アルゴリズム 21－2
（1991. 5. 29）

CAMを用いた機能メモリ型並列プロセッサ上での並列アルゴリズム

安浦寛人，渡邊章弘，左達隆吾，田丸啓吉
京都大学工学部

メモリの持つ高集積性を生かしたまま演算機能を付加した回路である機能メモリは，極めて高い並列度を持つ並列計算機構と考えることができる。機能メモリ型並列プロセッサアーキテクチャＦＭＰＰは、この点に注目し、機能メモリを並列アーキテクチャと見なす設計手法である。ここでは、代表的なＣＡＭ型のＦＭＰＰについて，その上での超並列アルゴリズムを示す．ＦＭＰＰの基本的な算術論理演算の計算能力とよく知られたデータ構造のＦＭＰＰ上での実現を示す．さらに，文字列のパターンマッチやグラフの最短経路問題に対するＦＭＰＰ上の並列アルゴリズムについて議論する．

Parallel Algorithms on Functional Memory type Parallel Processors Using a Contents-Addressable Memory

Hiroto Yasuura, Akihiro Watanabe
Ryugo Sadachi and Keikichi Tamaru

Department of Electronics, Kyoto University
Sakyo, Kyoto 606, JAPAN

Functional memories can be considered as highly parallel computation schemes with the integration density as same as RAMs or ROMs. FMPP (Functional Memory type Parallel Processor architecture) is a parallel architecture based on the feature of functional memories. In this paper, we show an FMPP architecture based on CAM (a Contents Addressable Memory) and parallel algorithms on it. First we show computation power of FMPP for the basic operations and implementations of basic data structures. As an application of them, we discuss algorithms for pattern matching and the shortest path problem.

1. Introduction

Although rapid progress of VLSI technology in this decade makes it possible to implement a computer system including more than millions of transistors, we have not yet established a highly parallel processor architecture which makes full use of the power of the VLSI technology. In fact, most parallel architectures, which are practically used or have been proposed in articles, can't directly utilize the result of increased integration which is often said "to increase four times every three years". The reason of this shortcoming is these architectures require large communication network to connect many processors with each other. Several parallel architectures suitable for VLSI implementation have already been proposed, such as, systolic and cellular array architectures, but there are some problems on integration of them into general purpose computer systems.

We have proposed a parallel processor architecture called FMPP (Functional Memory type Parallel Processor architecture)[1][2]. The main idea of FMPP architecture is to add some computation power to inside of memory circuits and to realize a highly parallel computation in the memory circuits. Since FMPP uses the LSI memory circuit structure, we can directly reflect the increase of integration density of memories to the increase of the number of processors. FMPP is an SIMD (Single Instruction stream Multiple Data stream) type architecture and is attached to a general purpose computer as a part of its main memory space. Communication between the host computer and FMPP is just an ordinary memory access. In FMPP, every word or group of words has a computation power and it works as a processor. So we can realize the number of processors as large as the size of the main memory space.

A content-addressable Memory (CAM) is a typical functional memory. Each word of CAM can be considered as a one-bit processor [1]. Recently several large scale CAM chips have been developed [3]-[6]. Several theoretical researches have been done on the computational power of CAM [1][2][7]-[9] and several special purpose systems are being implemented using these CAM chips[10]-[15]. Takagi etal proposed a new computation model based on CAM called FRAM (Random Access Machine with a Functional Memory) [9]. In Waseda University, LSI routing machine [10] is implemented using 4-Kbit CAM[3] developed by NTT. This 4-Kbit CAM chip is also used in the implementations of a Prolog machine ASCA at NTT[11] and a semantic network machine at ETL [12]. NEC developed a string-search chip which contains 8K-bit CAM [13][14]. These researches suggest that CAM is a very strong candidate of a basic circuit structure constructing an FMPP architecture.

In this paper, we present an architecture of FMPP and summarize the cost of computation realizing basic operations and data structures. As examples of parallel algorithms on FMPP, we show highly parallel algorithms for pattern matching and the shortest path problem. We show that we can easily parallelize algorithms designed for serial computers to FMPP.

