# A Neural Network Architecture for High-Speed Database Query Processing


Chun-Hsien Chen & Vasant Honavar *
Department of Computer Science
226 Atanasoff Hall,
Iowa State University,
Ames, IA 50011. U.S.A.
chen@cs.iastate.edu, honavar@cs.iastate.edu



**Abstract**

Artificial neural networks (ANN), due to their inherent parallelism and potential fault tolerance, offer an attractive paradigm for robust and efficient implementations of large modern database and knowledge base systems. This paper explores a neural network model for efficient implementation of a database query system. The application of the proposed model to a high-speed library query system for retrieval of multiple items is based on partial match of the specified query criteria with the stored records. The performance of the ANN realization of the database query module is analyzed and compared with other techniques commonly in current computer systems. The results of this analysis suggest that the proposed ANN design offers an attractive approach for the realization of query modules in large database and knowledge base systems, especially for retrieval based on partial matches.



*This research was partially supported by the National Science Foundation through the grant IRI-9409580 to Vasant Honavar.

# 1 Introduction

Artificial neural networks (ANN) offer an attractive computational model for a variety of applications in pattern classification, language processing, complex systems modelling, control, optimization, prediction and automated reasoning for a variety of reasons including: potential for massively parallel, high-speed processing, resilience in the presence of faults (failure of components) and noise. Despite a large number of successful applications of ANN in pattern classification, signal processing, and control, their use in complex symbolic computing tasks (including storage and retrieval of records in large databases, and inference in deductive knowledge bases) is only beginning to be explored (Honavar & Uhr, 1994; Sun & Bookman, 1994; Levine & Aparicioiv, 1994; Goonatilake & Khebbal, 1995).

This paper explores the application of neural networks to database query processing. Database query can be viewed as a process of table lookup which is used in a wide variety of computing applications. Examples of such lookup tables include: *routing tables* used in routing of messages in communication networks, *symbol tables* used in compiling computer programs written in high level languages, and *lexicons* or dictionaries used in natural language processing. Many applications require the lookup mechanism or query processing system to be capable of retrieving items based on *partial matches* or retrieval of *multiple* records matching the specified query criteria. This capability is computationally rather expensive in many current systems. The ANN-based approach to database query processing that is proposed in this paper exploits the fact that the database lookup task can be viewed at an abstract level, in terms of *binary mappings* which can be efficiently and robustly realized using ANNs (Chen & Honavar, 1994; Chen & Honavar, 1995a).

The rest of the paper is organized as follows:

- Section 2 reviews the mathematical model of multiple recall from partially specified input pattern in a neural network proposed in (Chen & Honavar, 1995a).

- Section 3 develops an ANN design for a high-speed library query system in detail based on the model described in section 2.

- Section 4 compares the performance of the proposed ANN-based query processing system with that of several currently used techniques.

- Section 5 concludes with a summary of the paper.

# 2 Multiple Recall from Partially Specified Input

Most of database management systems store data in the form of structured records. When a *query* is made, the database system searches and retrieves records that match the user's query criteria which typically only partially specify the contents of records to be retrieved. Also, there are usually multiple records that match a query (e.g., books written by a particular author in a library query system). Thus, query processing in a database can be viewed as an instance of the task of recall of multiple stored patterns given a partial specification of the patterns to be recalled. A neural associative memory for performing this task was proposed in (Chen & Honavar, 1995a). This section summarizes the relevant properties of the neural memory developed in (Chen & Honavar, 1995a).

## 2.1 Associative Memory with Bipolar Input and Binary Output

The associative memory used is based on 2-layer perceptron (with a layer of input neurons, a layer of hidden neurons, and a layer of output neurons; and hence two layers of *connection weights*). Given a set $U$ of $k$ bipolar input vectors $u_1$,...,$u_k$ of dimension $n$ and a set $V$ of $k$ desired binary output vectors $v_1$,...,$v_k$ of dimension $m$, such an associative memory module can be synthesized using one-shot learning as follows: The memory module has $n$ input, $k$ hidden and $m$ output neurons. For each associative ordered pair $(u_i, v_i)$, where $1 \leq i \leq k$, a hidden neuron $i$ is created with threshold $n - 2p_i$, where $p_i \in \mathbf{N}$ is the adjustable precision level for that associative pair. The connection weight from input neuron $g$ to hidden neuron $i$ is $u_{ig}$ and that from hidden neuron $i$ to output neuron $h$ is $v_{ih}$, where $u_{ig}$ is the $g$-th component of bipolar vector $u_i$ and $v_{ih}$ is the $h$-th component of binary vector $v_i$. The threshold for each of the output neurons is set to 1. The activation function at hidden node is binary hardlimiter function $f_h$, and that at output neuron can be binary hardlimiter function $f_h$ or identity function $I$, i.e., $I(x) = x$, depending on the particular hardware implementation employed. The binary hardlimiter function $f_h$ is defined by $f_h(x) = 1$ if $x \geq 0$ and 0 otherwise.

