EIS-TR-76-3

AN ANALYSIS OF PROGRAM BEHAVIOR

by

Tomoo Nakamura

Hajime Kitagawa

Hiroshi Hagiwara

July 7,1976

INSTITUTE
OF
ELECTRONICS AND INFORMATION SCIENCE

UNIVERSITY OF TSUKUBA

EIS-TR-76-3

AN ANALYSIS OF PROGRAM BEHAVIOR*

Tomoo Nakamura**

Hajime Kitagawa***

Hiroshi Hagiwara****

July 7, 1976

Abstract

As computer systems have increased their complexity, it has become very important to analyze the system efficiency. In the analysis of computer systems, it is fairly of use to have the knowledge of dynamic behavior of programs under execution. This paper is an empirical study of program behavior. We prepared an address tracer which recorded the memory reference patterns of programs and got the traced data for several programs. By using these data we analyzed some kinds of program behavior: run length; branch distance; memory demand; stack distance frequency for typical replacement algorithms (LRU and OPT); instruction buffering. In this paper we present the results of the measurement and analysis.

*This report is a revised edition of our paper appeared in Journal of Information Processing Society of Japan,Vol.15,No.1(1974).
**Institute of Electronics and Information Science, University of Tsukuba, Niihari-Gun, Ibaraki 300-31, Japan
***Data Processing Center, Kyoto University, Sakyo-ku, Kyoto 606, Japan
****Department of Information Science, Kyoto University, Sakyo-ku, Kyoto 606, Japan

I. Introduction

As computer systems have increased their complexity, it has become very important to evaluate the system performance or analyze the system efficiency. In the analysis of computer systems, there are many cases in which it is of use to have the knowledge of the dynamic behavior of programs under execution. Typical examples of the cases are the analysis of paged memory systems,[1,2,4] buffer memory systems,[7,8] interleaved memory systems,[6] and segmentationed memory systems.

The information about program behavior can be obtained by measuring and analyzing the memory reference patterns of actual programs under execution. Then, we prepared an address tracer which recorded the memory reference patterns of user programs, and got the traced data for several programs. By using these data we analyzed some kinds of program behavior: run length; branch distance; memory demand; stack distance frequency[5] for typical replacement algorithms (LRU and OPT); instruction buffering. In this paper we present the results of the measurement and analysis.

II. Address tracing

To measure the dynamic behavior of programs and to make the input data for memory reference simulation,[8] we have prepared the address tracer by which memory referencing patterns of user programs during execution can be recorded. The address tracer runs under the FACOM 230-60 batch operating systems. The tracer is linkage-edited with the object programs being traced, and only user program mode instructions of the object program are recorded during execution.*

The unit of data which is to be sequentially recorded on magnetic tape consists of,

(1) address of the instruction which references memory or causes transfer of control (jump or monitor-call), and its instruction code,

(2) address of the referenced memory (including all the referenced addresses in the case of indirect addressing),

(3) type of the reference; i.e., load/store as an operand or instruction fetch.

We selected several FORTRAN programs as the benchmarks and got the traced data while they were being compiled and being executed. (See Table 1.)

---

* The moderate slow-down rate of execution time under tracing is obtained; 50:1 to 80:1. The address tracer is written in FASP assembler language and its size is about 700 words.

Table 1

trace data	compile /execute	comp. type	prog. size (kilo-w)	f (x10⁴)	P_R %	P_W %	P_X %	
Differential equation RKG	C	FORTRAN-C non-optimize	*38	22	20.0	13.0	67.0	a1
	E		9	47	17.1	14.8	68.1	a2
EIGEN VALUE 10x10 symmetric matrix Jacobi	C		*38	83	21.8	11.2	67.0	b1
	E		14	79	31.0	10.5	58.5	b2
	C	FORTRAN-D optimize	*38	177	21.7	11.3	67.0	c1
	E		14	39	25.8	11.6	62.6	c2
Sorting	C		*38	38	21.0	12.8	66.2	d1
	E		9	**50	22.3	18.8	58.9	d2
BM Simulator	E		16	**50	26.7	13.8	59.5	e2

f : frequency of data reference by CPU,

P_R : frequency of operand fetch / f,

P_W : frequency of operand store / f,

P_X : frequency of instruction fetch / f,

P_F : frequency of instruction or operand fetch / f,

and $P_R + P_W + P_X = 1$, $P_F = 1 - P_W$.

* the amount of core memory used by the overlayed program

** Tracing was stopped when f = 500,000.

III. Measurements of program behavior

We made statistical analysis by using the traced data and characterized the dynamic behavior of programs in the following points of view.

3.1 Run length

A run length is defined by Sisson and Flynn[6] as "the number of addresses increasing in value by one over the previous address in an addressing pattern".

We define S as the distribution of run lengths such that: $S([m,n])$ is the probability that s is between m and n ($m < n$), where s is the number of instructions which are executed sequentially.

Some results of run length distribution are indicated in Fig.1. Fig.1 shows that s is less than about 5 with high probability for all the measured programs.

3.2 Branch distance

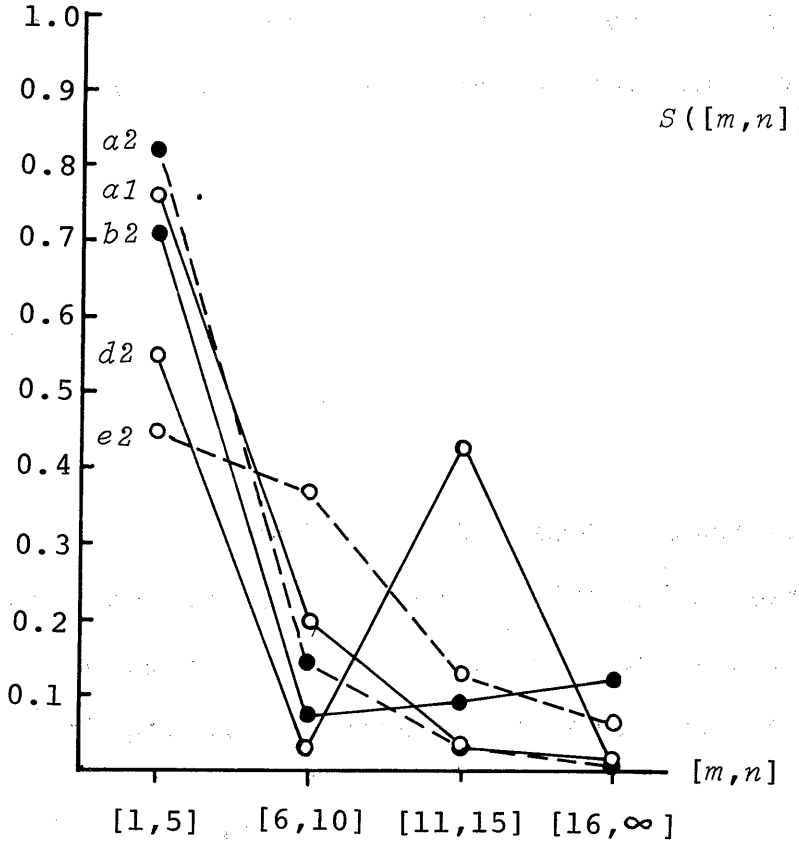
The branch distance distribution B is defined as follows: $B([m,n])$ is the conditional probability that b is between m and n ($m < n$), where b is the branch distance, i.e., $a_2 - a_1$, between a branch instruction (address a_1) and the next instruction (address a_2) when branch is executed. ($b \neq 0,1$)

The branch distance distribution as well as the run length distribution may indicate the characteristics of program-structures such as the size and occurence frequency of loops.

