Software measurements and their influence upon machine language design*

by L. PRESSER and M. A. MELKANOFF

University of California
Los Angeles, California

INTRODUCTION
At present, software development is responsible for a large part of the total cost of computer systems. A segment of this cost is traceable to the development of programming language (e.g., FORTRAN, ALGOL, PL/I) translators which constitute major components of any software package.

Albeit machine languages have evolved through the various computer generations, the machine language of present day conventional computers is still too elementary (e.g., string manipulation operations require considerable set-up time) for a simple and concise implementation of translators and for carrying out the actual translation process in a manner which makes effective use of the available hardware.

In order for future information processing systems to make effective use of the available technology, an appropriate evaluation1 of the complete operation of present information processing systems is needed. In particular, we are interested here in obtaining information concerning the manner in which the translation of programming languages may influence the machine language design of future computers, the objectives being the production of a well-balanced integrated design and the simplification of the translator writing task.

In this paper we shall discuss a specific programming language translator and a measurement control center, and associated software artifact, incorporated into the translator at implementation time in order to gather, optionally, information about the translation process.

Utilizing a number of benchmark programs measurements are collected to determine the relative effort spent in the various sections of the translation process by this particular translator. The type of information so obtained contributes to a better understanding of how the translation of programming languages may influence machine language design.

Translation technique
The system described here is essentially the META5 translator writing system.2 This system has been extended and implemented at UCLA on both the IBM SYSTEM/360 and the SDS SIGMA 7 computers. Henceforth, for purposes of this paper, we may view this translator writing system as a translator for the META5 language.

The META5 system employs a two-pass technique. The first pass transforms, interpretively, a META5 program into a pseudo-code or intermediate language (i.e., between machine and programming language) representation. The second pass interpretively executes the pseudo-code on the associated pseudo-machine.

Through a bootstrapping approach a pseudo-code version of the program which transforms META5 programs into pseudo-code programs was obtained.* Execution of this program, on the pseudo-machine, is what constitutes the first pass of the overall process.

* The work was supported by Atomic Energy Commission AT(11-1) Gen 10 Proj. 14; Advanced Research Projects Agency, SD 184; Information Systems Division Office of Naval Research. Reproduction in whole or in part is permitted for any purpose of the United States Government.

* First, Oppenheim's3 implementation was bootstrapped to the IBM SYSTEM/360; obtaining a META5 to PL/I translator in PL/I. The next bootstrapping produced a META5 to pseudo-code translator, in pseudo-code, on the SYSTEM/360.4
Measurements

The primary type of measurements in which we are interested are:

1. Event statistics (in particular as required to determine that part of the total translation effort contributed by the various sections of a translator):
   a. Time
   b. Frequency
2. Employment of resources (storage space in particular)
3. Interaction with the Operating System
4. Structure of translated programs

The measurement process can be carried out to varying degrees. Hence, the control information supplied by the Operating System to the translator may contain an indication of the amount of effort to be dedicated to the collection of measurement data.

At the completion of the translation process (in the case of META5 at the completion of each pass), the collected measurement information is processed, formatted, and recorded on the appropriate data set (organized collection of related information) which is then made available to the Operating System. This is done in the same manner as for the other various forms of output.

When a translator is being specified it is possible to indicate the type of measurements desired and the maximum cost one is willing to pay for the obtainment of the measurements. This cost may be stipulated as a percentage of the time employed by the measurement-free translation process as far as timing specifications is concerned, and as a percentage of the resources (storage space in particular) employed by the translator when it contains no measurement artifact, as far as the utilization of resources is concerned.

It is to be emphasized that the introduction of measurement artifact for collecting most of the type of measurements in which we are interested should be an inexpensive undertaking if this task is taken into consideration during the design of the translator; moreso if the translator is a well-organized and well-designed translator. What may involve some cost is the processing and formatting of the measurement data obtained. However, this is considered as a separate task subsequent to the translation process proper.

A special unit has been incorporated into the control section of the META5 translator with the sole purpose of controlling the collection of information about the translation process. This unit controls simple software measurement artifact which has been introduced into the system.

From the programmer's point of view, the measurement instrumentation consists of a set of possible measurements (i.e., MEASURE1, MEASURE2, ..., MEASUREi, ..., where I > 0) where each of MEASUREi gathers information on a specific set of events and each can be turned ON/OFF (for either the first or the second pass) at the META5 language level by appropriate control commands.

In the META5 case the measurements (only the collection of event statistics is discussed in the sequel) are obtained as follows. The META5 implementation consists, in part, of a pseudo-instruction decoder and a set of routines. Each time a pseudo-instruction is decoded a series of calls is made to the appropriate routines in order to execute the pseudo-instruction. Most of the measurement artifact has been introduced at the decoder level. Each time a routine is called the measurement artifact increments a frequency counter associated with that routine and records the clock; when that routine returns, the clock is recorded again and the elapsed time added to a time counter associated with the routine. These measurements provide statistics on the various pseudo-instructions.

Measurement artifact has been introduced at various other points (very few outside decoder) in order to record the activity associated with certain specific events. For instance, the clock is recorded right before and just after an I/O operation and the elapsed time recorded and associated frequency counter incremented. Also, measurements are gathered on the total time spent in such events as the pseudo-code loading process. At the end of a pass the collected measurement data are processed, formatted, and output in an easy-to-read form. Figure 1 displays a structural diagram of the META5 system which includes an indication of the location of its measurement artifact.