2. FMPP Architecture

The block diagram of the FMPP is shown in Fig.1. The FMPP has four kinds of input/output ports, an address bus, a data bus, command inputs and a detection line of selected words. The address bus is an n-bit bus and connected with the address register AR. The data bus is an m-bit bus and connected with the data register DR. An m-bit register, the mask register MR, is attached to the DR. The content of DR is masked by MR and sent into the memory cell array. The cell array contains 2^n m-bit words. Each word contains an m-bit RAM field and flags. The RAM fields can be accessed as a usual memory. In matching mode, the contents of words are compared with data in DR masked by MR (called a reference word). If the content of a word is equivalent to the reference word, a matched signal MS is sent to the flag part and a specified logical operation is applied to MS and the content of the S flag. We can specify a logical operation among AND, OR and THRU(through). The result of the selected operation between MS and the previous content of flag S is assigned into S. G flag is used as a garbage flag indicating garbage words. The detection line outputs 1, when there exists at least one word whose S flag is 1, called a selected word. The multiple response resolver resolves conflict when two or more words are selected simultaneously.

The following instructions can be used in the FMPP.

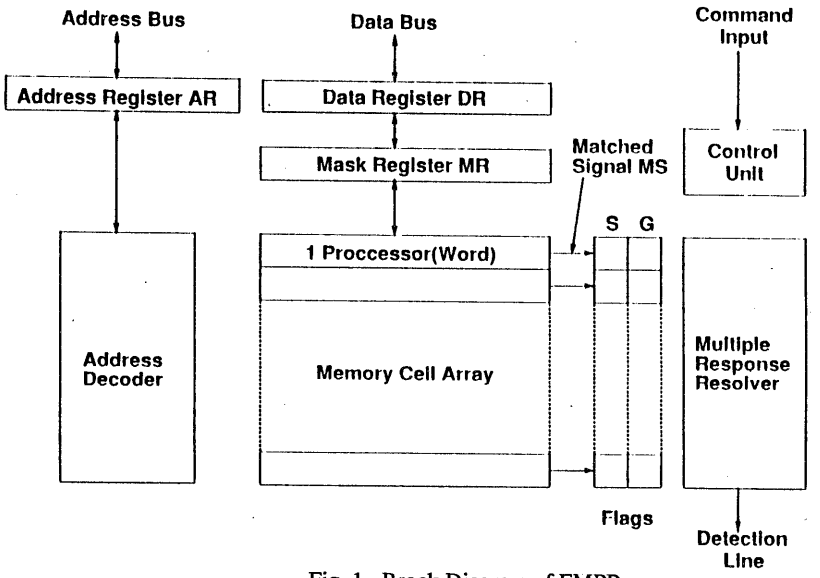

Fig. 1 Brock Diagram of FMPP

REF <function, data> : In each word, compare the content of the RAM field with the reference word specified by the data in DR masked by MR. If they are matched, accumulate the result MS into flag S after computing the specified logic operation with the previous value of S.

READS: Read the content of the RAM field and address of the selected word (S=1). If two or more words are selected, one of them are chosen by the multiple response resolver.

WRITES <data>:Write the input data masked by MR into all selected words in parallel.

READA <address>: Ordinary read operation by the address.

WRITEA <address, data>: Ordinary write operation by address.

MASKSET <data>: Set a mask data into MR.

SHIFT <direction>: Shift up/down the contents of S flags.

Each word is a processor with a 1-bit logical unit, *m*-bit register and 2 flag registers. Processors are connected by a bus corresponding to bit lines of memory. All processors execute the same instruction which is given from the command input terminal.

3. Basic Operations and Data Structures on FMPP

We assume that FMPP is embedded into a main memory space of a host computer. So each word of FMPP can be accessed by ordinary memory access and programs on the host computer can access the FMPP as a part of the main memory space. FMPP is also a parallel processors operating as an SIMD machine. Each word is a processor with 1-bit ALU and execute the same operation simultaneously. From the standpoint of algorithm design, we can see the FMPP from two different angles. The first one is to consider FMPP as an SIMD parallel processors. For this view, the computation power of FMPP on basic arithmetic and logical operations is important. The second one is that the FMPP is a memory with computation capability. From this viewpoint, implementations of basic data structures on FMPP are imoprtant.

