PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1988,
edited by ICOT.

ARTIFICIAL INTELLIGENCE RELATED RESEARCH ON THE CONNECTION MACHINE

David L. Waltz
Craig Stanfill
Thinking Machines Corporation
245 First Street
Cambridge, MA 02142 USA

ABSTRACT

Artificial Intelligence (AI) has a demand for high-performance non-numerical computing which, until the development of practical parallel computing engines such as the Connection Machine, was largely unmet. In this paper we summarize a variety of AI applications of the Connection Machine. deKleer's ATMS has been modified to run on the CM, which is both much faster and much simpler than the LISP-based ATMS system. A new AI technique, memory-based reasoning, has been developed and applied to the pronunciation of English language text. A parallel dynamic memory system (PARADYME) has been developed for the case-based reasoning system JULIA. A variety of connectionist systems, including backpropagation learning and simulation of a goldfish retina, have also been written. A variety of work has been done in the area of machine vision, including low-level algorithms such as edge detection and stereo vision and high-level vision such as object recognition. The Connection Machine has also bee applied to a difficult search problem in chess endgame analysis. Finally, a variety of work has been done in scaling up natural-language systems to produce commercially viable systems. All the above work has been done in two years by a small number of researchers. On the basis of this, rapid progress in certain areas of experimental AI seem likely over the next few years.

INTRODUCTION

Artificial Intelligence has always been a consumer of high-performance non-numerical computers. While numerical computers have been developed which deliver hundreds of millions of operations per second, non-numerical computing has lagged, with no practical system delivering more than 10 million operations per second. This plateau in computational power has partly led to a plateau in experimental artificial intelligence, as the size of problems which may be worked on and the computational methods which are fast enough to be practical has (for the most part) stood still.

With the recent development of practical parallel computers such as the Connection Machine System [23][50] many non-numerical computations may now be performed at rates of several hundred million to several billion operations per second. This paper will discuss the impact which this machine has had on AI research over the past two years.

2.1 Assumption-Based Truth Maintenance

One major application of this machine has been to assumption-based truth maintenance systems (ATMS's). An ATMS is a propositional inference system used to aid in a variety of problems which may be reduced to propositional logic or (equivalently) constraint satisfaction. Examples of such problems include qualitative physics, vision, diagnosis, natural language processing, and map coloring. Dixon and deKleer were able to realize a 70-fold speedup on a 16K processor CM, while reducing the size of the ATMS code from 60 pages to 3.

2.2 Memory-Based Reasoning

Memory-based reasoning (MBR) is an AI technique in which problems are solved by the intensive use of memory, including in particular the use of best-match associative retrieval as an inference mechanism. Such techniques generally require exhaustive searches of memory which, on a serial machine, is expensive if the database is at all large. On the Connection Machine System, these algorithms become fast enough to be used for real-time processing.

2.3 Dynamic Memory

Case-based reasoning is an AI technique in which problem solving is achieved through the retrieval and modification of solutions to formerly encountered problems. As was the case with memory-based reasoning, associative retrieval is an important issue as the experience of the system increases. A parallel dynamic memory (PARADYME) has been implemented on the Connection Machine System to improve the performance of this phase of CBR. The resulting memory exhibits constant-time retrieval regardless of the size of the memory base.

2.4 Connectionism

Connectionism is a branch of AI concerned with the emergent behavior of large networks of neuron-like computational devices. The most prominent example of a connectionist technique is backpropagation learning, in which networks have been used for a variety of tasks including pronunciation of English words and signal processing. A backpropagation system has been implemented on the Connection Machine, realizing a significant speedup over VAX-based implementations. A second project within the connectionist paradigm simulated the formation of connections between the retina and visual cortex of goldfish.

2.5 Machine Vision

Vision has long been frustrated by the sheer volume of information contained in images. There are many low-level operations (e.g. thresholding, edge detection) that have been long understood, but for which no fast serial operations exist. The result is that research on higher-level vision has been slowed: experimenters spend an excessive amount of time waiting for low-level algorithms to run. Also, the slowness of low-level vision hampers the application of machine vision to real applications, where response time is usually of critical importance. A variety of vision programs have been written for the Connection Machine, including stereo vision and object recognition.

2.6 Chess Endgames

The determination of optimal play for chess endgames represents an interesting task in problem solving because certain endgames can be classified as wins, losses, or draws, even though forcing a win in some positions requires 50 moves by each side. Recently a technique has been developed for retrograde analysis of such positions, in which an algorithm starts by computing all won positions for a given combination of pieces, then starts working backward, labeling accessible positions as wins or loses. This technique works because, for small numbers of pieces, the number of possible positions is relatively manageable. A program implementing this technique was developed for the Connection Machine, and was 4000 times faster than the fastest previous solution. Work in progress will apply this technique to problems which are otherwise infeasible to attempt.

2.7 Full Scale Natural Language

AI has developed a variety of techniques for dealing with natural language text, but most of them have been applied only to "toy problems" with small amounts of text. Use of the Connection Machine promises to allow many of these techniques to be scaled up. The best example here is a commercial project in Information Retrieval (IR), in which the Connection Machine is used as a search engine to locate documents in a large database, using techniques such as document-ranking and relevance feedback. Preparation of the database for search-

ing presents many opportunities for the application of natural language processing techniques, but because the databases are many gigabytes in size, issues of scale become quite important. This has led to research in parallel parsing and analysis techniques.

3. PROCESSOR ARCHITECTURE

The Connection Machine Model CM-2 parallel processing unit contains up to 64K (K=1024) data processors. Each data processor contains:

- an arithmetic-logic unit (ALU)
- four 1-bit flag registers
- router interface
- NEWS grid interface
- optional floating point accelerator
- I/O interface
- 64K bits of bit -addressable memory

The data processors are implemented using four chip types. A proprietary custom chip contains the ALU, flag bits, router interface, NEWS grid interface, and I/O interface for 16 data processors. The memory currently consists of commercial dynamic RAM chips. The floating point accelerator consists of a custom floating point interface chip and a floating execution chip; one of each is required for ever 32 data processors. A fully configured parallel processing unit of 64K data processors, and therefore contains 4096 processor chips, 2048 floating point interface ships, and 2048 floating point execution chips, and half a gigabyte (512 MB) of RAM.

