HARDWARE IMPLEMENTATION OF LOOP TRACE AND MICROPROGRAM SYNTHESIS

DR. A.M. ADB-ALLA and LAIRD H. MOFFETT
George Washington University and Naval Research Laboratory
Washington, D.C., U.S.A.

The modification or tuning of the microcode in a computer that utilizes a writable control store is one method whereby a program's execution time can be improved. A method for automatically performing a microcode tuning or synthesis has been developed by Drs. Karlgaard and Abd-alla and is discussed in detail in [1]. Presented is an extension of this effort which allows microcode synthesis to be performed "on-the-fly." This is accomplished by: (1) performing the required program trace with a hardware modification, (2) eliminating the statistics generation requirement, and (3) performing the synthesizing by using a microprogram rather than software. The implementation of this technique is described in this paper.

1. INTRODUCTION

An automatic microcode tuning procedure has been developed [1] whereby the execution time required to perform operations in a loop of a program could be reduced. Further work is being pursued on this topic to allow this tuning to be accomplished "on-the-fly," during run time. This paper presents briefly: the tuning algorithm developed in [1], the drawbacks of this method for not allowing the tuning to be performed "on-the-fly," a method of performing the trace to generate the required statistics, an approach to performing the actual synthesis, and future effort to be performed by the authors in examining the utility of the technique.

2. HEURISTIC SYNTHESIS ALGORITHM

A synthesis procedure was presented [1] whereby the time required to perform operations in a program loop could be reduced. Basically this method required a trace of the program in order to gather data concerning the program operation. This data would enable one to detect the presence of loops, the number of times a specific memory location has been addressed within the loop, and whether that address was an instruction or an operand location. Then, as shown in fig. 1, from statistics generated from the collected data the loop boundaries were determined and the most often used memory locations holding data referenced within the loop were determined. Those were placed in the GP microregisters and a new set of microinstructions were created which utilized a micro-operation stream equivalent to a register-to-register stream. After the loop was completed, a restore operation was performed. The synthesized microcode, along with the preload and restore operations, would then be called by a macro developed by the assembler or compiler. When this internal macro was called during program execution, the GP micro registers would be preloaded, the tuned microcode for the loop executed and the system restored for a contuation of the rest of the execution. This method has shown loop execution to be increased by a factor of 8 for a data movement program.

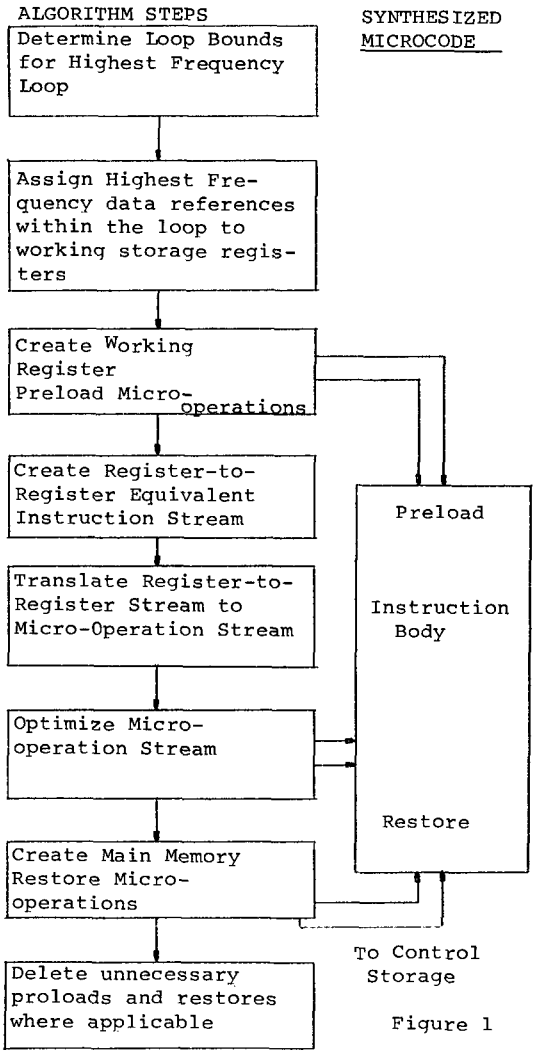

Figure 1

The primary drawbacks to utilizing this program in an on-line manner are that: (a) A trace program to generate usage data increases the amount of overhead for the program, (b) A program to generate the statistics must be run prior to selection of the data to be placed in the GP microregisters, (c) The synthesis is performed in software rather than by a microprogram.