Some results of branch distance distribution are indicated in Fig.2. Fig.2 shows that the probability $Pr(2 \leq b < 10)$ is high for all the measured programs.

$S([m,n])$

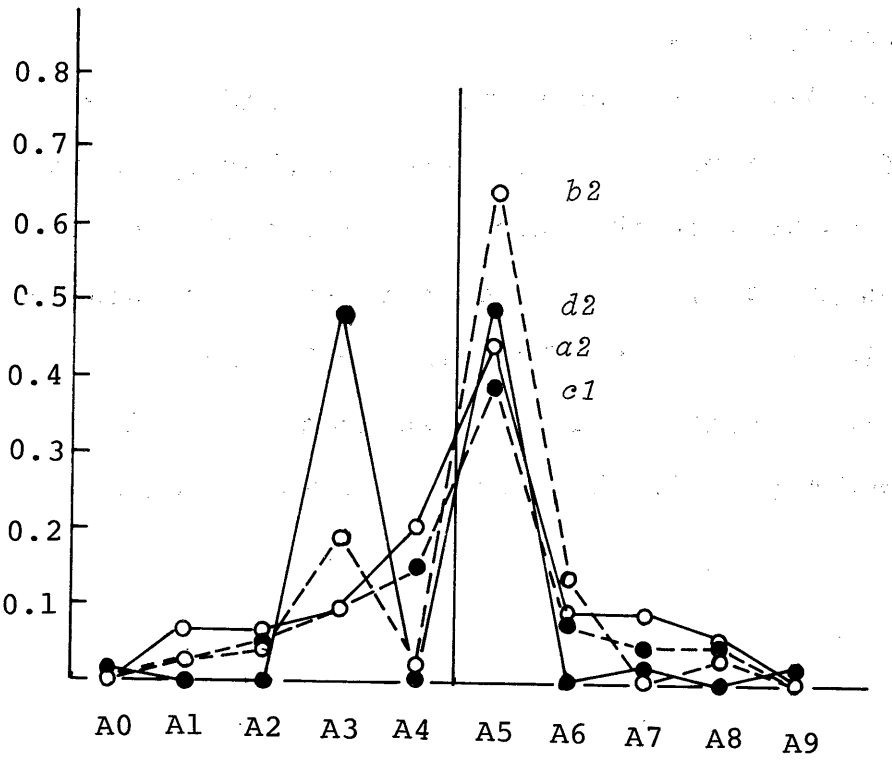
Fig.1 Run length



$$S([m,n]) = Pr(\ m \leqq s \leqq n\ )$$

$B([m,n])$

Fig.2 Branch distance



A0; $[-\infty, -10^4]$
A1; $[-10^4, -10^3]$
A2; $[-10^3, -10^2]$
A3; $[-10^2, -10]$
A4; $[-10, 0]$
A5; $[2, 10]$
A6; $[10, 10^2]$
A7; $[10^2, 10^3]$
A8; $[10^3, 10^4]$
A9; $[10^4, \infty]$

$$B([m,n]) = Pr(\ m \leqq b < n\ ) \ / \ Pr(\ b \neq 0 \wedge b \neq 1)$$

In Fig.2, the proportions of the number of branch instructions executed to the total dynamic instruction steps are as follows:

a2 : *28.8%*,

b2 : *32.6%*,

d2 : *14.2%*,

e2 : *14.2%*.

3.3 Memory demand

We define the amount of memory demand D as follows:
$D(r;m)$ is the average number of distinct blocks that are required by r successive memory references during execution of a program, where the memory space is divided into blocks each consisting of m contiguous addresses.

$D(r;m)$ corresponds to the average working set size[2] of each program, and r corresponds to the window size.[2] $D(r;m)$ is monotonically nondecreasing in r.

Fig.3 shows the results of the case where $m = 8$ and $r = 1 \sim 15000$. In the case of *a2*, *b2* or *e2*, $D(r;m)$ goes up sharply to the 'saturation points'. However, in the case of *c1* (one of typical patterns in compilation phase), $D(r;m)$ does not become 'saturated' while $r \leqq 15000$. As far as the data shown in Fig.3, the amount of memory required by *15000* references are less than *2000*words for all the programs and especially less than *1000* words for *b2*, *e2* and *d2*.