3.1 Data Processors

A CM-2 ALU consists of a 3-input, 2-output logic element and associated latches and memory interface. The basic conceptual AL cycle first reads two data bits from memory and one data bit from a flag; the logic element then computes two result bits from the three input bits; finally, one of the two results is stored back into memory and the other result into a flag. One additional feature is that the entire operation is conditional on the value of a third flag; if the flag is zero, then the results for that data processor are not stored.

The logic element can compute any two boolean functions on three inputs; these functions are specified by the sequencer as two 8-bit bytes representing the truth table for the two functions.

This simple ALU suffices to carry out all the operations of the instruction set for PARIS (PARallel Instruction Set, the CM's assembly language).

Arithmetic is carried out in a bit serial fashion: at about half a microsecond per bit, plus instruction decoding and other overhead, a 32-bit add takes 21 microseconds. With 64K processors, this produces an aggregate rate of 2500 Mips (that is, 2.5 billion 32-bit adds per second). All other PARIS operation are carried out in like fashion.

The ALU cycle is broken down into subcycles. On each cycle the data processors can execute one low-level instruction (called a nano instruction) from the sequencer and the memories can perform on read or write operation. The basic ALU cycle for a two-operand integer add consists of three nanoinstructions:

LOADA: read memory operand A, read flag operand, latch one truth table
LOADB: read memory operand B, read condition flag, latch other truth table
STORE: store memory operand A, store result flag

Other nanoinstructions direct the router, NEWS grid, and floating point accelerator, initiate I/O operations, and perform diagnostic functions.

3.2 The Router

Interprocessor communication is accomplished in the CM-2 parallel processing unit by special-purpose hardware. Message passing happens in a data parallel fashion: all processors can simultaneously send data into the local memories of other processors, or fetch data from the local memories of other processors. The hardware supports certain message-combining operation: the communication circuitry may be operated in such a way that processors to which multiple messages are sent receive the bitwise logical OR, SUM, MAX, or MIN of all the messages.

Each CM-2 processor chip contains one router node, which serves the 16 data processors on the chip. The router nodes on all the processor chips are wired together to form a boolean n-cube (this fact is not apparent to the programmer). For a fully configured CM-2 system, the network is a 12-cube connecting 4096 processor chips. Each router node is connected to 12 other router nodes; such that router node i

(serving data processors $16i$ through $16i+15$) is connected to router node j if and only if $|i-j| = 2^k$ for some integer k, in which case we say that routers i and j are connected along dimension k.

Each message travels from one router node to another until it reaches the chip containing the destination processor. The router nodes automatically forward messages and perform some dynamic load balancing.

The algorithm used by the router can be broken into stages called *petit cycles*. The delivery of all the messages for a Paris **send** operation might require a single petit cycle if only a few processors are active, but if every processor is active then several petit cycles are needed. It is possible for a message to traverse many dimensions, possibly all 12, in a single petit cycle, provided the congestion does not cause it to be blocked; the message data is forwarded through multiple router nodes in a pipe-lined fashion. A message that cannot be delivered by the end of a petit cycle is buffered in whatever router node it happens to have reached, and continues its journey during the next petit cycle. If petit cycles are regarded as atomic operations, then the router may be viewed as a store-and-forward packet-switched network. Within a petit cycle, however, the router is better regarded as circuit-switched network, where dimension wires are assigned to particular messages whose contents are then pumped through the reserved circuits.

Each router node also contains specialized logic to support virtual processors. When a message is to be delivered by a router node, it is placed not only with the correct physical processor, but in the correct region of memory for the virtual processor originally specified as the message's destination.

3.3 The Floating Point Accelerator

In addition to the bit-serial data processors described above, the CM-2 parallel processing unit has an optional floating point accelerator that is closely integrated with the processing unit. The hardware associated with each of these options consists of two special purpose VLSI chip (a memory interface unit and a floating point execution unit) for each pair of CM-2 processor chips. There are two possible options for this accelerator: Single Precision or Double Precision. Both options support IEEE standard floating point formats and operations. They each increase the rate of floating point calculations by more than a factor of 20. Taking advantage of this speed increase requires no change in user software.

3.4 The Role of the Front End

A front-end computer is a gateway to the Connection Machine system. It provides software development tools, software debugging tools, and a program execution environment familiar to the user. From the point of view of the user, the Connection Machine environment appears to be an extended version of the normal front-end environment. In addition to the usual suite of tools and languages provided by the front end, the environment includes at least one resident compiler or interpreter for a Connection Machine language. The front end also contains specialized hardware, called a Front-End Bus Interface (or FEBI), which allows communication with the Connection Machine.

A front end can be any computer system for which a FEBI exists. At the present times, a FEBI is available for most Digital Equipment Corporation VAX 8000 and 6000 series minicomputers, for Sun 4 workstations and compute servers, and for Symbolic 3600 series Lisp machine. Different types of front-end computers may be attached to the same Connection Machine and be running applications simultaneously. In addition, a single front-end computer may contain more than one FEBI to support up to four time-sharing users running Connection Machine applications simultaneously.

The front-end computer serves three primary functions in the Connection Machine system:

- It provides an applications development and debugging environment.
- It runs applications and transmits instructions and associated data to the Connection Machine parallel processing unit.
- It proves maintenance and operation utilities for controlling the Connection Machine and diagnosing problems.

3.5 Applications Development

Users create Connection Machine programs in the development environment provided by the front end. The editors, file systems, and

debugging tools are part of the front end's normal environment. The resident Connection Machine language, which contains parallel extensions to a language already familiar to the user, is used to express algorithms exploiting the data parallel structure of a problem. Thus, users with very little experience in data parallel programming may begin to use the Connection Machine immediately.

The native debugging facilities of the front end are augmented by simulators provided as part of the Connection Machine software system. The use of simulators can enhance productivity of users by allowing them to debug application programs, without tying up the Connection Machine hardware.