3. PERFORMING THE TRACE AND GENERATING THE REQUIRED STATISTICS

If one can perform a trace which does not increase the program exectuion overhead and can generate the statistics as the program is operating, then two major hurdles have been removed. This would allow the algorithm to be truly automatic. An approach to accomplishing this is to perform the trace and statistics calculation in hardware. This may seem very difficult at first but a relatively simple scheme for accomplishing this is described below.

It was learned that the average number of assembly language instructions in a program loop is 8 [2]. Therefore, if we examine every location as it is accessed and maintain a file of the last 16 or 24 locations used, we will encompass most program loops. These 16 or 24 file locations will contain the address of the instruction plus the address of any operands the instruction may utilize during its execution. The statistics that are required for each location accessed during the execution of the loop are the number of times the location is accessed, determination of whether that location contained an instruction or an operand, and the determination of whether or not it is a jump instruction.

This can be accomplished by the use of a content addressable memory (CAM) and a high speed random access memory (RAM) used in conjunction with the microstore. See fig.2.

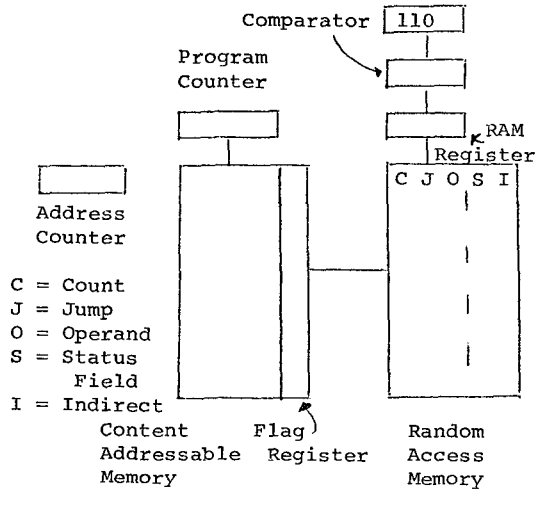

Figure 2a TRACE HARDWARE

RAM FIELD	FUNCTIONAL DESCRIPTION
C	If Flag set, set count to 1
J	Set to 1 if a jump instruction
O	Set to 1 if an operand
*	If comparison true adjust mapper to execute from synthesizer routine

FIGURE 2b Random Access Memory

The basic approach is as follows: as the computer reads an instruction from memory, the Content Addressable Memory is simultaneously searched using the location being addressed, i.e., contents of program counter as a target.

If there is a match, a flag in the CAM would be set corresponding to a repeated location. For each CAM word there is a corresponding word in the random access memory. Contained in that word is: (a) the count or whether that location has been recently addressed, (b) whether or not it is a jump instruction, (c) whether it is an instruction or an operand, (d) whether the instruction contains an indirect address, and (e) computer status information. The accessed RAM data is compared to 110 (count, jump and no operand) and if it is equal, the synthesis phase may be initiated. If it is not equal to 110, the count field is set to 1 and the word is stored back in the RAM location.

If there is no match, then the next available space in the CAM is loaded with that addressed location. (This space is determined by the CAM/RAM address counter which is modulo the number of words in the CAM.) The corresponding location in the RAM has its count set to 0 and its jump, operand, indirect and status locations set accordingly.

Whether the accessed location is an instruction or operand location can be determined by the computer phasing. To determine if it is a jump instruction or if the instruction contains an indirect either the instruction decoder has to set a flag or it can be determined in the microcode. The status information can be determined by examining the pertinent flip-flops and registers. This hardware will then generate the loop statistics required for synthesis. Notice that the above hardware will require the loop to be computed twice with the standard microcode before the synthesis phase of the tuning process begins.

4. PERFORMING THE SYNTHESIS

The synthesis procedure begins when the comparison of the data in the random access memory matches 110. The jump instruction is completed and the address of the next instruction is compared with contents of the CAM. If there is a match then the

program is placed under the control of the synthesis microprogram.