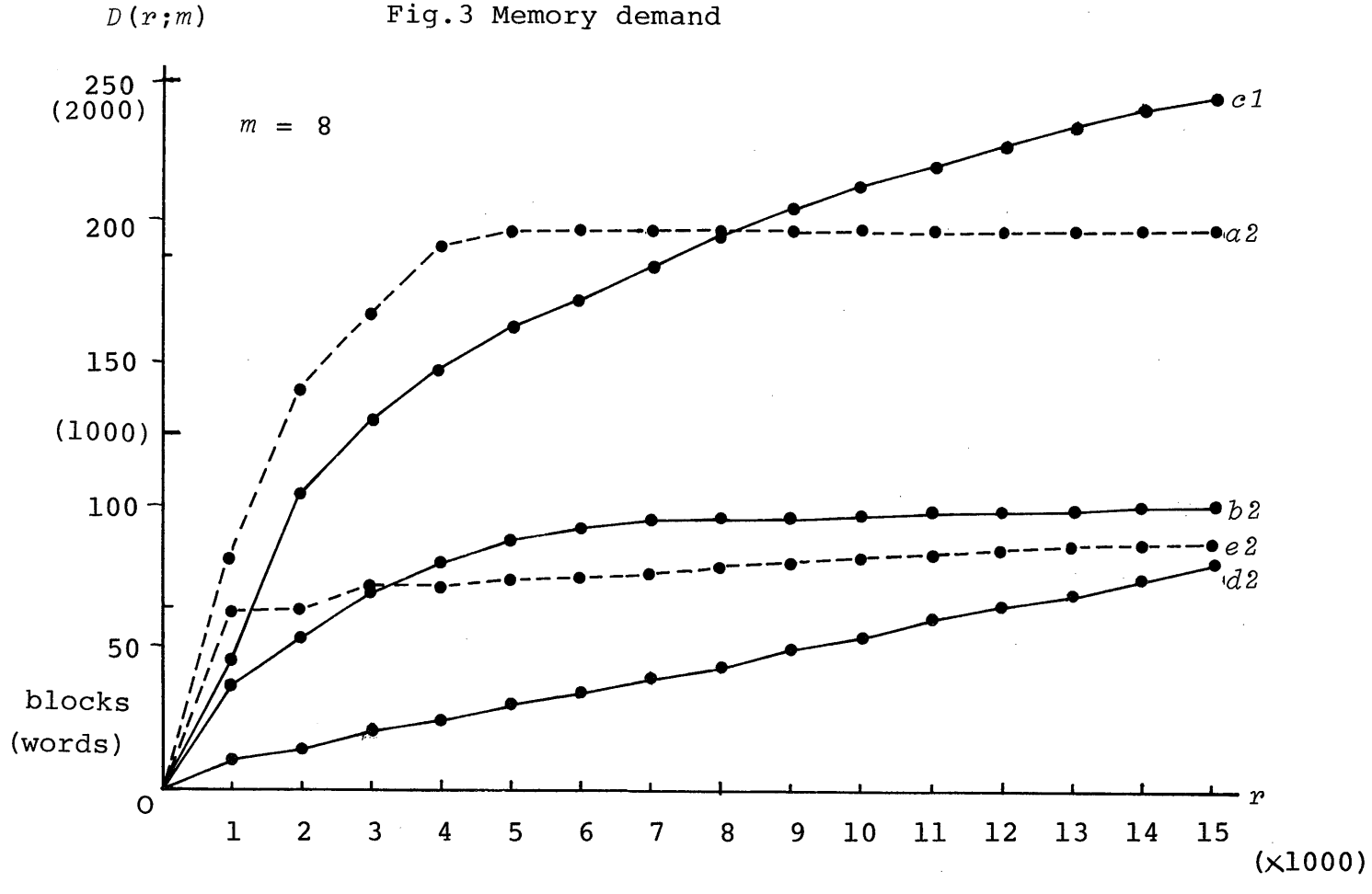


Fig.3 Memory demand

3.4 LRU stack distance frequency

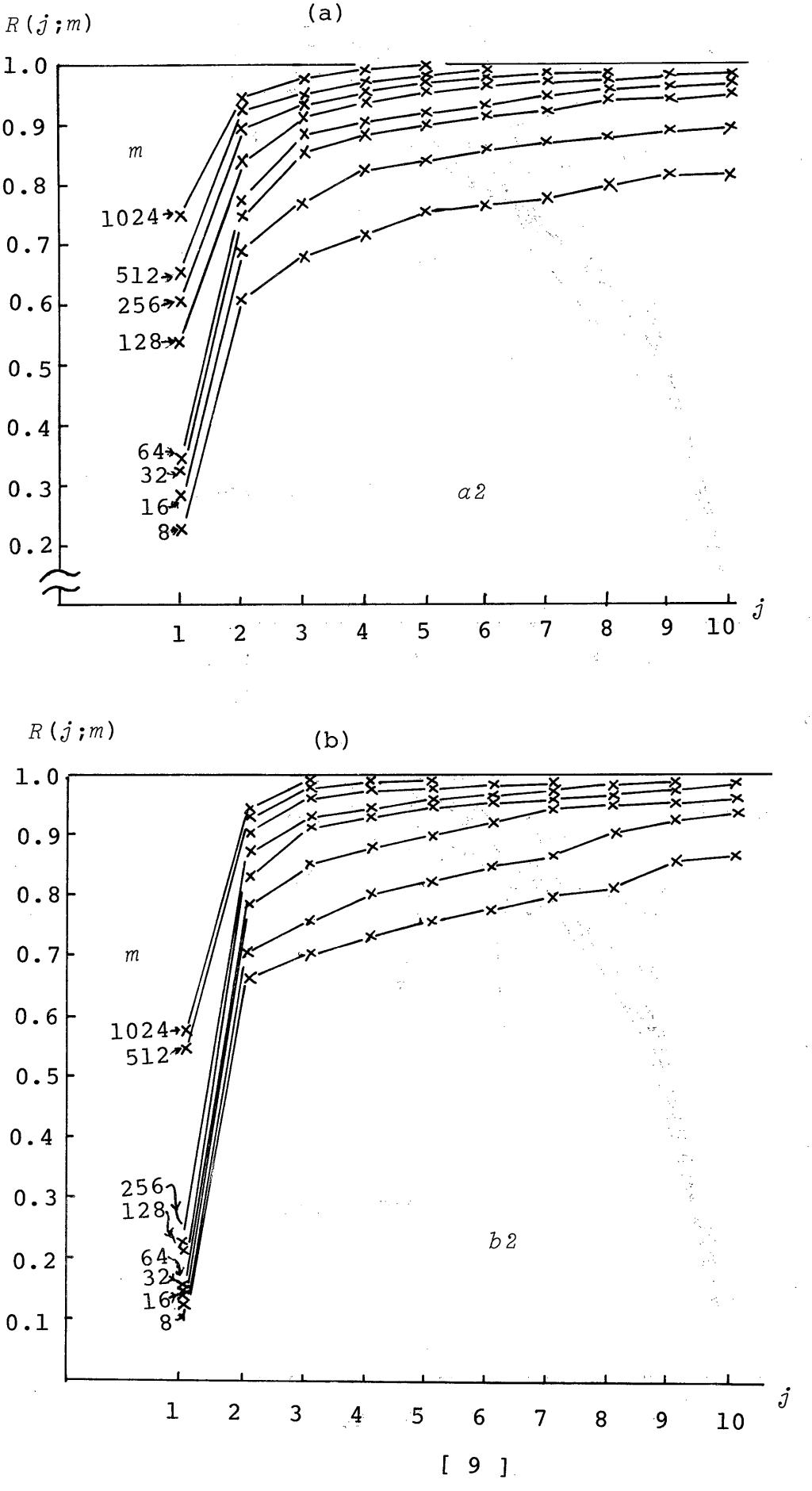
The LRU (least recently used) stack distance frequency defined by Mattson et al.[5] is a measure that indicates program's locality of memory reference, and can be used to obtain the access rates to buffer memory or main memory in the simulation of buffer memory systems or paged memory systems.