3.6 Connection Machine I/O Structure

The Connection Machine I/O structure allows data to be moved into or out of the parallel processing unit at aggregate peak rates as high as 320 megabytes per second for a system with multiple I/O controllers. Input/output is done in parallel, with as many as 2K data processors able to send or receive data at a time. All transfers are parity checked on a byte-by-byte basis.

The data processors send and receive data via I/O controllers, which interface through an I/O channel to Connection Machine data lines. These I/O controllers, in turn, operate under the control of the parallel processing unit sequencers. There may be as many as four sequencers in a fully configured system. The maximum I/O configuration for a 64K processor Connection Machine system includes eight I/O channels.

3.7 The DataVault

The DataVault is the Connection Machine mass storage system. It combines very high reliability with high transfer rates for large blocks of data. The DataVault holds five gigabytes of data, expandable to ten gigabytes. It transfers data at a peak rate of 40 megabytes per second and a sustained rate of 25 megabytes per second. Eight DataVaults, operating in parallel, offer a combined peak data transfer rate of 320 megabytes per second (200 megabytes pre second sustained) and hold up to 80 gigabytes of data.

Each DataVault unit stores its data in an array of 39 individual disk drives. Data is spread across the drives. Each 64-bit data chunk received from the Connection Machine I/O bus is split into two 32-bit words. After verifying parity, the DataVault controller adds 7 bits of Error Correcting Code (ECC) and stores the resulting 39 bits on 39 individual drives. Subsequent failure of any one of the 39 drives does not impair reading of the data, since the ECC code allows any single bit error to be detected *and* corrected. Although operation is possible with a single failed drive, three spare drives are available to replace failed units until they are repaired. The ECC codes permit 100% recovery of the data on the failed disk, allowing a new copy of this data to be reconstructed and written onto the replacement disk.

3.8 High-Resolution Graphics Display

The Connection Machine graphics system consists of a framebuffer module and a high-resolution 19-inch color monitor. The frame buffer, unlike the DataVault, is not connected to a Connection Machine I/O bus; instead it is a single module that resides in the Connection Machine backplane in place of an I/O controller. This direct back-plane connection allows the framebuffer to receive data from the Connection Machine processors at rates up to 1 gigabit per second.

The framebuffer contains a large video memory, which holds the actual raster image data. There are 28 planes of memory, divided into 4 buffer areas: red, green, and blue areas having 8 planes each, and an "overlay" area with 4 planes. Each plane provides one bit per pixel, and contains enough memory for 2^{21} pixels. There are also three color lookup tables (red, green and blue). Each color lookup table is 8 bits wide and has 259 entries; the first 256 entries handle data from the red, green, or blue area, and the last 3 entries are used for overlay processing.

The region displayed from the video memory planes is software configurable. Pan and zoom logix allows a specified subrectangle of the video memory to be displayed, magnified by an integral zoom factor. The subrectangle displayed at zoom factor 1 (no magnification) is typically 1280 x 1024 pixels.

3.9 Languages

The data parallel style of programming associates a processor with every element of a program's data. there are very few differences

between a data parallel program and a conventional serial program. In both cases, a single sequence of instructions is used, with a serial control structure. The Connection Machine system provides parallel processing without requiring the applications programmer to indicate synchronization explicitly in programs.

Because the data parallel and serial programming styles are similar, they utilize the same languages. The languages currently supported for the Connection Machine system are C*, Fortran, and *Lisp. The Fortran 8x array extensions to Fortran 77 are implemented directly, with no changes to the standard language definition. Each of the other languages is very close to the corresponding serial language specification, but in each case extends it by adding a new data type. Very little new syntax is added, the power of parallelism arising instead from extending the meaning of existing program syntax when applied to parallel data.

4. ASSUMPTION-BASED TRUTH MAINTENANCE SYSTEMS

Assumption-Based Truth Maintenance Systems (ATMS's), [13][14] are propositional inference engines, designed initially by Johan deKleer to support problem-solvers and other AI systems that need to search complex spaces efficiently. Recently an ATMS system was written for the Connection Machine by Dixon and deKleer. The CM provided a dramatic speed-up, by orders of magnitude over the previous fastest implementation[1]. ATMS's have been applied to problems in qualitative physics, vision, diagnosis, and natural language parsing [10][21][35].

4.1 Problem-Solving with Constraints

In the ATMS formalism, problem solutions consist of a set of variables which have been assigned specific values. If a particular variable, vi, can potentially take on values $\{a_0, a_1, \ldots a_n\}$, the ATMS formulation will create a set of assumptions, $\{vi \text{ <--- } a_0, vi \text{ <--- } a_1, \ldots vi \text{ <--- } a_n\}$, only one of which will be true. The full set of assumptions for all variables defines the search space of the ATMS. In order to decide which assumptions are true, the program must apply constraints, which allow the ATMS system to eliminate sets of assumptions that are inconsistent. (It should be clear that ATMS problems are closely related to satisfiability problems [9][55].)

4.2 ATMS on the Connection Machine

Two algorithms were tried, only one of which will be described here. In this algorithm, all assumptions are initially assigned a value of "unknown", and the full set of assumptions can thus be viewed as a solution vector, with all values "unknown". The ATMS system picks an assumption, and created two copies of the solution vector, one with the assumption marked "true" and the other with the assumption marked "false". The system then creates further copies of each of these vectors, assigning particular assumptions at each step to both "true" and "false", until the entire memory space of the CM-2 is used up. (Each of the copying steps can clearly be done in parallel; the last step before filling the machine copies K/2 vectors in parallel, where the total memory capacity of the CM-2 is K vectors. K may be much larger than 64K, through the use of the CM-2's virtual processor mechanism.)

One memory is filled, the ATMS system applies constraints. These constraints eliminate, in parallel, solution vectors that are impossible. Thus, for example, a constraint might state that that assumptions v_k <--- a_1 and vj <--- b_2 must have the same truth values. In that case, all vectors with v_k <--- a_1 = time and v_j <--- b_2 = false or with v_k <--- a_1 = false and v_j <--- b_2 = true will be eliminated.

Once half or more of the memory has been freed through the elimination of inconsistent vectors, another assumption can be applied, and so on. Clearly, the order in which assumptions are made has a critical effect on the overall speed of solution.

