The Cellular Neural Network Associative Processor,

C-NNAP.

Jim Austin[1]*, Martin Brown, Stephen Buckle, Anthony Moulds, Rick Pack,*

Aaron Turner.

January 1994

Advanced Computer Architecture Group
Department of Computer Science
University of York
York, YO1 5DD,
United Kingdom.

austin@minster.york.ac.uk, Tel. 0904-432734
brian@minster.york.ac.uk
sjbuckle@minster.york.ac.uk
rick@minster.york.ac.uk
amoulds@minster.york.ac.uk
aaron@minster.york.ac.uk

Key Words;
Associative Processor, ADAM, Neural Networks, Associative Memory, Image Analysis.

1. Please send any correspondence to this author.

Abstract

This paper describes a novel associative processor that uses neural associative memories as its processing elements. The machine has been designed to tackle problems in AI and computer vision and using nodes that allow rapid search using inexact information over very large data sets. Each processing element is an advanced distributed associative memory (ADAM) which is capable of storing large numbers of pattern associations and allow rapid access. As a neural network, the memory performs associative match, not by conventional CAM approaches, but by forming a mapping between the patterns to be associated. The benefit of this is rapid access, fault tolerance and an ability to match on inexact data. These abilities arise due to the distributed storage used in the memory, which also allows for high capacity in the memory. The memory is designed to work on large data word sizes, i.e. matching can take place on data items as small as 64 bits or as large as 1Mb. The machine is ideally suited as a pattern processing machine, which is where most of the application work has been centered. This is because it is capable of taking large subsections of images and performing matching and comparisons on these.

The paper describes a cellular neural network associative processor (C-NNAP) which supports a number of ADAM systems operating in parallel. The ADAM memory is particularly simple to implement in dedicated hardware, requiring a small amount of custom logic to allow operation at high speed. However, large amounts of memory are required for implementing any practical problems. The design of the processor card within the processor is described which consists of FPGA logic to support the ADAM evaluation, and a DSP processor for control and local memory for the ADAM associations.

1 Introduction.

This paper describes an associative processor based upon a neural associative memory. The architecture of the processor is basically an array of Advanced Distributed Associative Memories (ADAM). These differ from conventional associative memories in that the associative recall is performed using a high speed correlation matrix memory where data is forced into a smaller space resulting in faster search times compared to conventional listing associative memories.

The machine is primarily aimed at solving problems in image recognition and analysis, although applications in general image processing problems have been considered. The machine is currently in use in our group where its limitations and extensions are under investigation.

In the following, section 2 describes the motivation for the work. Section 3 describes the neural associative memory used in the machine and why it is preferable to more conventional associative devices. Section 4 describes the abilities of the neural associative memory compared with conventional memories. Section 5 describes the architecture of the C-NNAP machine, and how it provides a view of the machine that is easy for a programmer to understand and use. Section 6 describes our current implementation, and describes the hardware trade-offs that we have had to deal with. Section 7 gives the software support and describes how the machine is currently programmed. Finally, section 8 shows how the machine is being used to solve image processing problems.

2 Motivation.

This work has been motivated by the need to construct architectures that support processing of pattern data rather than numeric data for applications where the data is inexact. In the current paper 'inexact' is taken to mean similar to data seen previously but not exactly like it, i.e., belonging to a previously known class or set. For example, in image processing raw image data is expressed as a set of pixel values that map onto some domain that may represent an edge or some other feature. In document processing, the pixels may map on to a letter in the alphabet or a feature of a letter. In more commercial applications one may want to see if a data set is in some sense 'typical', i.e. does this person's characteristics fit the type of person that would buy my product?. Our particular motivation for constructing such a machine comes from our study of image processing problems.

All these problems require a classification process to take place. The process requires a system to identify whether X belongs to CLASS Y. Unfortunately X will be unlikely to exist exactly in Y, and could exist in two or more classes. Any system that performs this operation must have a model of the classes and have a means by which an unknown example is seen as belonging to the classes.

This classification process is fundamental to many problems, and many inexact matching processes have been developed (from Bayes Theory to Fuzzy sets). The present work has investigated the use of neural networks to perform the classification process. We have been keen to implement systems that operate at high speed both in the process of learning classifications and in the process of recognizing unknown examples. This is required if the machine is to be used on realistically sized examples. Unfortunately many neural network methods suffer from long learning times for even small sets of data. The ADAM memory does not suffer from these problems. It can acquire new classes very rapidly and scales well as the problem size grows. Becuase it uses correlation matrix memories (see sec 3). The memory implements a classifier and an associative recall process, i.e., the memory is capable of matching an unknown example to a class and then recalling a 'typical' example, or stored label, for that class.

The use of this type of memory in image processing has been explored at many levels of the problem (Austin, Brown, Kelly and Buckle 1993). For example, in low level image processing it may be necessary to identify line segments, textures, etc. At this level, individual ADAM systems are trained (i.e., associations stored) on typical examples of lines and textures, then the image is convolved using the memory to identify the possible features. The memory is able to output a label at each point to indicate the feature found. This operation can be achieved by using an array of ADAM systems operating in parallel over the image. Later stages in the process may require memories to communicate their outputs to each other. For example current work is investigating an algorithm for extracting lines from images, i.e., labelling a set of co-linear line segments as belonging to the same line. This requires iteration between the memories and is described in more detail in Austin, Brown, Kelly and Buckle (1993).

These image processing examples lead to the design of the cellular neural network associative processor (C-NNAP). The machine supports arrays of ADAMs working in parallel and allows memories to communicate with each other in a cellular (local neighbourhood) fashion. An experimental system is available to our group.

3 The associative memory element.

As described in section 2, the C-NNAP machine uses an associative neural network, ADAM,

for its processing elements. This associative memory differs from conventional memories used in associative processors (for exmple, see Potter, 1992) in that it performs associative look-up through a mapping process rather than a listing process. Consider the problem of identifying if an item X is similar to any stored items and, if it is, to recall an associated item X' (this is the class of the item). The simplest way to do this is to match the input item X to all the stored items (listing approach), and select the most similar pattern as the best match. The ‘class’ this pattern belongs to is then reported. The similarity measure used between the input and all those stored can be the Hamming distance (when using binary patterns) or Euclidean distance (when using continuous patterns). The major problem with this approach is the time taken to perform the operation on large data sets. When many millions of examples are to be matched this can result in very long processing times, which can limit the usefulness of the approachIn many applications. The speed problem can be overcome using dedicated implementations of associative memories. Unfortunately, dedicated implementation of these types of memory is expensive.