3.1 Basic operations on FMPP

Here we summarize algorithms for basic logic and arithmetic operations on FMPP. For simplicity, we assume all data are unsigned k-bit integer. In FMPP we can freely define the positions of each data but here we explain under the following assumption. The left most bit of each word is the tag bit. Operations are effective only for the word whose tag bit is 1. The next k bits represent the first data (data 1) and k bits of the second operand (data 2) follow them. The last several bits are used as a working area. So the word is divided into four fields such as < tag, data 1, data 2, work-bits>.

Now we show an FMPP program for addition. In the following programs we assume k=3.

```
Add: data 2 <- data1 + data 2.
MASKSET   <1,001,001,10>    /*addition of LSB*/
REF thru  <1,--1,--1,0->
WRITES    <1,--1,--0,1->
REF thru  <1,--1,--0,0->
WRITES    <1,--1,--1,0->
MASKSET   <1,010,010,10>    /*addition of the second LSB*/
REF thru  <1,-0-,-0-,1->
WRITES    <1,-0-,-1-,0->
REF thru  <1,-0-,-1-,1->
WRITES    <1,-0-,-0-,1->
REF thru  <1,-1-,-1-,0->
WRITES    <1,-1-,-0-,1->
REF thru  <1,-1-,-0-,0->
WRITES    <1,-1-,-1-,0->
MASKSET   <1,100,100,10>    /*addition of the MSB*/
REF thru  <1,0--,0--,1->
WRITES    <1,0--,1--,0->
REF thru  <1,0--,1--,1->
WRITES    <1,0--,0--,1->
REF thru  <1,1--,1--,0->
WRITES    <1,1--,0--,1->
REF thru  <1,1--,0--,0->
WRITES    <1,1--,1--,0->
```

The addition and subtraction are done in bit-serial manner. The first bit of the work field is used for storing the carry. The number of instruction is $9k$-4. It is easy to extend the algorithm to treat negative integers represented in the 2's complement notation.

We summarize the number of instruction steps of basic operations required for k-bit data in Table 1. Since most operations are computed in bit serial manner, the computation time mostly depends on the length of data and not on the number of data.

3.2 Basic Data Structures on FMPP

Using the computation power of FMPP, we can realize popular data structures with reduced computation cost of their operations. Here we show three examples of implementations of basic data structures[17].

1) Priority Queue

Let's consider a priority queue with n elements. The basic operations of the priority queue are insert(x) and deletemax. Insert(x) is an operation for addig x into the queue. Deletemax means finding a maximal element in the queue and delete it from the queue. It is well known that the priority queue can be effectively implemented by a heap.

Operations	The number of steps
Data Transfer (*out* -> *data 1*)	1
Data Transfer (*data 1* -> *data 2*)	$3k$+4
Data Transfer (*word i* -> *word i+j*)	(j+1)k
Addition (*data 1* + *out* -> *data 1*)	$5k$
Addition (*data 1* + *data 2* -> *data 2*)	$9k$-4
Increment (*data 1* + 1 -> *data 1*)	$3k$
Comparison (Compare *data 1* with *out*)	$2k$+1
Comparison (Compare *data 1* with *data 2*)	$5k$+3
Find maximum values of (*data 1*)'s	$3k$+1
Logical Operations	
k-input AND, NAND OR and NOR	7
k-input EXOR and EXNOR	8

Table 1. The number of instruction steps for basic operations

The computation cost of the operations are both O(log n). On FMPP we can easily implement the priority queue by a simple way. We can store the element in each word. For managing empty words, we use one bit of each word as a tag indicating that the word is including the queue. Insert(x) is implemented by REF and READS operations to find the address of an empty word and WRTEA to write x into the word. The computation time is O(1). Deletemax is the same operation of Find Maximum Values shown in the previous subsection. Since the operation is done in the bit-serial manner, O(k) steps are required for deletemax, where k is the bit length of x.