4.3 Results

The ATMS system was run off a 16K CM-2, and was applied to a number of problems. For a n-queens problem, with n=13, the CM-2 required 60 seconds, vs. 4235 seconds for the fastest sequential implementation on a Symbolics Lisp Machine. A 64K machine should easily achieve a 4x speedup over this. Moreover, the implementation, with aid from Craig Stanfill of Thinking Machines, was completed in a very short time.

[1] on lisp machines

As Dixon and deKleer write:

> The development of the parallel ATMS has also dramatically demonstrated the degree to which working around the performance limitations of serial machines has complicated otherwise simple algorithms. In order to obtain adequate performance the Lisp Machine implementation uses complex representations and elaborately crafted algorithms. Its development and tuning has taken over a year, and the resulting code is about sixty pages long. The Connection Machine algorithms are much simpler, require three pages of code, and took about a week to develop. In doing so we were also led to a clearer analysis of the ATMS, unencumbered by the complexities of the serial implementation's representation[15].

5. MEMORY-BASED REASONING

Memory-based reasoning is an AI technique which solves problems by the intensive use of memory [45]. The basic idea is that everything the system experiences is stored in memory then, when a similar situation comes along, the memory of the past situation is used to derive the answer. The basic task here is to determine which, of all the things in the system's memory, most closely matches the current situation.

This sort of best-match associative memory has not been widely studied in artificial intelligence due to the high cost of performing the lookup: the general solution to the problem of finding the best match between the current situation and items in memory which involves an exhaustive search of all memories. The usual work-around is to index memory in such a way that the search can quickly be restricted to a small number of potential candidates. Unfortunately, this indexing process requires that the system decide in advance which aspects of a situation are important, and does not constitute a general solution to the problem.

On a parallel machine there is a very good solution to this problem: store each memory in its own processor. When it comes time to recall experiences from memory, each event can rate itself according to how well it matches the current situation. The result is a practical means of accessing memory.

This system has been used in the area of pronunciation [42]. In the pronunciation task, the system starts by memorizing all the words in a subset of a dictionary. It is then given a set of words which are not in that dictionary, and endeavors to pronounce them by finding analogous words in the dictionary. This is the same task as was performed by the connectionist NETtalk system previously mentioned. The results are probably close to the limit achievable on the task: not perfect, but correct within the limits of English's ambiguous spelling. Even with a Connection Machine System, these experiments in memory-based reasoning are time consuming, and would have been impractical on a serial machine.

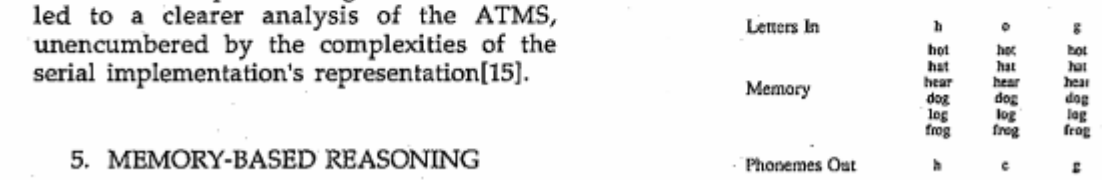

Figure 1: Memory Based Reasoning

6. PARALLEL RETRIEVAL FROM A HIERARCHICAL CASE MEMORY

Recently, a parallel system named PARADYME (Parallel Dynamic Associative Memory) has been implemented on the Connection Machine by Kolodner and Thau [30][48]. The goal of this system is to rapidly find a small number of best matching cases (cases typically represent events), given an incomplete set of retrieval cues. To provide a satisfactory cognitive model it is also important that retrieval time not increase as memory grows, and that generalizations as well as cases be retrieved as appropriate. Moreover, it should be reasonable easy to add new cases to memory.

PARADYME has been used with the JULIA meal planning database [25][28][29][41]. Items are found by first choosing a small set of categories to search, and then using the features of the case to be matched to drive the parallel traversal of the branches of the structures of cases in memory. Incorporation of new items into memory is accomplished by finding the best match in memory to the new item, and storing the new case near the best match. Specializations and generalizations are made as the item is incorporated.

7. CONNECTIONIST AND NEURAL NET MODELS ON THE CONNECTION MACHINE SYSTEM

Connectionism (also known as Neural Networks) is an approach to Artificial Intelligence in which computation is based on a large number of very small computational units, connected into a network, operating in a manner thought to be analogous to human neurons [20][37][54]. Each unit has "dendrites", "axons", and "synapses" connecting it to other units. As one unit in the network becomes "excited" it will, in turn, either "excite" or "inhibit" the other unites to which it is connected. The whole network is generally self-organizing, with mechanisms for growing connections or modifying connection strengths in response to the environment. The hope is that, if the correct methods of establishing and modifying connections can be found, connectionist systems will exhibit some of the flexibility and adaptability possessed by the human brain.

As the name "Connection Machine" would suggest, connectionist networks were an early stimulus to the development of the machine. The machine was, in fact, originally designed so as to enable the fast simulation of closely related "semantic networks"[6][19]. This led to the idea of building a machine with one processor for each node in the network. Such a machine clearly needs thousands or tens of thousands of processors. As the design progressed, it became clear that such massive parallelism was good for a wide variety of problems, and the Connection Machine became reality.

One prominent example of a Connectionist system is NETtalk, a system for leaning how to pronounce words [40]. This is done by feeding written text encoded as ASCII characters into one end of the network, pronunciations into the other end, and letting the network develop connections between the two via the back-propagation learning method [36]. When this learning phase is done, the network will be able to reproduce the pronunciations from the text, using the connections developed during the conditioning phase.

At this point, the CM has been used for several projects in the connectionist paradigm. NETtalk has been re-written for the Connection Machine [4][5]. This was done as a benchmark to establish the degree of speedup, but no new science as been done on it to date. More recently, the back-propagation system was used to predict the angles between adjacent bonds on amino acids, as part of a project to predict the tertiary structure of proteins, based on training on a corpus of known proteins[1].

