計算機アーキテクチャ　80−13
（１９９０．１．２６）

メモリ型並列計算機構をもつ計算システムについて

高木 直史　　武永 康彦　　矢島 脩三

京都大学　工学部

組合せ計算を高速化するメモリ型並列計算機構をもつ、新しい計算システムを提案する。本報告で提案するメモリ型並列計算機構は、アドレスの部分デコードによる複数のメモリセルへの並列アクセスが可能なメモリシステムである。現在の技術で、数ギガビットの容量をもつメモリ型並列計算機構が実現可能である。この計算機構を従来の計算機に付加することにより、組合せ問題を効率よく解くことができる新しい計算システムを構築できる。本報告では、更に、この計算機を計算モデルとして定式化し、その計算能力についても考察する。

On a New Supercomputer with a Memory-Based Parallel Computation Engine

Naofumi TAKAGI,　Yasuhiko TAKENAGA　and　Shuzo YAJIMA

Faculty of Engineering, Kyoto University

Yoshidahonmachi, Sakyo-ku, Kyoto 606, Japan

We propose a new supercomputer with a memory-based parallel computation engine which accelerates computations for solving combinatorial problems drastically. The memory-based parallel computation engine is a memory system in which several memory cells can be accessed in parallel through partial address decoding. Using today's technology, we can build a memory-based engine with the capacity of several gigabits. Attaching this engine to an ordinary serial computer, we can construct a new supercomputer which solves combinatorial problems efficiently. We also formalize the proposed supercomputer as a computation model, and investigate its computational power.

1 Introduction

A lot of combinatorial problems which we often encounter in several engineering areas are NP (Nondeterministic Polynomial time) complete or NP hard. Many researchers believe that the computation time to solve a problem of this kind on a serial computer is proportional to the exponential of the size of the problem instance. Many researches are being carried out to develop supercomputers to accelerate scientific numerical computations. Vector processors are now widely used and several parallel processor machines have been developed. However, a vector processor is a kind of serial computer and requires a large amount of time to solve a hard combinatorial problem. A parallel processor machine requires impractically many processors to solve such a problem efficiently. In this report, we propose a new supercomputer with a memory-based parallel computation engine which accelerates computations for solving combinatorial problems drastically.

Recent progress in the large-scale integration of semiconductor memory is remarkable. 4Mbit DRAM LSI's are available, and 16Mbit ones are developed in laboratories. Memory systems with the capacity of several gigabytes are built for vector processors. Although memories are originally for storing data, many researchers are aware that highly parallel computation can be obtained by adding some functions to memories [1, 2]. Several kinds of functional memory such as content addressable memories have been proposed and developed [3]. In this report, we consider a new type functional memory, by which we can achieve extremely parallel computation.

The memory-based parallel computation engine proposed in this report is a memory system in which several memory cells are accessed in parallel through partial address decoding. In the engine, 2^n different processings can be carried out on 2^n memory cells through n accesses according to their addresses, and therefore, exponential speed up can be obtained. Using today's technology, we can build a memory-based computation engine with the capacity of several gigabits, and can realize extremely parallel computation. Just attaching this engine to an ordinary serial computer, we can solve combinatorial problems efficiently.

We also investigate the computational power of the proposed supercomputer. We first formalize the proposed supercomputer as a computation model called an FRAM. Then, we show that the class of problems which can be solved by the FRAM within polynomial time is equal to Δ_2^p, i.e., the class of problems which can be solved by a deterministic polynomial time bounded oracle Turing machine with an NP oracle [4]. Δ_2^p includes the classes NP and co-NP.

In the next section, a memory-based parallel computation engine is proposed. In Section 3, a new supercomputer with this engine is proposed. In Section 4, the computational power of the proposed supercomputer is considered.

2 A Memory-based Parallel Computation Engine

We propose a memory-based parallel computation engine. The engine is a memory system in which several memory cells are accessed in parallel through partial address decoding. It has three inputs, i.e., an *instruction*, an *address* and a *mask* input, and one output. It accepts three instructions, RESET, WRITE1 and SEARCH0.

