Estimation of Execution Times of Process Control Algorithms on Microcomputers

R. G. RAJULU, MEMBER, IEEE, AND V. RAJARAMAN, SENIOR MEMBER, IEEE

Abstract—**An important question which has to be answered in evaluating the suitability of a microcomputer for a control application is the time it would take to execute the specified control algorithm. In this paper, we present a method of obtaining closed-form formulas to estimate this time. These formulas are applicable to control algorithms in which arithmetic operations and matrix manipulations dominate. The method does not require writing detailed programs for implementing the control algorithm. Using this method, the execution times of a variety of control algorithms on a range of 16-bit mini- and recently announced microcomputers are calculated. The formulas have been verified independently by an analysis program, which computes the execution time bounds of control algorithms coded in Pascal when they are run on a specified micro- or minicomputer.**

Key Words and Phrases—**Microcomputer evaluation, formulas for calculating execution time bounds, and process control algorithms.**

I. INTRODUCTION

THE OBJECTIVE of this paper is to present a method of computing the execution time bounds of control algorithms on microcomputers. As real-time systems are characterized by strict timing constraints, it is useful to obtain this time bound to enable a designer to choose an appropriate microcomputer for a given application. It would be desirable to obtain this time bound without writing the control program and executing it.

In the literature [7], [11], methodologies of evaluating microprocessors based on their arithmetic speeds, word length, and other hardware characteristics have been discussed. Farrar and Eidens [8] have developed and evaluated analytical procedures for establishing microprocessor accuracy, computational capability, and memory requirements for implementing linear quadratic Gaussian Control logic on Intel 8080 and LSI 11/2. To the best of our knowledge, no analytic method of evaluating the execution time bound for a class of control algorithms on a set of micro- and minicomputers has been reported. As our aim is to obtain execution time bounds of control algorithms for a class of microcomputers, we have assumed that the algorithms would be coded in a higher level language and translated. Specifically, we have assumed that the control algorithms would be coded in Pascal, translated to the P-code of the hypothetical stack machine by the P-compiler [2], [10], and that the P-code would then be translated to the machine language of an individual microcomputer. Keeping a library of machine language codes corresponding to each P-code instruction for a variety of microcomputers would facilitate software development for such a group of microcomputers [9].

In Section II, formulas for computing execution time bounds are derived for several control algorithms such as PID-controller algorithms, Kalman algorithms, Dahlin algorithms, Dead-beat algorithms, and finite time settling controller algorithms [13]. All these formulas are given in terms of execution times of arithmetic operations, data movements, and procedure linking for a specified microcomputer. The formulas are derived by inspection of the controller algorithms and without writing programs for implementing these algorithms.

In Section III, a closed-form formula for estimating the execution time bound for an on-line identification algorithm has been derived. This formula gives the execution time as a function of arithmetic speed, variable update time, time required for subscript manipulation, and overhead to set up and execute program loops in a specified mini- or microcomputer. In this case, also, no actual program for the application is written.

In Section IV, we briefly discuss another method of computing the execution time bound based on writing Pascal application programs. It is seen that the analytical method derived in this paper gives time bounds which are very close to that given by this method. In the last section, we conclude that for a large class of control algorithms reliable estimates of execution time bounds may be obtained analytically.

II. CONTROL ALGORITHMS AND DIGITAL FILTERS

This class of problems does not involve iterations or transcendental functions. For this class of applications, the number of additions, multiplications, and the variables to be updated determine the execution time bound. The number of multiplications, additions, and updates can be arrived at from the control algorithms, when it is given as a ratio of two polynomials in z. This is explained with respect to the following example. A typical control algorithm [14] is given by

$$D(z)=\frac{M(z)}{E(z)}=\frac{1-0.2238z^{-1}}{0.6535-0.5308z^{-1}-0.1227z^{-2}}. \quad (1)$$

Manuscript received August 31, 1981; revised March 23, 1982. This work was partially supported by funds from the project "Manpower Training for the Design of Microprocessor-based Systems," granted by the Electronics Commission, Government of India.

