ISE-TR-80-15

# A MONITORING MECHANISM OF PROGRAMS DURING EXECUTION IN A PRACTICAL ENVIRONMENT

by

Kozo Itano

April 15, 1980

INSTITUTE
OF
INFORMATION SCIENCES AND ELECTRONICS

UNIVERSITY OF TSUKUBA

# A Monitoring Mechanism of Programs during Execution in a Practical Environment

Kozo Itano

April 15, 1980

A part of this work is submitted to the Journal of Information Processing as "Can we use a slow computer comfortably?"

## Abstract

In order to establish a more comfortable user interface, a system should inform the user the behavior of his program during execution. For this purpose, a mechanism of monitoring a program during execution is implemented, and examples of monitor are disclosed for several practical cases in an experimental system.

## 1. Introduction

Interactive use of time sharing systems has been widespread among contemporary users, and rapid response of the system is indispensble for establishing a good user interface. For example, if the system responses within 0.1 seconds, the user cannot recognize the response time. However, when the response time exceeds several seconds, the user would be irritated against the "slow" computer, even if he knows that the execution of the program needs huge computation time. The essential reason why the user is irritated is that he is forced to be kept waiting for indefinite time and he is never given any information about when the execution of the program is finished. When the turn around time exceeds several seconds, the user have to keep his nerve at high tension until the system responses to him. In this envirnment, if the system could inform the user of the current state of his programs, a more comfortable user interface would be established. Also, it would be a useful debugging tool while his program might contain some bugs.

There are two kinds of turn around times: waiting time and execution time. A successful example of reporting system of the waiting time is Cambridge 370 system [1] which reports the user when his job will be executed. In cases of the execution time, however, there is no attempt to predict the whole execution time in advance, except that only the used cpu time is informed by the user's request [2].

The new idea devised can monitor the behavior of the program during execution and can predict when it is finished. In this paper the author gives the detailed mechanism,

implementation of the system, and examples of observation of the actual execution of programs.

## 2. How an interactive system should be designed

In this section, we would analyze a mechanism of interaction between a user and a computer. A typical example is a time sharing operating system, where as shown in figure 1 a user usually types a command through a terminal and a response is printed or displayed to his terminal. In the case of a program which needs strong interaction such as a text editor, a rather rapid response is desiarable to make a good user interface. However, we cannot expect such a short response time in the all cases which may need considerably big computation time. In most cases, the computation power of the machine is limited and the system cannot execute all jobs in an instant. In these situation, a user as a human beeing is forced to wait for innegligibly small time, and he usually cannot accept it.

One of the successful solutions to these situations is given in the UNIX operating system [14], which supports the two major capabilities: full duplex terminal input/output and multi-tasking. In this system, a user can input his commands and data independently of the message outputs or responses. He may not wait the response of the system unless necessary. Further, multi-tasking capability permits the user to execute several programs in parallel as an interactive mode. For example, he can use an editor during compilation of other programs interactively. This means that a user can start the next work and may not wait for the completion of the programs

which need big compuatation time.

In many cases, however, we must wait for the completion of the currently executed program. For example, we cannot execute or test the program until the compilation and likage are finished. Hence, "how to know the current state of the programs" becomes an important problem in order to use a computer comfortably!

## 3. Basic Algorithm

A basic strategy to know the behavior of the programs during execution is to monitor periodically an execution probe (E-probe is used here after) which indicates how the execution of programs is proceeding [3]. Usually, an E-probe is a function of several variables in the program or data to be monitored. In order to eliminate the overhead due to observation of programs, we evaluate the E-probe only when it is observed, for example, each one second. The E-probe is 0 when the execution begins, and 1 when the execution is finished. If this E-probe is completely proportional to the time from the execution begins, we can know precisely how the execution of the program is proceeding by monitoring the E-probe. Further, we can know how much time is necessary to finish the execution of the program. For example, when the E-probe is observed as 0.3, it indicates that 30 percents of tatal execution has been finished.

However, since an ideal E-probe which is completely proportional to the time cannot be implemented in practical programs, we have to use some approximation. As approximation

of the E-probes, we present two mechanisms below.

(1) Mechanism 1

In this mechanism, as an E-probe we use the size of data which are used during input or output. For example, let Q be the size of total data input to be processed, and q be the size of data input which is processed already. Then, the E-probe E is defined as:

$$E = q/Q.$$

Q and q are commonly used in most read routines, and the use of these variables would not produce any overhead during execution.