## 2.2 Associative Recall from Partially Specified Input Pattern

This section summarizes a mathematical model of associative recall from a partially specified bipolar pattern and its neural network realization. The model assumes that the unavailable components of an input pattern vector have a default value of 0. Thus, a partial input pattern is *completed* by filling in a 0 for each of the unavailable components.

Let $\dot{u}$ be a partially specified $n$-dimensional bipolar pattern with the values of some of its components being unknown; $bits(\dot{u})$, a function which counts the number of components with known values (+1 or -1) of partial pattern $\dot{u}$; $pad0(\dot{u})$, a function which pads the unavailable bits of bipolar partial pattern $\dot{u}$ with 0's; and $\mu \odot \nu$, a binary predicate which tests whether "*$\mu$ is a partial pattern of $\nu$*", where "*$\mu$ is a partial pattern of $\nu$*" means that the values of available bits of $\mu$ are same as those of their corresponding bits in $\nu$.

Let $\dot{D}_H(\dot{u}, \dot{v})$ denotes the Hamming distance between two bipolar partial patterns, with same corresponding unavailable components, $\dot{u}$ and $\dot{v}$ respectively. If $bits(\dot{u}) = j$, $pad0(\dot{u})$ is called as *padded j-bit partial pattern* derived from partially specified pattern $\dot{u}$. Define $\dot{U}^j_{Pi} = \{\dot{u} \mid bits(\dot{u}) = j \;\&\; \dot{u} \odot u_i\}$, $1 \leq j \leq n \;\&\; 1 \leq i \leq k$, i.e., $\dot{U}^j_{Pi}$ is the set of partial patterns, with $j$ available bits, of bipolar pattern $u_i$. Define $\ddot{U}^j_{Pi}(p_i) = \{pad0(\ddot{u}) \mid \exists \dot{u},\ \dot{u} \in U^j_{Pi} \;\&\; \dot{D}_H(\ddot{u}, \dot{u}) \leq \lfloor j/n \rfloor \times p_i\}$, $1 \leq j \leq n \;\&\; 1 \leq i \leq k$, i.e., $\ddot{U}^j_{Pi}(p_i)$ is the set of padded $j$-bit partial patterns having Hamming distance less than or equal to $\lfloor j/n \rfloor \times p_i$ to any one of the padded $j$-bit partial patterns of full pattern $u_i$.

Let $p_i = p, 1 \leq i \leq k$; $\ddot{U}^{c\sim n}_{Pi}(p) = \cup^n_{j=c} \ddot{U}^j_{Pi}(p)$; and $\ddot{U}^{c\sim n}_{P}(p) = \cup^k_{i=1} \ddot{U}^{c\sim n}_{Pi}(p)$. Let $f_P$ denote the function of recall from padded bipolar partial pattern. Then $f_P$ is defined as follows:

$$f_P : \ddot{U}^{c\sim n}_{P}(p) \rightarrow V$$

$$f_P(x) = v_i; \text{ if } x \in \ddot{U}^{c\sim n}_{Pi}(p),\ 1 \leq i \leq k$$

$f_P$ is a partial function and is extended to a full function $\hat{f}_P$ for recall from padded bipolar partial pattern using associative memory as follows:

$$\hat{f}_P : \ddot{\mathbf{B}}^{c\sim n} \rightarrow (V \cup \{<0^m>\})$$

$$\hat{f}_P(x) = \begin{cases} f_P(x) & \text{if } x \in \ddot{U}_P^{c\sim n}(p) \\ <0^m> & \text{if } x \in (\ddot{\mathbf{B}}^{c\sim n} - \ddot{U}_P^{c\sim n}(p)) \end{cases}$$

where $\ddot{\mathbf{B}}^{c\sim n}$ is the *universe* of $n$-dimensional vectors each of whose components is 1, 0, or -1 and which have at least $c$ non-zero components. It is assumed that the $m$-dimensional null pattern $<0^m>$ is excluded from $V$.

