The research addressed two important issues in the design of optimal languages for direct execution in an interpretive system: binding the operand identifiers in an executable instruction unit to the arguments of the routine implementing the operator defined by that instruction; and binding operand identifiers in an executable instruction unit to the arguments of the routine implementing the operator defined by that instruction; and binding operand identifiers to execution variables. These issues are central to the performance of a system, both in space and time. This report summarizes the results of this study.
DIRECTLY EXECUTED LANGUAGES

Final Report

Prepared For

Department of the Army
U.S. Army Research Office

Contract No. DAAG29-76-G-0001

July 15, 1975 - July 14, 1978

by

Michael J. Flynn
Digital Systems Laboratory
Department of Electrical Engineering
Stanford University
Stanford, CA 94305
Statement of the Problem

Representation is a key problem in computer systems. This manifests itself in many independent ways: the accuracy, the validity, the efficiency in human productivity, problems of design representation in the creation of new systems, etc. The research undertaken in this project involves one aspect of the representation issue; the production of highly efficient program representations for machine execution. This representation corresponds to a machine language in that it represents the commands which are interpreted by a machine. However, unlike conventional machine language approaches the representation is tailored to particular higher level language environments. The problem then is to find ways of synthesizing such very efficient language representations. We call languages thus derived, Directly Executed Languages or DELs.

Research Summary

A computer is largely defined by its instruction set. Of course, other issues such as space, power, algorithms used, may be important in certain applications but the user basically sees the instruction set of the machine. The instruction set, thus, is the interface between programs and resources. The program is a sequence of instructions that accomplish a desired user end. The instructions are interpreted by a control unit which activates the system's resources (data paths) to cause proper transformations to occur.

The instruction set is referred to as the architecture of the processor. It is actually a language whose usefulness is best measured by the space it requires to represent a program and time required to interpret these representations. Recent developments in technology allow a great deal more flexibility in control
unit structure while a variety of current research efforts have brought additional understanding in the nature of the instruction set. The purpose of our research was to explore these developments, especially the relationship between an arbitrary higher level language program representation and an "ideal" architecture for that language.

Specifically our research addressed two important issues in the design of optimal languages for direct execution in an interpretive system: binding the operand identifiers in an executable instruction unit to the arguments of the routine implementing the operator defined by that instruction; and binding operand identifiers in an executable instruction unit to the arguments of the routine implementing the operator defined by that instruction; and binding operand identifiers to execution variables. These issues are central to the performance of a system, both in space and time.

Historically, some form of "machine language" is used as the directly executable medium for a computing system. These languages traditionally are constrained to a single "n-address" instruction format; this leads to an excessive number of "overhead" instructions that do nothing but move values from one storage resource to another being imbedded in the executable instruction stream. We have developed techniques to reduce this overhead by increasing the number of instruction formats available at the directly executed language level [10].

Machine languages are also constricted with respect to the manner in which operands can be "addressed" within an instruction. Usually, some form of indexed base-register scheme is available, along with a direct addressing mechanism for a few, "special" storage cells (i.e., registers, and perhaps the zeroth page of main store). We developed a different identification mechanism--based on the Contour Model of Johnston. Using our scheme, only N bits are needed to encode
any identifiers in a scope containing less than $2^N$ distinct identifiers.

Together, these two results lead to directly executed language designs which are optimal in the sense that: (1) $k$ executable instructions are required to implement a source statement containing $k$ functional operators; (2) the space required to represent the executable form of a source statement containing $k$ distinct functional operators and $v$ distinct variables approaches $Fk + Nv$ -- where there are less than $2^F$ distinct functional operators in the scope of definition for the source statement, and less than $2^N$ distinct variables in this scope; (3) the time needed to execute the representation of a source statement containing $k$ functional operators, $d$ distinct variables in its domain, and $r$ distinct variables in its range approaches $d + r + k$; where time is measured in memory references.

In order to test the above results a novel directly executed language (DELtran) [9] tailored specifically to the FORTRAN source language, EMMY host, and scientific programming was constructed. DELtran is "transformationally complete" in that:

1. Code generation is linear with respect to the number of operators in a FORTRAN program.
2. Only $k$ DELtran instruction units are needed to represent a FORTRAN statement containing $k$ functional operators.
3. The space needed to represent a FORTRAN statement approaches $Nv + Fk$ -- where $v$ is the number of distinct variables in the statement, and $N$ and $F$ are the least integers such that there are less than $2^N$ distinct variables and $2^F$ distinct operators in the relevant scope of definition.

In addition, DELtran is "transparent" in that there is a 1-1 correspondence between DELtran operators and control constructs and FORTRAN operators and
control constructs, and "invertible" in that all sensible sequences of DELtran instruction units have a direct FORTRAN analogue.

The performance and vital statistics of DELtran on the host EMMY [8] is interesting, especially when compared to the 370 performance on the same system. The table below is constructed using a version of the well-known Whetstone benchmark and widely accepted and used for FORTRAN machine evaluation. The EMMY host system referred to in the table is a very small system—the processor consists of one board with 305 circuit modules and 4096 32 bit words of interpretive storage. It is clear that the DELtran performance is significantly superior to the 370 in every measure.

DELtran vs. System 370 Comparison for the Whetstone Benchmark

<table>
<thead>
<tr>
<th>Whetstone Source</th>
<th>System 370 FORTRAN-IV opt 2</th>
<th>DELtran opt 2</th>
<th>ratio 370/DELtran</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Size (static)</td>
<td>12,944 bits</td>
<td>2,428 bits</td>
<td>5.3:1</td>
</tr>
<tr>
<td>Instruction Executed</td>
<td>101,016 i.u.</td>
<td>21,843 i.u.</td>
<td>4.6:1</td>
</tr>
<tr>
<td>Instruction/Statement</td>
<td>6.6</td>
<td>1.4</td>
<td>4.6:1</td>
</tr>
<tr>
<td>Memory References</td>
<td>220,561 ref.</td>
<td>46,939 ref.</td>
<td>4.7:1</td>
</tr>
<tr>
<td>EMMY Execution Time</td>
<td>0.70 sec. (370 emulation approximates 360 Model 50)</td>
<td>0.14 sec.</td>
<td>5:1</td>
</tr>
<tr>
<td>Interpreter Size</td>
<td>2,100 words (excludes I/O)</td>
<td>800 words</td>
<td>2.6:1</td>
</tr>
</tbody>
</table>
Publications


Scientific Personnel

Michael J. Flynn
Professor, Electrical Engineering
Stanford University

Lee W. Hoevel
(Received Ph.D. degree in Electrical Engineering from Johns Hopkins University, June 1978)

Scott Wakefield
(to receive Ph.D. degree in Electrical Engineering June 1979)

Walter A. Wallach
(to receive degree in Electrical Engineering June 1979)