To overcome this problem the mapping memory does not store all the examples to be matched separately but, instead it stores a compact representation of each of the examples. It can be seen as forming a ‘function’ that, given any input pattern X, calculates the possible class, X', that X belongs to. The function is formed through ‘training’. Many neural networks operate in this way, which allows very rapid recall of associations as the function is quite small and can be evaluated quickly. Unfortunately, they nearly all take a long time to find this compact function (train). This is because the network needs to search for a compact function that optimally maps the patterns to be associated with each other. Our approach, used in the ADAM memory, does not need to search for an optimal solution, but 'builds' a mapping function through a set-based operation using a correlation matrix memory.

An ADAM memory is made up of a number of stages. An N-tuple pre-processor, a correlation matrix memory, an intermediate class and a second stage correlation matrix. The following describes its operation and the rationale for its design.

Fig 1 shows the correlation stage of the memory; this is the section of the memory that

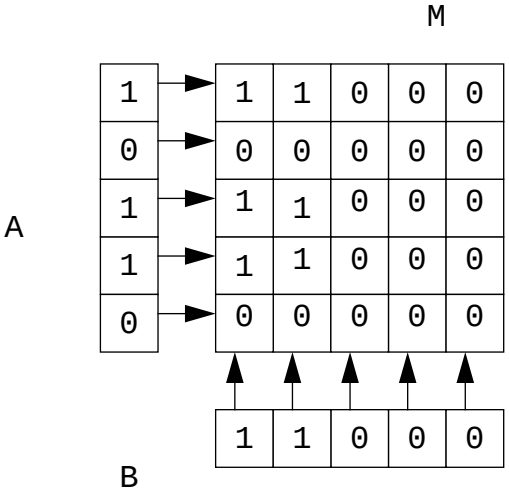

Fig 1. A correlation matrix trained on two patterns A and B. The matrix, M, represents the outer product of A and B.

effectively stores associations. The input pattern and output pattern to be associated are placed in the buffers A and B. Both patterns are binary. The memory can be viewed as a single layer neural network, but is best seen as a matrix, M, of binary digits, A_{max} x B_{max} in size, where Amax, Bmax are the size of the arrays 'A' and 'B'. During training, each weight records if a bit in the input pattern at logical 1 matched a bit in the output pattern at logical 1. If they do, the weight is set to logical one. (All weights are binary 0 to start with.). This is the same as taking the outer product (or tensor product) of A and B to produce M. Each subsequent pair of patterns results in another matrix, which is ORed on top of the original. Equation (1) describes this

$$M = \bigcup_{alli} \otimes \qquad (1)$$

operation which is used to train the network, where $\bigcup$ represents a logical OR, and $\otimes$ represents a tensor product (i.e. the outer product of two binary vertors) and where A_i and B_i are two example patterns to be trained. To recall from the memory using an unknown pattern 'A' a matrix vector multiplication is applied to M. This results in a real valued vector, C,

as shown in equation (2).

$$C = A^T M \qquad (2)$$

This operation is also illustrated in Fig 2. The next stage is to threshold the result, C, to recover the original binary pattern. ADAM uses a special K point thresholding method (Austin 1987) (sometimes termed L-max (Casasent, Telfer 1992)). In order to use this thresholding approach, every C pattern used in associations must have K bits set to one. If this is done then the recalled array, 'C', can be thresholded by selecting the highest N values from the array, and setting them to one, and setting all others to zero. It has been shown (Austin 1987) that this representation of data allows reliable recall of data.

Although the simple correlation matrix described above can be used to store associations, it suffers from poor capacity. i.e. the number of patterns it can store before recall results in an error is low. This is because (1) the memory suffers from excessive cross talk between associations stored, and (2) the size of the memory is bounded by the input and output array sizes. In the small example above, two 5 bit patterns are associated. The maximum number of patterns that can be stored in these memories is quite small (Wilshaw, Buneman, Longuet-Higgins, 1969).

To overcome these problems the following steps are taken. Firstly, an orthogonalizing

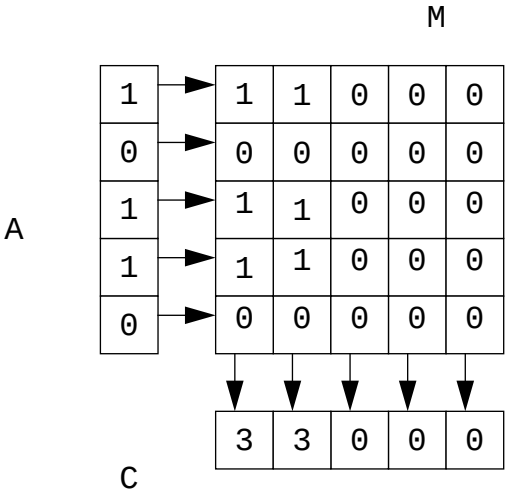

Fig 2. The recall of a previously associated pattern, 'A', using the matrix M, resulting in the class array'C'.

preprocessor is used which ensures that the input patterns are sufficiently separated (in pattern terms) to allow the storage and recall to be more reliable, secondly, a two-stage correlation memory is used to allow the size of the memory to be independent of the size of the patterns to be associated. These two stages are illustrated in Fig 3, which now shows the full ADAM system.

In this example of the memory, the input pre-processor consists of a set of binary decoders. These decoders each take a unique set of N bits from the input data (a tuple) and assign a unique state. The state can be determined in a large number of ways. In this example, the tuple is put through a 1 in N decoder (fig 4 shows the state assignments to each pattern). This is a fast and

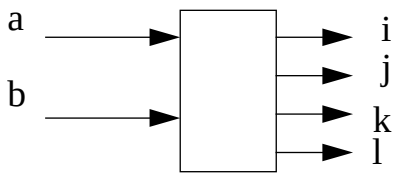

Fig 4. The N tuple decoder state assignments for the decoders shown in fig 3.