The neural associative memory designed for recall from a fully specified input pattern can be used for associative recall from a partially specified input pattern by only adjusting the thresholds of the hidden neurons as follows: Multiply the threshold of each hidden neuron by the ratio of the number of available components of a partially specified input pattern to that of a complete pattern. That is, reduce the threshold of each hidden neuron $i$ from $n - 2p$ to $(n - 2p) \times n_a/n = n_a - 2(p \times n_a/n)$, where $n_a \leq n$ is the number of available bits of a partially specified input pattern.

Note that $p$ is the precision level set for every memory pattern for recall from a partial pattern; $n_a$ equals the number of available bits of a partial input pattern and $p \times n_a/n$ is the new precision level.

## 2.3 Multiple Associative Recalls

The memory retrieval process in the neural associative memory described in previous sub-sections can be viewed as a two-stage process: *identification* and *recall*. During identification of an input pattern, the 1st-layer connections perform similarity measurements and sufficiently activate zero or more hidden neurons so that they produce high outputs of value 1. The actual choice of hidden neurons to be turned on is a function of the 1st-layer weights, the input pattern, and the threshold settings of the hidden neurons. During recall, if only one hidden neuron is turned on, one of the stored memory patterns will be *recalled* by that hidden neuron along with its associated 2nd-layer connections. Without any additional control, if multiple hidden neurons are enabled, the corresponding output pattern will be a superposition of the output patterns associated with each of the enabled individual hidden neurons. With the addition of appropriate control circuitry, this behavior can be modified to yield sequential recall of more than one stored pattern.

Multiple recalls are possible if some of the *associative partitions* realized in the associative ANN memory are not *isolated* (see (Chen & Honavar, 1995a) for details). An input pattern (a vertex of an $n$-dimensional bipolar hypercube) located in a region of overlap between several associative partitions is close enough to the corresponding partition centers (stored memory patterns) and hence can turn on more than one hidden neuron as explained below.

Suppose we define $\dot{U}_i^n(p_i) = \{u \mid u \in \dot{\mathbf{B}}^n \ \& \ D_H(u, u_i) \leq p_i\}$, $1 \leq i \leq k$, where $\dot{\mathbf{B}}^n$ is the universe of $n$-dimensional bipolar vectors; i.e., $\dot{U}_i^n(p_i)$ to be the set of $n$-dimensional bipolar vectors which have Hamming distance less than or equal to $p_i$ away from the given $n$-dimensional bipolar vector $u_i$, where $p_i$ is a specified precision level. To keep things simple, let us assume $p_i = p, 1 \leq i \leq k$. Suppose we define $f_M$ as follows:

$$f_M : \dot{U}^n(p) \rightarrow (2^V - \emptyset)$$

$$f_M(x) = \{v_i \mid x \in \dot{U}_i^n(p), \ 1 \leq i \leq k\}$$

where $\dot{U}^n(p) = \dot{U}_1^n(p) \cup \dot{U}_2^n(p) \cdots \cup \dot{U}_k^n(p)$, $\dot{U}_i(p) \cap \dot{U}_j(p) \neq \emptyset$ for some $i \neq j$, and $2^V$ is the *power set* of $V$ (i.e., the set of all subsets of $V$). The output of $f_M$ is a set of binary vectors that correspond to the set of patterns that should be recalled given the bipolar input vector $x$. $f_M$ is a partial function and is extended to a full function $\hat{f}_M$ to describe multiple recall in the associative memory module as follows:

$$\hat{f}_M : \dot{\mathbf{B}}^n \rightarrow (2^V \cup \{<0^m>\} - \emptyset)$$

$$\hat{f}_M(x) = \begin{cases} f_M(x) & \text{if } x \in \dot{U}^n(p) \\ \{<0^m>\} & \text{if } x \in (\dot{\mathbf{B}}^n - \dot{U}^n(p)) \end{cases}$$

$f_M$ can be further extended to function $f_{MP}$ for dealing with recall from a partially specified bipolar input pattern. $f_{MP}$ is defined as follows:

$$f_{MP} : \ddot{U}_P^{c \sim n}(p) \rightarrow (2^V - \emptyset)$$

$$f_{MP}(x) = \{v_i \mid x \in \ddot{U}_{Pi}^{c \sim n}(p), \ 1 \leq i \leq k\}$$

where $\dot{U}_{Pi}^h(p) \cap \dot{U}_{Pj}^h(p) \neq \emptyset$ for some $h$'s and $i \neq j$'s, $c \leq h \leq n$ and $1 \leq i, \ j \leq k$.