There are various levels of synthesis that can be performed. One method is a complete synthesis as performed in [1], where operand data is placed in microregisters. This required a new microprogram to be written for each instruction that utilized that data.

Another method is to place items in microregisters as before but only synthesize the instructions that would provide a significant improvement ratio such as double memory access instructions like an increment instruction. In this way, less overhead would be spent in synthesizing the loop and the improvement ratio would not be seriously reduced.

A third method that reduces the synthesis overhead greatly but also reduces the improvement ratio is to synthesize a pointer list that points to the location of the beginning of the microprogram for that particular instruction. See fig. 3.

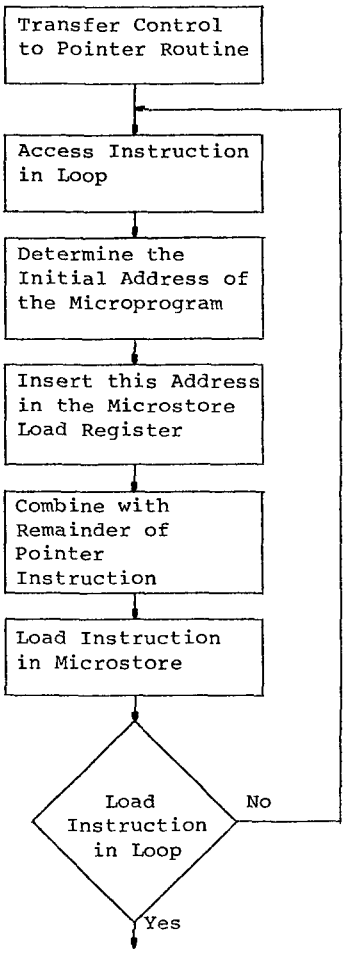

Figure 3 Synthesis Algorithm

In this approach, the instruction access time is reduced by the difference of the speeds between the main memory and the microstore. The savings here is in the synthesis time. Only pointers have to be implemented. A complete new routine does not have to be developed.

To load new instructions into the microstore requires 3 or 4 machine cycles. (A modified HP2100A, a machine that has a writable control store, was used as the basis for the numerical estimates.) The calculation to modify each of the instructions and to set up the microregisters with the data requires more specific calculations and a modification of each of the microprograms. On the HP2100A in real time, i.e. on-the-fly, this synthesis procedure is prohibitive. Therefore, the method chosen as most practical to examine through simulation is the third or pointer method.

5. IMPROVEMENT USING THIS TECHNIQUE

The major disadvantages of this scheme are the time required to actually perform the synthesis and the restore for continuation and the additional hardware required for statistics generation. The question immediately arises as to the trade-offs in the implementation. If it can be shown to be thruput effective then the additional hardware is justifiable. To determine this requires some analysis, a detailed simulation of the scheme and an investigation into typical loop profiles.

To begin with a determination of the cross-over point between performing the algorithm and not performing the algorithm in an operating situation is required.

A simple analysis to determine the cross-over point is presented below. Let us assume for simplicity that the time to perform the synthesis is directly proportional to the number of instruction in the loop.

Let t_n = time required to perform the synthesis; the loop is executed once as the synthesis is being performed.

Let t_1 = time to execute the loop once using unsynthesized instructions.

Let t_2 = time to execute the loop once using the synthesized instructions.

Also assume $t_n = bt_1$ where $b > 1$ and $t_1 = at_2$ where $a > 1$.

Let y = number of cycles through the loop.

Now let us examine two specific cases:

CASE 1

y = 2 Since two cycles are required (1) before the synthesis begins, then no time is gained or lost.

CASE 2

$y > 2$ The time to execute the loop y times with no tuning is $t_1 y$ and the time to execute the loop y times using the algorithm is (2)

$$2t_1 + t_n + t_2 (y - 3). \quad (3)$$

To determine the break-even point:

$$t_1 y = 2t_1 + t_n + t_2 (y - 3) \quad (4)$$

$$\text{or} \quad y = 2 + \frac{ab - 1}{a - 1} \quad (5)$$

To determine the values a and b will require a simulation. Let us take an example. Let the execution improvement be two hence a = 2 and the time to synthesize relative to regular loop operation be a factor of 5 hence b = 5 then

$$y = 2 + \frac{2 \cdot 5 - 1}{2 - 1} = 11 \text{ times} \quad 6)$$

through a loop before improvement occurs. So if the average number of times through a loop (which references a number of locations in the CAM) is greater than 11 the method is useful.

