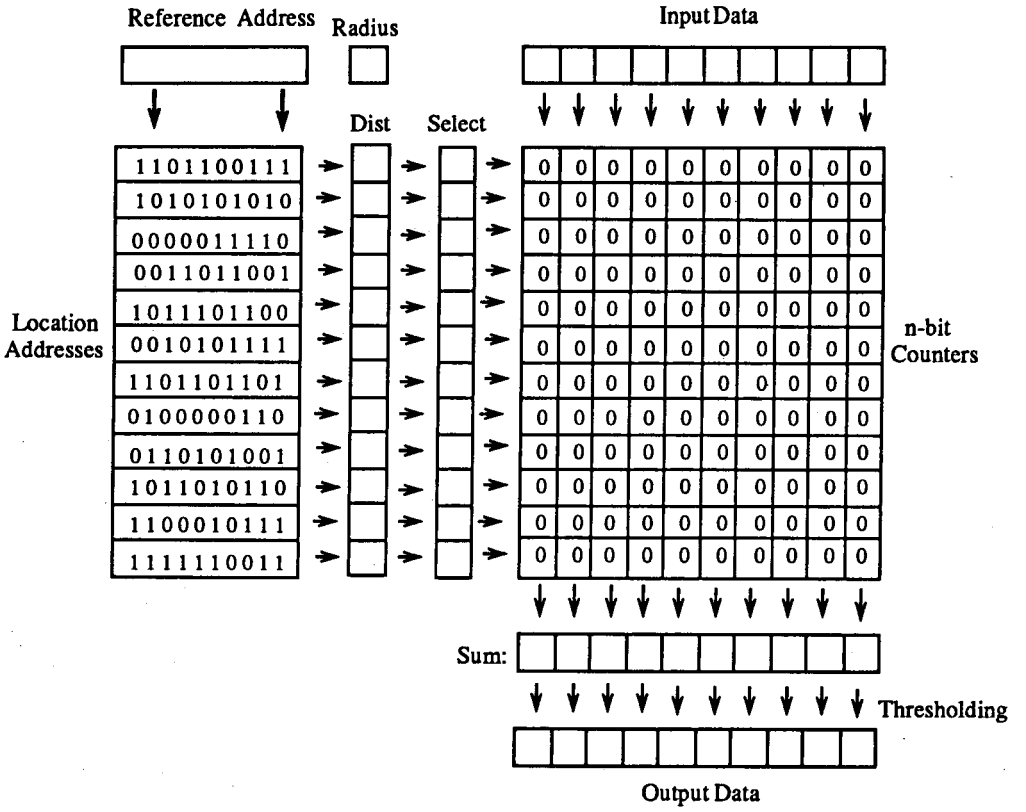

Fig 5. Structure of a sparse, distributed memory upon initialization. The location addresses have been assigned, and the data counters zeroed, but no reading or writing has been performed yet.

- Instead of looking for an exact match between the reference address and the location addresses, the memory calculates the *Hamming distance* between the reference address and each location address. Each distance is compared to a given *radius*; if it is less than or equal to that radius, then that location is selected. *More than one location is usually selected in this process.*
- The data registers are now *counters* instead of single-bit storage elements. These data counters are *n*-bits wide, including a sign bit. When writing to the selected locations, instead of overwriting, the memory *increments* a counter if the corresponding input data bit is a 1, and *decrements* a counter if the corresponding input data bit is a 0.
- When reading, the memory usually selects more than one location. The memory *sums* the contents of the selected locations columnwise, then *thresholds* each sum. If the threshold is zero, sums that are greater than or equal to zero correspond to output bits of 1, and sums that are less than zero correspond to output bits of 0.

These rules *define* the structure of a sparse distributed memory, but they do not offer an intuitive understanding of why this memory functions as it does. I do not present the mathematical foundations of SDM in this paper but instead offer a geometrical argument to explain the functioning of the model.

3.3 Geometric Argument for Sparse Distributed Memory

A property of associative memories in general, and sparse distributed memory in particular, is that the number of physical storage locations is much smaller than the size of the address space. (This is in contrast to random-access memory, which has one physical location for every address.) Indeed, for 1,000-bit addresses, the size of the address space is larger than the estimated number of atoms in the universe; it is inconceivable that one could build a memory having a physical location for each point in this address space. Thus, at best one could assign physical locations only to some very small subset of the address space.