$f_{MP}$ is a partial function and is extended to a full function $\hat{f}_{MP}$ for multiple recalls from padded bipolar partial patterns in associative memory as follows:

$$\hat{f}_{MP} : \ddot{\mathbf{B}}^{c\sim n} \rightarrow (2^V \cup \{<0^m>\} - \emptyset)$$

$$\hat{f}_{MP}(x) = \begin{cases} f_{MP}(x) & \text{if } x \in \ddot{U}_P^{c\sim n}(p) \\ <0^m> & \text{if } x \in (\ddot{\mathbf{B}}^{c\sim n} - \ddot{U}_P^{c\sim n}(p)) \end{cases}$$

# 3 Query Processing Using a Neural Associative Memory

This section describes the use of the neural associative memory developed in previous section to implement a high-speed database query system. A library query system is used to illustrate the key concepts.

A neural associative memory that can be used to support a library system queried by name can be designed as follows: Suppose the input is a name (provided in a format with last name followed by first name) of an author, and characters that appear in the name are represented using ASCII codes (extended to 8 bits by padding each character code with a leading *0*).

Assume the length of both last and first name are truncated to at most $L$ characters each. Since each ASCII code consists of 8 binary bits, we need $8L$ input neurons for the last name and $8L$ neurons for the first name. Each binary bit $x_b$ of the ASCII input is converted into a bipolar bit $x_p$ by expression $x_p = 2x_b - 1$ before it is fed into the ANN memory module for database queries. (This is motivated by the relative efficiency of the hardware implementations of binary and bipolar associative memories – see (Chen & Honavar, 1995a) for details).

Let output be a $M$-dimensional binary vector pointing to a record in the library database that contains information about a volume (or the binary vector can encode information about a volume directly). The output binary vector in turn can therefore be used to locate the title, author, call number and other relevant information about a volume. Using $M$ output neurons, we can access at most $2^M$ records from the library database. Each hidden neuron in the associative memory module is used to realize an ordered pair mapping an author's name to an $M$-bit pointer that points to a record which contains information about a corresponding volume. During storage of an associated pair, the connection weights are set as explained in section 2.1. During recall, the thresholds of hidden neurons are adjusted for each query as outlined in Section 2.2 (where for each query, the value of $n_a$ can be set either by centralized check on user's input, or distributed circuitry embedded in input neurons). The precision level $p$ is set at 0 for this associative ANN memory module.

The library query system can handle queries using wild cards as illustrated by the following:

- **Case 1** : Suppose a user enters "*Smith* $*$" to search for the books written by authors with last name *Smith*. In this case, that part of input for first name is viewed as unavailable, the corresponding input neurons are fed with 0, only the first $8 \times 5 = 40$ input neurons have input value either 1 or -1, and the threshold set at all hidden neurons is $8 \times 5 = 40$. Suppose the library database contains $k$ volumes written by authors with last name *Smith*. In this case, the ANN memory module contains $k$ hidden neurons for these $k$ volumes (one for each volume written by an author whose last name is *Smith*). During the recall process all these hidden neurons will have net sum of 0 and other hidden neurons have net sum less than 0 (see section 2 for details). The neurons with non-negative net sum get activated one at a time to sequentially recall the desired $M$-bit pointers pointing to the books written by authors with the specified last name.

- **Case 2** : suppose a user enters "$*$ *John*" to search for the books written by authors with first name *John*. In this case, that part of input for last name is viewed as unavailable, and the corresponding input neurons receive 0s as input, and only the input neurons numbered $8L + 1$ through $8L + 4 \times 8$ receive inputs of either 1 or -1, and the threshold set at all hidden neurons is $4 \times 8 = 32$. The recall of the associated pointers proceeds as in case 1.

- **Case 3** : Suppose a user enters "*Smith J*$*$" to search for the books written by authors with last name called *Smith* and first name beginning with a *J*. In this case, the rest of the characters of first name are viewed as unavailable, the corresponding input neurons receive 0s as inputs, and only the neurons numbered 1 through 40 and $8L + 1$ through $8L + 8$ have inputs of either 1 or -1, and the threshold set at all hidden neurons is $6 \times 8 = 48$. The recall of the associated pointers proceeds as in case 1.

