A Hardware-Accelerated Novel IR System

Michael Weeks, Victoria J.Hodge and Jim Austin
Advanced Computer Architecture Group,
Computer Science Department,
University of York,
Heslington, York, UK
(mweeks,vicky,austin)@cs.york.ac.uk

Abstract

AURA (Advanced Uncertain Reasoning Architecture) is a generic family of techniques and implementations intended for high-speed approximate search and match operations on large unstructured datasets. This paper continues the AURA II (Advanced Uncertain Reasoning Architecture) project's research into distributed binary Correlation Matrix Memory (CMM) based upon the PRESENCE (PaRallEl Structured Neural Computing Engine) hardware architecture [14]. Previous work has described how CMMs can be seamlessly implemented onto multiple hardware PRESENCE cards to accelerate core CMM operations. To demonstrate the system, this paper describes how a novel CMM-based information retrieval (IR) system, called MinerTaur, was implemented using multiple PRESENCE cards distributed across a cluster.

1 Introduction

Over the years the amount and range of stored text has expanded rapidly, overwhelming both users and tools designed to index and search the text. It is impossible to index this information dynamically at query time due to the sheer volume so an index must be pre-compiled and stored in a compact and fast data structure. Much of the text is unstructured so the data structure to index such a repository must be constructed solely from the unstructured text, storing associations between words and the documents that contain them. A search tool is also required to link to the index and enable the user to pinpoint their required information. The user's query and the unstructured text repository may contain spelling errors so a spelling mechanism is an essential component of the search tool.

In this paper we describe and empirically evaluate an information retrieval system [3], named MinerTaur implemented on a parallel hardware platform. MinerTaur comprises three modules, a spell checking pre-processor to identify errors in the user query, a synonym hierarchy to allow paraphrased documents to be matched and a word-document indexing module to identify documents matching particular query words. All modules rely on a fast efficient data structure to under-pin the system. The dedicated hardware provides a fast and efficient indexing data structure. We implement MinerTaur using binary Correlation Memory Matrices (CMMs) on PCI-based PRESENCE cards in a Beowulf PC cluster named Cortex-1. [1] This paper provides comparisons between a software implementation of the MinerTaur CMM data structures and the parallel hardware implementation to investigate the comparative retrieval speeds.

The paper first introduces the AURA architecture, including the AURA library, the PRESENCE PCI-card and the new hardware-based additions to the AURA library. This is followed by a description of the techniques used to map a CMM over a distributed system. The MinerTaur system follows and the seamless method by which MinerTaur is implemented onto the hardware. We detail the empirical comparisons between the software and multiple-PRESENCE card CMM implementations of MinerTaur. The paper closes with conclusions from the work, and future work.

2 AURA

AURA (Advanced Uncertain Reasoning Architecture) is a generic family of techniques and implementations intended for high-speed approximate search and match operations on large unstructured datasets [10]. AURA technology is fast, economical, and offers unique advantages for finding near-matches not available with other methods. AURA is based on a high-performance binary neural net-

[1] Cortex-1 consists of seven 500MHz PC nodes connected by 100Mbit Ethernet, six nodes of which contain 28 PCI-PRESENCE cards.

work called Correlation Matrix Memory (CMM). Typically, several CMM elements are used in combination to solve soft or fuzzy pattern-matching problems.

AURA takes large volumes of data and constructs a special type of compressed index. AURA finds exact and near-matches between indexed records and a given query, where the query itself may have omissions and errors. The degree of nearness required during matching can be varied through thresholding techniques.

In addition to the MinerTaur IR system, there are an increasing range of applications for AURA. These include a postal address matcher, high-speed rule-matching systems [11], high-speed classifiers (e.g. novel k-NN implementations) [15], structure-matching (e.g. 3D molecular structures) [13], and trademark-database searching [9].