2) Dictionary

A dictionary is a data structure with operations member(x), insert(x) and delete(x). This is a basic data structure to represent a set S. Member(x) examines whether x is in S or not. Insert(x) inserts x into S and delete(x) eliminates x from S if x is in S. For efficient implementations of the dictionary, several data structures called balanced binary trees are developed. In the most of them, all three operations need O(log n) time. By a naive implementation on FMPP of the dictionary, we can achieve constant time realization of the three operations. As same as the priority queue, we store each element of S in one word with a tag bit indicating whether the element is in S or not. Member(x) is done by REF and READS operations. Insert(x) is easily to be implemented by REF and WRITEA. Delelte(x) is done by REF and WRITES.

3)Data Structure for Union-Find [16]

This data structure is useful to treat partitions of a set. Let S be a set with n elements and S_1, S_2, $\cdots$, S_m be subsets of S mutualy distinct. We define two operations, union (S_i, S_j) and find (x). Union (S_i, S_j) merges S_j into S_i and find(x) returns the name of subset which includes the element x. We also implement these two operations in constant time on FMPP. In [16], we derived a simple algorithm for unification using the data structure on CAM.

4. Parallel Algorithms on FMPP

What kinds of problems are suitable for the computation on FMPP? FMPP is an SIMD machine with large number of simple processors each of which has ability of bit-serial operation and poor communication power. Thus the problems should be solved by many identical simple processes which can be computed independently without

frequent communications. Since each processor has a small computation power but the number of processors can be very large, the problems requiring a large number of simple processes are suitable for FMPP.

We developed several algorithms on FMPP for sorting, pattern matching, finding the shortest/longest path on graphs, logic simulation, test pattern generation for combinational circuits and combinatorial optimization problems[1]. Here we show algorithms for pattern matching and the shortest path problem.

To evaluate the performance of FMPP, we use a simulator on a work station and a prototype machine with 4-Kbit CAM chips [3]. The prototype machine was developed by NTT as a Prolog machine [11], which is not originally designed as an FMPP machine. We developed microprograms to emulate FMPP on the prototype machine. The machine contains totally 4K words (32bits/word) CAM, namely it is the FMPP with 4096 processors, and its cycle time is 250ns. Since the microprogram instruction set of the prototype machine is not tuned to FMPP architecture, instructions of FMPP defined in Section 2 require two or more cycles on the machine. In the simulation, we assume an ideal FMPP machine which executes each instruction in a 200ns machine cycle.

4.1 Pattern Matching

Consider a pattern matching problem on a linear text T containing t characters. The length of pattern P searched in the text is p. The idea of the pattern matching algorithm is shown in Fig. 2. Each character of the text is stored in each word of the FMPP in their order. First we initialize all S flags on. The first character in the pattern broadcasted from the host computer to FMPP and comparisons of the broadcasted character with stored characters are done in parallel by REF command. The results of the comparisons are accumulated in S flag and they are shifted down by SHIFT command . Then the next character of the pattern is broadcasted and comparisons, accumulations and shift are done in parallel. After broadcasting the final character of the pattern, the selected word indicates the tail of the patterns in the text. Only two FMPP instructions are required for every cycle.

In this algorithm, we don't move the text data which is much longer than patterns in the many practical applications. So the algorithm reduces the data transfer as small as possible. If we have enough words of FMPP to store the whole text, the computation time is linearly proportional to p, the length of the pattern, but independent of t, the length of the text. Since all sequential algorithms of the pattern matching requires at least linear time of t, it is expected that FMPP achieves quite a large speed-up for the long texts In Tabel 2, we summarize the performance of pattern matching on FMPP. In our prototype machine we have already achieve more than 60 times speed-up.

We can extend the algorithm to the cases in which patterns include wild characters or are defined by regular expressions. Depending on the complexity of the patterns, the information shifted between words are increasing. To treat complicated matching efficiently we need to introduce stronger communication mechanisim in FMPP.