This mechanism can be used in the case of translators such as compilers and assemblers. In the case of a multi-pass compiler, the E-probe becomes somewhat complex, because the relation between passes should be considered.

(2) Mechanism 2

In this mechanism, we should analyze the behavior of the program during execution and define the E-probe. As a simple example of this case, a matrix multiplication program P1 is shown below.

```
      DO 10 J=1,N
      DO 10 I=1,N
      C(I,J)=0
      DO 10 K=1,N
   10 C(I,J)=C(I,J)+A(I,K)*B(K,J)
```

P1. Matrix multiplication

In this program P1, the statement 10 is executed N**3 times, then the E-probe E is defined as shown below.

$$E=((J-1)N^{2}+(I-1)N+K-1)/N^{3}.$$

As an E-probe, also we can use E0 and E1 as an approximation of this E.

$$P0=(J-1)/N$$

$$P1=((J-1)N+I-1)/N^{2}.$$

By the use of this E-probe, we can predict the total execution time. Let t be the time from the execution begins, E(t) be the E-probe, and T be the total execution time to be predicted, then T is estimated as follows:

$$T = t/E(t).$$

And how much the execution has been proceeding is indicated by the E-probe E(t) itself. Though actually given T is not so

precise in the beginning of the program execution, it becomes more precise according as the execution is proceeding.

## 4. Implementation

An experimental system to monitor the execution of programs is implemented on the small computer TOSBAC40C which is equipped with 64K bytes of main memory, 5M bytes of magnetic disks, 2 magnetic tape drives, a character display console, a real time clock, and a printer. For easiness of the modification of the operating system, we have used MINIOS [4] which has been developed by the author.

For the implementation of the mechanism 1, we used a BCPL compiler [5] and runoff program [6] written in BCPL. In order to implement an E-probe, the data size of the file should be definite in advance the execution begins. For this purpose, we have installed the byte size of files in the file system of MINIOS. For the implementation of the mechanism 2, a Gaussian elimination program [7] was chosen. The results of the monitoring of the execution is displayed on the character display console in real time. This console is connected to the computer through a high speed communication line interface which allows quick update of the screen. An example of display format is shown in figure 2. In order to avoid a noisy message, the message is updated in the same position of the screen in each one second.

## 5. Precision of the Monitoring of Execution

Measurement of program behavior are made for the BCPL

compiler, the runoff program, and the Gaussian elimination program. The BCPL compiler has three phases: (1) AE tree generation, (2) OCODE generation, and INTCODE generation. Although the first and third phases of the compiler gave good results, the second phase gave rather poor one. This is because the input of the second phase is AE tree whose data structure is not a linear one. We show the relation of the E-probe and actual execution time as in figure 3, and the relation between execution time and size of input data as figure 4-7. The results measured for the Gaussian elimination is also shown in figure 8.

The quality of the monitoring is mostly dependent upon the E-probe. Therefore, it is most important problem to make a good E-probe in the program to be observed. There have been done many works of the analysis about the behavior of programs during execution [8-13] and there is much possibility to make up a good E-probe by the use of this kind of analysis.

## 6. Concluding Remarks

The hardware performance of the computer has been much improved, and the analysis of the software for speed up also made in the contemporary computer systems. However, the execution time of all programs cannot be reduced into negligibly small as we cannot feel. Therefore, the mechanism to monitor the behavior of programs during execution is useful for us to make "big" computation comfortably on a "slow" computer. This monitoring mechanism would be valid also in case of batch processing.

Acknowledgement

The author would like to express his thanks to Dr. Tetsuo Ida at Institute of Physical and Chemical Research and Mr. Kiyoshi Ishihata at University of Tokyo for their helpful discussions.

References:

[1] Steward, P. and Stibbs, R.J. Cambridge 370/165 user's reference manual, University of Cambrodge Computing Service (1976).

[2] DEC SYSTEM-20 User's guide, DEC (1976).

[3] Itano, K. Prediction of the actual execution time of programs and its application (in Japanese), Programming Symposium, Hakone Japan, 21(January 1980), 185-194.

[4] MINIOS Referene Manual (in Japanese), University of Tokyo (1974).

[5] Richard, M. BCPL: A tool for compiler writing and system programming, SJCC (1976), 557-566.