The large number of hidden neurons in such an ANN module poses a problem (because of the large fan-out for input neurons and large fan-in for output neurons) in the hardware realization of such a ANN module using electronic circuits. One solution to this problem is to divide the whole module into several sub-modules which contain same number of input, hidden, and output neurons. These sub-modules are linked together by shared input and output bus (see Figure 1). Such a bus topology also makes it possible to easily expand the size of the database. The *1*-dimensional array structure shown in Figure 1 can be easily extended to *2* or *3*-dimensional array structures.

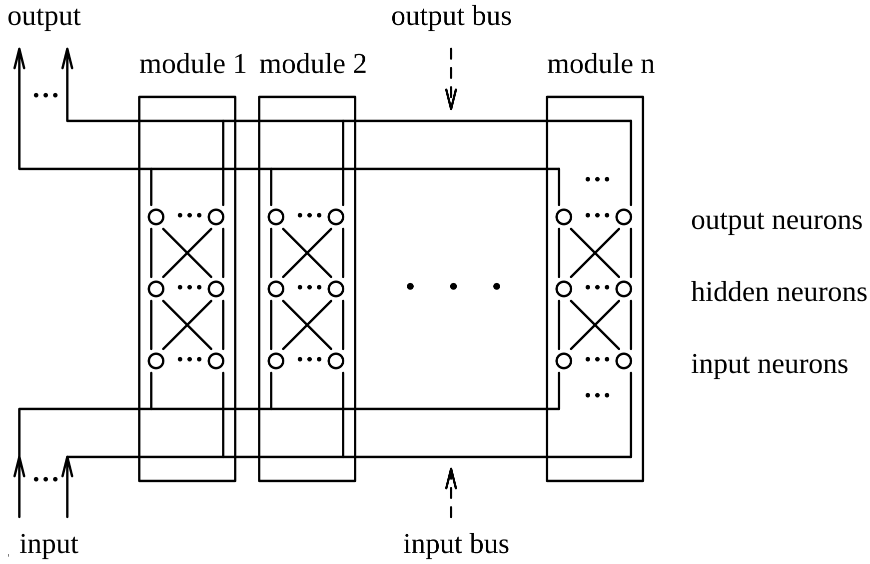


Figure 1 shows a modular design of the proposed ANN memory for easy expansion. This 1-dimensional array structure can be easily extended to *2* or *3*-dimensional array structures.

It is rather straightforward to modify the proposed database query system to make it case-insensitive. The following shows ASCII codes of English characters, which are denoted in hexadecimal and binary codes.

$$A = 41_{16} = 0100\ 0001_2,\ \ldots\ ,\ Z = 5A_{16} = 0101\ 1010_2$$

$$a = 61_{16} = 0110\ 0001_2,\ \ldots\ ,\ z = 7A_{16} = 0111\ 1010_2$$

The binary codes for the capital case and small case of every same English character only differ at the $3rd$ bit counted from left hand side. If that bit is viewed as "*don't care*", this query system will be case insensitive. This effect can be achieved by treating the corresponding input value as though it was unavailable.

# 4 Comparison with Other Database Query Processing Techniques

This section compares the anticipated performance of the proposed neural architecture for database query processing with other approaches that are widely used at present. Such a performance comparison takes into account the performance of hardware that is used in these systems and the process used for locating data items. First the performance of the proposed neural network is estimated, based on current CMOS technology for realizing neural networks. Next, the operation of conventional database systems is examined, and their performance is estimated and compared to that of the proposed neural architecture.

