PACT Loop Expansion*

GUS HEMPSTEAD AND JULES I. SCHWARTZ

The RAND Corporation, Santa Monica, Calif.

Introduction

Before beginning a discussion of the PACT loop expansion technique, it will be beneficial to review briefly the PACT instructions expanded in this section and the state of the partially compiled program at the time the loop generation begins.

The two orders used for generating loops are SET and TEST. Both commands describe subscripts and they are both necessary for a loop on any subscript. The SET order determines the initial value of a subscript and thus the initial values of the addresses for the generated code. The TEST order contains the final value of the subscript, which determines when the escape from the loop is to be made. If a step is specified in the TEST instruction, control will keep returning to this step until the escape has been made from the loop. If no step is specified, the return will be made to the step following the SET on this subscript.

The TESTs on a group of subscripts must appear in the inverse order of the SETs on these subscripts.

The USE operation determines a value for a subscript but is not an actual part of the loop mechanism.

Subscripts can be SET, USEd, and TESTed to numbers or variables. In the variable case, locations of the variables are available at loop expansion time rather than the actual numbers.

At the time this section begins, the PACT arithmetic and functional operations have been expanded, so that the actual machine language for most PACT instructions is known. Also, the relative assignment of variable storage has been made, so that with each subscripted instruction, the initial address of any array and the amount by which a variable address is to be modified for an increase in a particular subscript is known.

This discussion does not cover all the details of how loops are expanded in PACT. The main outline of the method will be discussed, omitting the description of special cases and some of the devices used to accomplish this expansion. On any machine, the actual writing of machine language for an expanded order is relatively easy once the necessary information has been gathered, so this aspect will not be discussed to any great degree.

Relation between PACT and Machine Language

In writing a loop generating routine, as in most generative routines, the first requirement is a thorough analysis of the code to be compiled and its relation

* Presented at the meeting of the Association, Sept. 14–16, 1955.

to the code to be generated. The code to be compiled, as in the case of PACT, is usually fairly simple. Thus, the fundamental problem is determining of the machine code necessary in the generated routine for a given series of instructions in the code to be compiled.

Forming a Machine Language Loop

The generated code for loop expansion consists of three main sections; the set up address orders, the address modification instructions, and the test instructions. This is true for any group of instructions which are to be enclosed in a loop. The set up and modify instructions are of a straight forward nature. The test can be on one of the addresses, in which case a test word must be formed and stored for later use, or a counter-type test. The latter means setting up a word with a tally in it which increases or decreases. Escape from the loop is made when the tally changes sign. Example 1 is assumed to be a series of five machine language instructions which are to be repeated a given number of times by the addition of other steps both at the beginning of the sequence of instructions and at the end.

Assuming the addresses A and B are to be modified with each repetition, we can draw several conclusions as to what additional instructions are necessary.

First, we wish to add the modify orders after step 7. These are a sequence of steps which will increase the addresses in steps 3, 4, 5, and 7 by a given increment after each repetition.

The nature of these orders is shown in Example 2. Any loop with modifiable addresses would necessarily contain a series of orders similar to these.

In our loop expansion, the method used for writing a series of modify orders such as these is the following:

Starting at the point at which the loop is to begin, every instruction is examined to determine if its address is modifiable. Those instructions which do have modifiable addresses are stored in a table in high speed memory, along with the amount by which their addresses are to be modified and with the initial addresses of their arrays. When the point in the program is reached where the loop is to end, the table is searched for all addresses which are to be modified at this time. It is at this point that the machine language modify instructions are written.

EXAMPLE 1

Mythical Machine Language

STEP	OP	ADDR.
3	XX	A
4	YY	B
5	ZZ	A
6	QQ	3
7	TT	A

To be enclosed in loop: Addresses A and B to be modified.

