Reprinted From
COMPUTER
Innovative technology for computer professionals

"The IXM2 Parallel Associative Processor for AI"

by Tetsuya Higuchi, Kennichi Handa, Naoto Takahashi, Tatsumi Furuya, Hitoshi Iida, Eiichiro Sumita, Kozo Oi and Hiroaki Kitano

IEEE COMPUTER SOCIETY
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

The IXM2 Parallel Associative Processor for AI

Tetsuya Higuchi, Kennichi Handa, and Naoto Takahashi
Electrotechnical Laboratory

Tatsumi Furuya, Toho University

Hitoshi Iida, Eiichiro Sumita, and Kozo Oi
ATR Interpreting Telecommunications Research Laboratories

Hiroaki Kitano, Sony Computer Science Laboratory

SIMD processing with large associative memories can support robust performance in speech applications. The IXM2 with 73 transputers has outclocked a Cray in some tasks.

Although researchers have been studying associative memory and associative processing for 30 years, the importance of associative memories as SIMD (single instruction, multiple data) devices has not been sufficiently recognized. Part of this is because large associative memories are difficult to develop, and the small associative memories available so far have allowed much less parallelism than needed to demonstrate effective SIMD processing on associative memories.

We describe the IXM2 associative processor and its main application in speech-to-speech translation. (IX stands for semantic memory system in Japanese, and the M stands for machine.) The IXM2 began as a faithful implementation of the NETL semantic network machine and grew into a massively parallel SIMD machine that demonstrated the power of large associative memories. In fact, it outperformed other significant machines in terms of language-translation tasks, which we discuss at the end of the article.

We selected speech-to-speech translation as our main application because it is one of the grand challenges of Massively Parallel Artificial Intelligence[1] (see the glossary on the next page). The social implications of successful automatic translation are enormous. Persons who speak different languages could communicate in real time by using interpreting telephony.

Speech-to-speech translation involves many computing operations because it requires such features as real-time response and a very large capacity for handling vocabulary and the corpus (see glossary for application terms). We focus on memory-based models of speech-to-speech translation that view episodic memory as the foundation of intelligence. This view naturally leads to implementing intelligent as-

0018-9162/94/ $4.00 © 1994 IEEE

Glossary

Corpus — Collection of texts consisting of documents, dialogue transcriptions, translations, and so forth.

Episodic memory — Recollection of a particular event characterized by a definite awareness that the event was personally experienced.

Grammar writing — Writing grammar rules for natural language analysis.

Massively Parallel Artificial Intelligence — Grand-Challenge-category AI problems being addressed through the computing power of massively parallel machines.

Parsing — Obtaining the syntactic structure of a sentence by applying grammatical rules to it.

Semantic distance calculation — (a) Calculating the semantic distance between words based on their meaning; (b) calculating the similarity or closeness of the meaning between words.

Semantic network — Schematic representation of knowledge that takes the form of a network. Network nodes represent concepts, and links represent the relationship between concepts.

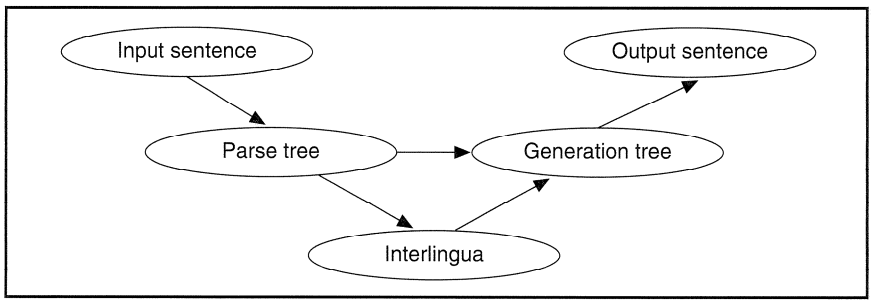


Figure 1. A traditional model of machine translation.

sociative memories. We first discuss the application as an illustration of associative processing and then shift to an exploration of the IXM2 itself.

Speech-to-speech translation issues

The traditional approach. The usual approach to natural language processing has been to rely on extensive rule application. This entails building an internal representation, such as a parse tree generated from an input sentence by using a set of rules. Figure 1 shows the main control flow in traditional models of translation.

Figure 2 is an example of traditional translation. First, the system inputs an English sentence for an analysis that uses grammar rules and a vocabulary dictionary. As the result, a parse tree is generated in which each node is a Japanese phrase or word. Then, the system applies rules and dictionaries to transform the parse tree into a generation tree. Finally, a Japanese sentence is output when generation rules and dictionaries are applied. Systems based on this approach inevitably face a number of drawbacks.

- **Performance.** Most existing machine-translation systems are not sufficient for such real-time tasks as speech-to-speech translation.
- **Scalability.** Current machine-translation systems are difficult to scale up because their processing complexity makes system behavior almost intractable.
- **Quality.** The intractability of system behavior combines with other factors to lower the translation quality.
- **Grammar writing.** By the same token, writing grammar rules is very difficult, since a complex sentence has to be described by piecewise rules. It is time-consuming (partly due to the intractability factor) to add these rules into the whole system.

Memory-based translation model. The alternative to the traditional approach is a memory-based model that uses examples for translation. This model was first proposed by Nagao[2] in 1984 and is the precursor of recent research on memory- and example-based translation models. Nagao argued that people translate sentences by using similar past examples of translation. This claim coincides with recent interest in memory- and case-based reasoning.[3]

Memory-based reasoning places memory at the foundation of intelligence. It assumes that numerous specific events are stored in people's memories and that they handle responses to new events by recalling similar events and invoking actions associated with them. This idea runs counter to most AI approaches, which make rules or heuristics the central thrust of reasoning.

The memory-based approach works well in machine translation, too. Figure 3 shows the control flow. At first, a sentence similar to the input sentence is retrieved. Then the retrieved sentence undergoes adaptation for generating an output sentence.

Let's consider the input sentence in Figure 2 again ("I would like to attend the party") and see how this sentence is translated into Japanese with a memory-based model. We used the ATR Interpreting Telecommunications Research Lab's corpus for conference registration as examples of translation. When the input sentence in Figure 2 is entered, the following sentences can be retrieved as the three most similar examples from the corpus. Each sentence is associated with a translated Japanese sentence.

- "I would like to register for the conference" becomes "Kaigi ni touroku shitainodesu."
- "I would like to take part in the conference" becomes "Kaigi ni sanka shitainodesu."
- "I would like to attend the conference" becomes "Kaigi ni sanka shitainodesu."

In the real system, each sentence will be assigned a similarity score. The next step is adaptation, in which differences between the input sentence and the example sentence are adjusted to create a translation. In this example, the differ-

ence is conference and party, so that the Japanese word for conference (kaigi) in the translation part of the example is replaced by the Japanese word for party (enkai).