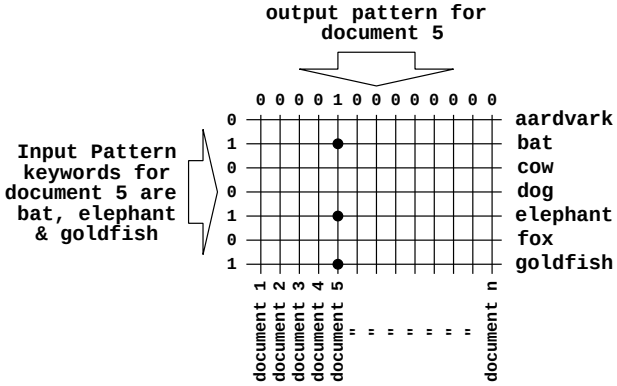

Figure 1. An example of an inverted index teach

2.1 Correlation Matrix Memories

CMMs are neural networks for the storage and retrieval of vector patterns. Each column of the matrix is seen as a neuron, and each row represents an input and synapses to each neuron. The PRESENCE hardware architecture consists basically of a binary correlation matrix neural network implemented in memory. In common with the CMM concept, the PRESENCE card has two main modes of operation; teach and recall. In teach mode, the input and output binary pattern vectors are supplied to the card over the PCI bus. Recall is achieved by issuing only the input binary vector, and on completion, the resulting summed column data of the CMM can be read unprocessed from the card for post-processing, or hardware thresholding can be applied to the data to sort the best matches. Two types of hardware thresholding can be applied (Willshaw or Lmax) dependent upon the application. Willshaw [16] thresholding compares the summed columns with a thresholded level, whilst Lmax [17] retrieves the top L matches from all of the summed columns. A detailed operation of CMM neural networks can be found at [18].

CMM techniques lend themselves for use in such applications as inverted indexes, whereby objects are stored in the CMM categorised by certain attributes. A keyword search of documents, for instance, where the output pattern associates one or more bits with a list of documents, and the input pattern describes keywords in the documents. Figure 1 illustrates how documents and keywords are taught into a CMM. A recall operation applies selected keywords as the input pattern, and the column counts of the output pattern contain matching documents.

2.2 AURA Library

Building high performance pattern matching architectures is not sufficient to exploit the architecture fully and easily, a powerful software library is also required. The object-oriented AURA library, written in C++, allows for the creation and manipulation of CMMs, in addition to pre- and post-processing algorithms. The library has been written for several machine-OS-compiler combinations, though we are initially concerned with Linux and the GNU gcc compiler.

AURA CMM objects can be instantiated so that they are mapped onto the PRESENCE hardware, or are simulated in software. CMM classes were initially designed as software simulations to investigate the CMM concept, to experiment with new features, and to enable application development whilst the hardware progressed.

This project has added two new CMM classes to the AURA library in the form of the NodeCMM and DistributedCMM implementations (see subsections 2.4 and 2.5). The NodeCMM object stripes a CMM across multiple PRESENCE cards in a node. The DistributedCMM object stripes a CMM across multiple nodes of a cluster via a client-server mechanism. The DistributedCMM can make use of either hardware-based NodeCMMs or software CMMs.

2.3 PRESENCE Architecture

This section gives a brief overview of the operation of the PRESENCE architecture. PRESENCE is the current family of hardware designs that accelerate the core CMM computations needed in AURA applications. A more detailed discussion of the PRESENCE architecture can be found in [12]. The PRESENCE card utilises the PCI-bus interface, to allow it to be used on standard desktop computer systems, and to give the CPU fast access to the card. The card is a PCI-slave device, and contains the pipelined **Summing and Thresholding** (SAT) processor that accelerates CMM

operation. The card has 128 MByte of low-cost DRAM for CMM weights memory, configured as 8Mwords of 128-bit width.

The recall performance of a Willshaw-thresholded single PRESENCE card is given in equation 1, where τ is the clock period of the binary neural SAT processor, S is the CMM output (separator) vector width in bits and I is the number of index or keyword terms that the recall is trying to match against.

$$T_{Willshaw} = \left\lceil \frac{S}{128} \right\rceil \cdot \tau \cdot (52 + 2 \cdot I) + 23 \cdot \tau \qquad (1)$$

The PRESENCE cards clock period, τ, is currently fixed at $\frac{1}{16MHz}$, therefore the performance of a Willshaw-thresholded recall operation is proportional to the variable terms I and S. Using a single PRESENCE board, the only option for reducing these variables is vector compression. However, vector compression increases the number of false matches leading to a need for computationally-slow post-process checking.

With multiple PRESENCE boards, I and S can be reduced by striping their respective vectors across the cards, and has the added benefit of scalable weights memory. However, whilst it is possible to stripe the index vector over multiple cards, a recall operation will require the retrieval of raw column data from the cards before performing column summation and thresholding in software. This will be restrictively slow as the parallelism and functionality of the PRESENCE card is bypassed.