In the RESET operation, 0 is set to all memory cells. In the WRITE1 and the SEARCH0 operation, an *address* and a *mask* are fed, then the memory cells whose addresses match to the fed *address* except the masked bits are accessed. (An *address* bit is mask when its cor-

responding *mask* bit is 1.) More than one cells may be accessed at the same time. In the WRITE1 operation, 1 is written to all accessed cells in parallel. In the SEARCH0 operation, the engine outputs 1 when there is at least one cell that stores 0 among the accessed cells.

A memory LSI for the engine is a modification (simplification) of a content addressable memory LSI [3]. It consists of five blocks, i.e., a memory cell array, a word operation block, a bit operation block, a search result gathering block and a control block. The memory cell array is a two-dimensional array of memory cells. Fig. 1 shows a static-type memory cell. We can design a dynamic-type memory cell by removing the flip-flop and the B_j line from the static-type one. (A mechanism for refreshing is required in a dynamic-type memory LSI.) The word and the bit operation block include a word and a bit address decoder, respectively. The address decoders are modifications of those in an ordinary memory LSI. Several WS_i (word-selected) and BS_j (bit-selected) lines may be activated at the same time in accordance with the *mask*. Fig. 2 shows the search result gathering block.

In the RESET operation, every W_i, every X_j, and every $\bar{B}_j$ line are set to 1 and every B_j line is set to 0. Then, every D_{ij} becomes 0. In the WRITE1 operation, each W_i and each X_j line are set to WS_i and BS_j respectively, and every B_j and every $\bar{B}_j$ line are set to 1 and 0 respectively. Then, the D_{ij}'s whose corresponding WS_i and BS_j are 1 (the accessed cells) become 1. In the SEARCH0 operation, every M_i line and the G line are precharged, each K_j line is set to BS_j and every W_i and every X_j line are set to 0. Then, the output becomes 1 if and only if at least one D_{ij}, whose corresponding WS_i and BS_j are 1, is 0. Otherwise, the output is 0.

Using today's technology, it seems easy to fabricate the proposed memory LSI with the capacity of 1Mbits or more. We can build a memory-based computation engine with the capacity of several gigabits by the above memory LSI's, in a similar way to building a main storage using ordinary random access memory LSI's. The major differences are that the engine requires mechanisms for partial address decoding and for gathering the search result. This engine can also be used as an ordinary random access memory system with a slight modification, and thus, can be used as an extended main storage.

3 A Supercomputer with a Memory-Based Parallel Computation Engine

Attaching the memory-based computation engine and adding the three instructions, RESET, WRITE1 and SEARCH0 to an ordinary computer, we can build a new supercomputer which solves combinatorial problems efficiently. We can obtain exponential speed up in combinatorial computations using the engine.

As an example of solving a hard combinatorial problem on the proposed supercomputer, we show a linear time algorithm to solve CNF-SAT which is a well-known NP-complete problem. CNF-SAT is a problem to answer whether there is an assignment for the variables that satisfies the given CNF (conjunctive normal form) Boolean formula. Here, we consider k-variable CNF-SAT.

[Algorithm CNF-SAT]

Step 1: Perform following (1) and (2) for each sum-term.

(1) Make an *address* and a *mask* according to each sum-term. In the *mask*, let the bits corresponding to the variables included in the sum-term be 0 and the others be 1. In the *address*, let the bits corresponding to the variables appearing as negative literals in the sum-term be 1 and the others be 0.

(2) Execute a WRITE1 instruction with the *address* and the *mask* made in (1).

Step 2:

(1) Make a *mask* such that the lowest k bits are 1 and the others are 0. Make an *address* whose bits are all 0.

(2) Execute a SEARCH0 instruction with the *address* and the *mask* made in (1).