[6] Ida, T., Itano, K. and Ishihata, K. Implementation of the alphanumeric text formatter: ROFF (in Japanese), Annual Report of Computer Centre, University of Tokyo, 7(1977).

[7] Forsythe, G. and Moler, C.B. Computer solution of linear algebraic systems, Prentice-Hall (1976).

[8] Usijima, K. and Harada, K. Tools for analysis and evaluation of software (in Japanese), Johoshori, 20, 8(1979), 703-711.

[9] Ingalls, D. The execution time profile as a programming tool, In design and optimization of compilers edited by Rustin, R., Prentice-Hall (1972), 107-128.

[10] Knuth, D.E. An empirical study of FORTRAN programs, Software Practice and Experience, 1(1971), 105-1433.

[11] Knuth, D.E. and Stevenson, F.R. Optimal measurement points for program frequency counts, BIT, 13(1973), 313-322.

[12] Ramamoorsey, C.V., Kim, K.H. and Chen, W.T. Optimal placement of software monitoring aiding systematic testing,
IEEE Trans. SE., SE-1, 4(1975), 403-411.

[13] Fosdick, L.D. and Osterweil, L.J. Data flow analysis in software reliability, Computing Surveys, ACM, 8, 3(1976),
305-330.

[14] Ritchie, D.M. and Thompson, K. The UNIX time sharing system, CACM, 17, 7(1974), 365-375.

Figure Captions

Figure 1. Scheme of interactive system.

Figure 2. Example of display format.

Figure 3. Relation between the E-probe and execution time in case of BCPL compiler.

Figure 4. Relation between execution time and size of input data in case of BCPL compiler (pahse 1).

Figure 5. Relation between execution time and size of input data in case of BCPL compiler (phase 2).

Figure 6. Relation between execution time and size of input data in case of BCPL compiler (phase 3).

Figure 7. Relation between execution time and size of input data in case of BCPL compiler (total).

Figure 8. Relation between the E-probe and execution time in case of the BCPL compiler, the runoff, and the Gaussian elimination.

