SETL Newsletter 109

Stephanie Brown
July 27, 1973

FASTER EXECUTION FOR THE LITTLE BASED BALM SYSTEM

Initial runs of the LITTLE written MBALM simulator indicate that it runs almost a factor of two times more slowly than the current FORTRAN based MBALM simulator. Allowing for expected improvements in the new LITTLE assembler the LITTLE based simulator can probably be made to perform as well as the FORTRAN version. Since the LITTLE based simulator will use the SETL run time library procedures in executing SETLB programs it will actually be considerably faster than the FORTRAN based version. However, it seems unlikely that the LITTLE based simulator will match the overall speed of our current MBALM to COMPASS translator system. Consequently a SRTL-compatible BALM translator will probably be necessary to make the LITTLE based system a viable alternative.

WHY A TRANSLATOR?

The BALM compiler generates code for a virtual machine which we call the MBALM. MBALM instructions are executed on the 6600 via a program which simulates the MBALM machine. Clearly this method of execution is inefficient. Runs of the SPY program on our Fortran written simulator show that about 70% of the total execution time is spent unpacking the MBALM instruction and jumping to the code which simulates it. Furthermore if the code were executed directly some simple register management could eliminate unnecessary loads and stores. In fact the MBALM to COMPASS translator runs between 6 and 10 times faster than the simulator system.

WHY NOT GENERATE COMPASS INSTEAD OF BALM?

Several reasons influenced the decision to write the existing translator as a FORTRAN program whose input is MBALM and whose output is a COMPASS code block.

1) MACHINE INDEPENDENCE

The BALM compiler is truely machine independent since it generates code for the virtual MBALM machine. It is easily transportable to other machines, needing only an MBLAM simulator program to run.

2) COMPLEXITY

The MBALM machine is ideally suited to running BALM programs. For example, it is a stacking machine and it supports lists and vectors. COMPASS is a much lower level language than MBALM. Problems of efficient register assignment and efficient storage of COMPASS code skeltons make generation of COMPASS more reasonable in a lower level language than BALM.

3) TRANSPORTABILITY

The translator program is written in FORTRAN. It has a table of skelton COMPASS code for each MBALM instruction and a table which directs parameter substitution into COMPASS instructions. Since the FORTRAN program is essentially table driven it was hoped that code for any machine could be contained in the tables.

4) STABILITY OF BALM

At the time the current BALM translator was being considered the BALM compiler was undergoing considerable changes. Shortcomings and problems pointed out by users were being rectified thus making it particulary unwise to maintain two versions of the BALM compiler - one into MBALM and one into COMPASS.

WHY NOT MODIFY THE BALM COMPILER TO GENERATE LITTLE?

One solution to the problem of faster execution for a SRTL compatible BALM system is to write a program which translates MBALM code into LITTLE.

A second solution is to modify the BALM code generator routines to produce LITTLE rather than MBALM.

Most of the objections to directly generating COMPASS disappear when we consider LITTLE. Machine independence is preserved since LITTLE is presumably machine independent. Transportability is the same whether a program is written to translate MBALM to LITTLE or whether BALM directly generates LITTLE. BALM is considerably more stable than it was a year ago. Maintenance of a second set of code generators is not likely to be made difficult by continuing changes to the compiler.

Complexity of the task is a bit more difficult to access. However in most respects LITTLE is a higher level language than MBALM. Granted that it does not include stacking, recursive calls and does not directly handle vectors or lists. However, generating LITTLE is not as difficult as generating COMPASS. The LITTLE compiler will handle register assignment and many of the other problems associated in generating COMPASS.

Producing LITTLE directly from BALM is an easier programming task than writing an MBALM to LITTLE translator. Much of the work such as label resolution which is done by the compiler must be repeated in the translator.

The general design of the BALM code generator would probably be retained even if LITTLE became the target language. This means that similar LITTLE code would be procuded regardless of whether it was generated directly from BALM or via an MBALM→LITTLE translator. However, it would be very easy to include special cases in BALM and quite difficult to add them to a translator.

SHOULD WE PRESERVE MBALM?

In one sense producing MBALM is an extra task. It will not result in a better final product (i.e., LITTLE code). Nor will it simplify the programming and maintenance tasks.

MBALM is very convenient if one wishes to MICRO program or mini computer. However, in this case the LITTLE translator would be as useless as BALM code generators for LITTLE.

One reason for keeping MBALM is that it provides a concise semantic definition of BALM or BALM SETL. If we directly produce LITTLE we no longer have a simple intermediate language.

<u>EXAMPLE</u>:

Consider the BALM statement

```
A = VECTOR (1, 2, X, 4, 5);
```

which is equivalent to the SETLB

```
A = <1, 2, X, 4, 5>;
```

The following MBALM code is produced:

```
LOAD 1
LOAD 2
LOAD X
LOAD 4
LOAD 5
VECTOR (5)
STORE A
```

The LITTLE code produced by a translator or initially by BALM would probably be something like this:

```
ARG1 = ROOTSINT;
EVAL ARG1 = 1;
ARG2 = ROOTSINT;
EVAL ARG2 = 2;
RESULT = VALTB (X);
PUSH1 (ARG1);          /*FREE A REGISTER */
EVAL (ARG1) = 4;
PUSH1 (ARG2);      /*FREE A REGISTER */
EVAL ARG2 = 5;
PUSH1(RESULT);      /*CLEAR REGISTERS AS */
PUSH1(ARG1);       /*GARBAGE COLLECTOR MAY BE */
PUSH1(ARG2);       /*CALLED */
```

```
ARG1 = ALLOTUP(5);
STP = STP+5;        /* POP STACK */
ARG2 = EPTR ARG1 +OFFTUPLE;
DOM (I,0,4);
   HEAP(ARG2+J) = STACK(STP-J); /* STORE TUPLE */
EDOM;
PUSH1(ARG1);
VALTB(A)=ARG1;
```

Treating this as a special case in BALM the following LITTLE code could probably be generated.

```
ARG1 = ALLOTUP(5);
PUSH1(ARG1);
ARG2 = EPTR ARG1 + OFFTUPLE;
RESULT = ROOTSINT;
EVAL RESULT = 1;
HEAP(ARG2) = RESULT;
EVAL RESULT = 2;
HEAP(ARG+1) = RESULT;
HEAP(ARG2+2) = VALTB(X);
EVAL RESULT = 4;
HEAP(ARG2+3) = RESULT;
EVAL RESULT = 5;
HEAP(ARG2+4) = RESULT;
VALTB(A) = ARG1;
```