The simulation of the pointer synthesis technique on the HP2100A was performed for two programs: a data move program in which data is moved from one area of core memory to the other, and a linear search program where the target is sequentially stepped through the memory locations containing the data being searched. The basic timing results are shown in fig. 4 and 5. As one can observe in both cases, the crossover

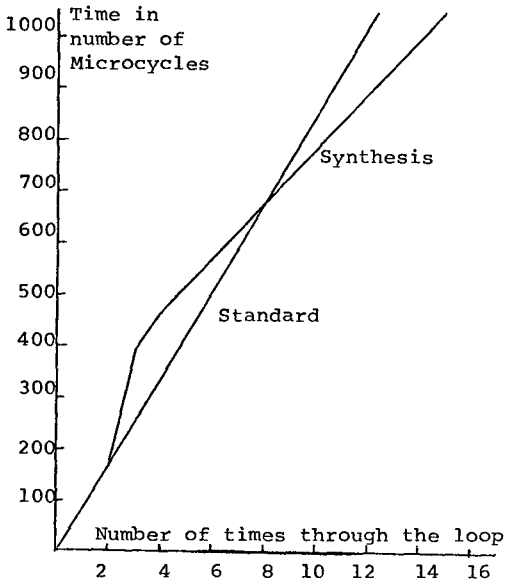

FIGURE 4; Number of microcycles versus the number of times through the loop for a data move program. There was a 55% improvement per loop using the synthesis technique.

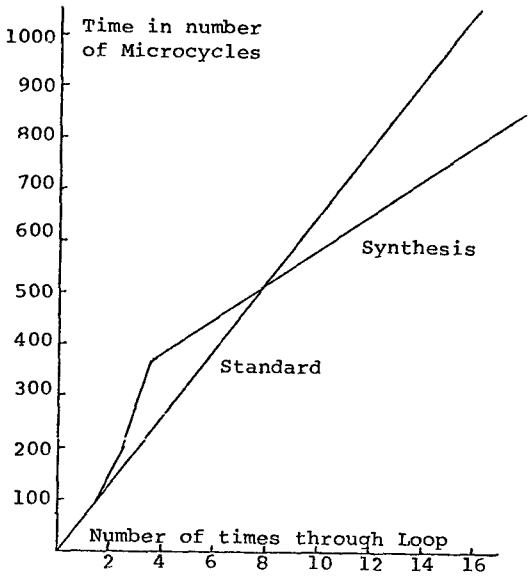

FIGURE 5; Number of microcycles versus the number of times through the loop for a linear search program. There was 86% improvement per loop using the synthesis technique.

between the standard HP2100A and the HP 2100A with the tuning technique employed is 8 times through the loop. The percent improvement in per loop performance is 55 with the data move and 86 with the linear search. Notice in the linear search simulation the entire point set phase had not been complete when the target had been located. Likewise, although to a lesser extent, with the data move because the return jump was not executed. Making linear approximations to the graphs at the synthesis points and placing those approximations into the analytical equation yields a crossover of 7.96 loops for the data move program and 7.95 for the linear search program.

An additional effort should be in eliminating the data fetch from memory for repeated operands. This would improve the crossover point between the standard loop and the synthesized loop if the overhead could be kept at a minimum.

To determine the usefulness of these methods, typical loop profiles in programs should be analyzed. This would include the determination of the "average" number of instructions per loop and the "average" number of times a single loop is executed in a given environment.

If the "average" number of time through a small loop is large then the synthesis procedure is a useful technique because there would be a definite thruput improvement.

REFERENCES

[1] Abd-alla, Dr. A.B. and Karlgaard, Dr. D.C., "Heuristic Synthesis of Micro-programed Computer Architecture," IEEE Transactions of Computers, vol. C-23, no. 8, August 1974, 802-807.

[2] Meltzer, Dr. A.C. Private Communication