Output vector striping, however, is feasible and fully utilises the characteristics of the PRESENCE architecture (see figure 2). Willshaw-thresholding scales naturally onto multiple cards, though not with Lmax-thresholding. Lmax-thresholding requires software thresholding on raw column data, or alternatively it can be achieved using repeat Willshaw-thresholding with a decrementing level. Fortunately, the use of Lmax-thresholding in applications is uncommon.

The following sections describes how a scalable multi--PRESENCE card CMM can be implemented in a distributed memory cluster such as Cortex-1. We identify two levels of PRESENCE scalability, board-level striping within a node, and node-level striping within the cluster.

2.4 PRESENCE Board-level CMM Striping

On a single node of the Cortex-1 cluster we can install and operate up to five PRESENCE cards in parallel over the PCI-bus. This gives a combined CMM memory space of 640MByte. Whilst this technique greatly increases the size of the CMM that can be implemented, it presents challenges when allocating, teaching and recalling.

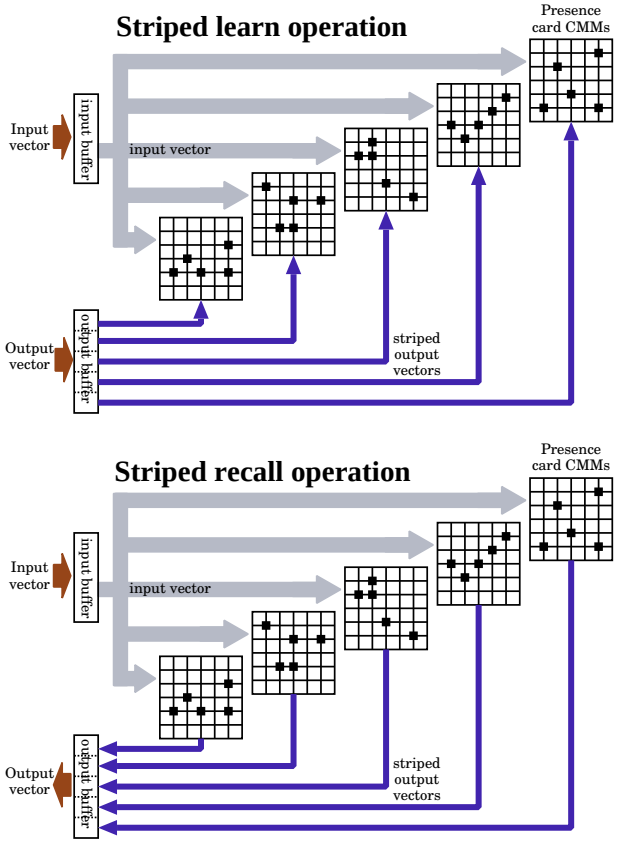

Figure 2. Data-flow for separator striped learn and recall operations

The NodeCMM class allows the transparent creation and manipulation of CMM objects within a single PC node. The CMM will be allocated on one or more PRESENCE cards, such that the user does not know, or need to know the location(s) of the data across the cards. It is almost identical in use to the software CMMs provided in the AURA library, though some differences exist which are explained below.

Striping a CMM across multiple cards decreases the recall time considerably. Therefore, for CMMs with a separator size greater than 128-bits, multiple PRESENCE cards are used.

In theory, the five PRESENCE cards operate in parallel, so there should be only a small penalty over the single card operation. In practice however, the PRESENCE driver, which is a sequential process, must initiate the five boards, check on their progress, then retrieve the data upon completion. Tests show that whilst Willshaw-thresholded recall times are substantially lower than the five cards operating individually, the overhead on the multiple recall is higher than expected. Analysis proved that this overhead is in the

section of driver code that retrieves the data from the card, via the PCI bus.