## 4.1 Performance of Current Electronic Realization for Neural Networks

Many electronic realizations of artificial neural networks (ANNs) have been reported (Graf & Henderson, 1990; Grant et al., 1994; Hamilton et al., 1992; Lont & Guggenbühl, 1992; Masa et al., 1994; Massengill & Mundie, 1992; Moon et al., 1992; Robinson et al., 1992; Uchimura et al., 1992; Watanabe et al., 1993). ANNs are implemented using mostly CMOS-based analog, digital, and hybrid electronic circuits. The analog circuit which mainly consists of processing elements for multiplication, summation and thresholding is popular for the realization of ANNs since compact circuits capable of high-speed asynchronous operation can be achieved (Gowda et al., 1993). (Uchimura et al., 1992) reports a measured propagation delay of 104 $ns$ from the $START$ signal until the result is latched by using digital synapse circuit containing an 8-bit memory, an 8-bit subtractor and an 8-bit adder. (Graf & Henderson, 1990) adopts a hybrid analog-digital design with 4-bit binary synapse weight values and current-summing circuits to achieve a network computation time of less than 100 $ns$ between the loading of the input and the latching of the result in a comparator. (Masa et al., 1994) adopts a hybrid analog-digital design with 5-bit (4 bits + sign) binary synapse weight values and current-summing circuits to implement a feed-forward neural network with 2 connection layers, and a network computation time of less than 20 *ns* is reported. (Grant et al., 1994) achieves the throughput at the rate of 10MHz (delay = 100 *ns*) in a Hamming Net pattern classifier using analog circuits. The 1st-layer and 2nd-layer subnetworks of the proposed neural architecture for database query processing are very similar to the 1st-layer subnetwork of a Hamming Net respectively, and the neural architecture with 2 connection layers in the proposed ANN is exactly same as that implemented by (Masa et al., 1994) except (Masa et al., 1994) uses discretized inputs, 5-bit synaptic weights, and sigmoid-like activation function. The proposed ANN uses bipolar inputs, weights in $\{-1, 0, 1\}$ and binary hardlimiter as activation function. Hence the computation delay of the proposed ANN can be expected to be at worst of the order of 100 nanoseconds and at best 20 nanoseconds given the current CMOS technology for realizing ANN.

The development of specialized hardware for implementation of ANNs is still in its early stages. Conventional CMOS technology that is currently the main technology for VLSI implementation of ANN is known to be slow (kumar, 1994; Lu & Samuleli, 1993). Other technologies, such as BiCMOS, NCMOS (kumar, 1994), pseudo-NMOS logic, standard N-P domino logic, and quasi N-P domino logic (Lu & Samuleli, 1993), may provide better performance for the realization of ANNs. Thus, the performance of the hardware implementation of ANNs is likely to improve with technological advances in VLSI.

## 4.2 Analysis of Query Processing in Conventional Computer Systems

In database systems implemented on conventional computer systems, given the values for a key, a record is located quickly by using key-based organizations including *hashing, index-sequential access* and *B-trees* (Ullman, 1988). For a very large database, the data is usually stored in secondary storage devices like hard disks. Conventionally, estimated cost of locating a record is based on the number of physical block accesses of secondary storage devices (Ullman, 1988). The cost-effective access time with current disk systems appears to be around 10 *ms*. In comparison, the proposed ANN-based implementation is clearly far superior when speed of access is the prime consideration. However, in order to ensure a fair comparison between conventional database systems and the proposed ANN based system, we will assume that the former uses $RAM$ (random access memory) to store all the data.

In the following comparisons, it is assumed that all program and data codes for processing queries using current computer systems are pre-loaded from main memory to caches (instruction, data and secondary caches) since accessing program instructions and data from main memory, the cost-effective access time of which is around 60 ns currently, slows down the performance of whole process. To simplify the comparison, it is assumed that all program instructions run on a currently cost-effective 100 $MIPS$ (million instructions per second; the effects of processor clock, code/data caches, pipeline, superscalar design and branch prediction are combined into this simple measure for the speed of a microprocessor. Note that these techniques take decades and many researchers to invent and evolve.) processor with 32-bit code/data bus and that every instruction takes 10 ns on average to access and execute although some instructions like those involving multiplication or division would take two or more times. The effect of data dependency among instructions which offsets pipeline and superscalar effects and thus much reduces the average performance of current computer systems is not considered here.

### Analysis of Query Processing Using Hashing Functions

Hashing structure is the fastest of all key-based organizations. However, although it is effective in locating a single record, it is inefficient for locating several related records in response to a single query. Let us consider the time needed for locating a single record using a hash function in current computer systems. Commonly used hash functions are based on multiplication, division and addition operations (Kohonen, 1987; Sedgewick, 1988). Assume the computer systems have dedicated hardware to execute multiplication and division. (Otherwise, their performance would not be as efficient without the help of dedicated hardware). In hardware implementation addition is faster than multiplication which is far faster than division. If each input field in a query contains $N$ bits, computing a hashing function will take several multiplications and/or divisions in at least $\lceil N/n \rceil$ cycles with an $n$-bit processor dedicated to each such field, depending on the algorithm of hashing function employed. Other overhead of computing hashing function in such systems includes the time for handling the potential problem of *collisions* in hash function. If the virtual single-CPU 32-bit 100 MIPS processor is used, it is expected that the total computation time for locating a single record will typically be far in excess of $100ns$ (If $L = 15$ and every cycle takes 5 instructions, the total computation time is about $\lceil N/n \rceil \times 5 \times 10ns = \lceil 8 \times 2L/n \rceil \times 50ns = \lceil 240/32 \rceil \times 50ns = 400ns$).