Although at a glance this seems a naive approach, the method essentially is the same as or better than current MT (Machine Translation) systems. Given the fact that large-scale MT systems have a few thousand grammar rules for the exception handling of each specific case, memory-based translation is a straightforward and tractable approach to translation because it uniformly handles regular and irregular cases. In addition, there is a difference in the solution spaces of the two methods. Assuming that there is a set of grammar rules to cover a certain portion of natural language sentences, the grammar not only covers sentences that actually appear in the real world but also covers sentences that are grammatical but are never produced in reality. An empirical study revealed that only 0.53 percent of possible sentences considered to be grammatical are ever produced. The memory-based approach never generates implausible sentences because the memory contains only examples of actual sentences used in the past.

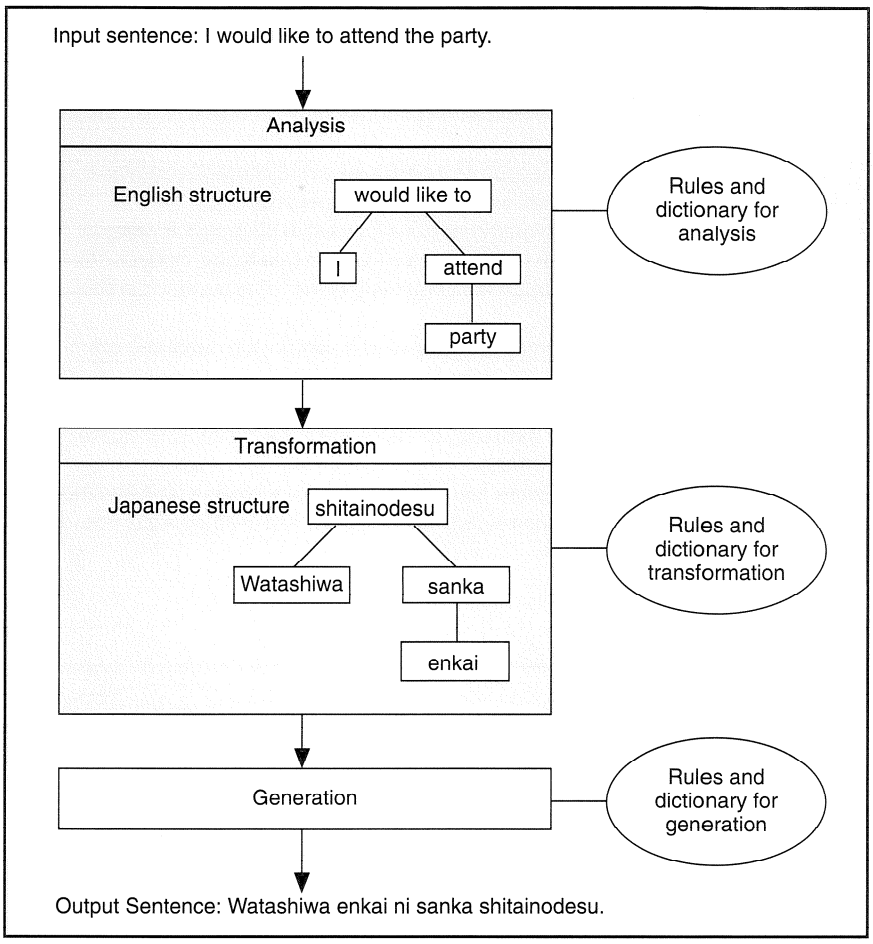


Figure 2. An example of traditional machine translation.

Parallel processing with associative memories

Apart from these linguistics discussions, the memory-based approach is well suited for data-parallel processing, as its performance is critical to search and retrieval of translation examples.

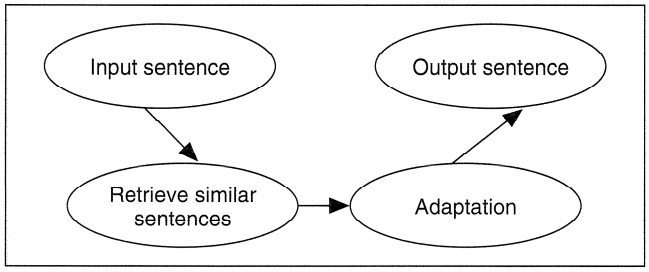


Figure 3. The control flow in memory-based machine translation.

Associative memory as an inexpensive SIMD device. The most time-consuming aspect of memory-based translation is data retrieval from an example database, which can be done in a SIMD manner. Therefore, commercial SIMD parallel machines, such as ICL's Distributed Array Processor (DAP), Goodyear's Massively Parallel Processor (MPP), and Thinking Machine's Connection Machine 2 (CM-2), are promising as computing platforms. In memory-based translation operations, however, the dominant operation is to retrieve data that matches input data; scientific calculations are less frequent than in other problems to which commercial SIMD machines are applied. Therefore, associative memories represent the best candidate for speech-to-speech translation because they are both compact and inexpensive.

Let's examine why this is true. First, when we focus on associative search operations, the "parallelism per chip" is the largest among various LSI implementations. At all the bits of associative memory, comparisons with input data bits are done in parallel; the parallelism is 20,000 within a 20K-bit Nippon Telegraph and Telephone (NTT) associative memory (AM) chip, which is the type used in the IXM2. Thus, large parallelism within a limited chip area can be obtained, since associative memory has regular cell structures suitable for LSI implementation, resulting in a compact and highly parallel device. Second, associative memories can be installed easily in a computer system; for example, they were installed in the IXM2 as ordinary memory devices. This leads to simple design and reduced cost. Third, no additional parallel software or software interface is required when using associative memories because they can be handled as ordinary memory devices. As described later, the associative

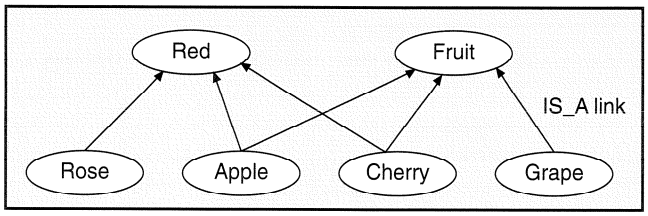


Figure 4. Network example.

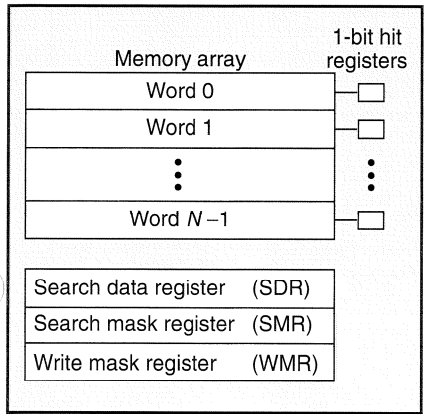


Figure 5. The architectural model of associative memory.