Different associative memories handle the assignment of physical location addresses in different ways; for SDM, the physical locations are assigned to *randomly selected* points in the address space. This provides a relatively even distribution of physical locations over the space. (This is where the adjective *sparse* comes from; the physical locations are sparse relative to the size of the address space.) If we pretend that the address space is a blank sheet, then the physical locations would be seen scattered across the page, as shown in Figure 6. The physical locations are represented by small white or grey squares on the page.

Once the location addresses have been set, a process is needed to decide which locations are selected for a given reference address. Again, different associative memories use different schemes; for sparse, distributed memory, *all* locations within a given radius of the address point are selected. For example, with the reference address A in Figure 6, the six locations shown within the selection radius are selected.

To read from this memory, the information from the selected locations is summed and thresholded. To write, the increment/decrement procedure described above is used. (This is where the adjective *distributed* comes from; the data pattern is not stored in any one physical location but is distributed over a large number of locations.)

Using this figure, it is possible to see why the memory is associative. If a reference address A' sufficiently close to A is used, it will cause the selection of most of the same physical locations selected by A. Thus, the sums retrieved by reading at A' will be nearly the same sums retrieved by reading at A, and so thresholding will likely give nearly or exactly the same output data.

If a random reference address is chosen, say B, there is a risk of selecting some of the locations that were previously selected by A. (In the figure, L is an example of such a location.) L will contain a mixture of *both* data patterns. The output data from reading at A or B will remain uncorrupted as long as the overlap between the selected sets stays small; this is because the reading process *averages* the values from all the selected locations.

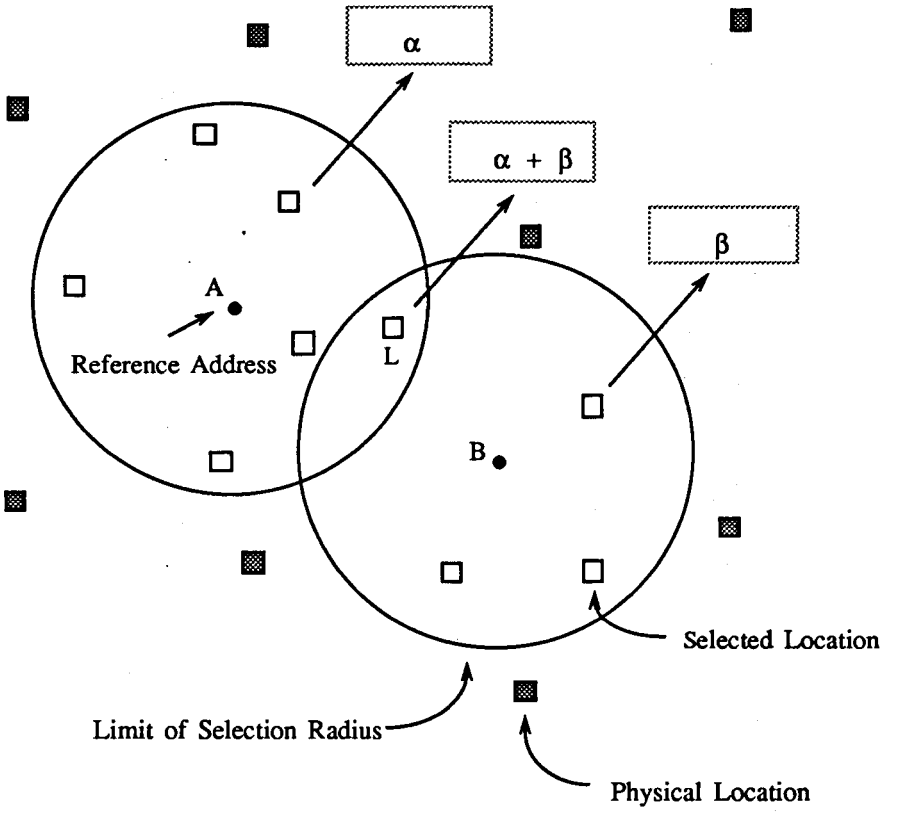

Fig 6. Distributed storage in a sparse memory.