The perturbation introduced into the translation process by the measurement activity is relatively minor, as shown below.

Measurement data

The results of a number of benchmark measurements are tabulated in Tables I through III.

The benchmark programs utilized were:

META5 to pseudo-code. Denotes the translation (first pass) of META5 (which is written in META5) to pseudo-code.

ECR to pseudo-code. Denotes the translation (first pass) of a META5 program to pseudo-code.

This program analyzes
Figure 1—Structural diagram of the META5 system

FORTRAN programs for purposes of parallel execution.

RM to pseudo-code. Denotes the translation (first pass) of a META5 program to pseudo-code. This program accepts as input certain logical equations and produces as output wire-list information.

CK to pseudo-code. Denotes the translation (first pass) of a META5 program to pseudo-code. This program reformats META5 programs into a specified format.

Execution of ECR. Denotes the execution (second pass) of the ECR program. The input data used was a small set of logic equations.

The input data used was a small set of logic equations.

Tables I through III are self-explanatory; however, some of the salient points should be noted:

1. Status restoring, in the backtracking process, represents the most time-consuming (about 29 percent) section of a META5 translation (first pass) process.

2. Inputting and outputting together represent the second most time-consuming (about 25 percent) section of a META5 translation (first pass) process.

3. String comparisons and table look-ups together represent the third most time-consuming (about 20 percent) section of a META5 translation (first pass) process.

4. Arithmetic operations constitute a relatively minor part (about 2 percent) of a META5 translation (first pass) process.

5. The effect of the measurement artifact is relatively minor (≤ 6 percent).

6. The system is printer bound when source listings are requested. If no listings are required the translator is compute bound.

7. The backtrack stack requires many levels (37 levels for the benchmark programs used here).

Finally, the following definitions may clarify the headings appearing in Table III.

The inputter is a unit which obtains, decodes, and places in a repository buffer, elements from the input stream. It permits backtracking the head of the input stream if required, and allows the optional advancing of the head of the input stream past certain characters

<table>
<thead>
<tr>
<th>Time spent in interpreter (excluding pseudo-code loading time) in ms.</th>
</tr>
</thead>
<tbody>
<tr>
<td>META5 to pseudo-code</td>
</tr>
<tr>
<td>---------------------</td>
</tr>
<tr>
<td>Listing of source program and gathering of measurements . . .</td>
</tr>
<tr>
<td>Listing of source program and no gathering of measurements . .</td>
</tr>
<tr>
<td>No listing of source program and gathering of measurements . . .</td>
</tr>
</tbody>
</table>
CONCLUSIONS

This paper suggests the stipulation of a measurement control center and associated artifact as a regular feature of programming language translator specifications. Experience shows that the introduction of the software measurement artifact required for the collection of the type of measurement data discussed here is a simple and inexpensive task if carried out at translator design time. Furthermore, the sample data presented here indicate that the optional measurement activity places a minor burden on the translation process.

The information to be gathered with such systems can have great influence on future computer architecture, especially machine language design, and in the simplification of the translator writing process. In addition, this type of data aid in the determination of the direction of effort for future translator improvement and in the evaluation of newer translator versions.

It should be emphasized that the results presented here were obtained on a single translator and specifically on the META5 SIGMA 7 implementation with which the authors were involved. Thus, even though there are no reasons to consider the implementation in question atypical, the data must be viewed in proper perspective.

It is also worth noting that META5, as implemented, is an interpretive system. If it were a compiling system with, for example, elaborate optimization as a goal, the relative percentages would show some changes.

These measurements substantiate the fact that the translation of programming languages is not an arithmetic-oriented problem but rather a string-manipulation problem. Hence, as far as the translation of programming languages is concerned, a machine language based on string manipulations rather than arithmetic operations, should allow the effective, simple, and concise implementation of programming language translators. These considerations suggest some very interesting nonconventional machine organizations.

As far as the determination of syntactic structure is concerned, the high cost of backtracking serves as strong justification for the considerable effort spent by many researchers on precedence techniques which do not require the backtracking mechanism.

Finally, it is indeed worth mentioning some simple hardware generally absent from present-day computers, which would facilitate the coding of software measurement artifact. To wit:

\[ \text{(e.g., blanks). The inputter is a component of the lexical analyzer.} \]
1. A variety of clocks, especially very fast clocks (of the order of an instruction cycle) with a very fast access time, as well as clock manipulation instructions. Another type of clock which would be useful would be clocks which accumulate time intervals.

2. A number of counters, as well as counter manipulation instructions.

There is no reason why these features could not be incorporated into a system at system design time in the same manner, for instance, that hardware facilities have been added to the newer generation of computers for purposes of error detection and correction.

REFERENCES

1. G ESTRIN et al
   Snuper computer—A computer instrumentation automaton, Proc S J C C 1967

2. D K OPPENHEIM
   The META5 language and system, System Development Corp Report TM-2396 1966

3. R MANDELL
   Private communication

4. E C RUSSELL
   Private communication

5. G ESTRIN
   Measurement definitions and discussions, Unpublished paper

6. L PRESSER
   The structure, specification, and evaluation of translators and translator writing systems, PhD dissertation Dept of Engineering University of California Los Angeles 1968

7. C KLINE
   Private communication

8. M A MARIN
   Investigation of the field of problems for the Boolean analyzer, PhD dissertation Dept of Engineering University of California Los Angeles 1968

9. T E CHEATHAM JR