EXAMPLE 2

Modify and Test Orders

	STEP	
MODIFY	8	RESET ADD STEP 4
	9	MODIFY B BY INCREMENT
	10	STORE ADDRESS IN STEP 4
	11	RESET ADD STEP 3
	12	MODIFY A BY INCREMENT
	13	STORE ADDRESS IN STEPS 3, 5, AND 7
TEST	14	SUBSTRACT STEP S (TEST WORD)
	15	TRANSFER PLUS (OR MINUS) TO STEP 3

The next consideration in loop writing is the forming of the test instructions. If it is possible at this point to compute the last value of one of the addresses that are being modified, then the simplest test to write is a test on an address. It is possible, with a few exceptions, when the subscripts are tested against numbers. To do this the last address is computed, the operation of the word to be tested is added to it, and this test word is stored in a step of constants, say step S. Then steps similar to 14 and 15 are added to the code.

In the case when the final address is not computable at this time, for example, when the subscript is tested against a symbol, a counter can be set up, and this type test can be used for the escape from the loop, so that a different and somewhat longer series of instructions would be necessary after step 13.

Writing the modify and test orders is the main part of the first, or forward, pass of the PACT loop expansion. At the same time these steps are written, of course, other functions are being performed. The main secondary function of this pass is the writing of various flags on information which will be used in the second, or backward, pass, the writing of the set-up orders. In some cases, there are additional instructions which must be written. These instructions will be discussed shortly.

The example in machine language so far discussed has been rather elementary. In PACT language, it would have appeared somewhat similar to Example 3. Of course, the one-to-one ratio would not ordinarily hold. Other machine lan-

EXAMPLE 3

PACT Code with Single-Subscripted Variables

STEP	OP	FACTOR	S_1	S_2
2	SET		I	1
3		X	I	
4	+	Y	I	
5	+	X	I	
6	−	3		
7	EQ	X	I	
8	TEST		I	V

guage instructions would probably have been added in actual cases of operation expansion.

The method used for expanding this series of PACT instructions into a loop would thus take the following course: Beginning with the SET instruction, all operations are examined for subscripts. All steps containing variables with subscripts are placed in a table in high-speed memory along with the amount their addresses are to be increased for an increase in the subscript, and the initial address of each array. In this case, steps 3, 4, 5, and 7 would be entered in this table. When the TEST instruction is found, all variables in the table are examined for the subscript being tested. For each one found, the necessary modify instructions are added in the TEST step position. The actual test instructions are then added. The decision as to whether an address or counter type test is to be made is determined by the test word being computable which is usually possible if V is a number, or not computable which is the case if V is a variable, or occasionally when V is a number.

Fundamentally, this is the method used in generating all expanded PACT loop instructions containing single-subscripted variables. However, life is not always this simple. More elaborate PACT codes lead to more interesting and complicated machine-language cases.

Inside and Outside Loops

By making the variables used in the previous examples elements of a matrix or double-subscripted variables, some additional problems can be seen.

The major difficulty encountered in Example 4 is the necessity of producing machine language code which, in effect, resets the inside subscript to its initial value every time escape is made from the inside loop, I in this case. The initial setting of the subscripts makes the variable addresses equal to X_{11} and Y_{11}. Each repetition of the loop will increase I by 1, and the variables will be effectively at X_{71} and Y_{71} at the first escape from the inside loop. Before leaving step 8, the expanded instructions have to include instructions which reset I to 1. Then upon entering step 9, the increase of J will leave the variables at X_{12},

EXAMPLE 4

PACT Code with Double Subscripted Variables

STEP	OP	FACTOR	S_1	S_2
1	SET		J	1
2	SET		I	1
3		X	I	J
4	+	Y	I	J
5	+	X	I	J
6	−	3		
7	EQ	X	I	J
8	TEST		I	7
9	TEST		J	7

Y_{12} as we would want, rather than at X_{72} and Y_{72}. For each succeeding escape from the inside loop, the same resetting is necessary.

The resetting of the subscript instructions are written in the following manner: When the modify orders described previously are written, each entry in the table is erased when both, or the only subscript, are accounted for, so that if after writing the test sequence an entry remains with the subscript being tested, this entry is recognized as containing the inside subscript of a double subscripted variable, and thus reset instructions must be written.