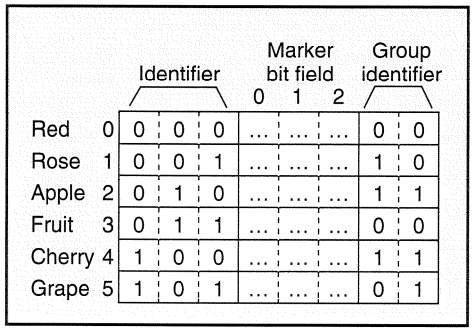


Figure 6. Network representation of associative memory.

memory operations in the IXM2 can be directly specified in the high-level language Occam2.

We next describe parallel processing using associative memory. We have chosen network-structured data processing (such as a semantic network) because it is widely used in natural language processing.

Marker bit operations. In memory-based translation systems, a network data representation is often used to describe a knowledge base such as a thesaurus. In processing network-structured data, three fundamental operations are used intensively: *association*, *set intersection*, and *marker propagation*. Association locates a particular node in a network, set intersection finds common members of two sets, and marker propagation obtains members belonging to a set. These operations are performed efficiently when marker bits are assigned to each network node. A marker bit is a one-bit flag that contains processing results.[4] Processing with markers is suitable for implementations with associative memories because execution times become independent of the amount of data, that is, O(1).

Figure 4 represents the marker bits. Suppose the network in the figure is asked which fruits are red. The solution will be the intersection of the set *fruit* and the set *red*, obtained in the following manner.

First, an association operation is executed to set marker bit number 0 of the node red. Next, marker propagation sets marker bits 0 of all the nodes that belong to the hierarchy of node red along the links, descending from node red. These nodes represent set members of node red. Similarly, marker bits 1 for members of node fruit are set using association and marker propagation. Finally, a set operation is executed to find the intersection of set red and set fruit. This operation is performed in two steps: Find nodes where both marker bits 0 and 1 are set, and then set marker bit 2 on those nodes. In this case, the answers are the apple and cherry nodes.

Parallel processing with associative memories. We next describe how these operations are efficiently performed with associative memories.

Required functions. The architecture model of associative memory for network data processing consists of a memory array, one-bit hit registers, a search mask register (SMR), a search data register (SDR), and a write mask register (WMR), as shown in Figure 5. In the memory array, problem data is stored word-wise, and data contents are compared in parallel at every bit with data in the SDR. A one-bit hit register is associated with each word of the memory array. This register is set when the word exactly matches the data in the SDR. The SMR is used when a part of each word (rather than the entire word) should match the corresponding part of the SDR. If a particular bit in SMR is 1, the search is conducted to the same bit position in the memory array. Otherwise, the search is disabled. A WMR is effective only when particular bits should be written; if a particular bit in a WMR is 1, the corresponding bit in the memory array is allowed to be written.

Representation for network data. Now we give data representations of this memory model for the network in Figure 4. The representation is node-based in that each network node occupies one word of associative memory. Each word has an identifier for its node, marker bits, and a group identifier for parallel marker propagation (described later), as shown in Figure 6. For example, the rose node occupies a word at address 1 of associative memory and its identifier is 001. Bit lengths in Figure 7 are chosen for this example. Information on connections between nodes is not kept in associative memory, but rather in RAM. (For clarity, this has been omitted from Figure 6.)

Set intersection in associative memory. At first, the red node is located by a search operation. Specifically, the SDR is loaded with the red's identifier, 000, at the leftmost position. The SMR is loaded to match only on the position of the node's identifier. The contents of the SDR and SMR are shown in Figure 7. After the search, word 0 is hit, and its one-bit flag register is set. Following this, a parallel-write operation is performed. This operation allows data writing only into those words whose one-bit hit registers are set. By this parallel-write operation, marker bit 0 of word 0 is set, though only word 0 is written in this case.

Second, members of the red node are obtained by marker propagations. Marker propagation can be done either sequentially or in parallel. Sequential marker propagation uses connection information stored in RAM, such as "Red is connected to the rose node with an IS_A link." (We use the term IS_A to indicate that a node belongs to another node conceptually.) Starting from a particular node like the red one, sequential marker propagation traverses a link and sets the marker bit of a node connected by that link. For example, executing sequential marker propagation three times

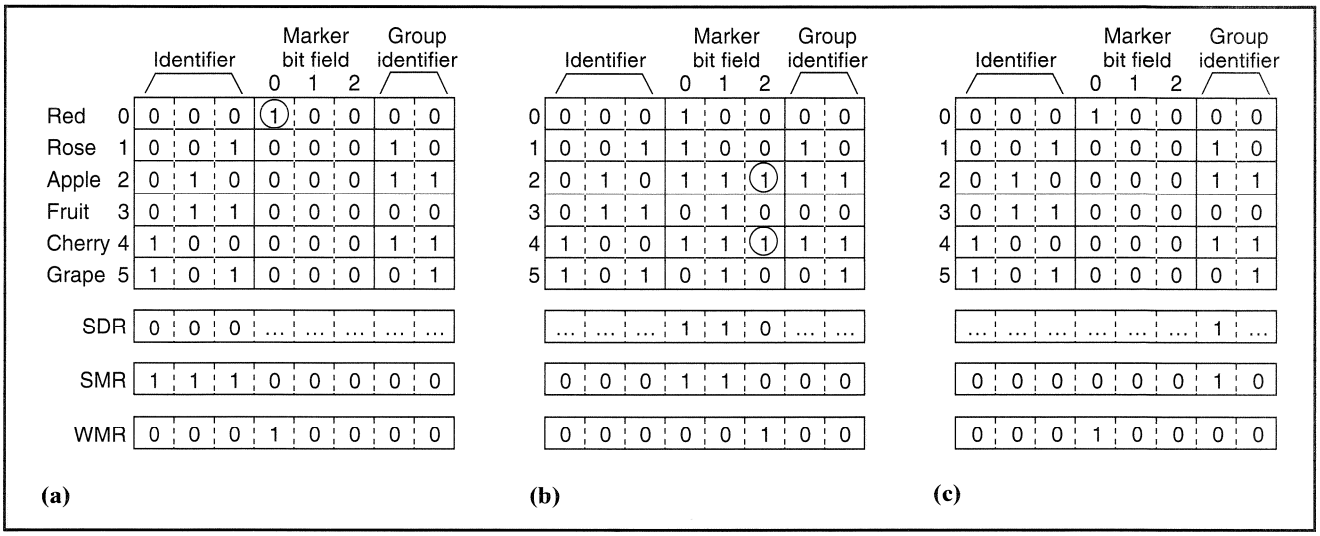


Figure 7. Register settings (SDR, SMR, and WMR) for marker bit processing: search (a), set intersection (b), and parallel marker propagation (c).

sets marker bits 0 of nodes rose, apple, and cherry. Figure 8a shows the result. Members of fruit are obtained in a similar manner; the fruit node is located by association and then its members, apple, cherry, and grape, are obtained by marker propagations (Figure 8b).