Figure 7 shows the results of a read operation at 0101010110 after writing the data 1011101010 at 1011001010 and writing the data 0001110101 at 0101010110. The patient reader is invited to confirm the results shown.

4. Sparse Distributed Memory and the Cerebellum

For certain kinds of tasks, such as pattern recognition, the human brain is by far the most powerful 'supercomputer' that exists. However, the complexity of its structure and the deficiencies of our technology have led us away from that structure in most of our forty-year history of building general-purpose computers. The current interest in computational 'neural networks' is just the beginning of research that must be done to understand the mechanisms of the brain. Close collaboration between computer scientists and neuroscientists, with each side proposing theories that the others can test, will be important in the work that lies ahead

The cerebellum of the brain seems a possible topic around which this collaboration can begin. The cerebellar cortex is highly regular in structure, with only seven primary kinds of cells. It is a major part of the brain, occupying ten percent of the

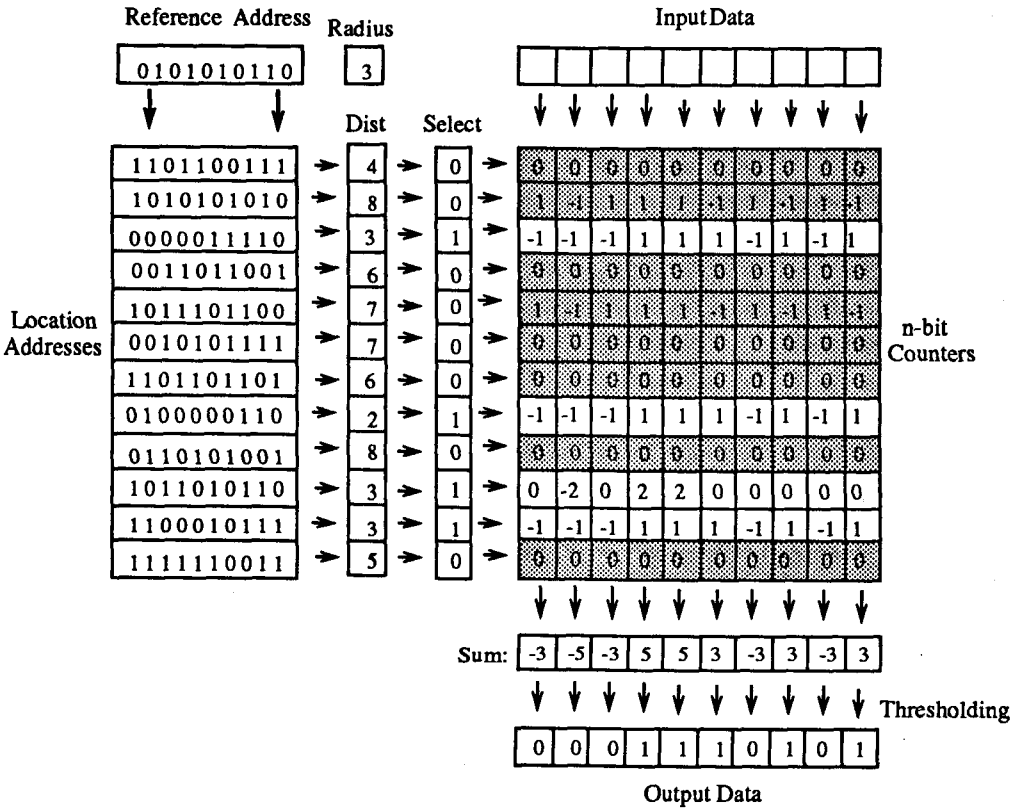

Fig 7. Reading from a sparse distributed memory after two write operations.

brain by volume and possibly half of all the neurons in the brain. Like the cerebral cortex, and unlike most other parts of the brain, the cerebellar cortex has undergone spectacular increases in size in the higher primates. Finally, its involvement in motor control, acuity, and learning would make information concerning its mechanisms valuable for current research work in robotics. [5]

The structure of the cerebellar cortex is shown in Figure 8. The system has two inputs, the mossy fibers and the climbing fibers. It has one output, the Purkinje cells. The granule, basket, stellate, and Golgi cells act internally to the system.