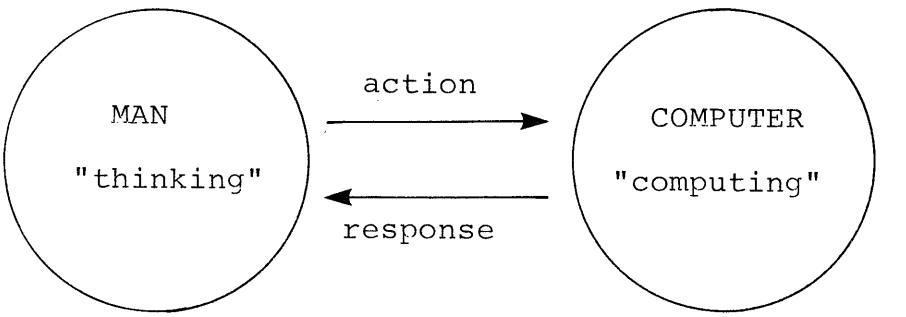


Figure 1. Scheme of interactive system

Figure 2. Example of display format

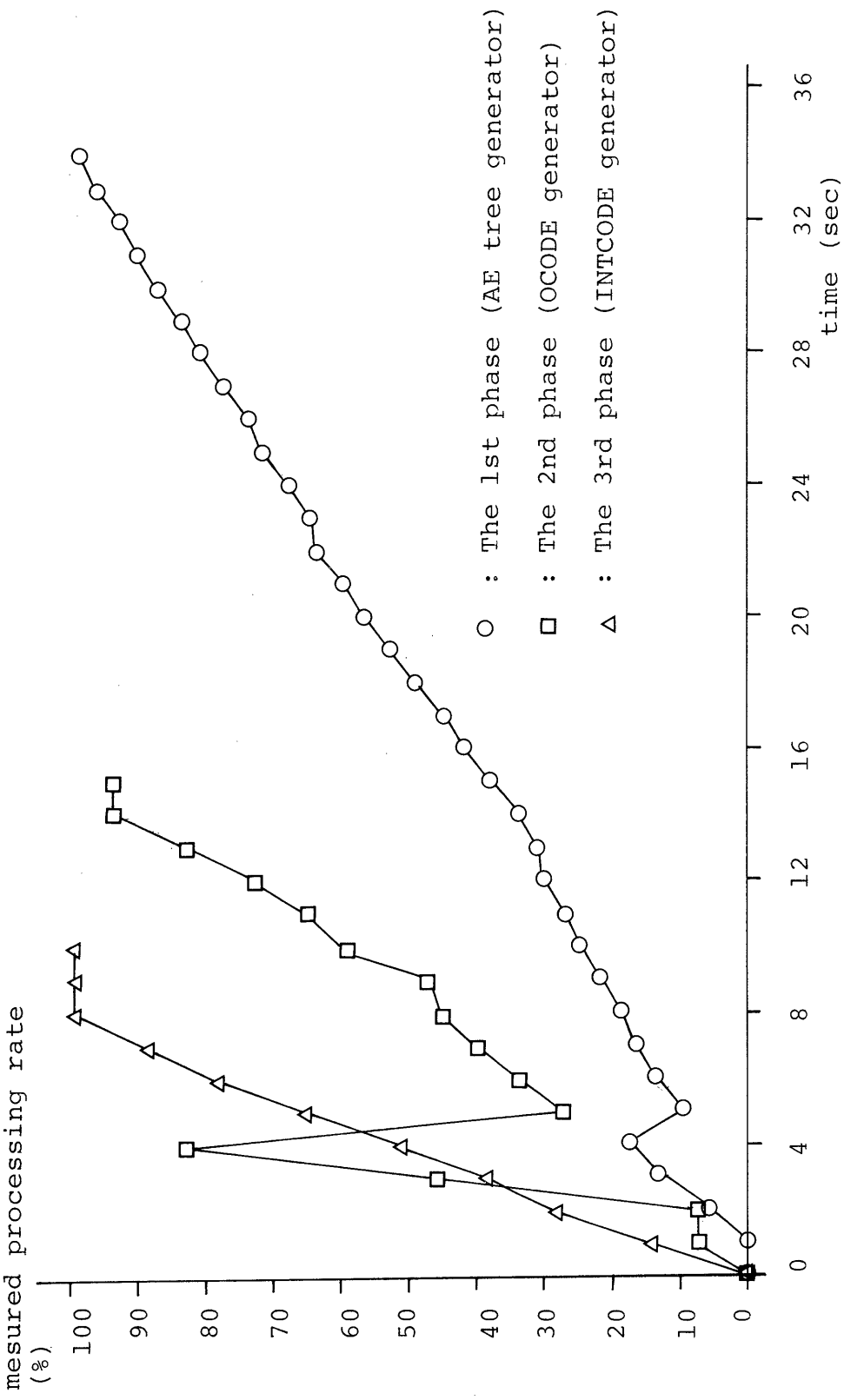


Figure 3. Relation between the E-probe and execution time in case of BCPL compiler