If the output of the engine is 1, the answer is 'yes'. □

We assume that the computation engine is reset by a RESET instruction before the execution of the algorithm. In Step 1, for each given sum-term, we check the assignments for variables which do not satisfy the sum-term by writing 1 in the corresponding memory cells. For example, for a sum-term $x_5 + \bar{x}_2 + x_1$ in a five variable CNF Boolean formula, we make the *mask* 0...01100 and the *address* 0...00010, then write 1's in the four corresponding cells, i.e., cells at 0...00010, 0...00110, 0...01010 and 0...01110. After Step 1, only the memory cells, which correspond to the assignments to the variables that satisfy the given formula, keep 0. Therefore, in Step 2, we examine whether there is such a cell or not by searching 0. The computation time to solve CNF-SAT is proportional to the number of sum-terms in the given formula.

When we have a memory-based computation engine with the capacity of 4Gbits, we can solve CNF-SAT with up to 32 variables very efficiently. We can also accelerate the computation to solve CNF-SAT with more than 32 variables drastically. It is well known that many combinatorial problems are reducible to CNF-SAT. We can solve some of those problems efficiently on the proposed supercomputer.

By the above algorithm, we can only know whether there is a truth assignment or not. We can get a truth assignment by adding the following procedure to the algorithm.

[Procedure GET]

Perform following (1) through (3) while there is a 1 in the *mask*.

(1) Change the most significant 1 in the *mask* to 0.

(2) Execute SEACH0 operation.

(3) If the output of the engine is 0, change the *address* bit corresponding to the *mask* bit changed in (1) to 1. (Note that initially it is 0 from Step 2 (1).)

The assignment corresponding to the final *address* is one of the answers. □

We can get another truth assignment by executing WRITE1 operation with the final *address* and the final *mask* (all 0) and then performing Step 2 and [Procedure GET]. Repeating these operations, we can get all truth assignments.

As another example, we consider the eight queen problem. This problem is well known as a problem that requires a trial-and-error or backtracking algorithm. On the supercomputer with the memory-based engine, it can be solved straightforwardly. We put one queen on each row and express the position (the column number) of each queen by a 3-bit binary number. Namely, we express all arrangements of the eight queens by 24 bits (eight 3-bit numbers). There are unsuitable arrangements among them, and therefore, we check the unsuitable arrangements.

[Algorithm 8-QUEEN]

Step 1: For every combination of two rows, i.e, the i-th and the j-th row where $i < j$, perform the following (1) and (2).

(1) Make a *mask* in which the bits corresponding to the i-th and the j-th row are 0 and the others are 1.

(2) For every unsuitable position pair in the combination of the two rows, perform following (2-1) and (2-2). (For each position on the i-th row, there are at most three unsuitable positions on the j-th row, i.e., on the same column and on the same diagonal lines.)

(2-1) Make an *address* in which the bits correspond-

ing to the rows are the column numbers of the unsuitable position pair and the others are 0.

(2-2) Execute a WRITE1 instruction with the *address* and the *mask* made in (2-1) and (1) respectively.

Step 2:

(1) Make a *mask* such that the lowest 24 bits are 1 and the others are 0. Make an *address* whose bits are all 0.

(2) Execute [Procedure GET] to get a queens' arrangement. □

4 The Computational Power of the Supercomputer

To investigate its computational power, we first formalize the proposed supercomputer as a computation model. The model consists of a RAM (Random Access Machine), a memory-based computation engine and a search result register, as shown in Fig. 3. The RAM is one of the most common serial computation model [5]. The memory-based engine is basically the same as that we proposed in Section 2, but we assume that there are infinitely many memory cells, and therefore, assume that the *address* and the *mask* are infinitely long. The search result register is a 1-bit flag. 0 or 1 is loaded automatically into the register according to the result of a SEARCH0 instruction. We call this model an FRAM.

Table 1 shows the instruction set of the FRAM. The instructions except SEARCH0 and WRITE1 are the same as those of an ordinary RAM. The search result register can be an operand. SEARCH0 and WRITE1 are instructions which use the memory-based engine. The contents of the operands are regarded as binary numbers and 0's are assumed in the higher bits out of consideration. Namely, when n-bit binary numbers are given as operands, the 2^n memory cells at the positions from 0 through $2^n - 1$ are processed. All memory cells of the engine initially store 0. The computation time on the FRAM is defined to be the number of executed instructions.