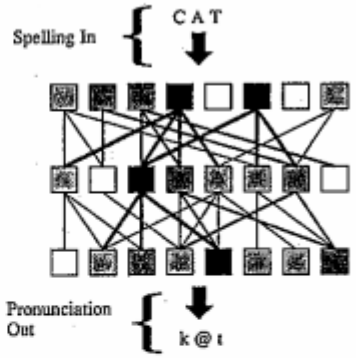

Figure 2: NETtalk

In addition, Robertson and Hillis have written a "retina simulator", which models the adaptive growth of connections between a "retina" and a "visual cortex" [24]. The program starts with a retina, a visual cortex, and a set of nerve fibers connecting the two. The problem is that the nerve fibers do not quite grow straight, and must organize themselves to that the visual cortex receives a coherent image. This is done by projecting onto the cortex a series of checker-board images, and allowing the units in the cortex to gradually develop connections to the nerve fibers so as to preserve the local coherence of the pattern.

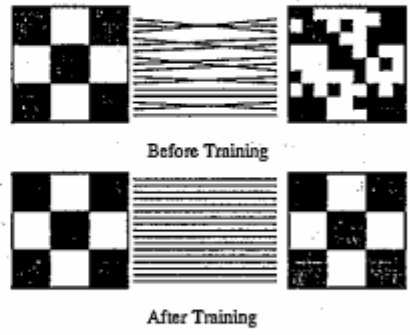

Figure 3: Retina Simulation

[1] This work was done by Xiru Zhang of Brandeis University and Dean Pomerleau of Carnegie Mellon University as part of their summer 1988 projects at Thinking Machines. No results have yet been published.

A final piece of work in the are of network-like representations is Blelloch's CIS (Concurrent Inference System)[3]. This system implements rule-based inference via a network representation, using node-activations to effect inference in the presence of uncertain data and uncertain rules of inference.

8. VISION

Computer vision research [1] has always been hampered by the lack of sufficiently powerful hardware. Low level vision - including such operations as edge detection and contrast enhancement - has always struggled with the difficulty that, even though the algorithms for these operations are quite simple, the amount of information contained in an image is quite large, and serial machines simply are not fast enough to process this information in a timely manner. The result is that experimentation becomes very slow, sometimes requiring hours to fully process a single image. In any event, processing times are so long that, even if the systems work, it is hard to see how they will be of use outside the laboratory.

High level vision - which attempts to perform such perceptual-level tasks as object recognition - suffers from its own set of computational problems. The first problem is the low-level vision task mentioned above: applying low-level algorithms to detect lines and perform other image transformation. Second, the results of these low-level operations are always ambiguous; where a human might see a single unbroken line in an image, a low-level vision algorithm might find two, and the line that is really there might be broken into three unconnected segments. There are methods of coping with these problems, but they are computationally quite demanding.

One low-level vision system which was implemented on the Connection Machine System was a stereo-vision algorithm [16] [17]. The system is given two images take from a pair of cameras. The images are then loaded into the CM's memory, and a series of operations applied. First, the images must be corrected: optical distortion and mechanical misalignment always cause the points in an image to end up some distance from where, in theory, they ought to be. This is solved by having a calibration map, which allows the CM to move every pixel in the image from wherever its apparent position is to where it would end up if the camera system were perfect. Second, edges must be detected, requiring a second pass over the entirety of both images. Third, edges in the two images must be matched. Once this is done, the differing positions of the two lines in the image allow the algorithm to determine how close it is to the camera. Finally, the height of areas which contain no line must be determined by interpolation.

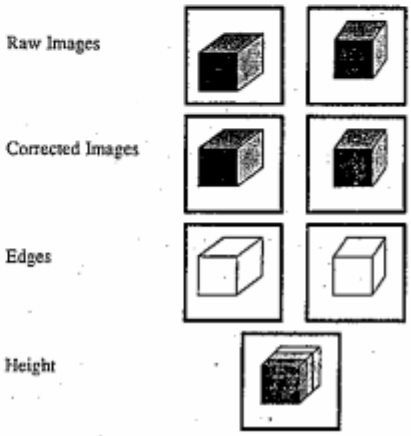

Figure 4: Stereo Vision

A high-level object recognition system has also been implemented on the Connection Machine [52][53]. It operates by matching edges extracted from images against a database of known objects. This system is unusual in that it allows a scene to contain multiple overlapping objects. Both the low-level processing needed to extract edges and the high-level algorithm for matching known objects against the image are computationally expensive, and benefit from increased processing speed.

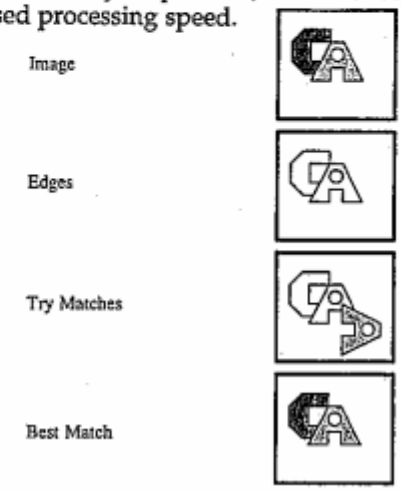

Figure 5: Object Recognition

9. CHESS ENDGAMES

Recent research by Stiller [47] has shown that chess endgames can be solved on a 32K Connection Machine (CM-2) about 4000 times faster than the best previous solution. The speed on a 64K CM-2 on the same problem is expected to be 4-5 times faster yet, about 17,000 times as fast as the previous solution[1] (1.5 minutes for a full five-piece endgame vs. 25,000 minutes). In Stiller's program the search space is encoded in the network topology, and the motion of tokens through the topology compute the search space in parallel.

9.1 The Chess Endgame Problem

Chess is played by two players who alternate moves using six different types of pieces. An endgame is roughly the portion of the endgame when most of the pieces have been captured (and thus removed from the board). Computer programs that have exhaustively searched five-piece end games have shown that traditional wisdom is incorrect concerning the outcomes of many of these games. For example, chess rules have stipulated that a game that lasts more than 50 moves without a change of "material balance" (i.e. a capture of one or more pieces) is a draw. Exhaustive search of five piece endgames has shown that there are some forced wins using more than 70 moves.