Length of text	Sequential Program (25 MIPS CPU)(ms)	Simulation of FMPP (ms)	Emulation on the prototype machine (ms)
256	76.8	4	19.8
1K	307.2	4	19.8
4K	1,228.8	4	19.8
16K	4,915.2	4	-
1M	314,572.8	4	-

Table 2. Performance Analysis for Pattern Matching (p=10)

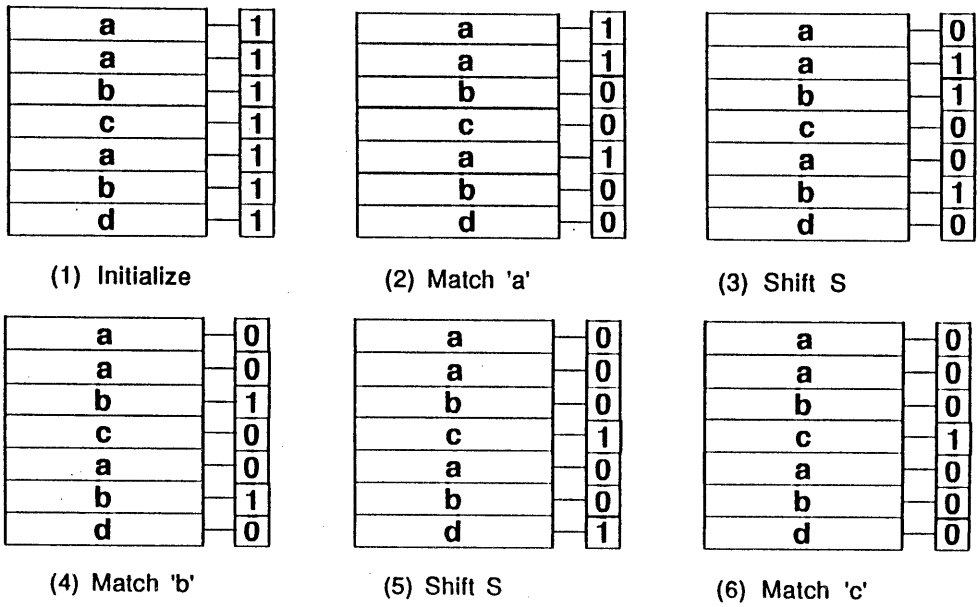

Fig. 2 Pattern Match on FMPP

4.2 Shortest Path Problem

For the shortest path problem on a directed graph with n nodes and m edges, it is well known that Dijkstra's algorithm works $O(m \log n)$ time using a heap[17]. The factor of $\log n$ is comming from the basic operations for the priority queue implemented by a heap. So using the data structure on FMPP, we can reduce time for the operations into $O(1)$ and $O(k)$ as shown in 3.2. Then we get a parallel algorithm for the shortest path problem with $O(mk)$ time complexity.

Warshall-Floyd algorithm is a dinamic programming algorithm for the shortest path problem [17]. The time and space complexity of the algorithm is $O(n^3)$ and $O(n^2)$, respectively. On FMPP we can compute the updated values of each iteration of the algorithm in parallel. The time complexity is reduced into $O(kn^2)$, where k is a bit length of data representation. We can use this parallelization techniques for other dinamic programming algorithms.

If we have enough space to store all paths from the source to the sink, we can use a parallel exhaustive search [1]. Computation of the distance on each path is done independently each other, we can compute the distance in parallel in each word. The time complexity is $O(mk)$ and the space complexity is $O(2^m)$.

5. Conclusion

We proposed an FMPP architecture based on CAM and presented parallel algorithms for basic operations and data structures. In the parallel algorithm for pattern matching we show a processing with the smallest data transfer. It is a very promising approach to construct a highly parallel algorithm by the current VLSI technology. In the algorithms for the shotest path problem, we show that FMPP is useful to parallelize well-known algorithm design methods

such as the greedy algorithm, dynamic programming and exhaustive search.