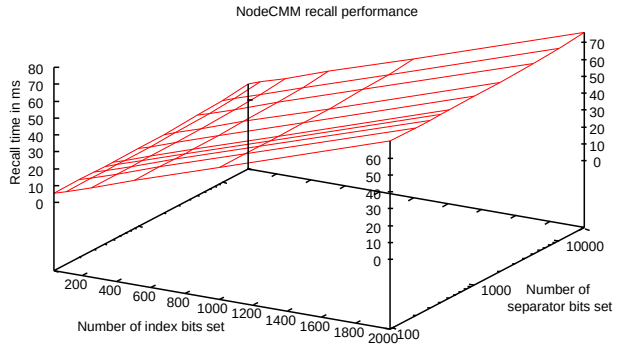

Figure 3. 32,768 x 32,768 NodeCMM Willshaw-thresholded recall performance

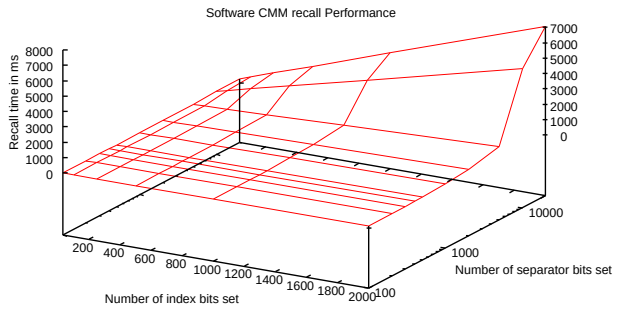

Figure 4. 32,768 x 32,768 EfficientCMM Willshaw-thresholded recall performance

An application was written to compare the performance of the NodeCMM and EfficientCMM classes. The EfficientCMM is a type of software simulated CMM that stores bits in the CMM matrix in a compacted format. Each CMM type was initialised with both input and output vector widths of 32768 bits. Recall operations were then performed with varying input and output vector saturation levels that were randomly set. Figures 3 and 4 compare the Willshaw-thresholded recall performance of the two CMMs.

2.5 Node-level Parallelism

The limitations of the simulated CMM are that its storage capacity scales with system memory, and large highly-saturated CMMs can be slow. Using PRESENCE cards, storage capacity is limited to 128MByte or 640MByte with a maximum of 5 cards per machine. To scale the CMM's storage capacity further, it must be distributed across multiple machines. The CMM is distributed across multiple nodes in the Cortex-1 cluster via separator striping, similar to that described in section 2.4. A client-server framework was created that allowed the creation of remote CMMs across a network. The remote CMMs can be either software or the PRESENCE-based NodeCMM. For communication tasks we used the ACE library (Adaptive Communication Environment) [2], which provides a library of common communication software tasks across a range of operating system platforms. A wrapper class, called *DistributedCMM* encapsulates the underlying cluster infrastructure and PRESENCE cards so they are hidden from the application programmer. In this way, converting the existing applications to distributed PRESENCE cards would simply be a matter of re-declaring the CMM objects and re-compiling.

3 MinerTaur IR System

Many IR systems employ inefficient data structures. Glimpse [8] uses a two-level index that is only suitable for small system storage. Bayesian networks and node-based neural networks suffer the explosion of nodes as the number of nodes increases exponentially with the number of documents stored. Other systems minimise storage by incorporating various data compression techniques. Latent Semantic Indexing [6] reduces the size of the word-document matrix. However, low-level information may be factored out and the decomposition process is exceedingly time consuming and computationally expensive. In the system evaluated in this paper, we use orthogonal vectors to represent words and documents. Orthogonal vectors ensure accurate retrieval and thus maximise system speed and throughput. Orthogonal vectors produce no false matches unlike other vector compression process such as vectors with multiple bits set [7].

Our overall IR system [3] comprises three modules: a spell checker, a hierarchical thesaurus and finally a word-document association index. We briefly describe each module below with citations to more detailed descriptions.

The first module is a **front-end spell-checking system** [4] to isolate any mis-spelt query words by validating each query word against the lexicon. The module uses 2 CMMs: one for Hamming Distance and n-gram spelling and one for phonetic matching. If a word is not validated we assume a spelling error and our spell-checking module provides a list of the best candidate matches for the user to select from. We integrate the outputs of the 2 CMMs and score the candidate matches. All query words are correctly spelt in our evaluation, so only a validation step is necessary.

The second module is a **hierarchical thesaurus** we generate automatically from the corpus. We employ a statisti-

[2] available from http://www.cs.wustl.edu/∼schmidt/ACE.html

cal gathering and inference methodology to automatically evolve a hierarchical thesaurus from word co-occurrence statistics in the text corpus. Words are grouped in to clusters by their contextual similarities using the average context vectors and our TreeGCS [5] growing hierarchical clustering algorithm, built upon Fritzke's Growing Cell Structures [1]. We can then exploit the distances within the synonym hierarchy to ascribe scores to query words and their synonyms. The scores are added to the matching documents output from the word-document matrix.