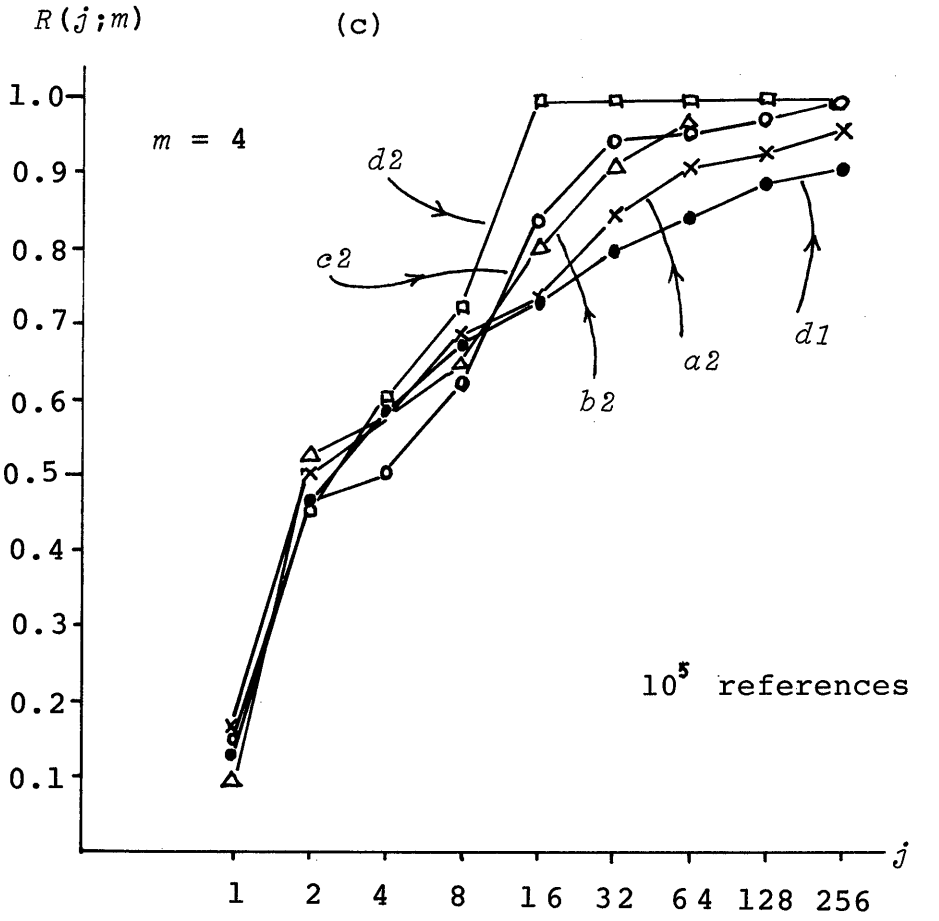
We define $R(j;m)$ as the cumulative distribution of LRU stack distance frequency such that:
$R(j;m)$ is the cumulative distribution of the relative frequency that the address of memory referenced is contained in the j "most recently used" distinct blocks if the memory is divided into blocks of m contiguous addresses.

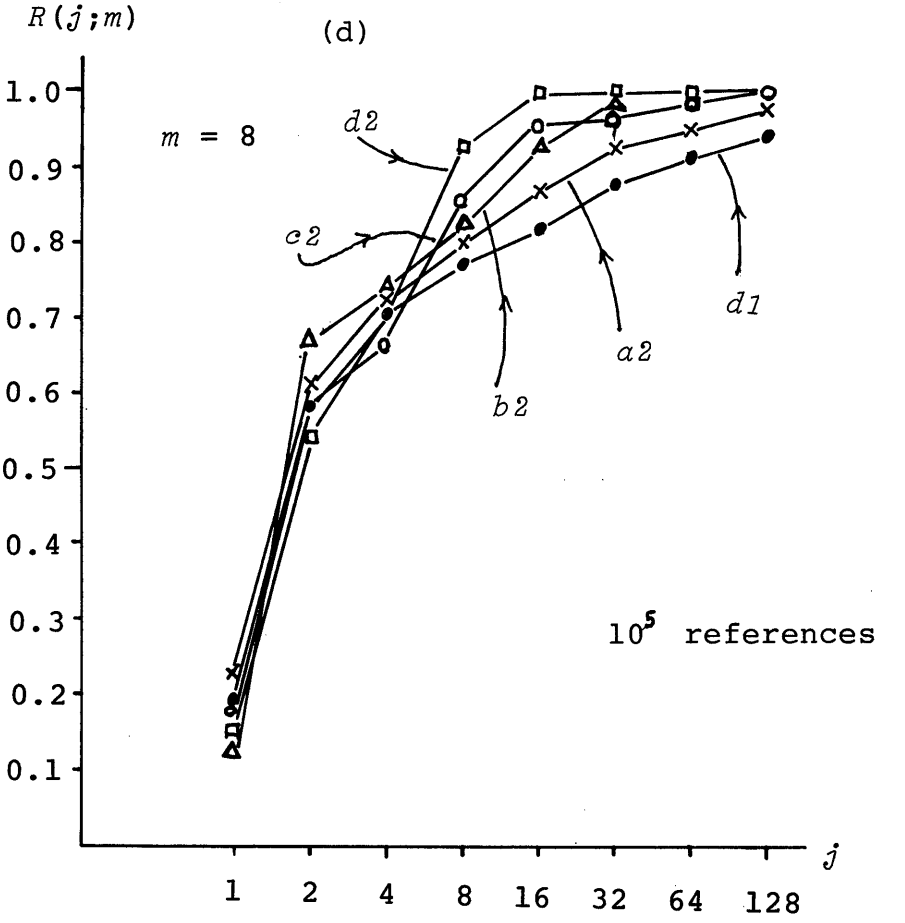
Some results are shown in Fig.4(a)～4(f). In Fig.4(a) and 4(b), the separate curves represent different block sizes. On the other hand, in Fig.4(c)～4(e), the separate curves represent different traced data. It is remarkable that the rapid ascent is observed by increasing j from *1* to *2* in any case. This is explained by the fact that memory reference strings consist of the two streams; i.e., the instruction stream and the operand (data) stream.

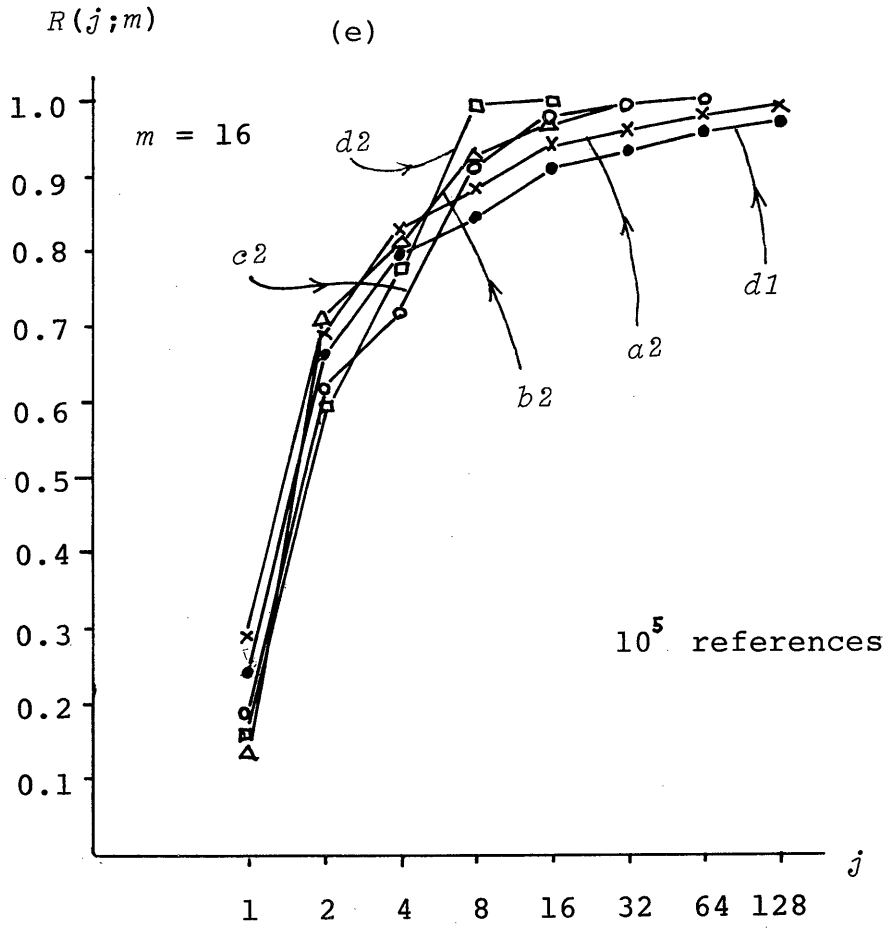
Fig.4(c), 4(d) or 4(e) directly leads to the variation in the percentage of accesses to buffer memory (block size m) with buffer memory capacity $m \times j$ in the simulation of buffer memory system which is controlled by fully associative mapping and LRU replacement algorithm. From these figures, one can see that the access rates to buffer memory made up of *2* blocks of *4* word-block is about *50 %*, and that the difference of the access rates among