Table 1:

a	b	i	j	k	l
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

effective way to ensure all possible 2^N states of a tuple are given a unique state. Every bit in the input data is passed uniquely to a tuple. The result of this operation is an array of states, 2^N states for each tuple. As mentioned above, this operation acts to make the possible set of patterns presented to the correlation matrix more sparse, and thus more different from each other. This 'orthogonalization' process reduces the possibility of pattern recall failure. The effect is most noticeable if the tuple size, N, is equal to the size of the data array. This would result in a state being defined for every possible input pattern to be assigned by the decoder. The memory would then be a simple look-up device. Unfortunately, this approach is not feasible for pattern sizes involved in most tasks including image processing. Furthermore, the system would not generalize, i.e., be able to recognize patterns that were not exactly like those taught into the associative memory. In practice, setting the tuple size to a value between 4 and 8 results in good

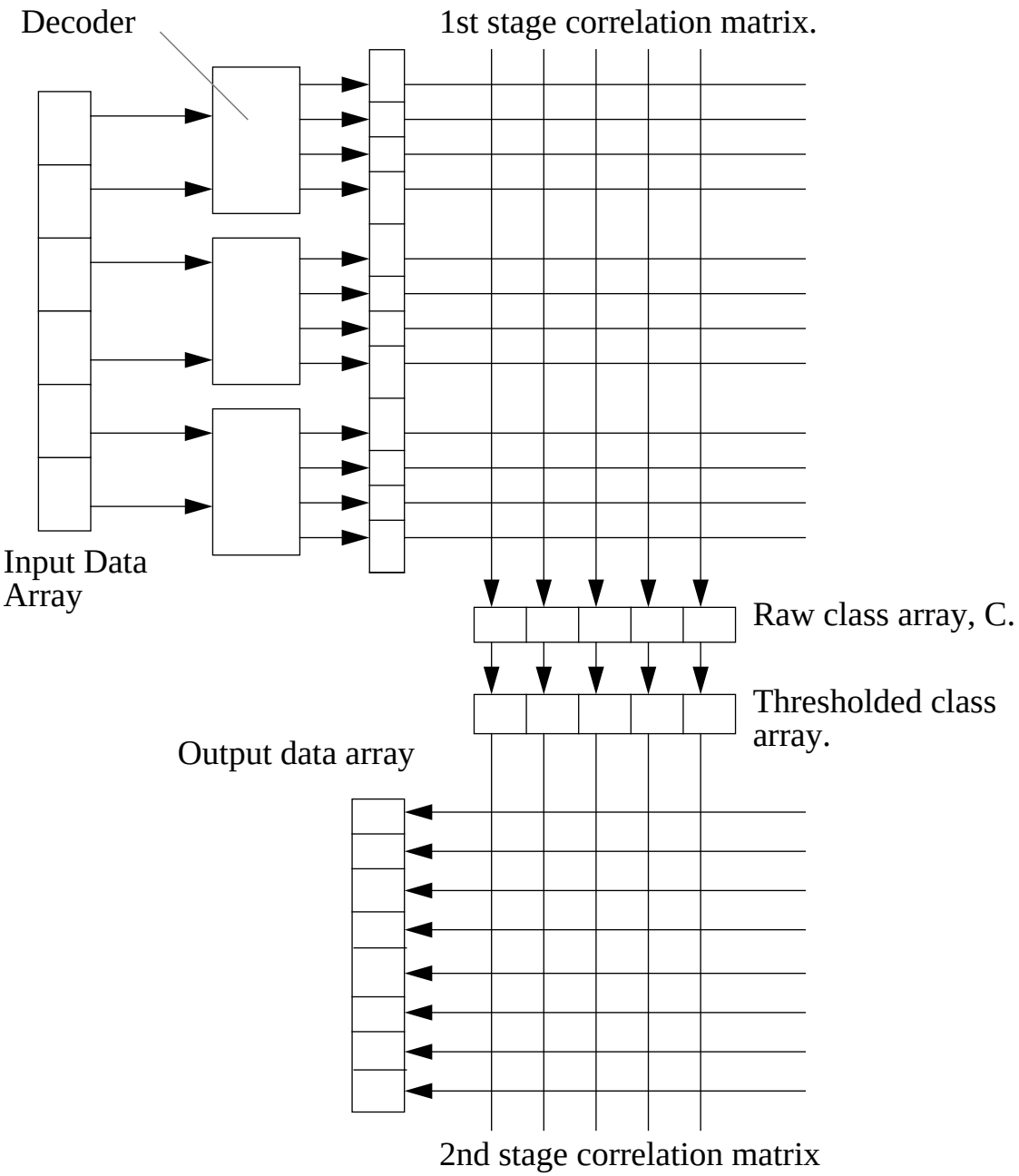

Fig 3. The ADAM memory showing the use of two correlation matrix memories and the N tuple pre-processing.

orthogonalization between the input patterns while maintaining an ability to generalize. This technique is described in (Aleksander, 1985) which presents an analysis of the N tuple size against generalization.

The next stage shown in fig. 3, is to pass the state assignments to the correlation matrix. As is shown, the array of tuple states forms an access list to the rows of the correlation matrix. The correlation matrix is taught and tested as described by equations (1) and (2).

Fig 3 shows two correlation matrices with an intermediate class pattern. The aim of this arrangement is to allow control of the storage in the memories. If two data arrays were associated in one matrix the size of the matrix (assuming no N tuple pre-processing) would be Amax x Bmax, where A_{max} and B_{max} are the size of the arrays to be associated. Unfortunately, this results in a fixed size of memory, determined solely by the size of the data arrays to be associated and not by the number of associations to be stored. As a result the number of associations that can be stored is fixed. By using a two stage memory, the number of associations becomes independent of the size of the input and output arrays. It now depends on the size of the intermediate class array, which can be large or small, as required by the problem.