The third module is a **binary word-document CMM** for fast training and rapid partial match retrieval [2]. Each row of the matrix effectively indexes a particular word and each column effectively indexes a specific document and a bit is set at position ij if $word_i$ occurs in $document_j$. To retrieve all documents matching a query word, we activate the matrix rows indexed by the words and retrieve all columns where a bit is set. We multiply this output vector by the word score to score the matching documents.

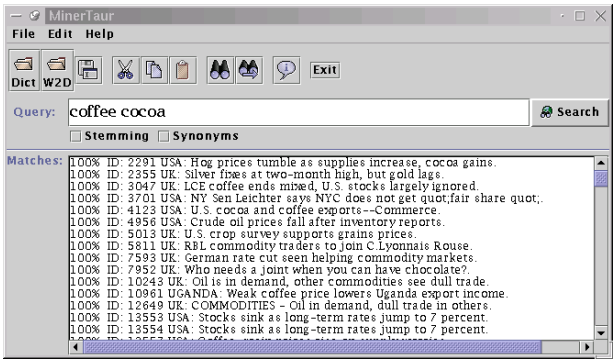

Figure 5. Snapshot of MinerTaur's Java user-interface

MinerTaur uses a Java front-end (figure 5) to interface with a back-end C++ library that implements the main components of the IR system. The Java code accepts keyword inputs and passes them to the C++ library. We pass each query word through each of the three modules in turn, retrieving a set of matching documents from the word-document matrix. We produce a separate vector for each query word with an attribute for each document representing the document's score with respect to the specific query word. We can then rank the documents by summing all query word vectors to generate a cumulative document score vector. Assuming that the keywords chosen are spelt correctly and are present in the document list, the library code returns the list of documents that contain the keywords and their degree of confidence.

The initial document corpus was 15.5 MBytes in size and contained 18249 Reuters document abstracts [19]. 47,985 keywords were extracted from the file to search the documents. The three MinerTaur CMMs: the spelling, phonetic, and word-to-document CMMs are initially taught with input-output associations from data files. The size of the individual CMMs are given in table 1.

CMM	*Rows*	*Columns*	*Memory size*
Spelling	2000	48767	1.393 MByte
Phonetic	68	48767	389 kByte
Word-Doc	48767	18250	7.989 MByte

Table 1. MinerTaur CMM configuration

To test the performance of PRESENCE in real life applications it was decided to implement the MinerTaur application's Word-Doc CMM using the NodeCMM and the DistributedCMM objects, using various numbers of PRESENCE cards. A batch query was added to the MinerTaur back-end code in order to gather system performance data. The batch query consists of 1200 sets of randomly generated query word combinations. The combinations are organised as 100 examples each of 1 word to 12 word queries. The batch query was performed with both **word** and **word and synonym** searches.

Converting the code from the software CMM class to the two PRESENCE-based CMM class should be trivial. Theoretically, a simple re-declaration of the Word-Doc CMM object should suffice. In practice however, when recalling data, the software must be optimised to maximise the performance gains obtainable from PRESENCE. For instance, the application initially made use of raw column-count recalls that were then thresholded using software algorithms. In hardware, Willshaw-thresholded recalls are an order of magnitude faster than a raw column-count recall, and requires no threshold post-processing by the system CPU. Therefore, for raw CMM recalls that are then Willshaw-thresholded in software, the two operations can be replaced by a single hardware Willshaw-thresholded recall. This required some sections of the code to be rewritten and certain functions to be in-lined due to complexities in the flow of data.