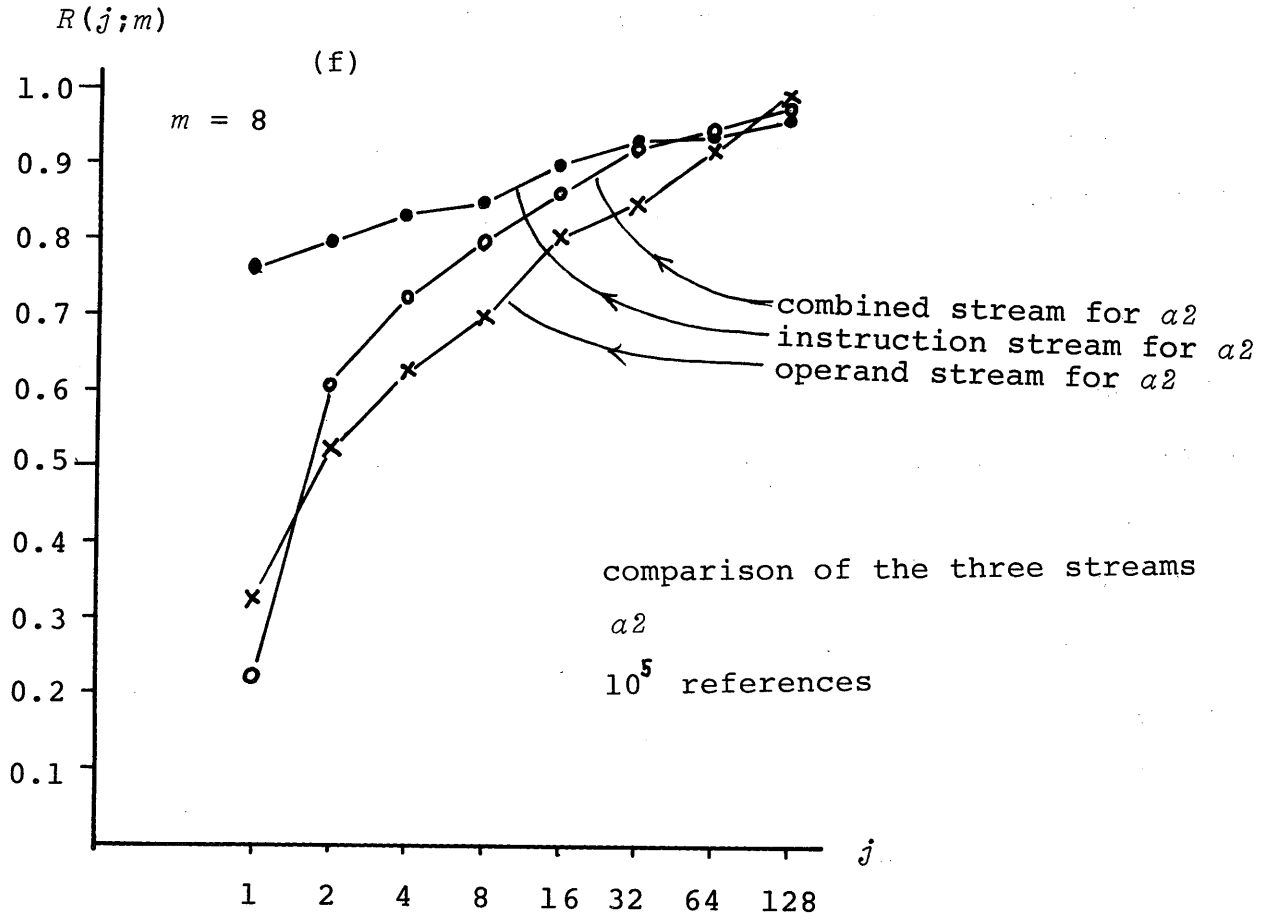
Fig.4 LRU stack distance frequency




R(j;m)
(c)
m = 4
d2
c2
b2
a2
d1
10^5 references
j
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
1
2
4
8
16
32
64
128
256


R(j;m)
(d)
m = 8
d2
c2
b2
a2
d1
10^5 references
j
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
1
2
4
8
16
32
64
128


R(j;m)
(e)
m = 16
d2
c2
b2
a2
d1
10^5 references
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
j
1 2 4 8 16 32 64 128


R(j;m)
(f)
m = 8
combined stream for a2
instruction stream for a2
operand stream for a2
comparison of the three streams
a2
10^5 references
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
j
1 2 4 8 16 32 64 128

the measured programs is unexpectedly small at $m \times j = 8$, 16 and 32. It is also seen that there is no great difference in the effects of various buffer memory structures at $m \times j = 32$. Fig.4(f) compares the characteristics of the three different streams for $a2$; i.e., instruction stream, operand stream, and combined stream. In this example, $R(j;m)$ of the instruction stream is above that of the operand stream while $j \leqq 32$, and $R(j;m)$ of the operand stream crosses over that of the instruction stream between $j = 64$ and $j = 128$.