The operation of the two stage memory is simple. During associative storage, the two patterns to be trained are presented and a unique class is selected. The class pattern is an N point pattern as described above. The data is associated by training the correlation memories as described above. Recall is also straight forward. The key pattern is presented into buffer A, then the N tuple process prepares an expanded version of the data. This is then presented to the first correlation matrix and the first memory tested. The raw class, C, is recalled and K point thresholded. This pattern is then presented to the second memory and this part of the memory tested to recall the data associated with the key. The thresholding used on the data recalled from the second stage is a simple 'Wilshaw' thresholded (Wilshaw. Buneman, Longuet-Higgins, 1969). It involves setting any element set to K to 1, the rest to 0, where K is the number of bits set in the class.

The number of patterns that can be stored in the associative memory is approximately given by

equation (3). K is the number of bits set in the class, C_{max} is the size of the class, I is the input

$$T \approx \frac{\ln\left(1 - \frac{1}{C_{\max}^{(\frac{1}{V})}}\right)}{\ln\left(1 - \frac{K \times V}{C_{\max} \times I}\right)} \qquad (3)$$

array size to the correlation matrix, and V is the number of bits set to one in the input array. It includes the number of patterns that can be stored in the correlation matrix used in the first stage and the second stage of the ADAM memory. The full derivation can be found in Austin (1987).

The structure of the memory described here is the original ADAM described in Austin (1987). There have been many enhancements to this structure since then. The use of non binary input data, the use of continuous weights, the use of an optimizing process for class selection and the use of an optimizing pre-processor (for the last of these see Bolt, Austin, Morgan, 1992).

4 Recall reliability against speed in the processor node

The most important aspect of the memory is how it allows faster associative recall than conventional listing memories (i.e., classical content addressable memories), while coping with inexact input data. The memory trades off speed against recall reliability. For example, to store a fixed number of associations, the ADAM memory may be made small and fast, but with a high probability of recall failure, conversely it may be made large with a slower recall time, but with a lower probability of recall failure. The approximate relationship between probability of recall failure and size is given by equation (4), which gives the probability of successfully recalling a

$$P \approx 1 - \left(1 - \left(1 - \left(\frac{K \times V}{C_{\max} \times I}\right)^{T}\right)^{V}\right)^{C_{max}} \qquad (4)$$

data item from a correlation matrix (from Austin (1987). This relates to the first or second stage correlation matrix. The fact that the memory may fail can be quite worrying in a lot of applications. However, in many real time applications it is better to get any answer in a given time than no answer at all, as long as you can tell if the answer is correct or not. For example, consider a listing memory that takes a fixed time M to recall data, but the memory recalls data that is always correct. The mapping memory takes time Q to recall, where Q << M. An application requires a result less than time P to meet a hard real-time deadline. It may be the case that M > P but Q < P. The listing memory cannot be altered to achieve this dead-line, but the mapping memory can, at the cost of a lower possibility of recall success (it is explained later how this probability of recall failure can be improved). Typical applications that require this behaviour are in aircraft tactical systems, that must make a decision before an obvious real time deadline. This is also found in the nuclear industry, in chemical plants, or any application where any result will be better than none at all.

The ADAM memory can detect a recall when using exact input data (noiseless) through the K point threshold mechanism. If the class cannot be thresholded so that exactly K elements are set to 1, the recall has failed (i.e., if more than K bits have the same value). Most recall failures are of this type. However, it is possible that patterns taught to the memory cause a 'ghost' pattern to be recalled. This is a result of unintended internal correlations in the memory, where two or more patterns coincide to produce a third pattern that is incorrect. Both cases of recall failure can be completely detected and removed by validating the memory after training by comparing the data recalled against what was trained. Any recall failures or ghosts, can be identified and the memory retrained to remove them. Without validation, there remains a small probability that failures can not be detected.

A simple comparison will show the recall speed and accuracy of a correlation matrix against a listing memory. Consider performing recall on one pattern against M data items of I bits in size. A listing memory would require M lots of I bit matches, where a match would be a hamming distance comparison. Assuming the hamming distance calculation took H(I) time to compute, the operation would take in the order of MxH(I) operations to perform. A correlation matrix, as used in the first stage memory of ADAM would take the input pattern (I bits in size) and

associate it with a class of C_{max} bits in size (assuming no N tuple pre-processing). If the operation of the correlation matrix is examined, the raw class recovery is equivalent to C_{max} number of I bit hamming measures. Each element of the raw class is the result of performing a hamming distance measure between the input data and the column in the matrix[1]. Thus the computation time is in the order of C_{max}xH(I) operations. As long as $C_{max} < M$ the recall speed of the correlation matrix will be faster than the listing memory. Unfortunately, as shown above, the correlation matrices are inflexible and error prone on their own .

The ADAM memory uses two correlation matrix memories for its computation, as well as a constant time overhead to do the N tuple pre-processing and class thresholding. However, to fully compare the two methods, the listing method requires some means of recalling the pattern associated with the input data. For this reason we ignore the time to compute the second ADAM correlation matrix in this analysis.

The N tuple pre-processing has the effect of increasing the array size of the input data prior to its application to the correlation matrix. The subsequent computation is not increased by this operation. This is because the number of bits set in the data pattern after N tuple pre-processing are less than or equal to the number of bits set to one in the original input data (inspection of the N tuple method will make this clear). The number of bits set to one in the data array applied to the correlation matrix is equal to the number, V, of N tuple samples taken from the input data, as can be seen from the architecture in fig. 3 and the use of the decoders as laid out in fig. 4. For each active bit on the input to a correlation matrix, a row of the correlation matrix is added into the raw class. Thus, V rows of the correlation matrix are summed into the class. Hence, the computation is in the order of V x A(C_{max}), where A() is the time to add one bit in the correlation matrix into a raw class element. If we assume that the computation required for a bit comparison in a Hamming distance calculation is equal to the time to add, then A() = H().

These results suggest a ratio of computation of ADAM: Listing memory given by:

1. Note that in the hamming distance measure all the elements that are different between two binary patterns are counted, whereas in the correlation matrix memory only the elements that are at logic 1 in the input data are compared, and counted.

V x H(C_{max}): M x H(I),

Where V = Number of N tuple samples = number of bits set to one in data presented to the first correlation matrix in ADAM;

C_{max} = The number of elements in the class in ADAM;

I = The size of the input data array in the listing approach;

M = The number of patterns to be stored in the listing memory.