Finally, a set operation is performed to get the answer nodes; nodes where both marker bit 0 and 1 are set are located by a search operation, using SDR and SMR set as in Figure 7b. The search will locate words 2 and 4 (that is, the apple and cherry nodes) as the answer, followed by a parallel-write operation that writes marker bit 2 on those words (See Figure 7b).

While sequential marker propagation takes place in a time proportional to the number of links on which markers are propagated, parallel marker propagation can be done with constant times, regardless of the number of links. Parallel marker propagation is powerful in handling large network data where a node with many links (we call this a large fan-out node) often exists. In such a node, marker propagation has to be repeated as many times as the number of links connected to the node, resulting in an execution bottleneck.

In parallel marker propagation, a large fan-out node is referred to as a base node, and nodes connected to it are referred to as descendant nodes. For example, when members of the red node are obtained by marker propagation, the red node is a base node; and the rose, apple, and cherry nodes are descendants.

The basic idea of parallel marker propagation is to search descendant nodes by using a group identifier associated with each base node, and then write one particular bit at a time into each descendant node with a parallel-write operation. An allocator that encodes semantic networks into the representation on the IXM2's associative memories generates a group identifier for each group. A group consists of a base node, link type, and descendant nodes. For example, the network in Figure 4 has two groups: {RED, IS_A, {ROSE, APPLE, CHERRY}} and {FRUIT, IS_A, {APPLE, CHERRY, GRAPE}}. An associative memory word for each descendant node has a group identifier. Therefore, a base node can locate its descendant nodes by a search operation with the group identifier, and following this, a parallel-write operation sets each particular marker bit one at a time on each descendant node word. For example, via parallel marker propagation, descendant nodes of the red node can be located by a search operation with a value of 1, which is on the left bit of a group identifier in Figure 6, followed by a parallel-write operation. Figure 7c shows the register setting for those operations.

The IXM2

The IXM2 is the first massively parallel associative processor to clearly demonstrate the computing power of a large associative memory.[5] It attains 256K-fold parallelism by using the NTT AM chip.

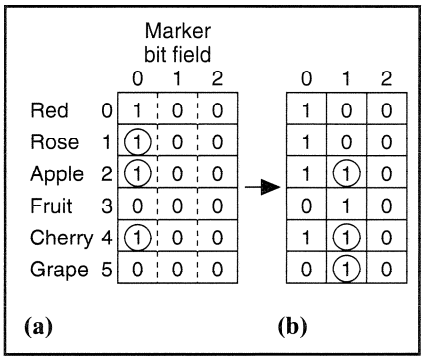


Figure 8. States of marker bit field: members of red (a) and fruit (b).

The system consists of 64 associative processors (APs) and nine network processors (NPs). The IXM2 is actually a coprocessor under the control of a host SparcStation (see Figure 9 on the next page). Each AP has an associative memory of 4 Kwords of 40 bits, plus a 17.5-MHz IMS T800 transputer. The IXM2 also employs a complete connection scheme to speed up marker propagation among APs.

In an AP, eight AM chips are connected to the transputer bus via simple interface logic just as additional memories are connected to the bus. No independent AM controllers are required.

The NTT AM chip. The associative memory chip is an NTT 20-Kbit associative memory (512 words by 40 bits) designed by

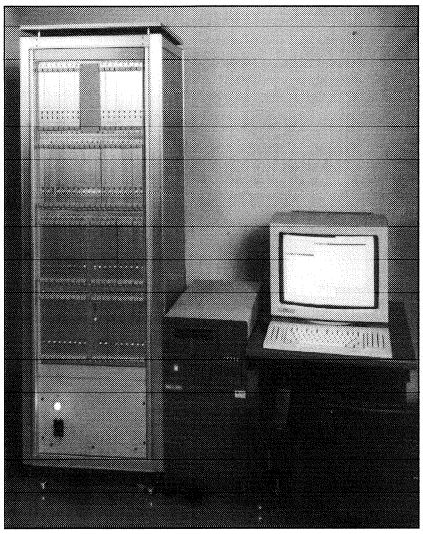


Figure 9. The IXM2 and its host Sun SparcStation-2.

Ogura[6] and his colleagues. The chip is a processor rather than a memory, executing 26 instructions classified into four types: search, write, read, and garbage collection. This combination provides powerful functions, such as relational search and long-word data processing. The parallel-write instruction, which writes a pattern in parallel into all the words just selected by a previously executed search instruction, is extremely useful in SIMD processing, especially when intermediate results of problem solving are stored as marker bits, as described above.

Programming with a high-level language. Associative memory processing can be specified directly with Occam2. User-defined reserved words are used in programming statements directly corresponding to NTT AM instructions. For example, AM.Init is declared to be a literal with a particular value. When this address value appears on the transputer address bus, a part of this value is interpreted as the AM initialization instruction by the associative memory interface. Thus, IXM2 programming is simple and requires no software-interface routines, which can cause processing overhead.

Memory-based translation

There are two lines of research based on this idea. The first one is to use a memory-based approach to identify top-level syntactic structures; the second is to use memory-based translation for translating phrases. These two approaches are complementary. Using sentence-structure matching in a memory-based approach lets the system segment phrases and identify macrostructures. The system can then decompose phrases and use memory-based translation to translate phrases at a high-quality level.

We describe three IXM2 memory-based systems: Astral[7] (Associative model of Translation of Language), the EBMT (Example-Based Machine Translation) and the TDMT (Transfer-Driven Machine Translation). Astral focuses on top-level tasks, and the EBMT focuses on phrase-level tasks. The TDMT is an effort to hybridize Astral and the EBMT.

Astral: An experimental system. This is an implementation of memory-based translation on the IXM2. Figure 10 shows the overall architecture. The memory consists of four layers: phoneme, phoneme sequence, lexical entry, abstraction hierarchies, and conceptual sequence.

- Phonemes represented as nodes in the network connect to each instance of a phoneme in the phoneme-sequence layer. Weights are associated with links that represent the likelihood of acoustic confusion between phonemes. The phoneme sequence of each word is represented in the form of a network (see Figure 11).
- The lexical (word) entry layer is a set of nodes. Each one represents a specific lexical entry.
- The class/subclass relation is represented using IS_A links. The highest (the most general) concept is *all, which entails all possible concepts in the network. Subclasses are linked under the *all node, and each subclass node has its own subclasses.
- Concept sequences that represent patterns of input sentences are represented in the form of a network. Concept sequences capture linguistic knowledge (syntax) with semantic restrictions.