The major part of the input to the cerebellar cortex comes through the *mossy fibers*. Synapsing onto the mossy fibers are *granule cells*; each cell touches between 3 and 6 mossy fibers. The granule cells send their axons upwards, where they split in two and travel down a straight line in opposite directions. These axons are called *parallel fibers*. The *Purkinje cells* send their dendritic trees into these bundles of parallel fibers, synapsing on tens of thousands of different fibers. The output of the Purkinje cell is the only information to leave this system.

Of special interest is the *climbing fiber*, which connects to the system in a very specific way. Each Purkinje cell receives input from exactly one climbing fiber.

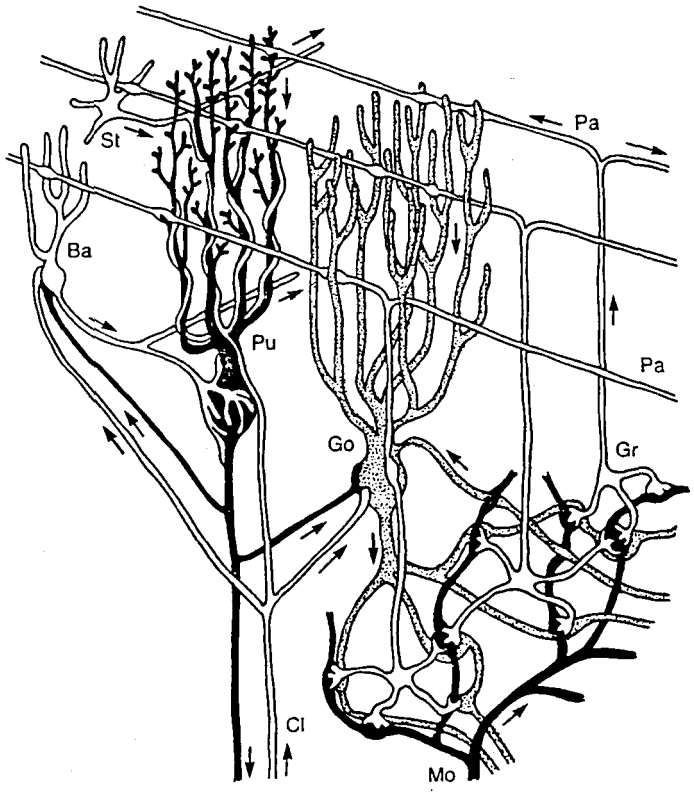

Fig 8. The structure of the cerebellar cortex of the brain. Pu = Purkinje cell (black); Go = Golgi cell (dotted); Gr = granule cell; Pa = parallel fiber; St = stellate cell; Ba = basket cell; Cl = climbing fiber; Mo = mossy fiber (black)

Even more provocative is the manner in which the climbing fiber branches to follow the dendritic tree of a Purkinje cell to its synapses with the parallel fibers.

In 1970, David Marr proposed that the function of the cerebellum is pattern learning.[6] Marr postulated that the firing of the climbing fiber, coincident with the activation of parallel fibers, caused changes in the Purkinje-cell--parallel-fiber synapse which facilitated future firings across those synapses. James Albus, and, later, Pentti Kanerva, independently proposed similar models for the cerebellum. [2,7,8]

These models are now recognized as essentially equivalent. I will refer to this model as the Marr-Albus-Kanerva (or MAK) model of the cerebellum. Of interest in this paper is the relationship of the MAK model of the cerebellum to the sparse distributed memory algorithm. This is best described using the simplified model of the cerebellum shown in Figure 9. (For simplicity, the Golgi, basket, and stellate cells have been left out of the figure.) In this proposed relationship, the mossy fibers are transmitting the reference address for the memory. Each granule cell is