The goal of the program is to assign a value to every possible arrangement of the pieces in an endgame. The value represents either (1) the minimum number of moves to a forced win by the player to move, or (2) a notation that the position leads to a draw (or tie).

9.2 Problem Layout

Let us assume that we have a five piece endgame: two kings, white and black (the point of the game is to capture the opponent's king) and three other pieces P_1, P_2 and P_3. There are 64 possible squares on which each of these pieces can sit. Thus, we can represent all the possible arrangements of P_1, P_2, and P_3 by a 64 x 64 x 64 array of points (or, as it is actually done on the CM-2, by an (8 x 8) x (8 x 8) x (8 x 8) array). At each point in this array, we can represent all the possible positions of the two kings by a 450-bit vector. A number of points will correspond to impossible positions, i.e. when two or more pieces occupy the same square, these points are specially marked. The entire array is laid out on the CM-2 so that adjacent points correspond to positions that can be reached by moving the pieces.

9.3 Program Operation

To start the program, we mark all the positions that correspond to wins for one of the players (say white). We then find all the positions which force a win in one move; all these positions (points in the array) are marked with a value of "1", meaning that there is a win within one move. We then repeat this process until a step marks no new positions. The entire endgame is then solved.

One can envision the process in the following analog: suppose that each point in the array is either a slot that can hold a marble, or corresponds to a hole, so that marbles arriving there fall out of the array. All the points that correspond to impossible positions are made to be holes. We can place marbles painted with different colors in each of the slots that corresponds to a winning position. We can find all the ways to reach these positions by tipping the array, and letting the marbles roll, leaving paint, until they fall out of holes or off the edge of the board. We repeat with new marbles, tipping the array in a different direction, until all possible ways of reaching the position are marked. (There are four directions: up/down, right/left, and the two diagonals.) By placing marbles either on these paint-marked slots, or the complement of these sequences (all the slots without paint) on alternating moves, we can trace out the entire endgame space for both players.

The needed operations involve only NEWS-grid moves and bit comparisons (of the king vectors or the slots) and thus this operates very rapidly on the CM-2.

9.4 Related Problems

Similar methods will apply to domains that are small enough to be searched exhaustively, which can also sometimes prove useful at the leaves of a larger alpha-beta search. An example domain is for a "super optimizing" peephole compiler [11][26][31][34]. Another is the alternating Turing Machine [7]. We hope within the next year to begin exploring six-piece

[1] By Ken Thompson [50] on a sequent balance 12-processor MIMD machine.

endgames; these require approximately 64 times the memory required by the five-piece endgames. This problem will thus have to be solved using DataVault disk units for swap space.

10. FULL-SCALE NATURAL LANGUAGE SYSTEMS

For most of its history, AI has been concerned with "toy problems". Scaling up presents difficulties: at the two extremes of a spectrum, one can hand-code [33], or one can use methods to automatically build NLP systems. To date there have been very few practical applications of natural language processing[1], and this fact has dampened the enthusiasm of funding agencies and companies that support research in this area[2]. Fortunately, there are signs that this situation can be improved by strategically merging AI/NLP and Information Retrieval technologies. Over the last several years we have been involved with such AI/IR research; it offers novel opportunities both for automatic learning, and for building systems that have immediate practical value.

A series of experiments and discoveries led researchers at Thinking Machines, most notably Craig Stanfill and Brewster Kahle, to devise a document retrieval system that works in parallel on the Connection Machine [44]. The resulting system, now a commercial product, provides a high quality search through a clever interface that can be used effectively by a computer-naïve person after only about five minutes of training. The basic idea is this: A database of documents (e.g. news articles, abstracts, books, etc.) is distributed to each of the many processors of a Connection Machine (if documents are 2K bytes long, each processor can hold about twelve documents). The user types a few words (a question, description, or list of terms will do) and a carriage return; the terms are broadcast to all the processors in parallel along with a numerical "weight" indicating the importance of each term[1]. Each processor compares the terms with the contents of its documents, and adds the "weight" to the score for each document in which the term occurs. The headlines for the documents with the highest total scores are then displayed to the user. The user can view the text of each of these documents by clicking a mouse button while pointing to the headline. when the user sees a document (or paragraph of a document) that answers his request, s/he can mark it "good" by pointing and clicking the mouse. The system collects all the terms from all the documents marked "good" along with the initial words the user typed, and repeats the search process described above, but now with many more terms (often several hundred). Each search requires less than a second, even on databases up to 10 gigabytes. This method, called "relevance feedback"[39], generally produces substantially better search than is possible with boolean search systems [2][2].

This system offers two interesting opportunities to extend natural language processing systems:

1. It is highly desirable to add natural language pre- and post-processing to the existing system, to improve its performance, and to extend its capabilities. For example, we are building recognizers that can find, label, and store lists of terms that refer to company names, geographic locations, names of persons, etc. Ultimately, this will help users to ask and obtain answers to questions that would be very difficult to phrase as boolean queries. For example: typing "Earnings reports for New England utility companies" would expand to "Earnings reports for Maine, New Hampshire, Vermont, Massachusetts, Connecticut, Rhode Island, utilities, power companies, electric companies, Edison, power and light...". In addition, natural language processing systems will allow us to post-process retrieved documents, to filter out irrelevant articles, and thus improve the performance of the system from the user's point of view.

---

[1] Commercially available natural language systems include INTELLECT [22], a product of Artificial Intelligence Corp., which allows users to get information from a database by typing questions in English: NL Menu [48], a product of Texas Instruments, that lets users build natural language "front ends" for programs – one uses a mouse to select the words for each sentence from a set of menu choices, insuring that the system will "understand" any user input; Q and A, a natural language database front end from Semantic Inc., and EPISTLE, a grammar correcting system from IBM.

[2] DARPA, the Defense Advanced Research Projects Agency, expects to spend only about $1.4 million in fiscal 1989 on natural language processing, compared to over $7 million the previous year. Much of funding has been switched to support speech processing research.

[1] Term weights are assigned automatically by a program that pre-processes the text and updates the database. The number of occurrences of each term are saved, and weights computed proportional to the negative log of the probability of occurrences of each term.