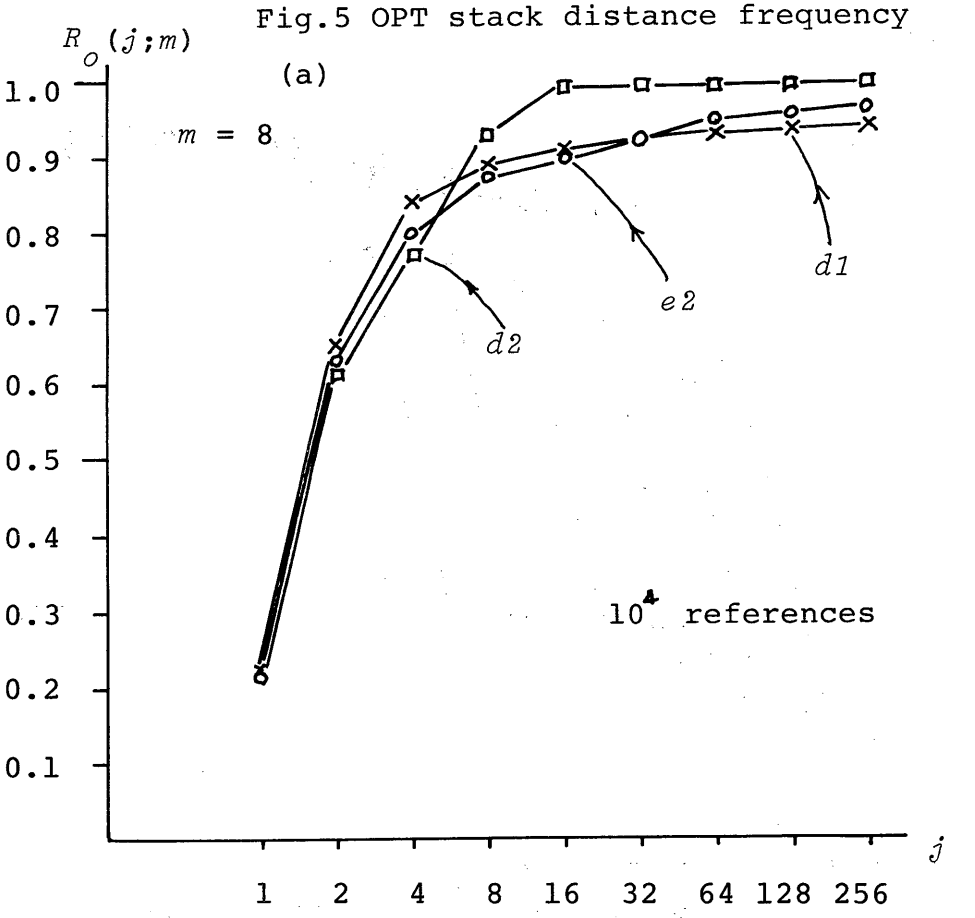
3.5 OPT stack distance frequency

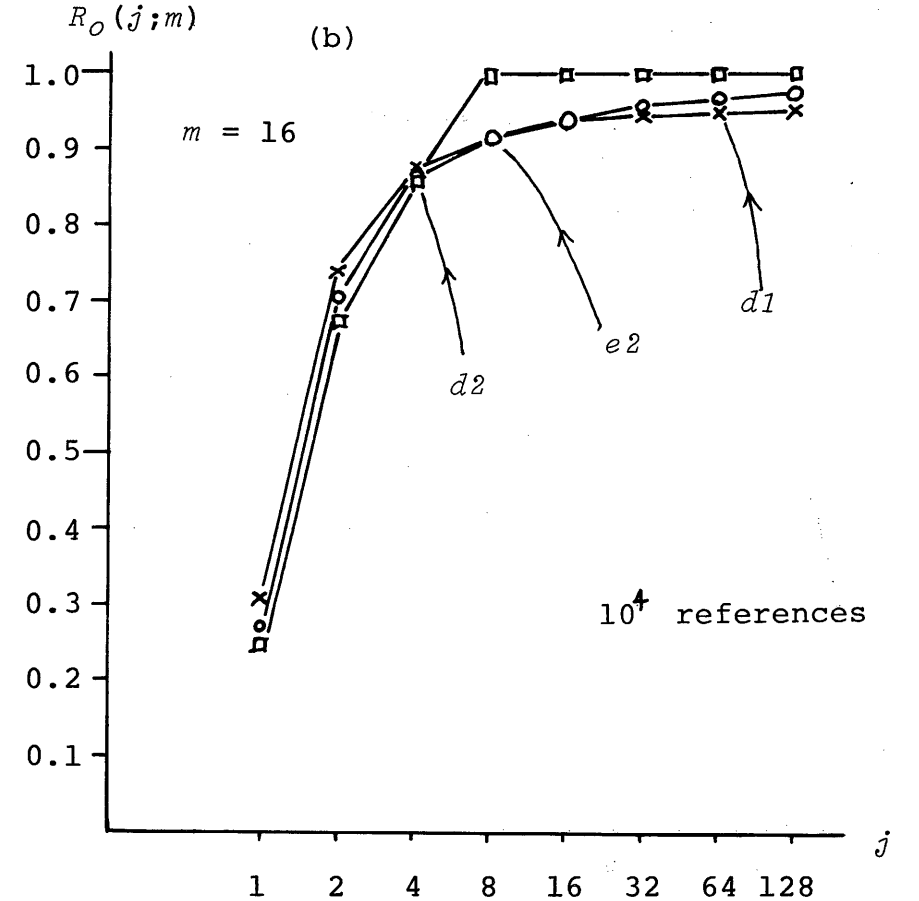
OPT stack distance frequency is the stack distance frequency by an optimum page (block) replacement algorithm called OPT.[5] OPT replacement algorithm is not realizable in an actual computer system, but is a useful benchmark for the evaluation of any other replacement algorithm.

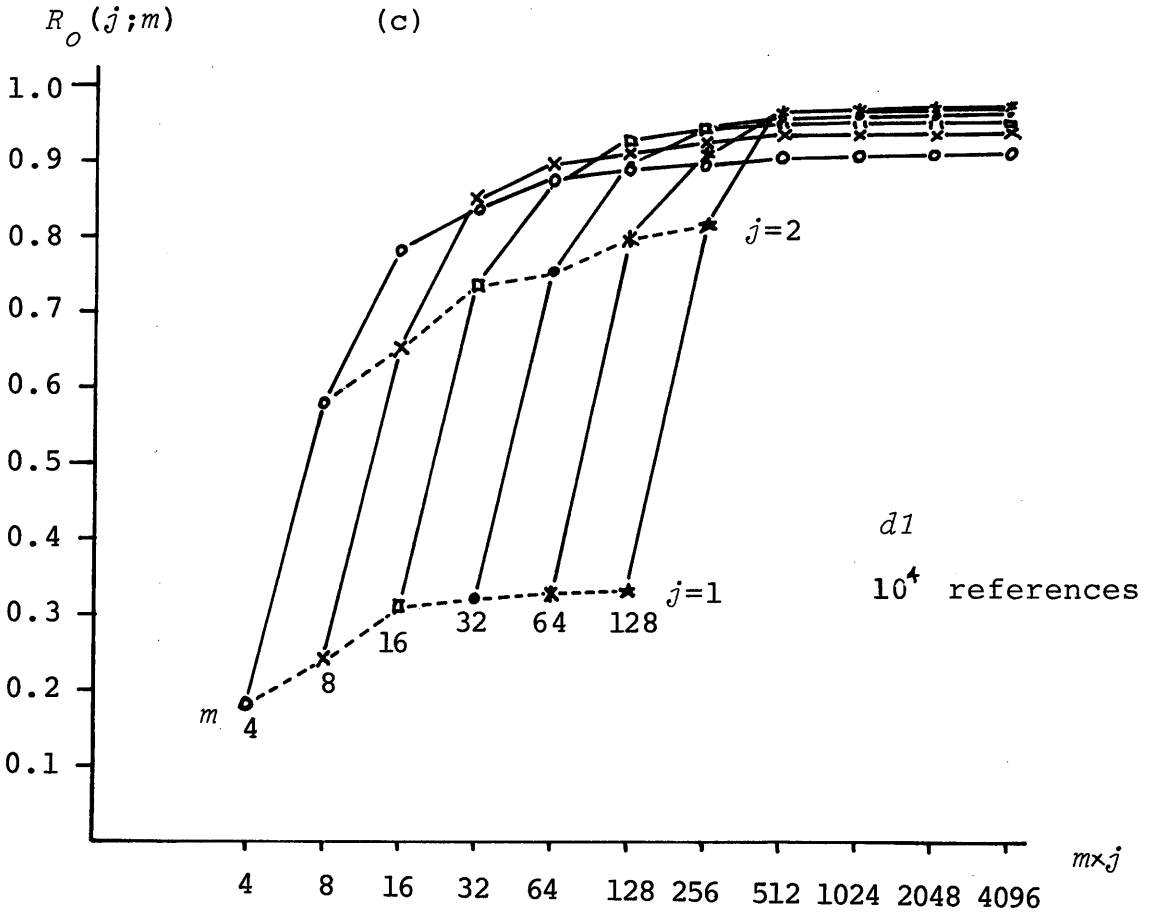
We denote $R_o(j;m)$ to be the cumulative distribution of OPT stack distance frequency as the function of the number of blocks j and the block size m.