Figure 11 shows a part of the network. The figure shows a node for the word "about," and how the phoneme sequence is represented. The left side of the figure shows how the network on the right side is represented in the IXL programming language.[8] (Refer to Kitano and Higuchi[7] for details of the mapping of semantic networks to IXM2.) We have encoded a network including *phonemes*, *phoneme sequences*, *lexical entries*, *abstraction hierarchies*, and *concept sequences* that cover the entire task of the ATR conference registration domain.

The vocabulary size is 405 words in one language, and more than 300 sentences in the corpus have been covered. The average fan-out of the network is 40.6. We did not set the weight in this experiment so that we could compare the performance with other parsers that do not handle stochastic inputs. (In real operation, however, a fully tuned weight is used.) We also used a hierarchical memory network to attain a wider coverage with smaller memory requirements.

The host computer contains the table for templates of the target language. The system creates the binding table between each concept and concept sequence, and specific substrings by using the corpus. When parsing is complete, the generation process is invoked on the host. It is also possible to perform distributed computing on 64 T800 transputers. The generation process is computationally cheap, since it only retrieves and concatenates substrings in the target language. These substrings are bound to conceptual nodes in patterns that mimic the concept sequence in the target language.

Algorithm. The parsing algorithm is simple. Activation markers (A-markers) and prediction markers (P-markers) control the parsing process. A-markers are propagated through the memory network from the lexical items, which are activated by the input. P-markers designate the next possible elements to be activated. When an A-marker collides with a P-marker, the A-marker is moved to the next element of the sequence. We have implemented several mechanisms to attain robust processing. These include tolerances against noisy phoneme insertion, phoneme misrecognition, and missing phonemes. This algorithm is similar to the basic framework of the ΦDM-Dialog speech-to-speech translation system.[9] (Refer to Kitano[10] for details.)

System performance. We carried out several experiments to measure system performance. The system attained parsing on the order of milliseconds. PLR is a parallel version of Tomita's LR parser. Its performance provides only a general

idea of the speed of traditional parsing models. Since PLR machines and grammars differ from those in our experiments, we cannot directly compare the two processes. However, the total time required and the exponential increase of parsing time in the case of PLR clearly demonstrate the problems inherent in the traditional approach.

The memory-based approach on the IXM2 (with the MBT) shows an order-of-magnitude faster parsing performance. Also, the parsing time increases almost linearly to the length of the input sentences, as opposed to the exponential increase seen in traditional parsing algorithms. The CM-2 runs at a slow speed but exhibits similar characteristics to those of the IXM2. The CM-2's speed is due to the processing element capabilities and the machine architecture. The fact that the CM-2 shows a similar curvature indicates the benefits of the MBT. The Sun-4 implementation that the performance degrades drastically as the size of the knowledge base (KB) grows, as discussed below.

We have also tested scalability by increasing KB size. This size is measured by the number of nodes in the network. Performance degradation grows more slowly than a linear function of the grammar KB, due to the local activation of the algorithm. This trend is the opposite of the traditional parser, in which parsing time grows faster than a linear function of the grammar KB size (which generally grows as the square of the size of grammar rules), due to a combinatorial explosion of serial rule applications. The CM-2 shows a performance trend similar to that of the IXM2, but is much slower due to the slow processing capability of 1-bit PEs. The Sun-4 has a disadvantage in a scaled-up KB due to its serial architecture. Particularly, the MBT algorithm involves extensive use of set operations to find nodes with an A-marker, P-marker collision, which is suitable for SIMD machines. Serial machines search the entire KB, which leads to the undesirable performance.

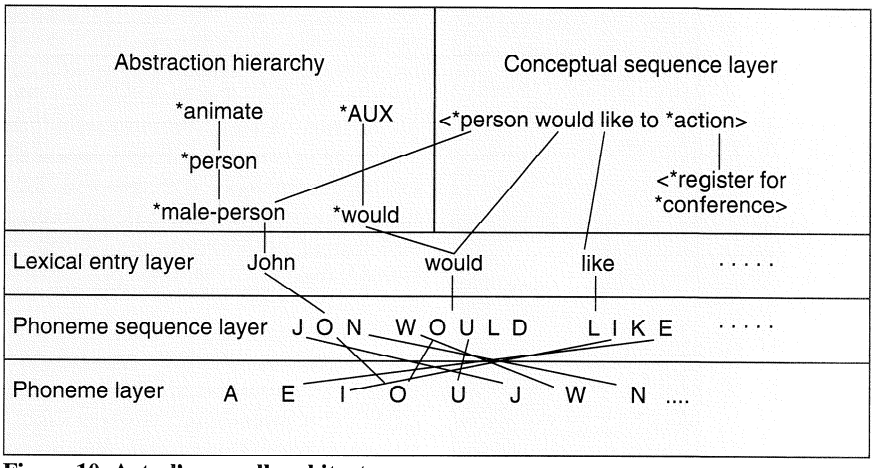


Figure 10. Astral's overall architecture.

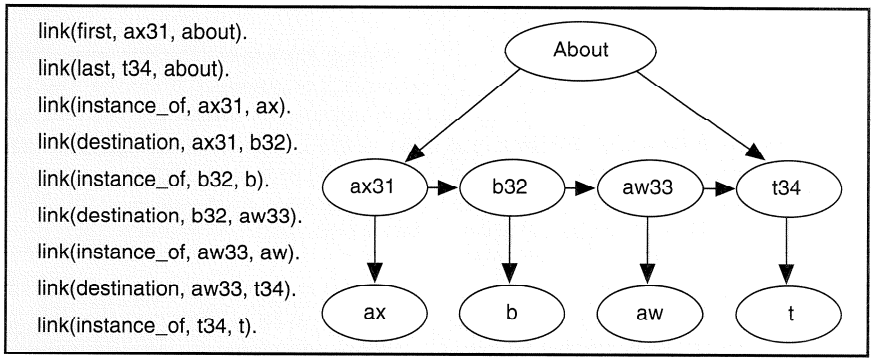


Figure 11. Network for the word "about" and its phoneme sequence.

Example-Based Machine Translation. While Astral focuses on identifying entire syntactic structures, the EBMT focuses on producing translation at the phrase level. The EBMT

(1) prepares a database consisting of translation examples,
(2) retrieves examples whose source part is most similar to the input phrase or sentence from the example database, and
(3) obtains the translation based on retrieved examples.

By focusing the task at the phrase level, we can collect a larger effective dataset to produce translation. This is a critical issue in attaining high-quality translation. One of the key issues in the EBMT is semantic distance calculation, and the IXM2 has been proven to be computationally effective for this task.

The other key issue in the EBMT is to determine a set of examples with minimum semantic distance from the input. In other words, the EBMT involves the full retrieval of the most similar examples from the example array.