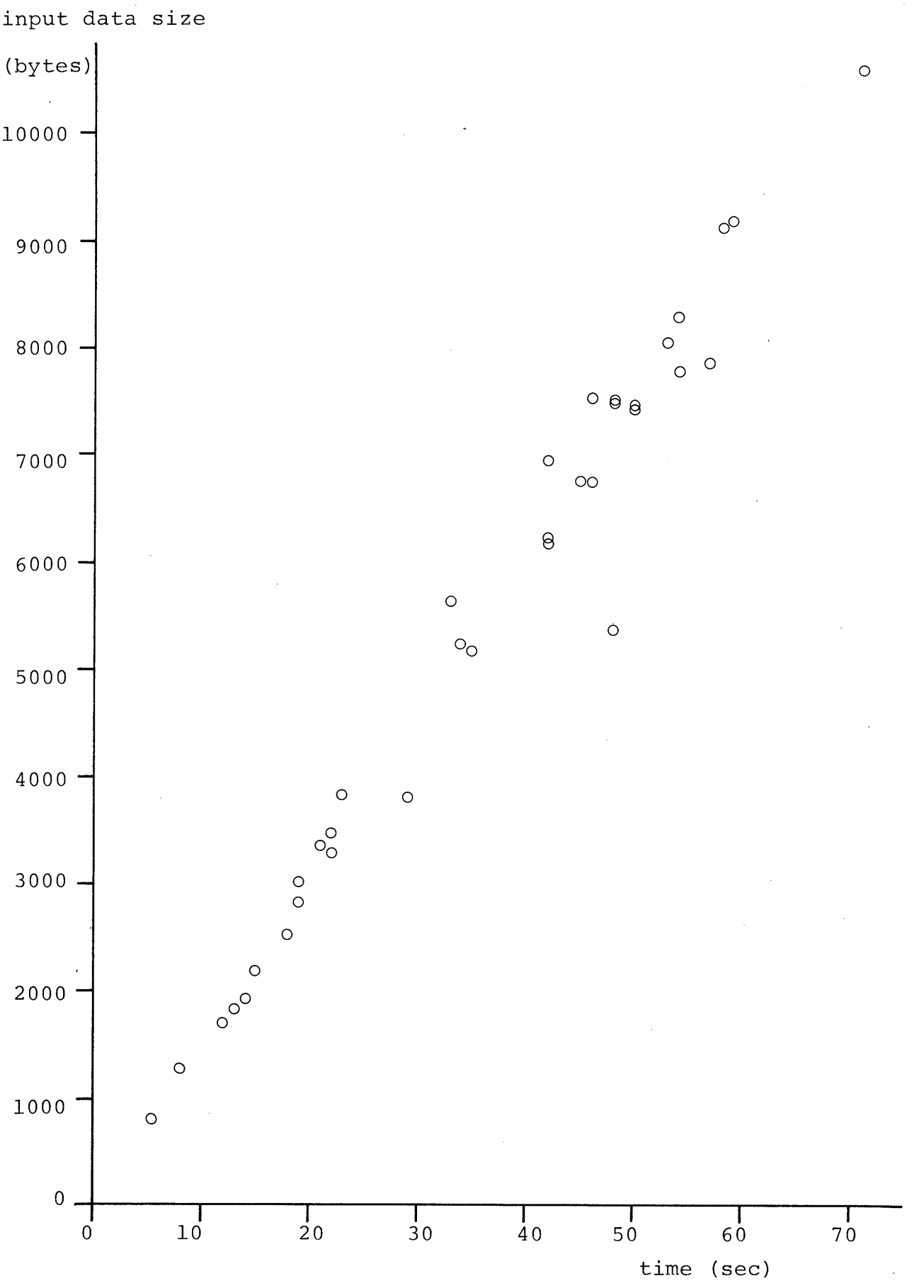


Figure 4. Relation between execution time and size of input data in the case of BCPL compiler (phase 1)