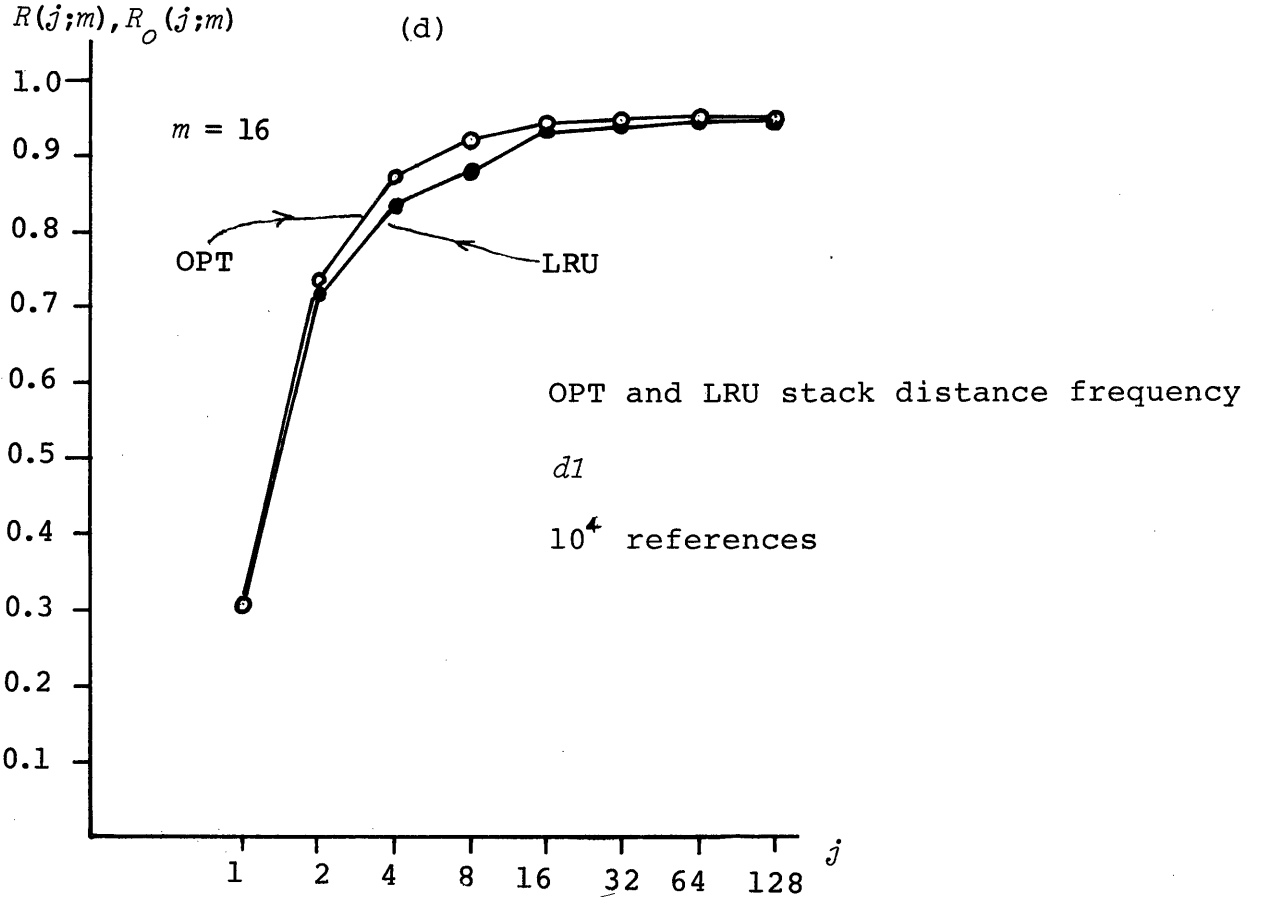
Some results are shown in Fig.5(a)~5(d). Fig.5(a) and 5(b) show the variations of $R_o(j;8)$ and $R_o(j;16)$ with j. Fig.5(c) shows the variation of $R_o(j;m)$ with $m \times j$, in which the effects of block size m on $R_o(j;m)$ are also indicated. It is obviously seen from Fig.5(c) that buffer memory has a good performance while $j \geqq 4$. The effects of using OPT and LRU algorithm are compared in Fig.5(d). Some difference appears at $j = 4$ and $j = 8$, but is not so great.

Fig.5 OPT stack distance frequency






$R_o(j;m)$
(c)
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
j=2
j=1
m
4
8
16
32
64
128
$d1$
10^4 references
4 8 16 32 64 128 256 512 1024 2048 4096
$m \times j$


$R(j;m), R_o(j;m)$
(d)
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
m = 16
OPT
LRU
OPT and LRU stack distance frequency
$d1$
10^4 references
1 2 4 8 16 32 64 128
j