Experiments. Suppose Input I and Example E are composed of n words. Our semantic distance measure is the summation of the distance, $d(I_k, E_k)$ at the k-th word, multiplied by the weight of the k-th word, w_k.

$$\sum_{k=1}^{n} d(I_k, E_k) \times w_k$$

The distance $d(I_k, E_k)$ is determined by a method we devised using a thesaurus. Each word corresponds to its concept in the thesaurus. The distance between words is reduced to the distance between concepts. The distance between concepts is determined according to their positions in the thesaurus hierarchy. The distance varies from 0 to 1. When the thesaurus is $(n+1)$ layered, the distance, (k/n) is given to the concepts in the k-th layer from the bottom ($0 \le k \le$ n). The weight w_k is the degree to which the word influences the selection of the translation. For example, the Japanese words "kaigi" (conference) and "kaisetsu" (commentary) have a distance of 1/3 because a minimal common node in the thesaurus hierarchy can be found at 1/3 of the total hierarchy layer.

But, the Japanese words "kaigi" (conference) and "soudan" (meetings) have a distance of 0, since the words are subsumed by a concept at the bottom layer. This is a very crude way of determining the distance, but our experiments demonstrate that such methods work reasonably well in the domain.

In fact, it has been demonstrated[11] that the EBMT outperforms traditional machine translation in the accuracy of target word selection in function words (for example, Japanese particles and English prepositions).

In the experiments, we translated the Japanese noun phrases of the form "A no B" into English. (A and B are nouns, and "no" represents an adnominal particle.) They are translated into various English noun phrases such as "$\tilde{B}$ of $\tilde{A}$," "$\tilde{B}$ in $\tilde{A}$," "$\tilde{B}$ for $\tilde{A}$," "$\tilde{B}$ at $\tilde{A}$" and so on ($\tilde{A}$ and $\tilde{B}$ are English translations of A and B.) They are representative because the examples are composed of three variables, or words (two or three variables is typical), and they are difficult in traditional machine translation, although the EBMT can handle them well. The "A no B" examples were collected from the ATR corpus, which consists of Japanese sentences and their English translations concerning the conference registration task.

Experimental results. We have experimented with 1,000 examples. The processing time was 270 milliseconds on the SparcStation-2 and 21 milliseconds on the IXM2. The IXM2 attains a speed almost 13 times greater than that of the SparcStation-2. A fully equipped IXM2 (with 64 associative processors) loads 64,000 examples. For this set size, the IXM2 can attain a speed $13 \times 64 = 832$ times greater than the SparcStation-2 if communication overhead is not considered.

The IXM2 can perform in this manner because it carries out matching processes by using parallel search and parallel-write operations for the AM chip. Distance calculations are triggered in constant time and are executed in linear time by the transputer.

On the other hand, another massively parallel machine, the CM-2, took 240 milliseconds. Although the CM-2 conducts matching processes in constant time, the example collection process is executed linearly by the host machine, a SparcStation-2. Unfortunately, the communication between CM-2 and SparcStation-2 occurs through the VMEbus, which is very slow. Thus, the advantage of a data-parallel search was completely lost in the communication bottleneck of data collection by the host.

Transfer-Driven Machine Translation. The TDMT is an effort to design and develop a novel machine-translation architecture based on our previous experiments with Astral and the EBMT.[12,13] the TDMT is essentially a combination of two processes: sentence pattern identification and phrase-level translation. After the sentence pattern is identified, phrase translation proceeds concurrently.

The TDMT approaches translation by determining how concepts expressed in one language should be converted to another.

Although most machine-translation models consider translation as a pipelined process of parsing-transfer-generation, the TDMT approaches translation as a problem of determining how concepts expressed in one language should be converted into another. The TDMT discards the notion of pipelined processing and adopts a transfer-driven view. This approach resulted in vastly different machine-translation architectures. Translation is performed by the transfer module using stored empirical transfer knowledge. Other modules, such as lexical processing, analysis, generation, and contextual processing, help the transfer module apply transfer knowledge and produce correct translation results.

In the TDMT, transfer knowledge is the primary knowledge used to solve translation problems. Most of the knowledge is described by an example-based framework. Transfer knowledge describes the correspondence between source-language expressions (SEs) and target-language expressions (TEs) in certain meaningful units, preserving the translational equivalence.

The TDMT uses the EBMT's semantic distance-calculation method to determine the most plausible target expression and structure in a transfer. The semantic distance between words is reduced to the distance between concepts in a thesaurus.[8,9]

Transfer knowledge in an example-based framework is described as follows:

$$\begin{array}{lll} \text{SE} & \Rightarrow & \begin{array}{ll} \text{TE1} & (\text{E11}, \text{E12}, \ldots), \\ \vdots & \vdots \\ \text{TEn} & (\text{En1}, \text{En2}, \ldots) \end{array} \end{array}$$

Each TE is associated with examples. E_{ij} indicates the j-th example of TE_i. To select a TE, the input sentence is matched against all examples in the memory base using an extended version of the semantic distance calculation adopted in the EBMT. An example with the smallest distance from the input is chosen, and the TE of that example is extracted to make the most plausible target sentence.

Massively parallel TDMT. The most important consideration in implementing the TDMT strategy on a massively parallel machine is to maximize parallelism. Since semantic distance calculation is the most computationally demanding step in the TDMT, our successful implementation of this calculation on the IXM2 was significant. Careful analysis of the computational cost in the sequential version of the TDMT reveals that the semantic distance calculation for the top 10 patterns accounts for nearly 94 percent of the whole semantic distance-calculation time.

Accordingly, we have decided to implement the semantic distance calculation of the top 10 patterns on the IXM2 to substantially reduce the computing time. Each pattern of the top 10 is assigned to each associative processor. Examples of each form are stored in one associative processor. Thus, examples of the top 10 patterns are stored in 10 APs.

Performance analysis of sequential vs. massively parallel TDMTs. To attain real-time performance for spoken language translation, we assume that the translation time per sentence must be a few seconds, at most.

In a test run of 746 sentences with sequential TDMT,

(1) half were translated in less than 2 seconds,
(2) most of the rest were translated in more than 2 but less than 10 seconds, and
(3) 25 were translated in more than 10 seconds.

In terms of the experiment mentioned in the EBMT section, the semantic dis-

tance calculation time is reduced to about 1/13 when using the IXM2. As a result, the total translation time per sentence is reduced by half on average and is a maximum of 2.5 seconds for 97 percent of the test sentences (Figure 12). Therefore, the massively parallel TDMT will attain a speed sufficient for spoken language translation.

Discussion

The most salient feature of memory-based translation is that matching input with an example database can be performed with fine-grained parallelism, but example collections cannot be parallelized after the matching procedure. Execution time for the example collection is influenced heavily by differences in computer architecture. To examine this, let's look at the EBMT results again.