Note that V << I always holds. In ADAM, for perfect recall the class size is made equal to the number of data items to be stored (M), using a K point size of 1, i.e. C_{max} = M. Thus, the correlation matrix method is faster by a factor I/V. This speedup is entirely due to the use of the N tuple pre-processing. In practice I/V is between 4 and 8. Unfortunately, the gain is offset by the need to perform N tuple pre-processing, which adds to the computation time. As this depends on implementation, under these conditions (where C_{max}=M) we can assume that the ADAM approach is as fast if not faster than the listing memory approach. It will be noted that large N tuple sizes will result in a smaller value for V and thus a corresponding speedup. However, it can result in unacceptably high memory use. The memory used, U, by the first stage correlation memory in ADAM is given by;

$$U = \frac{I}{N} \times 2^N \times C_{max}$$

where C_{max} is the size of the class array, I is the input data array size and N is the tuple size.

The interesting case is when the size of the class is made smaller than M. In this case the class has more than one bit set. If K bits are set then the number of unique class patterns are C_R^K, which is the maximum number of patterns that can be stored (note that typically the memory would be saturated long before this maximum is reached). When the class array size C_{max} is less than the number of stored examples, M, the probability of recall becomes less than 1 (see equation 4).

The main application of the memory has been in image processing, where fast lookup is required. In this application the data presented to the memory is different from the training data. The memory is able to identify the closest stored association to the data inputted, and recall the associated pattern.

5 The processor architecture.

Our interest in the ADAM system has been for image processing. For our problem, we require an array of associative memories, each inspecting a small part of the image (Austin, Brown, Kelly, Buckle 1993). To allow us to build a machine that would be easily programmable, we have defined a simple high-level architecture that is generic enough to allow us to specify a wide range of specific architectures for particular problems. From this, we have defined the programming interface and constructed a machine to allow a user to easily build and execute a given architecture.

The simple model we use has two major components; ADAM units and input/ output memories. The ADAM units can be connected to the input/output memory by a mapping function which defines where each data bit of the input to ADAM is connected to in the input/output memory (io-memory). The io-memory is logically a 1-dimensional array of 8 bit bytes. Any number of ADAM units can be connected to an io-memory, and a system can have as many io-memories

as required (see Fig. 5). The ADAM units consist of a number of stages of processing, which

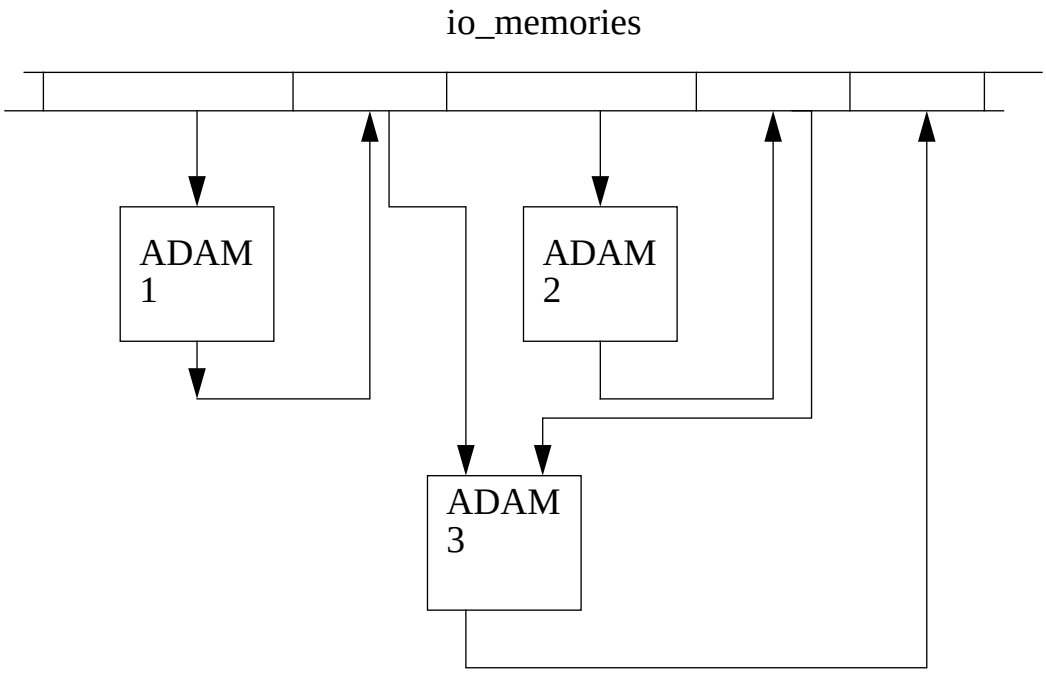

Fig 5. A C-NNAPs architecture using io_memories and ADAMS.

are controlled by a number of functions;

1) Map in; Force the data in the io-memory onto the input of an ADAM by using the mapping function defined for the memory.

2) Group the data in the ADAM input buffer into tuples.

3) For each tuple, find the state assigned to it (using a decoder as shown in Fig. 4).

4) Access or recall, using the tuple state assignments, from the 1st stage correlation matrix. Produce a raw class.

5) Threshold the raw class to produce the class pattern.

6) Access or recall using the class array. Produce the raw recalled data item.

7) Threshold the recalled data pattern.

8) Map the thresholded data pattern back onto the io-memory using a stored mapping function.

The software that allows the manipulation of the C-NNAPs machine also allows specification of the N tuple mapping function, i.e., any individual bit of a binary N tuple can be mapped anywhere in a particular io_memory space.

The handling of ADAM units and io_memories is by a single 'handle', which is a name given to these units by a programmer. The construction of a system takes place by 1) defining and creating the required number of ADAM memories and io_memories, 2) Connecting the ADAMs to the io_memories and 3) specifying the order in which the memories are evaluated and outputs and inputs mapped onto the io_memories. It is left open to the programmer how data arrives into the io_memory and is read from it, as this depends widely on the application of the system.

6 The current implementation.

From the simple architecture definition given in the previous section the physical implementation of the machine was considered. Our major constraints in achieving the implementation were;

1) The system should be made available on a network via a HP workstation.

2) The system should be extensible to allow a large number of ADAM systems to be physically implemented.