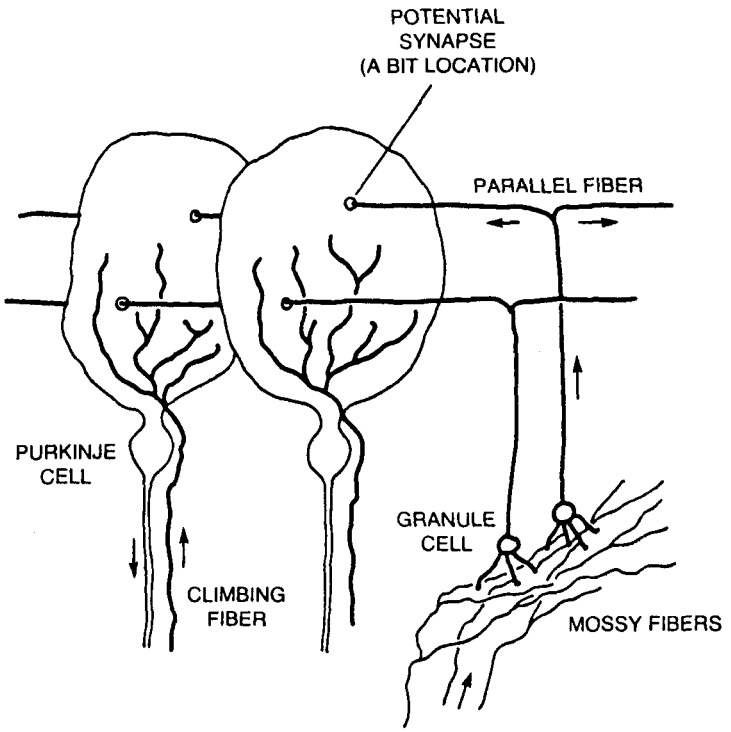

Fig 9. Simplified structure for the cerebellar cortex of the brain.

acting as a memory location; it only fires when the address transmitted along the mossy fibers is close enough to the address it represents. *The firing of a granule cell can be considered equivalent to the selection of a location in the SDM model.* (The Golgi cells, which are not shown in our simplified drawing, may be involved in setting the radius for the memory.)

A major question is the site in the cerebellum corresponding to the data counters in SDM. Kanerva postulates that the data counters are the synapse points where parallel fibers, climbing fibers, and Purkinje cell dendrites meet. This is the same location where Marr postulated learning to take place in the system. If this hypothesis about these synapses is correct, then the climbing fibers could be carrying the input data for the memory. This would explain the careful construction of the cerebellum, where each Purkinje cells receives input from exactly one climbing fiber.

The Purkinje cells provide the output data from the system. As the natural function of a neuron is to sum its inputs and fire if over threshold, they would serve admirably in this capacity and mirror the functioning of a column in a SDM (see Fig. 5).

While this correspondence is suggestive, there is little direct evidence to support it. Even 18 years after Marr suggested a site for plasticity in the cerebellum, there is still a lively debate among neuroscientists as to whether this plasticity

exists.[5,9] This interplay between two branches of science is bound to increase as computer designers begin to depend on the reverse-engineering discoveries of the neuroscientists. The current interest in neural-network models of processing is only the beginning of such a process

5. Sparse Distributed Memory as a Neural Network

Previous sections showed how SDM is related to both random-access memory and the cerebellum of the brain. Given the resurgence of neural-network models of processing, it is of interest that the SDM model can be described as a fully-connected three-layer feed-forward neural network, which is the same network architecture used for backpropagation algorithms.[10] A neural-network equivalent to sparse distributed memory is shown in Figure 10.