We will show that the class of problems which can be solved by the FRAM within polynomial time is equal to Δ_2^p, i.e., the class of problems which can be solved by a deterministic polynomial time bounded oracle Turing machine with an NP oracle [4].

We first show the following lemma.

[Lemma 1]

All the sets in NP ∪ co-NP can be accepted by the FRAM within polynomial time.

Proof

All sets in NP are reducible to CNF-SAT within polynomial time by a RAM. CNF-SAT can be solved in the same way as [Algorithm CNF-SAT] shown in Section 3. Since it takes O(k) time to make an *address* and a *mask* in Step 1 (1) on the FRAM, it requires O(mk) time to solve k-variable CNF-SAT where m is the number of sum-terms. Thus, all sets in NP can be accepted within polynomial time. Since the FRAM can see that an input is rejected, it is also clear for a set in co-NP. □

Now, we show the following theorem on the computational power of the FRAM.

[Theorem 1]

The class of sets accepted by a polynomial time bounded FRAM is identically Δ_2^p.

Sketch of Proof

We show that an OTM (oracle Turing machine) with an NP oracle and the FRAM can simulate each other within polynomial time. The basic idea of this proof is that a series of instructions to the memory-based engine corresponds to an oracle for CNF-SAT.

First, we show a simulation of polynomial time bounded OTM with an NP oracle by the FRAM. Except the query state, the simulation technique is the same as

that in a simulation of a Turing machine by a RAM [5]. When the OTM enters the query state, the FRAM reduces the contents of the oracle tape to CNF-SAT. Since the oracle set of the OTM is in NP, it can be reduced to CNF-SAT within polynomial time by the FRAM without using the memory-based engine [6]. CNF-SAT can be solved within polynomial time by [Algorithm CNF-SAT]. Therefore, a polynomial time bounded OTM with an NP oracle can be simulated by the FRAM within polynomial time.

Next, conversely, we show a simulation of the FRAM by an OTM with an NP oracle. The instructions of the FRAM except WRITE1 and SEACH0 are the same as those of a RAM, and can be simulated by a deterministic Turing machine within polynomial time [5], when they do not refer to the search result register as an operand. The results of the computations on the engine are referred to only through the search result register. The OTM enters the query state only when the search result register is referred to as an operand. Whenever the FRAM executes a WRITE1 instruction or a SEARCH0 instruction, the OTM writes the contents of the *address* and the *mask* into the oracle tape. As is shown in [Algorithm CNF-SAT], they correspond to a CNF Boolean formula. Therefore, the content of the search result register is given by a CNF-SAT oracle. Therefore, the polynomial time bounded FRAM can be simulated by the OTM with a CNF-SAT oracle within polynomial time. □

5 Conclusion

We have proposed a new supercomputer with a memory-based parallel computation engine for accelerating combinatorial computations. We can solve several hard combinatorial problems very efficiently by the proposed supercomputer. We can achieve extremely parallel computation by using recent progress in the large-scale integration of semiconductor memory. The function that we added to a memory, i.e., the parallel accessibility through partial address decoding, is very interesting as a new function to be added to functional memories [7, 8].

We have also investigated the computational power of the proposed supercomputer. We have formalized the computer as a computation model and have shown that the class of problems which can be solved by the model within polynomial time is equal to Δ_2^p. The model is very interesting as a new computation model because of its simplicity and realizability.

References

[1] T. Kohonen, *Content Addressable Memories.* 2nd ed., Springer-Verlag, 1987.

[2] L. Chisvin and R. J. Duckworth, "Content-addressable and Associative Memory: Alternatives to the Ubiquitous RAM," *IEEE Computer*, vol.22, pp.51-64, Jul. 1989.

[3] T. Ogura, J. Yamada, S. Yamada and M. Tan-no, "A 20-kbit Associative Memory LSI for Artificial Intelligence Machines," *IEEE J. Solid-State Circuits*, vol.24, no.4, pp.1014-1020, Aug. 1989.

[4] L. J. Stockmeyer, "The Polynomial Time Hierarchy," *Theoretical Computer Science*, vol.3, no.1, pp.1-22, 1977.