[2] A commercial software system based on these ideas has been developed at Thinking Machines and purchased by Dow Jones, Inc.

2. The retrieval system itself can be adapted to extract phrase sentence and paragraph "templates" or patterns, in order to aid the building of recognizers for particular topics or types of stories. Such processing can provide empirical data on language usage that would be very difficult to find or invent any other way, leading to "dictionaries" of multi-word and multi-sentence language patterns and to FRUMP-like systems [12] with broad subject coverage[1].

Other related research, using dictionaries or thesauruses, has become popular in recent years. Some striking successes have been achieved by Ken Church and co-workers at AT&T Bell Laboratories [8][18] using the augmented "Brown Corpus" [32]. The Brown Corpus consists of one million words of text, chosen to represent a wide range of text types and styles (newspaper and magazine articles, books on history, economics, etc.). It was "augmented" by Kucera and Francis by assigning each word in the corpus to one of about 450 classes, covering standard grammatical categories (noun, verb, adjective) but also including substantially finer distinctions (e.g. noun + agent of sentence; verb + complements of particular types). Church collected statistics on the probabilities that various words would follow particular other word (or category) combinations. This system has been used to judge the most likely categories for words in novel text taken from newswire sources. Success rates for Church's system are in the range of 98-99%, much higher than for the best syntactic parsers (in the range of 33% [38]).

All these current lines of research emphasize breadth of coverage, rather than depth of coverage, and are thus complementary to the goals of traditional AI NL processing research. All present attractive alternatives to hand-coding [33], and all can be used to accelerate the research into deep processing. We believe these general approaches will have great importance in the ultimate story of the achieving of truly intelligent systems.

[1] FRUMP recognized about 70 types of stylized or "scriptal" newspaper articles: auto accidents, diplomatic visits, burglaries, etc. FRUMP contained quite specific, hand-coded sets of patterns to match each of the types of information needed to fill in the values in its scripts for each story type.

11. SUMMARY AND PROSPECTS

In the experimental sciences, an improvement in the tools available to experimenters usually leads to a series of rapid advances. Such an improvement has just taken place in the area of artificial intelligence. As the discussion above demonstrates, the speed of certain computations has just increased by a factor of 100-17,000. To the extent that artificial intelligence is empirically based, we can reasonably expect that this improvement in computational tools will lead to advances in science, both basic and applied. Indeed, this is already happening on a small scale.

It may be argued that the Connection Machine does not speed up *all* algorithms; that there are tasks for which the Connection Machine provides no speedup. It may also be argued that there are significant unsolved problems in artificial intelligence for which increased computational power is simply not required.

Both arguments are, of course, correct, and we do not intend to suggest that the Connection Machine is the answer to all AI's questions. It is clear, however, that many areas of research have long been stifled by the lack of adequate tools, and that others have never made it out of the laboratory due to the lack of computational resources sufficient to support commercial application.

We believe there are prospects for rapid progress in AI over the next several years, as the research community learns how to use this new tool. As with any new technology, some time will be required before a community of users facile in its use will develop. Some ideas that had been thought promising will fail, and other ideas that nobody had taken seriously will prosper. One thing is, however, clear: these vastly improved tools will yield new results previously unobtainable. The next several years should be exciting ones for AI.

REFERENCES

1. Ballard, D., and C. Brown, Computer Vision, Prentice Hall, Englewood Cliffs, NJ, 1982.

2. Blair, D. and M. Marion, "An Evaluation of Retrieval Effectiveness for a Full-Text Document Retrieval System" *Communications of the ACM* 28 3 pp 289-299, March 1985.

3. Blelloch, G., "CIS: A Massively Concurrent Rule-Based System" *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986.

4. Blelloch G., and C. Rosenberg, "Network Learning on the Connection Machine" *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, 1987.

5. Blelloch, G., and C. Rosenberg, "An Implementation of Network Learning on The Connection Machine", chapter in Connectionist Models and Their Implications, eds. D. Waltz and J. Feldman, Ablex Publishing, Norwood, NJ, 1988.

6. Brachman, R., "On the Epistemological Status of Semantic Networks" in Associative Networks: Representation and Use of Knowledge by Computers, ed. N.V. Findler, Academic Press, NY, 1979.

7. Chandra, A., D. Kozen, and L. Stockmeyer, "Alternation" *Journal of the Association for Computing Machinery*, **28** 1 pp. 114-133, January 1981.

8. Church, K., "A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text" Unpublished manuscript, AT&T Bell Labs, Murray Hill, NJ, 1988.

9. Cook, S., "The Complexity of Theorem Proving Procedures" *Proceedings of the Third Annual ACM Symposium on Theory Computing*, 1971.

10. D'Ambrosio, B., "A Hybrid Approach to Uncertainty" *International Journal of Approximate Reasoning*, to appear.

11. Davidson, J., and C. Fraser, "Automatic Generation of Peephole Optimizations" *Proceedings of the ACM SIGPLAN 1984 Symposium on Compiler Construction*, pp 111-116, June, 1984.

12. DeJong, G., "An Overview of the FRUMP System", chapter in Strategies for Natural Language Processing, eds. W. Lehnert and M. Ringle. Lawrence Erlbaum Assoc., Hillsdale, NJ 1982.

13. deKleer, J., "An Assumption-Based TMS" *Artificial Intelligence*, **28** pp 127-162, 1986.

14. deKleer, J., "Extending the ATMS" *Artificial Intelligence*, **28**, pp 163-196, 1986.

15. Dixon, M., and deKleer, "Massively Parallel Assumption-Based Truth Maintenance" pp 199-204, *Proceedings of Seventh National Conference on Artificial Intelligence*, St. Paul, MN. August 1988.

16. Drumheller, M., "Connection Machine Stereomatching" *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, PA 1986.

17. Drumheller, M., and T. Poggio, "On Parallel Stereo" *Proceedings of the IEEE International Conference on Robotics and Automation*, San Fransisco, CA 1986.

18. Ejerhed, E., "Finding Clauses in Unrestricted Text by Stochastic and Finitary Methods" unpublished manuscript, AT&T Bell Labs, 1988.