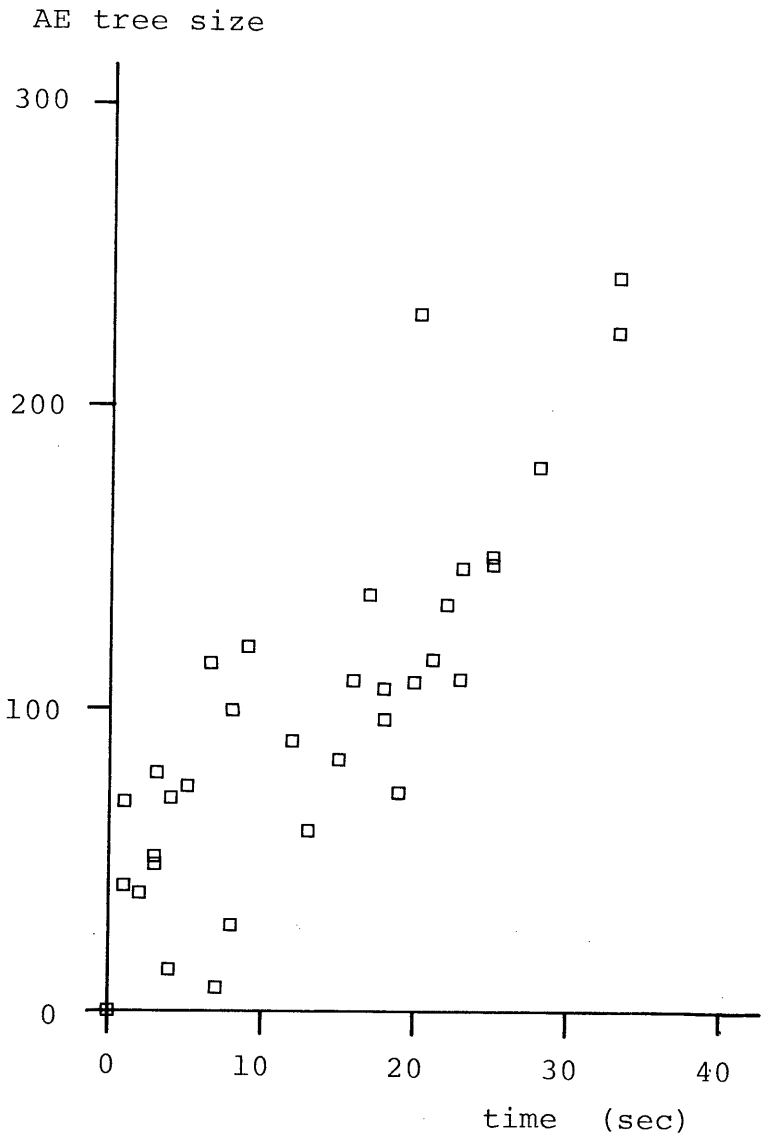


Figure 5. Relation between execution time and size of input data in case of BCPL compiler (phase 2).

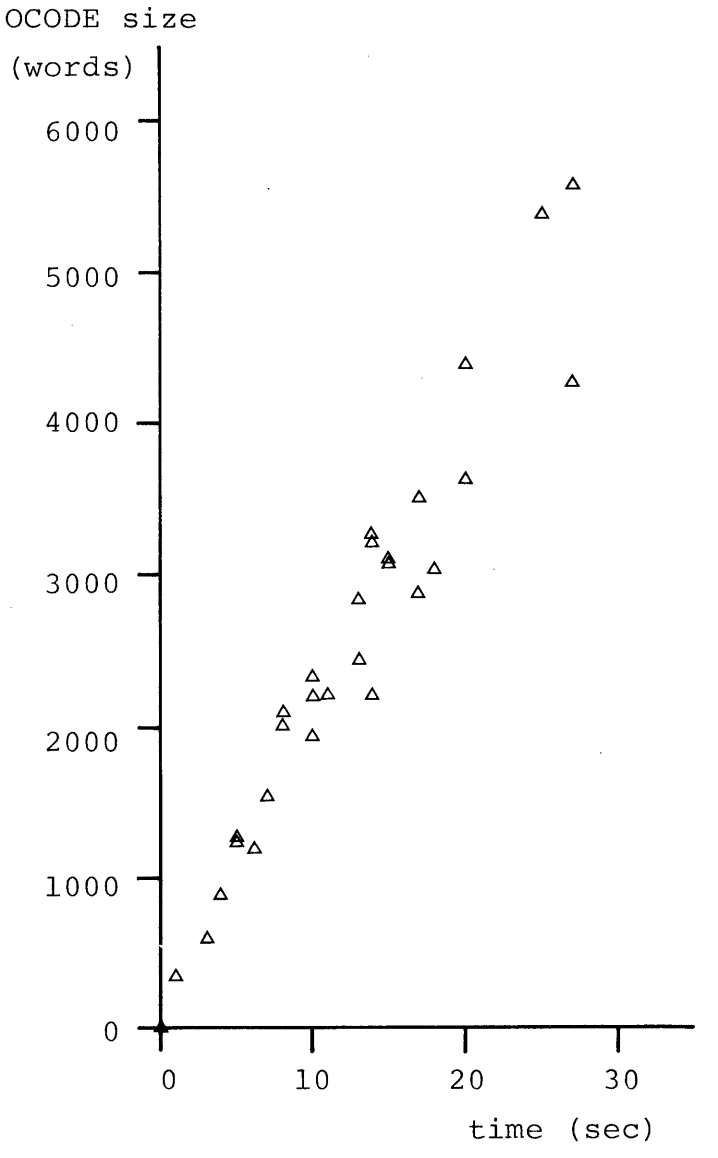


Figure 6. Relation between execution time and size of input data in the case of BCPL compiler (phase 3)