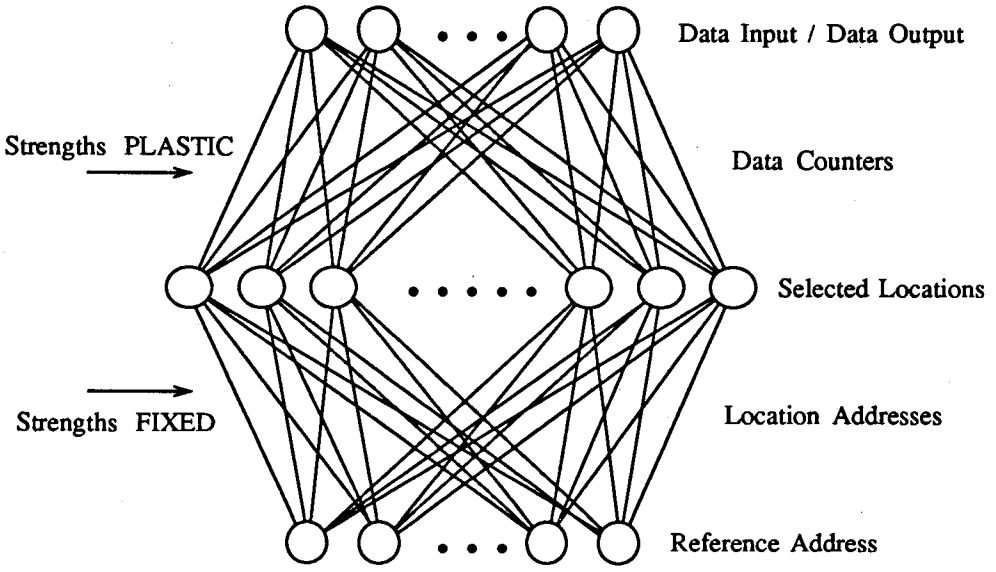

Fig 10. Neural-network representation of a sparse distributed memory. For a reasonable size memory, there might be 1,000 nodes in each the top and bottom layers and 1,000,000 nodes in the "hidden" layer.

The bottom layer is where the reference address is given; that is, there is one node in this layer for each bit of the reference address. These nodes are locked at either 1 or -1 depending on whether the corresponding bit of the reference address is 1 or 0.

The connections between the bottom layer and the nodes of the so-called *hidden* layer are either 1 or -1 in strength. These strengths are never changed, as they determine the address of the physical memory locations.

Each of the hidden-layer nodes corresponds to a memory location in the SDM model. A memory location is selected if the sum of its inputs (i.e., the

dot product of the reference address and the location's weight vector) is greater than or equal to its threshold. This threshold corresponds to the radius in the SDM model, and the sum of the inputs is effectively taking the Hamming distance between the memory location's address and the reference address.

The top layer is where the output data appear. Each hidden-layer node is fully connected to the top-layer nodes. The data counters of a memory location are represented in the strengths of the connections between a hidden unit and the output nodes. This is the only part of the network that is plastic.

Reading the memory involves setting the values of the reference address and reading the output from the output nodes. Writing to the memory involves setting both the reference address *and* the data input nodes to the desired values; internal nodes that are active then add the value of each data input node (one or minus one) to its connection.

In this form, SDM appears quite similar to other neural architectures. However, there are two major differences that may prove important:

- The number of hidden-layer nodes is much larger than is commonly used for neural networks. A reasonable size memory may have an address and data size of 1,000 bits, which would correspond to 1,000 nodes in each of the top and bottom layers. This is large, but not beyond the capabilities of current neural-network algorithms. However, if the memory has 1,000,000 memory locations, this would correspond to a network with 1,000,000 nodes in the hidden layer. It is unclear how standard algorithms, such as backpropagation, would perform with such a large number of units in the hidden layer.
- With backpropagation algorithms, the strengths of the connections in *both* layers are changed. In the SDM model, only the strengths in the *second* layer of connections are changed. This preserves the relationship between the input representation and the nodes in the hidden layer. Without this, learning that occurs in the second layer can become obsolete as the first layer changes. With the large number of hidden units, such obsolescence would be very costly to the system.

The relationship of sparse distributed memory to neural-network models suggests a possible direction for research: neural networks with massive numbers of nodes in the hidden layer. Like SDM, this work has been difficult in the past due to the computational resources available. The Connection Machine seems suited to advance research in this area.

6. Implementational Issues for Sparse Distributed Memory

However attractive an algorithm appears, it is difficult to conduct research on its properties if it is not a good fit to current computational technology. For example, consider a sparse distributed memory with the following parameters:

- 1,000-bit address and data size
- 1,000,000 physical memory locations
- 8-bit data counters
- Radius chosen so that addressing results in approximately 1000 selected locations

Approximate computational and storage requirements for this memory are:

- 100M byte-XORs needed to calculate the select vector
- 1M byte-ADDs to calculate the sum of the selected locations
- Up to 1,000M tests when adding only selected data counters
- 1,000M bytes of storage for data counters

Current computers have too little memory and are too slow for a reasonable implementation of this memory. The storage requirement is unavoidable; any computer would be required to furnish this much storage. The computational requirements could be met either by building specialized hardware or by using a computer that took advantage of the natural parallelism available in the SDM algorithm.