### Analysis of Query Processing Using Index Search

Perfectly balanced binary search tree is another popular data structure used in conventional database systems to locate a single record. Suppose every node in the binary search tree links two child subtrees and there are $(2^M - 1)$ nodes in the tree. Following discussion is based on the case of section 3. Since the binary search structure is assumed to be stored on RAM for high speed (and for fair comparison with the proposed ANN-based model), it is estimated that the computer system needs at least $(2L+6) \times (2^M - 1)$ bytes (or 72 $MB$, if $L = 15$ and $M = 21$ - The library at Iowa State University has over 2 million volumes) of RAM to store the binary search structure (8 $\sim$ 16MB RAM is typically the size of working memory in current PCs and workstations). The data structure for each node at least contains an index of $2L$ bytes and two 3-byte pointers (might need to use 4 or 8-byte pointers of type *long integer* or *double precision*, depending on computer systems used) pointing to $(2^M - 1)$ records. The average number of nodes visited for locating a desired record would be $\frac{1}{2}(\log 2^M) = 0.5M$. On an average, every visit takes about $4 \times 5 = 20$ instructions executed in a 32-bit processor (Assume $L = 15$, the comparison cycle for every 32 bits takes 5 instructions on average, and on average half of the 240 bits of key are compared in every visit). If the virtual 100 MIPS processor is used, the overhead will be about $0.5M \times 20 \times 10ns = 100M\ ns$ ($= 2100\ ns$ if $M = 21$) which is far more than $100ns$. Once again, note that this is only the cost for locating a single record. The cost for locating several related records using user-entered data for multiple fields (could be partially specified) is examined in next section.

**The Cost of Partial-Match Queries**

One of the most commonly used data structures for processing partial-match queries on multiple fields is *k-d-tree* (Bentley, 1975). In the worst case, the number of visited nodes in an ideal *k-d-tree* containing $M$ nodes (one for each record stored) for locating the desired records for a partial-match query is

$$\frac{(J+2)2^{K-J-1}-1}{2^{K-J}-1}[(M+1)^{(K-J)/K}-1] \approx O(M^{(K-J)/K}) \qquad (1)$$

where $K$ is the number of fields used to construct the *k-d-tree*, and $J$ out of $K$ fields are explicitly specified with query criteria by user. For typical values of $M$, $K$, and $J$, the performance of such systems is far worse than that of the proposed ANN based model according to expression (1).

# 5 Summary and Discussion

Artificial neural networks, due to their inherent parallelism and potential for fault tolerance, offer an attractive paradigm for robust and efficient implementations of a broad range of information processing tasks. In this paper, we have explored the use of artificial neural networks for query processing in large databases. The use of the proposed approach was demonstrated using the example of a library query system. The performance of a CMOS hardware realization of the proposed database query processing system was estimated and compared with that of other approaches that are widely used in conventional databases implemented on current computer systems. The comparison shows that ANN architectures for query processing offer an attractive alternative to conventional approaches, especially for dealing with partial-match queries in large databases. With the increased use of large networked databases over the Internet, efficient architectures for high-speed query processing are of great practical interest. Extensions of the proposed ANN architecture for query processing to deal with compound queries as well as syntax analysis is in progress (Chen & Honavar, 1995b; Chen & Honavar, 1995c).

# References

J. L. Bently, Multidimensional Binary Search Trees Used for Associative Searching, *Communications of the ACM*, vol. 18, no. 9, pp. 507-517, 1975.

C. Chen & V. Honavar, Neural Network Automata, *Proc. of World Congress on Neural Networks*, vol. 4, pp. 470-477, San Diego, 1994.

C. Chen and V. Honavar, Neural Associative Memories for Content as well as Address-Based Storage and Recall: Theory and Applications. *Submitted.* Preliminary version available as Iowa State University Dept. of Computer Science Tech. Rep. ISU-CS-TR 95-03, 1995a.