3) The speed of the system was not of prime importance, only how the system could be physically implemented to allow constraints to be examined, and projected speedup calculations to be done.

4) Software support that allows execution without the physical system being available should be provided to allow work to continue on the hardware as well as the software.

5) Off-the-shelf components and standards should be used as much as possible to reduce construction cost and time.

It was clear from the start that a number of factors made the implementation of this system interesting, including;

1) Each ADAM may use a large amount of memory.

2) The most computer intensive part of the implementation was the evaluation of the correlation memories.

Expertise in our group existed in VME bus system design and support, in digital signal processors and in PCB technology. We did not have support for VLSI facilities, but have support for field programmable gate arrays (FPGAs).

From these considerations the system was designed and is now into its third revision.

The mapping of the architecture onto the physical system is shown in Figs. 6 and 7. They show how a distributed memory is used for the io_memories, with local weight memory for each card. This arrangement allows us to place a collection of ADAM memories on each physical card, each card served by a local disk to hold a backup of the correlation matrix memories. Each ADAM sees a local io_memory, confined to the card that it 'lives' on. Any access by individual ADAMs to memories on other cards is via a VME bus access. Each card is allowed VME bus mastering, which allows it to take exclusive access of the bus for card to card io_memory access. Each card has its own disk, to cut down the load on the shared VME bus.

The implementation of each card is shown in Fig. 6. Each card has a processor (currently a DSP 32C) for controlling the evaluation of an ADAM. It is our long-term intention to implement the evaluation of an individual ADAM in dedicated hardware. However, we have decided to maintain a flexible architecture to allow us to implement a wide variety of ADAM variations on a card. The only part built out of dedicated logic on the current implementation is a function that takes an N bit word and adds each bit into a set of accumulators (called bit accumulation) followed by hardware K point thresholding. This allows us to implement the binary correlation matrix very efficiently. For the bit accumulators we have used ACT devices, which although flexible, are not particularly fast compared to VLSI implementation.

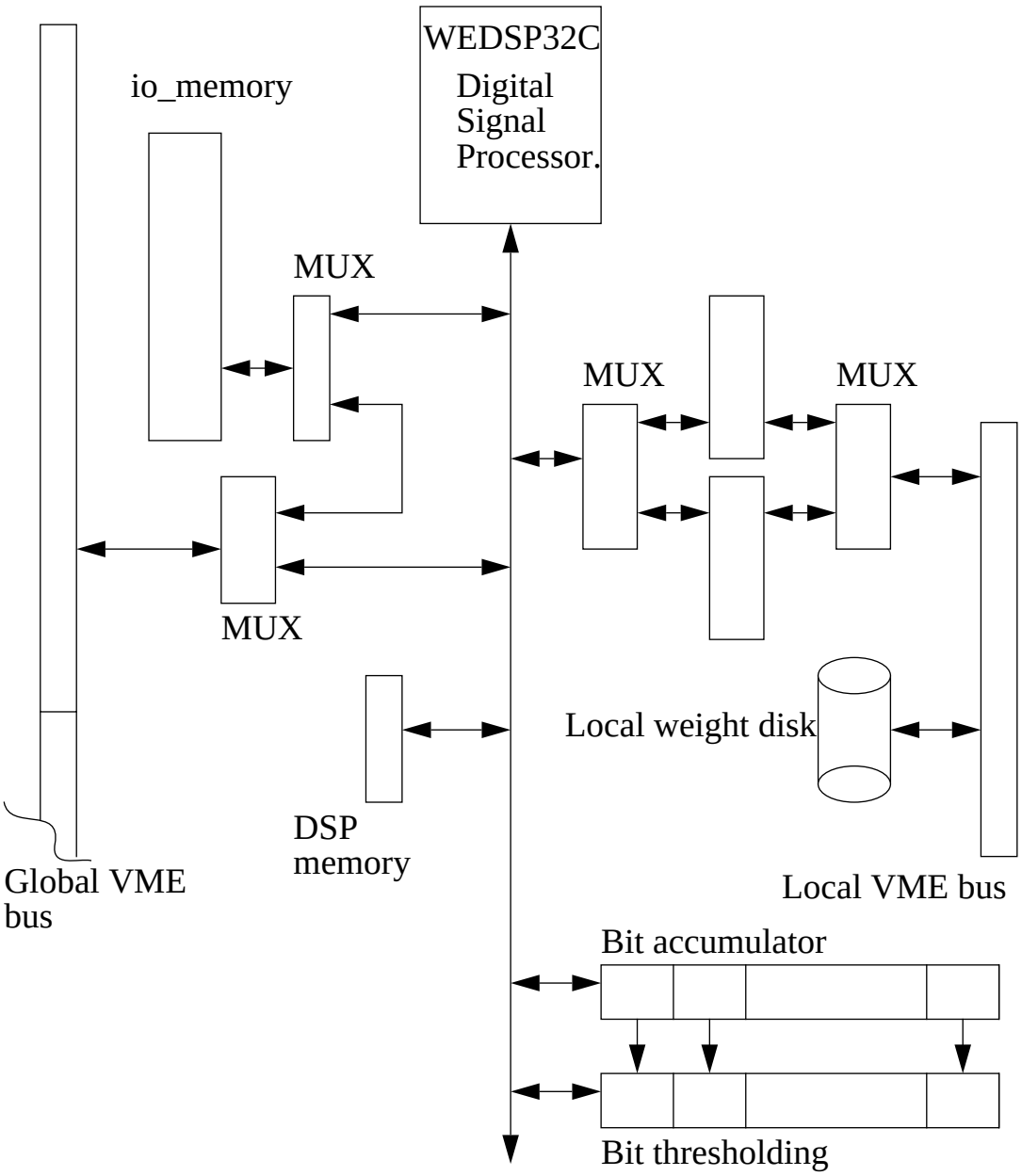

Fig 6. The architecture of a single processor card.
MUX is a 2 to1 multiplexer.