6.1 Parallelism in the Sparse-Distributed-Memory Algorithm

The algorithm for sparse distributed memory contains many opportunities for parallel computation. However, the parallelism may not be accessible for some parallel architectures. Implicit in figure 7 are three major opportunities for parallelism:

- Each memory location, in parallel with all other locations, calculates the hamming distance between its address and the reference address and tests the result against the radius to see if that location is selected. In other words, the address decoders for the location addresses in figure 7 operate in parallel. This requires up to 1,000,000 processors.
- Each selected memory location, in parallel with all other locations, signals its row of counters that they will participate in the upcoming summation.
- Each column of data counters, in parallel with all other columns, computes the sum of the signalled counters. This requires up to 1,000 processors in parallel.

Note that the first and second opportunities for parallelism are *rowwise*, with a large number of processors and small numbers of operations per processor; the third is *columnwise*, with a small number of processors and large number of operations per processor. The same machine must behave with two different 'grain sizes' to utilize fully the parallelism in the SDM algorithm.

(It is also possible to further parallelize parts of these processes; for example, the data counters could be summed using a binary fan-in tree. While recognizing the role that such optimizations can play, these are minor opportunities compared with those of items 1-3 above.)

6.2 Utilizing the Parallelism in the Sparse-Distributed-Memory Algorithm

At first, it would appear that many vector machines would be able to take advantage of the three opportunities presented in the previous section. However, this is unlikely: not only would some of the vectors have to be very large (1,000,000) but the machine would have to be able to mix vectors of different lengths and communicate efficiently among these different representations. No vector machine currently has this ability.

The Connection Machine has the ability to emulate a variety of parallel machines. That is, it can simulate machines with different numbers of processors at different times during one computation. It is this crucial ability that allows it to exploit the inherent parallelism in the SDM algorithm.

In my implementation of SDM, the most time-consuming step in the algorithm is the transfer of data from the rowwise format to the columnwise format. The actual operations of computing, calculating Hamming distance and summing, are relatively fast compared to this communication step.

6.3 Hardware versus Software Issues

The SDM project at RIACS, in collaboration with Stanford University, has developed a hardware prototype of sparse distributed memory. [11] We have also developed a software prototype on the Connection Machine. [12] There are advantages to each.

The advantages of the Connection Machine software are:

- The Connection Machine has more memory (~500Mbytes) than the current generation of hardware under consideration.
- The software can be changed and new designs for the memory tested.
- A software version of sparse distributed memory could be used to help prototype a new generation of hardware.

The advantages of the Stanford hardware implementation are:

- Faster (~50 memory cycles per second versus ~3 memory cycles per second with 256-bit addresses and 8,000 physical memory locations).
- Hardware significantly cheaper than a Connection Machine.
- Hardware can be distributed as an add-on to conventional minicomputers.

Rather than conflicting, the considerations show that the hardware and software complement each other. The flexibility and size of the Connection-Machine-based simulation is offset by the high cost of general-purpose supercomputers. Specialized hardware such as the Stanford SDM prototype is much cheaper than a general-purpose supercomputer. In the future, such hardware will be important in distributing access to these algorithms.

6.4 Results of Implementation of SDM on the Connection Machine

The algorithm for sparse distributed memory was implemented on the Connection Machine. The entire program took ~5,000 lines of source code, written in *LISP (pronounced "star-lisp"). The core algorithm is about a hundred lines long, but as the plans for the software call for public distribution, the software includes a large amount of user-interface and optimization code.

For comparison with the hardware prototype, I created a memory with a 256-bit address and data size and 8,192 physical locations. These parameters correspond to the dimensions of the current-generation hardware prototype. (These parameters, however, are easily changed to model larger memories of various specifications.) The Connection Machine simulator of sparse distributed memory runs at ~3 cycles per second while the hardware prototype runs at ~50 cycles per second. While this is slower than the hardware prototype, it is still acceptable for many applications.

However, it is also possible to build much larger memories on the Connection Machine than are currently available in hardware. On a 64K-processor CM-2, the parameters can be set to create a memory with 1,000,000 physical locations. This memory would be the largest version of a sparse distributed memory ever built, and among the largest neural-network-type programs currently implemented. [13] It would be over 100 times larger than the current hardware prototype. This enormous size allows the development and testing of applications that require significantly more memory than the hardware can provide, such as continuous-speech recognition or shape recognition. The planned future release of the SDM simulator should encourage the development of these and other applications.