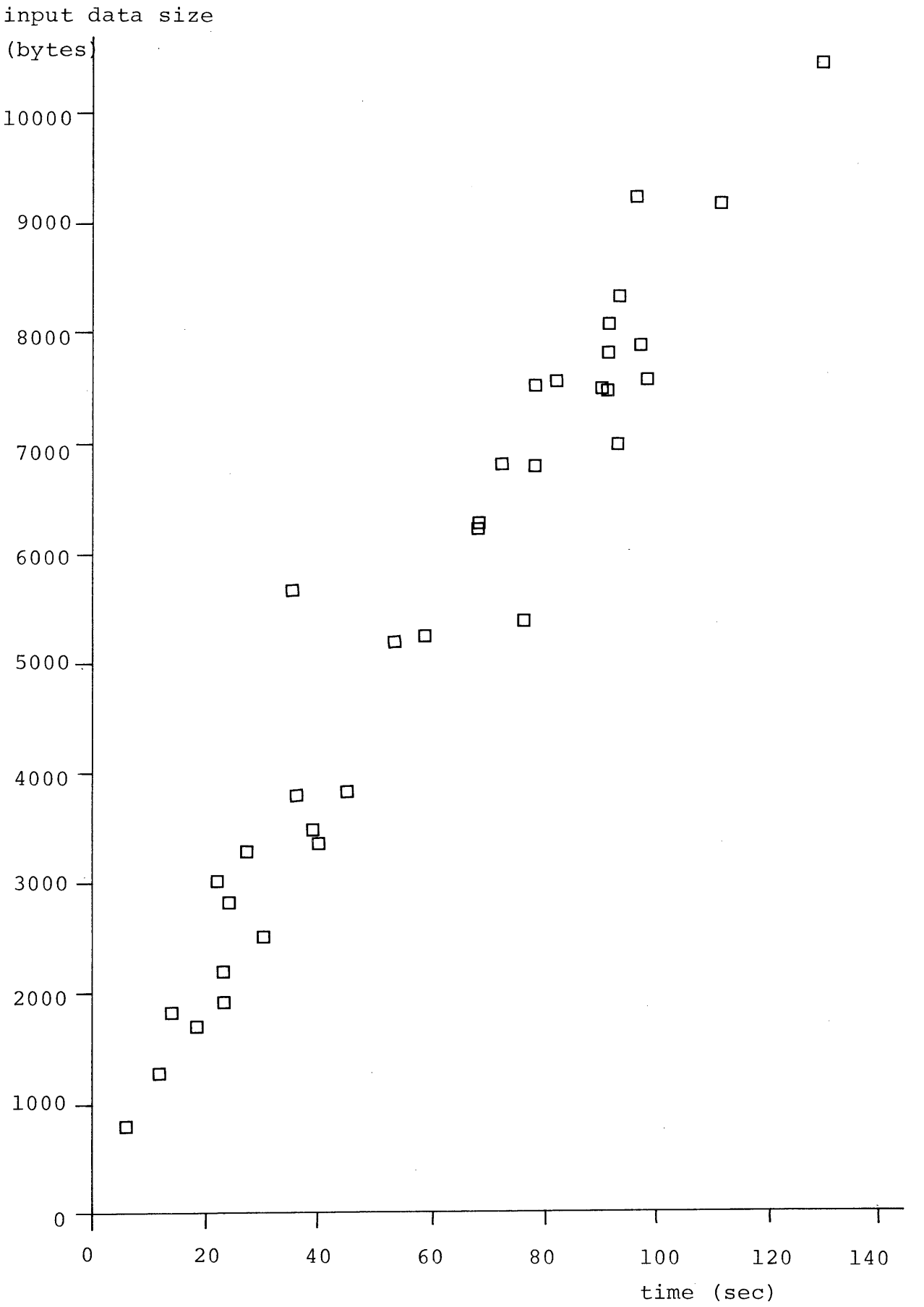


Figure 7. Relation between execution time and size of input data in the case of BCPL compiler (total).