Future improvements to the hardware performance can only be beneficial if current application performance is heavily degraded by CMM recall operations. Data from the batch query was analysed for the amount of time spent performing CMM recalls during a MinerTaur query was recorded in table 2. Data is given for **word** and **word and synonym** searches implemented using both locally implemented hardware and software CMMs. From this data it can be seen that when MinerTaur is implemented using a 5-card NodeCMM, 67-76% of program execution time is spent in recall operations. This provides justification for improvements to the hardware which will improve query response

	Software implemented CMM					
	Word			Word + Synonyms		
	%Spell CMM	%w2d CMM	%Total CMM	%Spell CMM	%w2d CMM	%Total CMM
Mean	78.0	1.0	79.0	11.6	74.9	86.5
Std Dev	0.5	0.1	0.5	3.7	5.0	2.5
Min	74.4	0.8	75.3	5.8	44.9	75.8
Max	78.8	1.6	79.8	32.6	87.8	93.6

	5 PRESENCE NodeCMM					
	Word			Word + Synonyms		
	%Spell CMM	%w2d CMM	%Total CMM	%Spell CMM	%w2d CMM	%Total CMM
Mean	74.6	1.1	75.7	24.1	42.8	66.9
Std Dev	0.9	0.1	0.9	6.3	5.1	2.0
Min	69.8	0.8	70.9	9.8	16.5	57.1
Max	76.5	1.6	77.5	52.1	55.9	76.8

Table 2. Proportion of CMM recall time during MinerTaur document search

	Word		Word + Synonyms	
	Total query time (ms)	W2D CMM time (ms)	Total query time (ms)	W2D CMM time (ms)
SW CMM	89.62	0.89	653.77	495.39
1 card CMM	238.66	1.14	547.02	242.32
2 card CMM	139.35	0.91	364.97	159.24
3 card CMM	106.61	0.89	307.83	134.42
4 card CMM	90.94	0.88	280.12	122.95
5 card CMM	78.21	0.86	263.64	115.99

Table 3. The mean performance per query word for a MinerTaur document search using local CMMs

times substantially.

4 Results

Training time takes longer in hardware than in software. The word-to-document CMM training time is 14.09s in software, or 25.71s for a 5 PRESENCE card NodeCMM. A DistributedCMM using 28 PRESENCE cards over 6 slave nodes trains in approximately 10 minutes.

Using the previously described batch query, tables 3 and 4 show the mean performance of CMMs that are locally-implemented and distributed across various quantities of PRESENCE cards. The mean time per query word for the basic word search is 89.2 ms in software, 78.2 ms using the 5-card NodeCMM, and 112.2 ms for a 6-node 28-card DistributedCMM. The mean recall time for the word-to--document CMM was 0.89 ms in software, 0.86 ms using the 5-card NodeCMM, and 22.85 ms for a 6-node 28-card DistributedCMM.

		Word		Word + Synonyms	
# cards	# Nodes	Total query time (ms)	W2D CMM time (ms)	Total query time (ms)	W2D CMM time (ms)
10	2	118.23	27.07	2902.86	2728.79
15	3	117.95	25.82	2773.97	2620.19
20	4	116.14	24.11	2728.10	2548.78
25	5	112.27	23.92	2698.6	2517.22
28	6	112.16	22.85	2662.46	2485.54

Table 4. The mean performance per query word for a MinerTaur document search using distributed hardware CMMs

5 Conclusion

This paper gives details of how multiple PRESENCE cards have been implemented in order to provide scalability in terms of CMM recall performance and data storage. It has shown that the PRESENCE hardware and the new scalable CMM classes can be successfully taken out of the lab and used in real-life applications on the MinerTaur Information Retrieval system. Whilst performance gains using locally implemented PRESENCE cards may appear small for the additional cost of the hardware, it must be remembered that PRESENCE currently operates at 16MHz, whereas the software CMM was executing on a 500MHz Pentium III CPU. The next generation PRESENCE card will operate at a far higher operating frequency and with greater parallelism.

The recall performance of the DistributedCMM is approximately 25 times slower than the CMMs implemented locally. This is largely due to the communications overhead when passing vectors between the nodes. Therefore, the DistributedCMM should be avoided if a dataset is small enough to be implemented locally. Larger data corpora, however, may be unable to be implemented locally, or may be restrictively slow, and so the DistributedCMM should be used. For example a larger Reuters database [20] was used that associated 477,952 documents with 62,903 words, in order to fill the whole weights memory available on 28-cards over six slave nodes. This dataset is a 571 MByte file and contains far too much data to be implemented locally. The mean time per query word for the basic word search was 193.7 ms, of which the mean time spent recalling from the word-document CMM was 31.36 ms.

The synonym-based search is restrictively slow using the DistributedCMM, as the program performs multiple recall operations as the synonym tree structure is traversed. To avoid this problem, the synonym tree has been replaced using the k-nearest neighbour method.