The CM-2 has a serious communication-overhead problem. The VMEbus bottleneck is between the host and the CM-2. The CM-2 took only 9 milliseconds to calculate each semantic distance. The data-parallel nature of the EBMT makes this a quick process. This computing time is constant up to 64K examples because one processor has only one example to calculate. Despite this high performance in distance computing, the CM-2's overall response time is 240 milliseconds, indicating the significant impact of the communication overhead.

Intel's iPSC/2 loosely coupled multiprocessor system took 328 milliseconds for 1,000 examples with a 16-PE configuration, though this is slower than the 270 milliseconds achieved by a SparcStation-2. Interprocessor communication absorbs 110 milliseconds of the 328 milliseconds. Based on our experiments, a 256-PE iPSC/2 would require 220 milliseconds for communication, whereas semantic distance calculation is completed in about 14 milliseconds (for the 1,000 examples). Thus, communication overhead is also a serious problem in a loosely coupled multiprocessor.

Nor is the Cray X-MP 216 a good choice; it was slower than more recent workstations such as the 99-MHz HP735: 101 milliseconds for the Cray and 90 milliseconds for the HP with 1,000 examples. The main reason for this is that translation does not involve many floating-point operations. In addition, the average vector length was about 60, which is not suitable for a vector supercomputer.

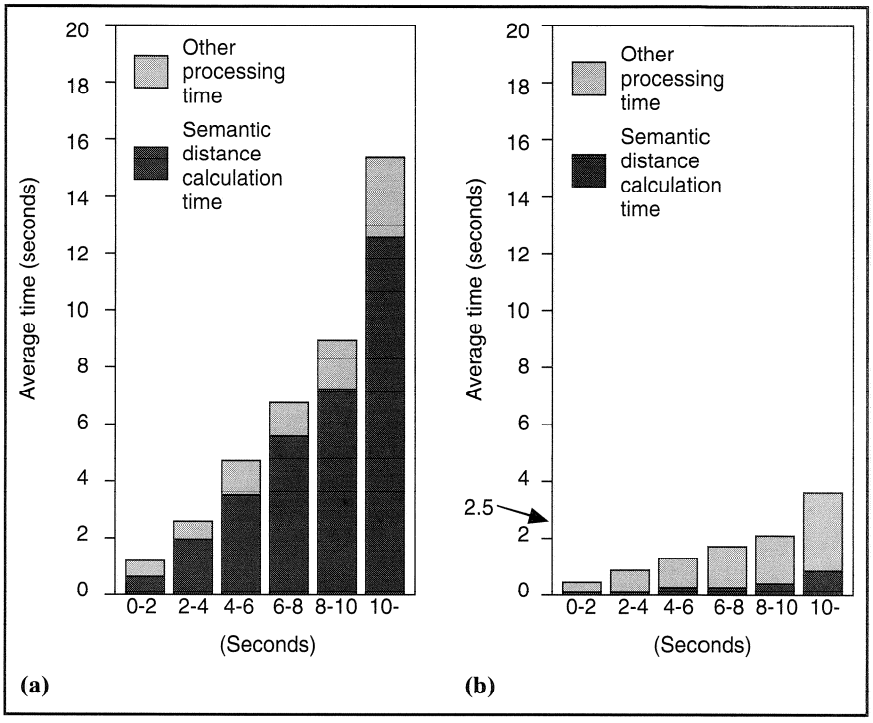


Figure 12. Translation time for sequential TDMT (a) vs. massively parallel TDMT (b).

The IXM2 architecture was superior to other machines because of high-performance example search by associative memory and minimum communication overhead due to the use of the transputer for collection examples. In fact, the interprocessor communication required for the EBMT is simply the transmission of the chosen examples to the host. Experimental data indicates that the IXM2 took 14 milliseconds for matching and 7 milliseconds for collection. Though execution time only for IXM2 distance calculation is slower than that for the CM-2 (9 milliseconds), the IXM2's overall performance is better.

Requirements for associative memory as a SIMD parallel device. Because of our experiences in using associative memories for massively parallel computing, we believe three functions are required for associative memories: (1) partial writing, (2) parallel writing, and (3) a multiword function.

In marker bit processing implemented on associative memories, as we described, each marker bit is used as temporary storage of intermediate processing results, just as a microprocessor uses register files to store intermediate results. This usage of associative memory is inevitable for SIMD parallel processing on associative memories. To realize this usage, partial writing capability that permits bits to be written selectively is required.

The parallel-write operation is second only to the search function in terms of contributing the most to parallel processing on associative memories. As previously described, this function reduces the algorithmic complexity of the problem; for example, set intersection on a sequential machine takes $O(N)$, where N is the number of data, whereas it takes O(1) on the IXM2. Set intersection with associative memory is very powerful, outperforming a Cray by two orders of magnitude for 64K data.[5]

A key issue in using associative memories is how to represent (or encode) necessary information on associative memory in the most compact form; however, multiple words are usually required. A multiple-word function is needed to represent information that cannot be represented with one word. To implement a multiword function, the search result of one word (that is, the content of a hit register) should be propagated to an adjacent word. This function is implemented in the NTT AM chip to represent one EBMT example with four words.

Though the IXM2 attains better performance in memory-based translation than other computers

Status report

The IXM2 is the successor of the IXM1, which began in 1987 as an implementation of the NETL semantic network machine proposed by S. Fahlman. By 1990, the IXM2 grew into a massively parallel SIMD machine with large associative memories. Research was mainly performed by T. Higuchi in collaboration with Carnegie Mellon University and ATR with the overall goal of developing a machine that could answer one of the Grand Challenges — automatic real-time speech translation between languages.

The IXM2 uses 64 associative processing transputers and nine communications transputers, having a total of 256 Kwords of associative memory (one of the largest associative memories developed). It can outperform a Cray supercomputer by an order of two magnitudes in set-intersection operations, which are heavily used in such AI applications as speech translation and knowledge-base processing.

The IXM2 is being upgraded with faster transputers and memories, and a TDMT that uses 5,000 examples. Initial experiments show that the IXM2 can maintain its current level of response while processing this increase in examples.

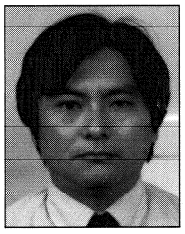
do, there is much room for performance improvement. The first area is the associative memory. The NTT AM chip used for the IXM2 was made in 1988 with a 1.2-micron VLSI design rule, whereas a 0.5 micron design rule is now common. The chip operates at 4.4 MHz, which is quite slow compared with the current technology standard VLSI implementation. In addition, the silicon chip size of the NTT AM is almost one fourth of today's VLSI standard. Considering recent progress, there is no doubt that we can make much more powerful AM chips. Ogura, who designed the NTT AM chip, estimates that were the NTT AM chip to be designed again with current technology; a 256-Kbit, 40-MHz AM chip would be feasible. This means roughly a ten-fold increase in both speed and density is possible with current technology, resulting in a desktop associative processor with a database of at least one million examples.