After writing the modify and test instructions in step 8, an examination of the table would reveal steps 3, 4, 5 and 7 still in the table and the necessary reset instructions would be written. After writing the modify and test orders in step 9, however, no entries would remain in the table, so no reset orders would be written.

The other difference which appears in this case is the fact that the test word which is used for the test on I is a function of J as well as I. Thus the test word on I must be increased when J increases. For example, if step 3 has been used as the word for the test on I, and it is recognized that I is the inside loop, the outside subscript, J, would be stored in another table in highspeed memory along with the location of the test word which has J as the outside subscript.

The very first requirement, then, of expanding the PACT TEST instruction is the search of this table of test words to be modified.

When a subscript in this table agrees with the subscript being tested, the necessary test word modification instructions are written prior to entering the writing of the address modification instructions. This would occur when the expansion of step 9 begins.

Interlocking Loops

When Example 4 is first examined, the reasoning for this mode of operation is not immediately obvious. However, when one examines a case involving inter-

EXAMPLE 5

Matrix Multiplication (Interlocking Loops)

STEP	OP	FACTOR	S_1	S_2
1	SET		I	1
2	SET		K	1
3	CL	C	I	K
4	SET		J	1
5		A	I	J
6	X	B	J	K
7	+	C	I	K
8	EQ	C	I	K
9	TEST		J	7
10	TEST		K	7
11	TEST		I	7

locking loops, such as is used for the matrix multiplication in Example 5, the reason for the methods used becomes apparent.

Between Steps 9 and 10 in Example 5 it is necessary to reset J, so that steps 5 and 6 will have J = 1 for the beginning of the loop on K. Similar reasoning holds for resetting K after Step 10. The test word on J is not modified until the outside loop is reached, so that, if A_{IJ} were used in the test word for J, the test word would not increase until I is increased.

Second Pass

As stated previously, the expanding of the USE and SET orders is the second pass of the loop expansion. SET orders are used to set up addresses of variables, values of counters, and test words which are to be modified. The USE order is for the sole purpose of setting up addresses.

Variable set-up orders are written in a similar manner to the modify orders of the first pass. This time the program is read in a backward direction, storing all subscripted variables in a table. Upon finding a USE or SET order, this table is examined and the necessary orders are written.

Information about counters and test words has been saved from the first pass and their necessary set-up orders are written accordingly.

Conclusion

On both passes of this loop expansion, the basic method has been to examine the program in a forward or backward direction, storing various information in tables of undetermined length in high-speed storage, then using this information to write the necessary machine language.

The main advantage of this method is the speed at which information is accessible. However, the main handicap is the amount of storage available. There is no limit to the amount of code that one might wish compiled at one time. But there is a limit to the amount of space available for a given table. Two results arise from this situation. One is the possibility of a program not working, i.e. the compiler breaking down when the number of loops or variables is too large. To date this situation has not arisen. The lengths of the tables have been adequate for any problem run until now.

(Actually, the restrictions are not too encumbering. As many as 33 SET instructions are allowed within one region, for example.)

Of course, when one becomes familiar with PACT coding, he learns certain devices are possible which alleviate much of the possibility of a program breaking down. For example, if the number of variables seems excessive, the use of a Result of a Step can be used to eliminate many of them (and also save space in the generated code). Subregions can be utilized effectively when necessary.

Another result of this mode of operation is the restriction on the expanded code. In attempting to conserve space for tables, a pattern was set up to handle the general case, and thus some special cases whose machine language code may

have been handled very cleverly are forced to follow the general pattern and thus might be less efficient in machine language.

This situation, we believe, will have to be tolerated in any routine of this sort at the present time. To attempt to make more frequent use of secondary storage, such as tapes or drums, with their present relatively slow access speed would slow the compiling time by a considerable factor, and the gain to be gotten in machine language efficiency or impossibility of breakdown might be overshadowed by the fewer number of routines capable of being compiled in a given time.

It seems then that full utilization of high-speed storage and a minimum of secondary storage is both a practical and sufficient method of generating programs of this type.