The project also identified improved thresholding functionality in order to enhance the spelling CMM's recall performance. These improvements can be trivially implemented in hardware, and will be incorporated onto future

PRESENCE designs. We also identified previously unseen performance bottlenecks in the AURA library. For instance, the XOR operator for the AURA library's BinaryBitVector (BBV) class, was found to be restrictively time consuming when dealing with large heavily saturated BBVs. This problem was analysed and a more efficient algorithm implemented. Finally, analysis of the original MinerTaur code showed that the use of CMM operations were inefficient when applied to PRESENCE. Therefore, application programmers must be aware of the hardware, and write CMM applications with PRESENCE characteristics in mind.

References

[1] B.Fritzke, Growing Cell Structures - a Self-organizing Network for Unsupervised and Supervised Learning. TR-93-026, ICSI, Berkeley, CA, 1993.

[2] V.Hodge & J.Austin, An Evaluation of Standard Retrieval Algorithms and a Binary Neural Approach. Neural Networks 14(3): Elsevier Science, 2001.

[3] V.Hodge & J.Austin, An Integrated Neural IR System. In, Procs of the 9th European Symposium on Artificial Neural Networks, April 2001.

[4] V.Hodge and J.Austin, A Novel Binary Spell Checker. To appear in, IEEE Transactions on Knowledge and Data Engineering.

[5] V.Hodge & J.Austin, Hierarchical word clustering - automatic thesaurus generation. To appear in, Neuro-Computing: Elsevier Science.

[6] S.Deerwester, S.T.Dumais, T.K.Landauer, G.W.Furnas, and R.A.Harshman. Indexing by Latent Semantic Analysis. Journal of the Society for Information Science, 1(6):391–407, 1990.

[7] J.Kennedy, The Design of a Scalable and Applications Independent Platform for Binary Neural Networks, PhD thesis, Department of Computer Science, University of York, 1997.

[8] U.Manber and S.Wu, GLIMPSE: A Tool to Search Through Entire File Systems. In, 1994 Winter USENIX Technical Conference, 1994.

[9] S.Alwis and J.Austin. A novel architecture for trademark image retrieval systems. In *Electronic Workshops in Computing*. Springer, 1998.

[10] J.Austin, J.Kennedy, and K.Lees. The advanced uncertain reasoning architecture. In *Weightless Neural Network Workshop*, 1995.

[11] J.Austin, J.Kennedy, and K.Lees. A neural architecture for fast rule matching. In *Artificial Neural Networks and Expert Systems Conference (ANNES '95)*, Dunedin, New Zealand, December 1995.

[12] A.Moulds, R.Pack, Z.Ulanowski, and J.Austin. A high performance binary neural processor for PCI and VME bus-based systems. In *Weightless Neural Networks Workshop*, 1999.

[13] A.Turner and J.Austin. Performance evaluation of a fast chemical structure matching method using distributed neural relaxation. In *Fourth International Conference on Knowledge-Based Intelligent Engineering Systems*, August 2000.

[14] M.Weeks, J.Austin, A.Moulds, A.Turner, Z.Ulanowski, and J.Young. Mapping correlation matrix memories onto a beowulf cluster. International Conference on Artificial Neural Networks (ICANN2001), Vienna, Austria, 21 –25 August 2001.

[15] P.Zhou and J.Austin. A PCI bus based correlation matrix memory and its application to k-nn classification. In *MicroNeuro'99*, Granada, Spain, April 1999.

[16] D.J.Willshaw, O.P.Buneman, H.C.Longuet-Higgins. Non-holographic associative memory. Nature 222(1969), 960-962.

[17] D.P.Casasent and B.A.Telfer. High capacity pattern recognition associative processors. Neural Networks 5(4), 251-261, 1992.

[18] Jim Austin. Distributive associative memories for high speed symbolic reasoning. Int. J. Fuzzy Sets Systems, 82, 223-233, 1996.

[19] Reuters-21578. The Reuters-21578, Distribution 1.0 test collection is available from David D. Lewis' professional home page, currently: http://www.research.att.com/∼lewis.

[20] Reuters Corpus. Volume 1: English language, 1996-08-20 to 1997-08-19, at http://www.reuters.com/researchandstandards/corpus

The research detailed in this paper was funded by EPSRC grant numbers GR/L74651 and GR/K41090.