C. Chen and V. Honavar, Neural Network Architecture for Parallel Set Operations *In preparation*, 1995b.

C. Chen and V. Honavar, A Neural Network Architecture for Syntax Analysis. *Submitted.* Preliminary version available as Iowa State University Dept. of Computer Science Tech. Rep. ISU-CS-TR 95-18, 1995c.

Goonatilake, S. and Khebbal, S. (Ed.) Intelligent Hybrid Systems. *Wiley*, London, 1995.

S. M. Gowda et al., Design and Characterization of Analog VLSI Neural Network Modules, *IEEE Journal of Solid-State Circuits*, vol. 28, no. 3, pp. 301-313, 1993.

H. P. Graf and D. Henderson, A Reconfigurable CMOS Neural Network, *ISSCC Dig. Tech. Papers*, pp. 144-145, San Francisco, CA, 1990.

D. Grant et al., Design, Implementation and Evaluation of a High-Speed Integrated Hamming Neural Classifier, *IEEE Journal of Solid-State Circuits*, vol. 29, no. 9, pp. 1154-1157, 1994.

R. L. Greene, Connectionist Hashed Associative Memory, *Artificial Intelligence* **48**, pp. 87-98, 1991.

A. Hamilton et al., Integrated Pulse Stream Neural Networks: Results, Issues, and Pointers, *IEEE Transactions on Neural Networks*, vol. 3, no. 3, pp. 385-393, 1992.

V. Honavar and L. Uhr (Ed.) Artificial Intelligence and Neural Networks: Steps Toward Principled Integration. *Academic Press*, New York, 1994.

T. Kohonen, Content-Addressable Memories, 2nd ed., *Springer-Verlag*, Berlin, 1987.

R. Kumar, NCMOS: A High Performance CMOS Logic, *IEEE Journal of Solid-State Circuits*, vol. 29, no. 5, pp. 631-633, 1994.

D. Levine and M. Aparicioiv (Ed.) Neural Networks for Knowledge Representation and Inference. *Lawrence Erlbaum*, Hillsdale, NJ, 1994.

J. B. Lont and W. Guggenbühl, Analog CMOS Implementation of a Multilayer Perceptron with Nonlinear Synapses, *IEEE Transactions on Neural Networks*, vol. 3, no. 3, pp. 457-465, 1992.

F. Lu and H. Samueli, A 200-MHz CMOS Pipelined Multiplier-Accumulator Using a Quasi-Domino Dynamic Full-Adder Cell Design, *IEEE Journal of Solid-State Circuits*, vol. 28, no. 2, pp. 123-132, 1993.

P. Masa , K. Hoen and H. Wallinga, 70 Input, 20 Nanosecond Pattern Classifier, *IEEE International Joint Conference on Neural Networks*, vol. 3, Orlando, FL, 1994.

L. W. Massengill and D. B. Mundie, An Analog Neural Network Hardware Implementation Using Charge-Injection Multipliers and Neuron-Specific Gain Control, *IEEE Transactions on Neural Networks*, vol. 3, no. 3, pp. 354-362, 1992.

M. Minsky and S. Papert, Perceptrons: An Introduction to Computational Geometry, MIT Press, Massachusetts, 1969.

G. Moon et al., VLSI Implementation of Synaptic Weighting and Summing in Pulse Coded Neural-Type Cells, *IEEE Transactions on Neural Networks*, vol. 3, no. 3, pp. 394-403, 1992.

M. E. Robinson et al., A Modular CMOS Design of a Hamming Network, *IEEE Transactions on Neural Networks*, vol. 3, no. 3, pp. 444-456, 1992.

R. Sedgewick, Algorithms, 2nd ed., *Addison-Wesley*, 1988.

R. Sun and L. Bookman (Ed.) Computational Architectures Integrating Symbolic and Neural Processes. *Kluwer*, New York, 1994.

K. Uchimura et al., An 8G Connection-per-second 54mW Digital Neural Network with Low-power Chain-Reaction Architecture, *ISSCC Dig. Tech. Papers*, pp. 134-135, San Francisco, CA, 1992.

J. D. Ullman, Principles of Databases and Knowledge-base Systems, vol. I, Chapter 6, *Computer Science Press*, Maryland, 1988.

T. Watanabe et al., A Single 1.5-V Digital Chip for a $10^6$ Synapse Neural Network, *IEEE Transactions on Neural Networks*, vol. 4, no. 3, pp. 387-393, 1993.