The io_memory is implemented as a dual port arrangement, one side is seen by the DSP and the other side is seen by the VME bus. The memory map of the VME maps each board into a specific slot in the full 2^{32} address memory space. The memory for the ADAM weights is triple

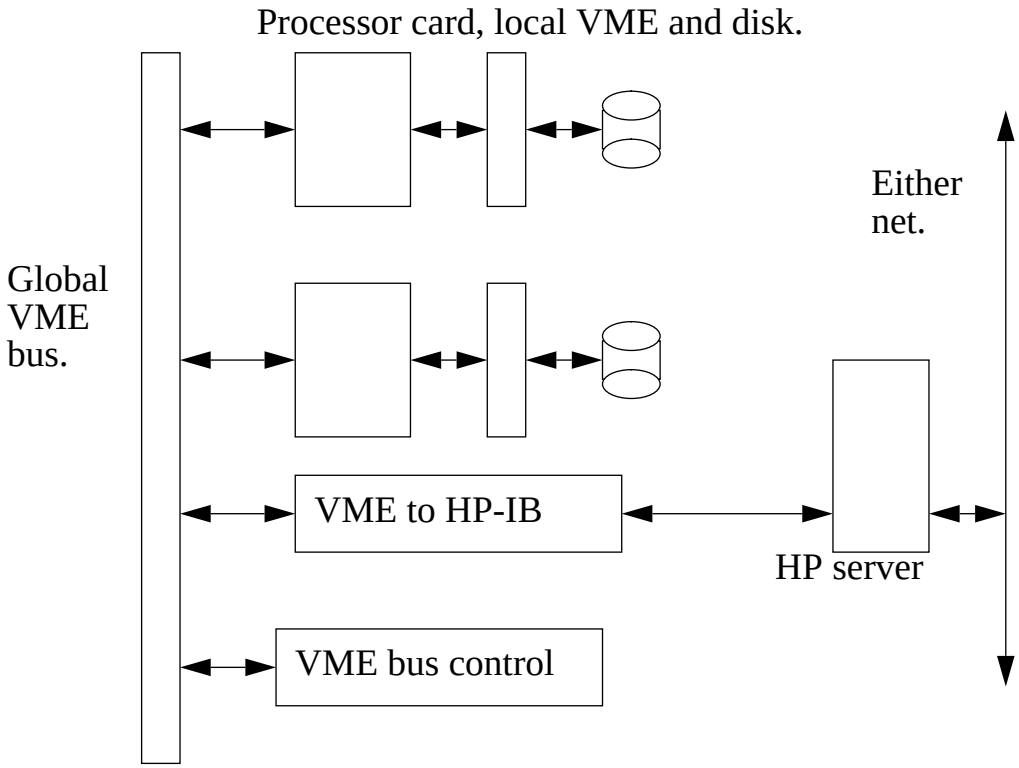

Fig 7. The high level architecture of the machine.

ported between the disk sub-system, VME bus, and the DSP. This allows 1) access by the disk while the DSP is currently evaluating an ADAM, and 2) host loading of weights for individual ADAMs.

At the moment the system is under control of a host, which is a HP 9000 series file server. The VME bus is accessible under UNIX via a parallel link. Unfortunately, the bandwidth of this link is quite low, but it allows us to evaluate the architecture. A comprehensive library of functions are implemented (see below) which allow control of the C-NNAPs machine. A command from the host is sent to the processor which executes the function. Low level as well as high level functions (teach, test etc.) functions are currently implemented .

Three levels of parallelism are potentially available on the machine, but, at present, only two of them are in use. The first is at the level of the correlation matrix; the special bit accumulator function permits large correlation matrices to be evaluated in a small number of operations and

has already been implemented. The second level is at the ADAM memory level and has not been implemented yet. It is to pipeline the evaluation of a number of ADAM functions for individual teach and train cycles (as shown in section 5). This has not been implemented yet as we have not evaluated the possible data, structural and control hazards. To do this we are using the present system to evaluate the speed of the various stages that could be defined. This is to be incorporated in the next design revision. The third level of parallelism is at the system level, which allows cards to operate in parallel on a number of ADAM memories. This level has been implemented. Currently we have two cards that can operate in parallel.

Currently we have implemented a dedicated piece of logic in an FPGA device to speed up the binary matrix multiplication. The main function required was one that took an N bin number and added each bit into a separate register. This function is used to accumulate the raw class array (see Fig 3.). This was identified as the slowest part of the ADAM function in software simulations. Unfortunately, no commercial micro-processor implements such an operation in one instruction. To add to the efficiency of the device we deigned, we also implemented a hardware thresholding process. This allowed us to take the N register values and convert them to an N bit number before reading the result from the device. This reduced the communication overhead between the device and the main processor. Our current implementation consists of a 16bit device. We have now designed a device that can accumulate up to 64K bits, and contains most of the control hardware for performing the binary matrix multiplication. This will be implemented by the end of 1994.

The construction of the machine is currently based on 9U height VME cards. The next revision will incorporate greater use of surface mount components and an ability to upgrade memory, processor and ADAM dedicated hardware independently of each other by a card consisting of a mother board (memory and buses) and two daughter boards (control processor and dedicated ADAM logic). We are aiming at a 6U height per card system.

The ADAM weights are stored on local disk and paged onto static memory when required. This paging process is the major bottleneck in the system. The ADAM hardware can evaluate data much faster than the weights can be paged from disk. Certain problems will not require a large

number of ADAMs to be evaluated within any given time, thus local weight memory will hold all the required weights. However, as soon as a new memory is required the weights are paged from disk to the static memory. The current system implements SCSI via a second VME bus. This allows us to use multiple disks per main card and use disk array methods should the bandwidth be a problem. Unfortunately the access characteristics of an individual memory tend to be random, i.e., they do not have a small and regular frame of reference, which precludes the use of cache memory methods.

The system relies heavily on the DSP for memory shuffling, N tuple state assignment etc. Although flexible, most of the features of a DSP are not used. The largest amount of DSP time is spent accessing memory and passing data to the bit accumulator, an operation that can be implemented in simple dedicated hardware. This will be incorporated in the next design. The present design has allowed us to evaluate different implementations of ADAM.

The description above has pointed out a large number of limitations of the present design, and indicated how we are in the process of extending the machine to overcome them. A number of other extensions are also possible as follows. It has been clear from the start that the VME bus is the second major bottleneck in the machine. If there is no interaction between the ADAM memories this does not prove to be a problem as in that case each ADAM will only use local memory for inputs and outputs. The VME bus is then used to download the image data to each card. The problem arrises when the input data and output data of each ADAM is shared between ADAM memories. This requires communication between the cards over the VME bus. The bandwidth of the VME bus is about 20Mb per second. It is clear to us that this bus should be used for control and a high speed dedicated bus provided for data communication between ADAMs.