19. Fahlman, S., NETL: A System for Representing and Using Real-World Knowledge, MIT Press, Cambridge, MA, 1979.

20. Feldman, J., and D. Ballard, "Connectionist Models and Their Properties" *Cognitive Science*, Vol. 6 3 pp 205-254, 1982.

21. Forbus, K., "The Qualitative Process Engine" University of Illinois Technical Report UIUCDS-R-86-1288, 1986.

22. Harris, L., "Robot: A Full Performance Natural Language Data Base Query System" *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pp 903-904, 1977.

23. Hillis, D., The Connection Machine, MIT Press, Cambridge, MA, 1985.

24. Hillis, D., and G. Robertson, personal communications, 1987.

25. Hinrich, T. "Towards an Architecture for Open World Problem solving" *Proceedings of the DARPA Workshop on Case-Based Reasoning*, 1988.

26. Kessler, P., "Discovering Machine-Specific Code Improvements" *Proeedings of the ACM/SIGPLAN Symposium on Compiler Construction*, pp 249-254, June 1984.

27. Kolodner, J., "A Process Model of Case-Based Reasoning in Problem Solving" *Proceedings of the international Joint Conference on Artificial Intelligence*, LA, 1985.

28. Kolodner, J., "Capitalizing on Failure Through Case-Based Inference" *Proceedings of the 1987 Conference of the Cognitive Science Society*, 1987.

29. Kolodner, J., "Extending Problem Solver Capabilities Through Case-Based Inference" *Proceedings of 1987 International Machine Learning Workshop*, 1987.

30. Kolodner, J., "Retrieving Events from a Case Memory: A Parallel Implementation" *Proceedings of the DARPA Workshop on Case-Based Reasoning*, May 1988.

31. Krumme, D., and D. Ackley, "A Practical Method for Code Generation Based on Exhaustive Search" *Proceedings of the ACM SIGPLAN*, pp 185-196, June 1982.

32. Kucera, H., and W. Francis, Frequency Analysis of English Usage, Houghton Mifflin Company, Boston, 1982.

33. Lenat, G., M. Prakash, and M. Shepherd, "CYC: Using Common Sense Knowledge to Overcome Brittleness and Knowledge Acquisition Bottlenecks" *AI Magazine*, 4, pp 65-85, Winter 1986.

34. Massalin, H., "Superoptimizer - a Look at the Smallest Program" *Proceedings of the Second International Conference on Architectural Support for Programming Languages and Operating Systems*, pp 122-126, 1987.

35. Morris, P., and R. Nado, "Representing Actions With an Assumption-Based Truth Maintenance System" *Proceedings of the National Conference on Artificial Intelligence*, Seattle, WA, July 1987.

36. Rumelhart, D., G. Hinton, and R. Williams, et al., "Learning Internal Representations by Error Propagation", chapter in Parallel Distributed Processing, Vol. I, pp 318-362, MIT Press, Cambridge, MA, 1986.

37. Rumelhart, D., and J. McClelland, et al., Parallel Distributed Processing, MIT Press, Cambridge, MA, 1986.

38. Salton, G., personal communications, 1988.

39. Salton, G. The SMART Retrieval System: Experiment in Automatic Document Processing, Prentice-Hall, Englewood Cliffs, NJ, 1971.

40. Sejnowski, T., and C. Rosenberg, "NETtalk: A Parallel Network That Learns to Read Aloud" Electrical Engineering and Computer Science Department, Johns Hopkins University Technical Report # JHU/EECS 86/01, 1986.

41. Shinn, H., "Abstractional Analogy: A Model of Analogical Reasoning" *Proceeding of the DARPA Workshop on Case-Based Reasoning*, 1988.

42. Stanfill, C., "Parallel Computing for Information Retrieval: Recent Developments" Thinking Machines Corporation Technical Report # DR88-1, 1988.

43. Stanfill, C., "Memory-Based Reasoning Applied to English Pronunciation" *Proceedings of the Sixth National Conference on Artificial Intelligence*, pp 577-581, Seattle, WA, 1987.

44. Stanfill, C., and B. Kahle, "Parallel Free Text Search on the Connection Machine System" *Communications of the ACM*, Vol. 29, No. 12, December 1986.

45. Stanfill, C., and D. Waltz, "Artificial Intelligence on the Connection Machine System: A Snapshot" Thinking Machines Corporation Technical Report # G88-1, 1988.

46. Stanfill, C., and D. Waltz, "Toward Memory-Based Reasoning" *Communications of the ACM*, Vol. 29, No. 12, pp 1213-1228, 1986.

47. Stiller, L., "Parallel Analysis of Certain Endgames" Boston University Department of Mathematics Manuscript, September 1988.

48. Thau, R., "Details of an Implementation of a Frame System on the Connection Machine" Supplement to Kolodner, J. "Retrieving Events from a Case Memory: A Parallel Implementation", 1988.

49. Tennant, H. "Menu-Based Natural Language Understanding" *Proceedings of the National Computer Conference*, 1984.

50. Thinking Machines Corporation, "Connection Machine® Model CM-2 Technical Summary" Technical report # HA87-4, 1987.

51. Thompson, K., "Retrograde Analysis of Certain Endgames" *Proceedings of the International Joint Conference on Artificial Intelligence*, Vol. 9, No. 3, 1986.

52. Tucker, L., "Data Parallelism and Computer Vision Using the Connection Machine" *Proceedings of the Third International Supercomputing Conference*, Boston, MA, 1988.

53. Tucker, L, C. Feynman, M. Drumheller, D. Fritzsche, and D. Waltz, "Model-Based Object Recognition Using the Connection Machine" *Proceedings of th SPIE Conference on Intelligent Robots and Computer Vision*, 1988.

54. Waltz, D. and J. Feldman, eds. Connectionist Models and Their Implementations, Ablex Publishing, Norwood, NJ, 1988.

55. Zabih, R., and D. McAllester, "A Rearrangements Search Strategy for Determining Propositional Satisfiability" *Proceedings of the National Conference on Artificial Intelligence*, St. Paul, MN, August 1988.