The transputer can also be improved. The type used for the IXM2 is slow compared with current RISC chips, both in the MIPS value and the communication speed. Recently, intercommunication for multiple computers employing wormhole routing has achieved much higher performance in communication than the transputers in the IXM2.

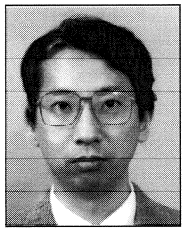
Considering these potential improvements, memory-based translation supported by an associative multicomputer system seems the most promising choice for real-time speech-to-speech translation. ■

References

1. *Massively Parallel Artificial Intelligence*, H. Kitano and J. Hendler, eds., AAAI Press/MIT Press, 1994.
2. M. Nagao, "A Framework of a Mechanical Translation between Japanese and English by Analogy Principle," *Artificial and Human Intelligence*, A. Elithorn and R. Banerji, eds., Elsevier Science Publishers, B.V., Amsterdam, 1984, pp. 173-180.
3. C. Stanfill and D. Waltz, "Toward Memory-Based Reasoning," *CACM*, ACM, New York, Vol. 29, No. 12, Dec. 1986, pp. 1,213-1,228.
4. S. Fahlman, *NETL: A System for Representing and Using Real-World Knowledge*, MIT Press, Cambridge, Mass., 1979.
5. T. Higuchi et al., "IXM2: A Parallel Associative Processor," *Proc. 18th Int'l Symp. Computer Architecture*, ACM, New York, 1991, pp. 22-31.
6. T. Ogura et al., "A 20-Kbit Associative Memory LSI for Artificial Intelligence Machines," *IEEE J. Solid-State Circuits*, Vol. 24, No. 4, 1989, pp. 1,014-1,020.
7. H. Kitano and T. Higuchi, "High-Performance Memory-Based Translation on IXM2 Massively Parallel Associative Memory Processor," *Proc. AAAI-91*, American Assn. for AI, Menlo Park, Calif., 1991, pp. 149-154.
8. K. Handa, "Flexible Semantic Network for Knowledge Representation," *J. Information Japan*, Vol. 10, No. 1, 1986, pp. 13-19.
9. H. Kitano, "ΦDM-Dialog: An Experimental Speech-to-Speech Dialog Translation System," *Computer*, Vol. 24, No. 6, June, 1991, pp. 36-50.
10. H. Kitano, *Speech-to-Speech Translation: A Massively Parallel Memory-Based Approach*, Kluwer Academic Publishers, Boston, 1994.
11. M. Sumita et al., "Example-Based Machine Translation on Massively Parallel Processors," *Proc. IJCAI-93*, Morgan Kaufmann Publishers, San Mateo, Calif., 1993, pp. 1,283-1,288.
12. O. Furuse and H. Iida, "Cooperation Between Transfer and Analysis in Example-Based Framework," *Proc. Fifteenth In'l Conf. Computational Linguistics*, Assoc. Computational Linguistics, Bernardsville, N.J., 1992, pp. 645-651.
13. K. Oi et al., "Toward Massively Parallel Spoken Language Translation," *Parallel Processing for Artificial Intelligence 2*, H. Kitano, V. Kumar, and C.B. Suttner, eds., Elsevier Science Publishers, B.V., Amsterdam, 1994, pp. 177-184.

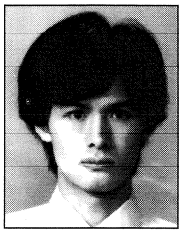


Tetsuya Higuchi heads the computational models section at Electrotechnical Laboratory. His research interests include parallel processing and associative processing. Higuchi received his BS, ME, and PhD degrees in electrical engineering in 1978, 1980, and 1984 from Keio University. He is a member of the Information Processing Society of Japan (IPSJ).



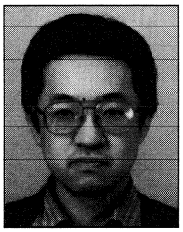
Kennichi Handa is a senior researcher in the machine inference section of Electrotechnical Laboratory. His research interests include knowledge representation, machine learning, and the multilingual computer environment. Handa received his BS, ME, and PhD degrees in electrical engineering in 1981, 1983, and 1994 from Tokyo University.

Tatsumi Furuya is a professor in the Information Science Department at Toho University. He was previously with the computational models section at the Electrotechnical Laboratory. His research interests include parallel processing, associative memory, and neural networks. Furuya received his BS and PhD degrees from Seikei University in 1971 and 1985. He is a member of the IEEE.



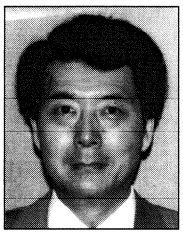
Naoto Takahashi is an MITI official at the Electrotechnical Laboratory. His interests include natural language processing and neural networks. He received his BE, ME, and PhD degrees in engineering from the University of Tsukuba in 1987, 1989, and 1992. He is a member of the IPSJ, the Japanese Society of Artificial Intelligence (JSAI), and the Association for Natural Language Processing (ANLP).

Eiichiro Sumita is a senior researcher at ATR Interpreting Telecommunications Research Laboratories. His research interests include natural language processing, information retrieval, and parallel processing. Sumita received his BE and ME degrees in computer science from the University of Electro-Communications in Tokyo in 1980 and 1982.

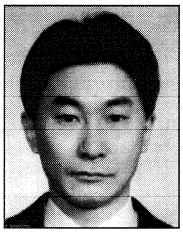


Hiroaki Kitano is a researcher at the Sony Computer Science Laboratory in Tokyo. He has been a visiting researcher at the Center for Machine Translation, Carnegie Mellon University, since 1988. His recent research interest includes evolutionary large-scale chaos, massively parallel AI, spoken language translation, evolutionary computation, and entertainment science.

Kitano graduated from the International Christian University in 1984 with a BA in physics and received a PhD in computer science from Kyoto University in 1991. In 1993, he received the Computers and Thought Award from the International Joint Conference on Artificial Intelligence. Currently, he is serving as a member of the IJCAI advisory board, the International Executive Committee for Evolutionary Computation, and an associate editor of *Evolutionary Computation.*



Hitoshi Iida joined the Electrical Communication Laboratorics of Nippon Telegraph and Telephone in 1974 and has temporarily moved to ATR Interpreting Telecommunications Research Laboratories. His interests are dialogue comprehension and speech translation. Iida received a BS and an MS in mathematics in 1972 and 1974 from Waseda University.



Kozo Oi is an ATR Interpreting Telecommunications Research Laboratories researcher on loan from Sanyo Electric Co. His research interests are natural language processing, parallel processing, and artificial intelligence.

Oi received a BE degree in electronic engineering from Doshisha University, Kyoto, Japan, in 1984.

Readers can contact Tetsuya Higuchi at higuchi@etl.go.jp.