7. Conclusions

The advent of the Connection Machine changes profoundly the world of supercomputers. Its highly nontraditional architecture makes possible the exploration of algorithms, such as that for SDM, that were impractical for standard Von Neumann architectures.

The Connection Machine and the algorithm for sparse distributed memory have proven to be an excellent match. The capabilities of the machine are fully utilized by the algorithm, and the opportunities for parallelism in the algorithm are well-exploited by the machine.

The importance of sparse distributed memory lies not only in its intrinsic value, but in its relationship to other, known, algorithms. SDM occupies a particularly valuable position, as it is related to three important computational systems: random-access memory (RAM), neural networks, and the cerebellum of the brain. Studies of the behavior of SDM has implications for these systems as well.

The discovery and development of algorithms that have little counterpart on sequential computers is still a young process, and I expect that the current practice of measuring algorithmic success on supercomputers in MFLOPs will be with us for some time to come. However, this work suggests that the combination of a new

generation of nonstandard supercomputers such as the Connection Machine, and a new generation of algorithms, such as that of sparse distributed memory, may someday change our viewpoint of what success with these giant 'number crunchers' really means.

Acknowledgements

This work was supported in part by Cooperative Agreements NCC 2-408 and NCC 2-387 from the National Aeronautics and Space Administration (NASA) to the Universities Space Research Association (USRA). Funding related to the Connection Machine was jointly provided by NASA and the Defense Advanced Research Projects Agency (DARPA). All agencies involved were very helpful in promoting this work, for which I am grateful.

I would also like to acknowledge Thinking Machines, Inc., for their original grant to me when I was working as a postdoctoral fellow at MIT. It was either great foresight or extraordinarily good luck on their part that they chose someone who would end up working on the machine they were busy creating.

The entire RIACS staff and the SDM group has been supportive of my work. Mike Raugh was an especially strong backer of this project. I also thank the researchers at the NAS project for making me feel at home in a strange building.

Finally, I'll get mushy and thank those who supported my spirits during this project, especially Pentti Kanerva, Rick Claeys, Bruno Olshausen, John Bogan, and last but of course not least, my parents, Philip and Cecilia. Love you all.

References

1. Kanerva, P., *Sparse Distributed Memory* (MIT Press, Cambridge, MA, 1988).
2. Kanerva, P., "Self-Propagating Search: A Unified Theory of Memory", Technical Report, Center for the Study of Language and Information, Stanford, CA, CSLI-84-7, March, 1984.
3. Hillis, D., *The Connection Machine* (MIT Press, Cambridge, MA, 1985).
4. Baum, E., Moody, J., Wilczek, F., "Internal Representations for Associative Memory," *Biological Cybernetics*, (1987).
5. Ito, M., *The Cerebellum and Neural Control* (Raven Press, New York, 1984)
6. Marr, D., "The cortex of the cerebellum," *J. Physiology* (1969) 437-470.
7. Albus, J.S., "A theory of cerebellar functions," *Math. Bioscience.* (1971) 25-61.
8. Albus, J.S., *Brains, Behavior, and Robotics* (BYTE Books/McGraw-Hill, Peterborough, NH, 1981).

9. Lisberger, S.G., "The Role of the Cerebellum during Motor Learning in the Vestibulo-Ocular Reflex. Different Mechanisms in Different Species?" *Trends in Neurosci.* **5** (1982) 437-441.
10. Rumelhart, D.E., Hinton, G.E., Williams, R.J., "Learning Internal Representations by Error Propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Volume 1: Foundations*, eds D.E. Rumelhart and J.L. McClelland (MIT Press, Cambridge, MA, 1985) pp. 318-362.
11. Flynn, M.J., Kanerva, P., Ahanin, B., Flaherty, P. Hickey, P., Bhadkamkar, N., Lochner, E., Zeidman, B., "Sparse Distributed Memory Prototype: Principles of Operation," Technical Report, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffett Field, CA, 88.12, 1988.
12. Rogers, D., "AARON: A Sparse Distributed Memory Simulator for the Connection Machine," Technical Report, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffett Field, CA, in press, 1989.
13. Brown, N.H., Jr., "Neural Network Implementation Approaches for the Connection Machine," in *Neural Information Processing Systems*, ed D. Anderson (American Institute of Physics, New York, 1988) pp. 127-136.