7 Software and tool support.

Good software support is essential for the practical use of any machine. The programming model given in section 5 allows the user to quickly grasp the machine’s capabilities. For efficient and practical use, it has been our aim from the outset to provide a software emulation of the machine. In fact thesoftware was developed first, while the first version of the hardware

was being developed. The software consists of library of routines written in the programming language C. In order to be compatible with the host UNIX operating system, and because C supports bit manipulation. The software has been written in three layers, low-, medium-, and high-level functions. These layers allow the programmer to interact with the machine at the most suitable abstraction for their task. The lowest level contains the functions to perform the correlation matrix multiplication, N tupling etc. as shown in section 6. The second level provides functions that group these into practical operations, such as training and testing a memory, saving weights to a disk and retrieving it, clearing memory, etc. These exist as C level functions. The highest level provides a UNIX level interface to the software which allows the machine to be used on the command line. The hardware replaces the level one functions. The pipeline support will eventually replace the software at the second level, but maintain the C function interface.

Any program written using the emulation of the machine can be re-compiled with a flag to use the C-NNAPs machine. If a one uses the low level functions, each function call will execute on the hardware. The medium level functions are built out of these low level functions. To allow efficient operation when a medium level function is called, all the low level functions it executes are run on the C-NNAPs machine (separate calls to the machine are not made from the host).

The software provides basic functions for displaying image data on an X windows system and basic functions for dumping data structures in the machine.

The software library (Version 3.1) is fully described in Austin, Kelly and Turner 1993.

8 Example application in image processing.

A major application of the machine has been in image processing. Currently we are working on a system that allows features of an image to be extracted and matched with high level descriptions (maps). Two typical operations in this work are to extract urban and field areas from infra-red images and to allow roads to be extracted from images. In both, these processes and an array of ADAM memories are used. The memories use the grey scale N tuple approach described in Austin 1988, which involves changing the decoder function for a function that can

assign a state to a tuple of real valued pixels. The image and results are shown in figs 8, 9 and 10. For a description of how these were extracted see Austin, Brown, I Kelly, Buckle (1993).

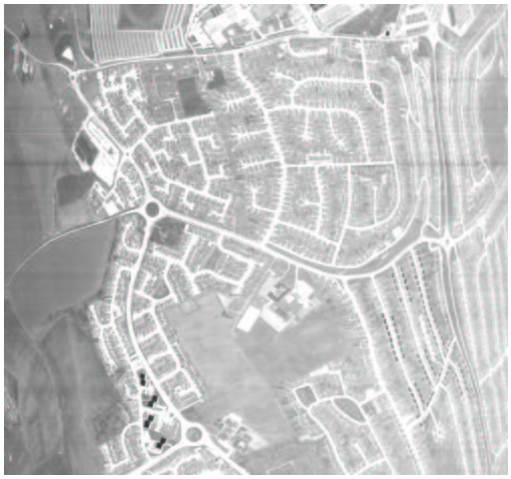

Fig 8. The raw image processed with an ADAM network

Fig 9. The urban areas found using ADAM.

see last page.

Fig 10. The roads found using ADAM.

9 Conclusion.

This paper has described the first steps in the design of a machine that uses associative mapping memories as computing elements. Although a great deal of research remains to be done, we are satisfied that the machine we have constructed is a valuable tool in our research on the application of the ADAM memory. The paper has shown the first steps foreword in producing an associative processor that uses a fast, efficient, and flexible associative memory.

10 Acknowledgments.

Parts of this work were carried out as a part of a joint Science and Engineering Research Council/Department of Trade and Industry project “Vision by Associative Reasoning”, Grant number GR/F 36330 (IED 1936). The hardware design and implementation was partly supported through funds provided by the Department of Computer Science, University of York.

11 References

J. Austin, T. J. Stonham, “An Associative Memory for use in Image Recognition and Occlusion Analysis”, Image and Vision Computing, vol. 5, no. 4, p. 251-261, Nov. 1987.

G.R. Bolt, J. Austin, “Operational Fault Tolerance of the ADAM Neural Network System”, Second International Conference on Neural Networks, p. 285-289, Bournmouth, United Kingdom, p18-20, November 1991.

J. Austin, M. Brown, I. Kelly, S. Buckle, “ADAM Neural Networks for Parallel Vision” in proceedings of the JFIT Technical Conference 1993, p. 173-180. The department of trade and industry.

D. J. Willshaw, O. P. Buneman, H. C. Longuet-Higgins, “Non-Holographic Associative Memory”, 1969, Nature, p 222, June 7, Vol. 9.

I. Kelly, J. Austin, “Implementing an associative memory environment”, In “Techniques and Application of Artificial Neural Networks”, 1993, M J Taylor, P J G Lisboa, Chapter 14, Ellis Horwood.

D. Casasent, B. Telfer, “High Capacity Pattern Recognition Associative Processors”, 1992, Neural Networks, Vol. 4, No. 5, p. 687-698.

I. Kelly, J. Austin, “Hardware support for distributed associative memories”, In the proceedings of WNNW workshop, York, UK, April 1993, Eds J Austin, N Allinson.

J. L. Potter, “Associative Computing”, 1992, Plenum.

I. Aleksander, “Advanced Digital Information Systems”, 1985, Prentice Hall International.

G. Bolt, J. Austin, G. Morgan, “Uniform tuple storage in ADAM”, 1992, Pattern Recognition letters, p. 339-344, vol. 13, North Holland.

J. Austin, “Grey Scale N tuple Processing”, In Pattern Recognition: 4th International Conference, Cambridge, UK, Berlin, 1988, Josef Kittler, Springer-Verlag, Lecture Notes in Computer Science, p110-120, vol. 301.

J. Austin, I. Kelly, A. Turner, S. Buckle, M. Brown, “The ADAM software manual Version 3.1" Oct 1993. University of York, Computer Architecture Group Internal Report.

Fig 10.