Acknowledgement

The authors express their sincere appreciation to Dr.Sakai, Dr.Ogura and Mr.Naganuma of NTT who lent us the prototype machine and CAM chips. This work is partly supported by Grant of Scientific Researches 02452160.

References

[1] H.Yasuura, T.Tsujimoto and K.Tamaru: "Parallel Exhaustive Search for Several NP-Complete Problems Using Content Addressable Memory", Proc. of 1988 IEEE Int. Symp. on Circuits and Systems, pp.333-336, June 1988.

[2] H.Yasuura, T.Tsujimoto and K.Tamaru: "Functional Memry Type Parallel Processor Architecture for Combinational Problems", Trans. of IECEJ, vol.J72-A, no.2, pp.222-230, 1989 (in Japanese).

[3] T.Ogura, S. Yamada and T.Nikaido: "4-Kbit Associative Memory LSI", IEEE J. of Solid-State Circuits, vol. SC-20, no.6, pp.1277-1282, 1985.

[4] H.Kadota, J.Miyake, Y.Nishimichi, H.Kubo and K.Kagawa: "An 8-Kbit Content-Addressable and Reentrant Memory", IEEE J. of Solid-State Circuits, vol. SC-20, no.5, pp.951-957, 1985.

[5] T.Ogura, S.Yamada and J.Yamada.: "A 20Kb CMOS Associative Memory LSI for Artificial Intelligence Machine", Proc. of ICCD'86, pp.574-577, 1986.

[6] T.Hamamoto, T.Yamagata, M.Mihara, T.Kobayashi and M.Yamada:"A 288kbit Fully Parallel Contents Addressable Memory Using Stacked Capacitor Cell Structure", 1991 Spring Convension Rec. of IEICE, C-649, vol.5, p240, 1991(in Japanese).

[7] T.Kohonen: "Content-Addressable Memories", 2nd ed., Springer Series in Information Science, Springer-Verlag, 1987.

[8] I.D.Scherson and S.Ilgen:"A Reconfigurable Fully Parallel Assosiative Processor", Journal of Parallel and Distributed Computing, vol.6, pp.69-89 (1989).

[9] N. Takagi, Y.Takenaga and S. Yajima:"A Memory-Type Parallel Computation Model and Its Computational Power -Yet Another Approach to Supercomputiong-", Trans of IPSJ, vol.31, no.11, pp.1565-1571, 1990(in Japanese).

[10] K.Suzuki, T.Ohtsuki and M.Sato: "A Gridless Router: Software and Hardware Implementation", VLSI87, pp.153-163, Aug. 1987.

[11] J.Naganuma, T.Ogura, S.Yamada and T.Kimura: "High-Speed CAM-Based Architecture for a Prolog Machine (ASCA)", IEEE Trans. on Computers, vol.37, no.11, pp.1375-1383, 1988.

[12] T.Higuchi, T.Furuya, H.Kusumoto, K.Handa and A.Kokubu: "The Prototype of a Semantic Network Machine IXM", Proc. of 1989 International Conf. on Parallel Processing, vol.1, pp.217-224, 1989.

[13] H.Yamada, M.Hirata, H.Nagai and K.Takahashi: "A High-Speed String Search Engine", IEEE J. of Solid-State Circuits, vol. SC-22, no.5, pp.829-834, 1987.

[14] M.Hirata, H.Yamada, H.Nagai and K.Takahashi: "A Versatile Data String-Search VLSI", IEEE J. of Solid-State Circuits, vol. SC-23, no.2, pp.329-335, 1988.

[15] N.Ishiura and S.Yajima: "Linear Time Fault Simulation Using a Content Addressable Memory", 1991 Spring Convension Rec. of IEICE, SA-3-9, vol.1, pp401-402, 1991(in Japanese).

[16] M.Ohkubo, H.Yasuura, N.Takagi and S.Yajima: "A Hardware Oriented Unification Algorithm Using Content Addressable Memory", Trans of IPSJ, vol.28, no.9, pp.915-922, 1987(in Japanese).

[17] T.Hirata: "Algorithms and Data Structure", MORIKITA-SHUPPAN, 1990 (in Japanese).