[5] S. A. Cook and R. A. Reckhow, "Time Bounded Random Access Machines," *J. Computer and System Sciences*, vol.7, no.4, pp.354-375, 1973.

[6] S. A. Cook, "The Complexity of Theorem Proving Procedures," *Proc. 3rd Annual ACM Symposium on the Theory of Computing*, pp.151-158, 1971.

[7] M. Ohkubo, H. Yasuura, N. Takagi and S. Yajima, "A Hardware-Oriented Unification Algorithm Using a Content Addressable Memory," *Trans. Information Processing Society of Japan*, vol.28, no.9, pp.915-922, Sep. 1987 (in Japanese).

[8] H. Yasuura, T. Tsujimoto, and K. Tamaru, "Functional Memory Type Parallel Processor Architecture for Combinational Problems," *Trans. Institute of Electronics, Information and Communication Engineers*, vol.J72-A, pp222-230, Feb, 1989 (in Japanese).

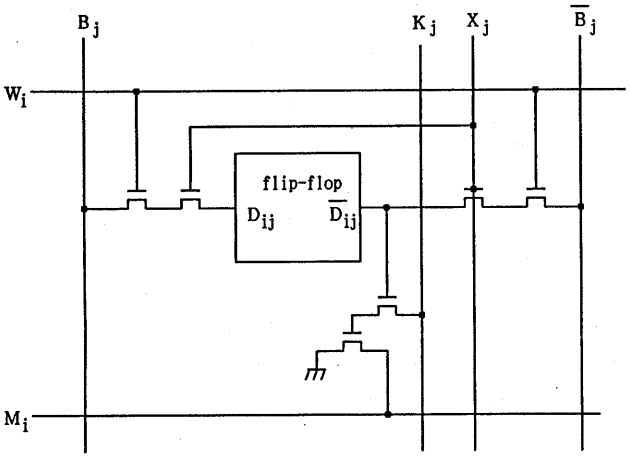

Fig. 1 A design of the memory cell (static-type).

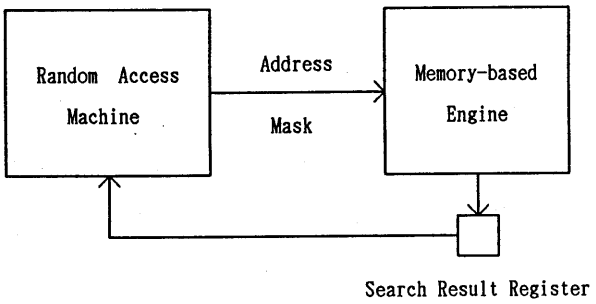

Fig. 3 A scheme of the FRAM.

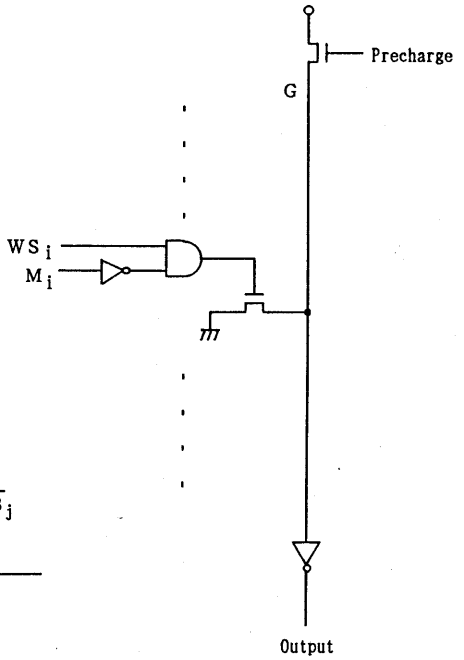

Fig. 2 A design of the search result gathering block.

Table 1. The instruction set of the FRAM.

Instructions	
LOAD	operand
STORE	operand
ADD	operand
SUB	operand
READ	operand
JUMP	label
JZERO	label
JGTZ	label
WRITE1	oper1, oper2
SEARCH0	oper1, oper2
ACCEPT	
REJECT	