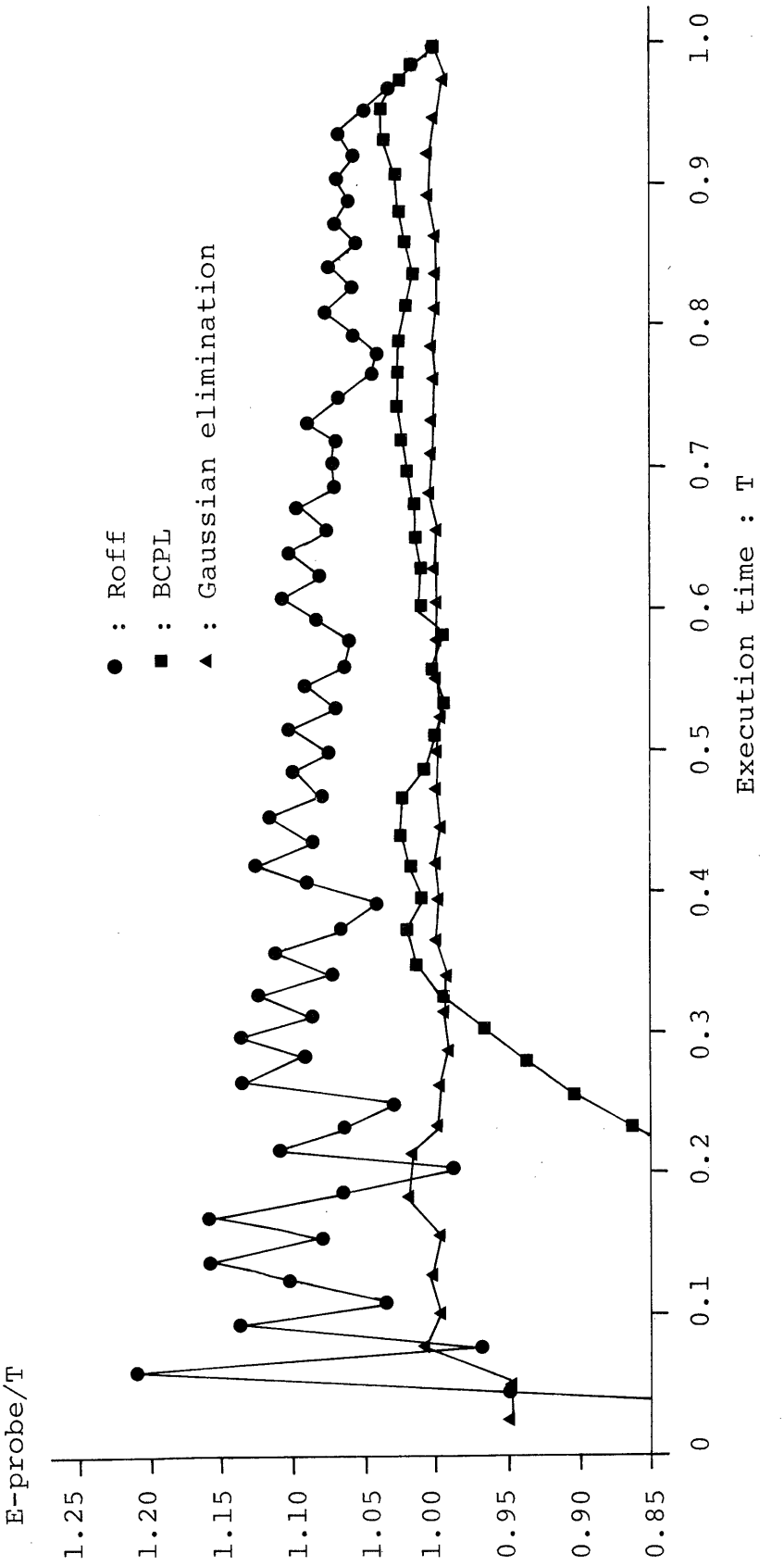


Figure 8. Relation between the E-probe and execution time in the case of the BCPL compiler, the runoff, and the Gaussian elimination.

INSTITUTE OF INFORMATION SCIENCES AND ELECTRONICS
UNIVERSITY OF TSUKUBA
SAKURA-MURA, NIIHARI-GUN, IBARAKI 305 JAPAN

REPORT DOCUMENTATION PAGE

REPORT NUMBER
ISE-TR-80-15

TITLE

A Monitoring Mechanism of Programs during Execution in a Practical Environment

AUTHOR(s)

Kozo Itano

REPORT DATE
April 15, 1980.

NUMBER OF PAGES
20

MAIN CATEGORY
Processors, Supervisory systems

CR CATEGORIES
4.1, 4.3

KEY WORDS
interactive systems, user interface design, monitors, operating systems, time sharing systems, programming systems, language processors, debugging techniques, software tools and prediction of execution time.

ABSTRACT

In order to establish a more comfortable user interface, a system should inform a user the behavior of his program during execution. For this purpose, a mechanism of monitoring a program during execution is implemented, and examples of monitoring are disclosed for several practical cases in an experimental system.

SUPPLEMENTARY NOTES