3.6 Effect of instruction buffering

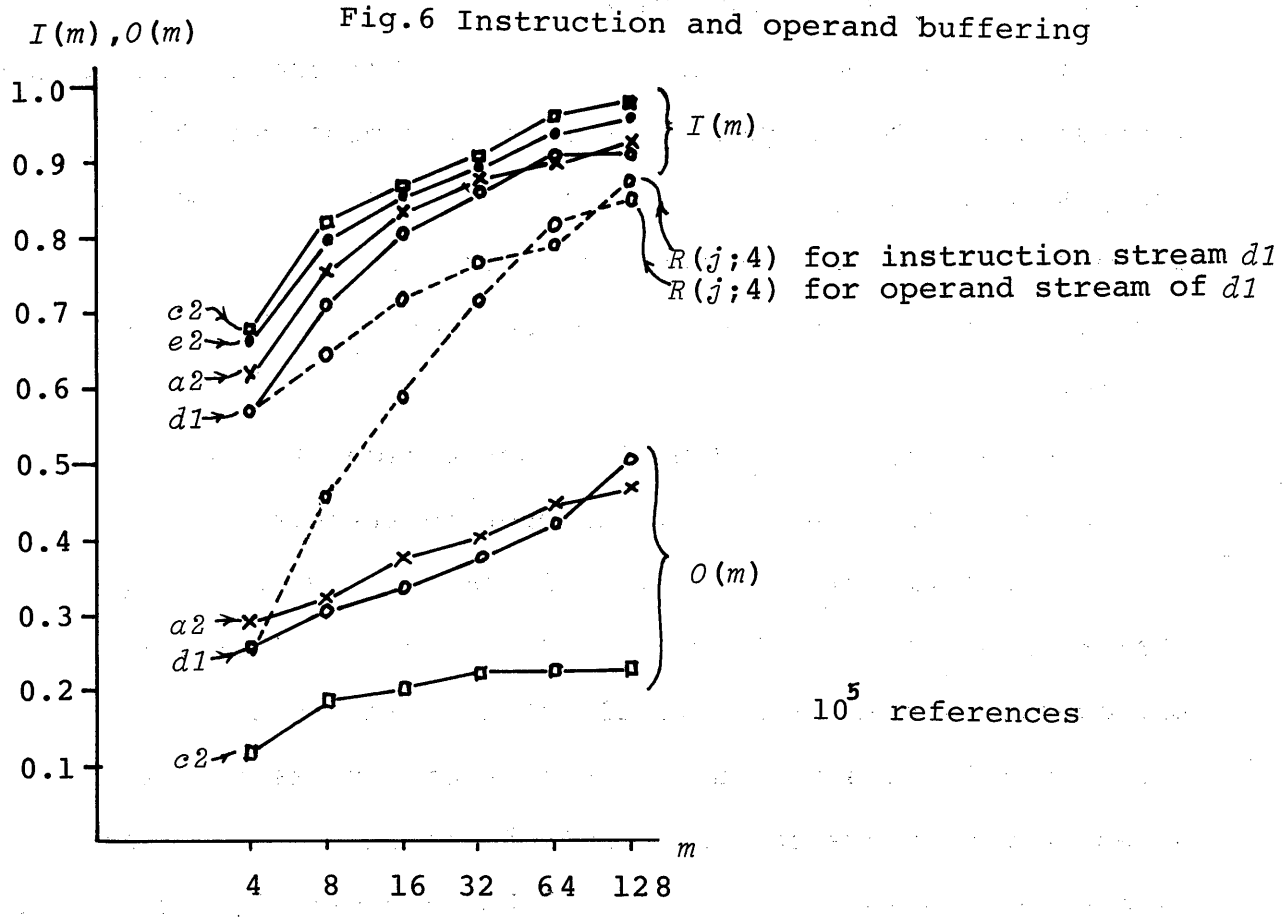
$I(m)$ is defined as the probability that the address of an instruction to be fetched is found in the block which contains the just previously executed instruction-word if memory space is divided into blocks each consisting of m contiguous addresses. $O(m)$ is defined as the corresponding probability for operand streams.

$I(m) = R(1;m) = R_O(1;m)$ for instruction streams

$O(m) = R(1;m) = R_O(1;m)$ for operand streams

The distributions of $I(m)$ and $O(m)$ are shown in Fig.6. The variation of $R(j;4)$ with m ($= 4 \times j$) for the instruction stream and operand stream of $d1$ are illustrated in Fig.6 for reference. From Fig.6 it is seen that the effect of increasing m on the access rates is greater than that of increasing the number of blocks for the instruction stream, but the effect of increasing the number of blocks is greater than that of increasing m for the operand stream.

Fig.6 Instruction and operand buffering



IV. Concluding remarks

This paper is a companion paper to " Simulation of a computer system with buffer memory " (EIS-TR-76-2).[8] We have prepared an address tracer to make the input data for memory reference simulation. At the same time, we have measured the dynamic behavior of the test programs in several points of view. In this paper the results obtained by measurements are presented. Although S and B show various patterns according to the difference of program structure, R has some similiarities between programs. It may be said that this is one of general characteristics of memory reference and it keeps the effectiveness of buffer memory or paging memory stable on various ordinary programs. Some of the results are also the outputs obtained by the memory reference simulation of buffer memory system of the smaller buffer capacities in comparison with the simulation model discussed in [8]. From the results one can see that the access-rates to buffer memory are about *50 %* if buffer capacity is as small as *4* (word/block)× *2* (blocks) and that the deviation of the access-rates among test programs is unexpectedly small if the buffer capacity is small.

Acknowledgment

The authors thank Nobuhiro Hayashi for implementing OPT simulator, and Masanori Kanazawa and Noriko Iida for their helpful comments and criticism. This work was supported in part by Project of Data Processing Center, Kyoto University.

References

1) Brawn,B.S. and Gustavson,F.G. Program behavior in a paging environment, AFIPS Conference Proceedings, FJCC, Vol.33, pp.1019-1032, 1968.

2) Denning,P.J. The working set model for program behavior, Comm. of the ACM, Vol.11, No.5, pp.323-333, 1968.

3) Freibergs,I.F. The dynamic behavior of programs, AFIPS Conference Proceedings, FJCC, Vol.33, pp.1163-1167, 1968.

4) Joseph,M. An analysis of paging and program behaviour, Computer Journal, Vol.13, No.1, pp.48-54, 1970.

5) Mattson,R.L., Gecsei,J.,Slutz,D.R. and Traiger,I.L. Evaluation techniques for storage hierarchies, IBM Systems Journal, Vol.9, No.2, pp.78-117, 1970.

6) Sisson,S.S. and Flynn,M.J. Addressing patterns and memory handling algorithms, AFIPS Conference Proceedings, FJCC, Vol.33, pp.957-967, 1968.

7) Liptay,J.S. Structural aspects of the System/360 Model 85 II The cache, IBM Systems Journal, Vol.7, No.1, pp.15-21, 1968.

8) Nakamura,T., Kitagawa,H., Kanazawa,M. and Hagiwara,H. Simulation of a computer system with buffer memory, EIS-TR-76-2, University of Tsukuba, 1976.

INSTITUTE OF ELECTRONICS AND INFORMATION SCIENCE
UNIVERSITY OF TSUKUBA
SAKURA-MURA, NIIHARI-GUN, IBARAKI JAPAN

REPORT DOCUMENTATION PAGE	REPORT NUMBER TR-76-3

TITLE

An Analysis of Program Behavior

AUTHOR(s)

Tomoo Nakamura (Institute of Electronics and Information Science, University of Tsukuba)

Hajime Kitagawa (Data Processing Center, Kyoto University)

Hiroshi Hagiwara (Faculity of Engineering, Kyoto University)

REPORT DATE	NUMBER OF PAGES
July 7	18
MAIN CATEGORY	CR CATEGORIES
Computer Systems	6.2

KEY WORDS

memory hierarchy, buffer memory, program behavior

ABSTRACT

As computer systems have increased their complexity, it has become very important to analyze the system efficiency. In the analysis of computer systems, it is fairly of use to have the knowledge of dynamic behavior of programs under execution. This paper is an empirical study of program behavior. We prepared an address tracer which recorded the memory reference patterns of programs and got the traced data for several programs. By using these data we analyzed some kinds of program behavior: run length; branch distance; memory demand; stack distance frequency for typical replacement algorithms (LRU and OPT); instruction buffering. In this paper we present the results of the measurement and analysis.

SUPPLEMENTARY NOTES