R. G. Rajulu is with the Elctrical Engineering Department, Regional Engineering College, Warangal, A.P., India.

V. Rajaraman is with the Computer Centre, Indian Institute of Science, Bangalore 560 012, India.

0278-0046/84/0200-0056$01.00 © 1984 IEEE

This control algorithm in the time domain is given below:

$$m_n = (e_n - 0.2238\,e_{n-1} + 0.5308\,m_{n-1} + 0.1227\,m_{n-2})/0.6535$$
$$= 0.153\,(e_n - 0.2238\,e_{n-1} + 0.5308\,m_{n-1} + 0.1227\,m_{n-2}) \quad (2)$$

where

m_n controller output at the nth instant,
m_{n-1} controller output at the $(n-1)$th instant,
m_{n-2} controller output at the $(n-2)$th instant,
e_n error at the nth instant,
e_{n-1} error at the $(n-1)$th instant,
$e_{n-1}, m_{n-2}, m_{n-1}$, and m_n to be updated.

From (2), we can see by inspection that the number of variables to be updated is 4, the number of multiplications is 4, and the number of additions/subtractions is 3.

For example, for Dahlin's second-order algorithms [5], the controller output m_n at the nth sampling instant is given by

$$m_n = a_1 e_{n-1} - a_2 e_{n-2} + a_3 e_{n-3} - a_4 m_{n-1} - a_5 m_{n-2} + a_6 m_{n-3} + a_7 m_{n-4}. \quad (3)$$

This equation involves 6 addition/subtractions, 7 multiplications, and 8 variables to be updated. The 8 variables to be updated are m_n, m_{n-1}, m_{n-2}, m_{n-3}, m_{n-4}, e_{n-1}, e_{n-2}, and e_{n-3}. As another example, consider the equations of the notch filter given by Cadzow [3]. They are

$$y_1(k) = U(k) + b_1 U(k-1) + U(k-2) - a_1 y_1(k-1) - a_2 y_2(k-2) \quad (4)$$

$$y_2(k) = y_1(k) + b_1 y_1(k-1) + y_1(k-2) - a_3 y_2(k-1) - a_4 y_2(k-2) \quad (5)$$

$$y_3(k) = y_2(k) + b_1 y_2(k) + y_2(k-2) - a_5 y_3(k-1) - a_6 y_3(k-2). \quad (6)$$

The above equations involve 12 addition/subtractions and 9 multiplications. The variables to be updated are 11. They are $U(k-1)$, $U(k-2)$, $y_1(k)$, $y_1(k-1)$, $y_1(k-2)$, $y_2(k)$, $y_2(k-1)$, $y_2(k-2)$, $y_3(k)$, $y_3(k-1)$, and $y_3(k-2)$. Table I gives a number of algorithms and the number of additions/subtractions, multiplications, and updates for each of the applications, along with the references.

As was pointed out in the introduction, we make the assumption that the algorithms would be coded in Pascal and translated into the P-code of the hypothetical stack machine [10]. The P-code is then translated into machine codes of individual microprocessors. In the Appendix, we give a list of P-code instructions pertinent to this paper.

TABLE I
DIGITAL CONTROL ALGORITHMS AND THEIR PARAMETERS

Algorithm	Additions n	Multiplications m	Updates k	Reference
Kalman I order with lag + deadtimes	3	4	5	[5]
Kalman II order with lag + deadtimes	4	5	6	[6]
Dahlin I order with lag + deadtimes	5	6	6	[5]
Dahlin II order with lag + deadtimes	5	6	6	[5]
Dahlin I order with ringing elimination	3	4	4	[14]
Deadbeat I order	4	5	5	[14]
Deadbeat II order	6	7	8	[14]
PID controller position form	4	3	3	[14]
PID controller velocity form	2	3	3	[14]
FTSC with FTSO	10	11	9	[1]
Notch filter	12	9	11	[3]
Butter-worth LPF	8	9	9	[3]
Optimal sampled-data control system	12	14	2	[4]

TABLE II
EXECUTION TIMES OF SOME P-CODE INSTRUCTIONS ON A SET OF 16-BIT COMPUTERS

Operation	P-Code Instructions	Execution time in microseconds					
		TDC316	LSI-11	PDP 11/40	TI9900	Z8000	8086
Addition	ADI T_1	5.2	7.7	3.4	10.9	5.7	6.6
Multiplication	LDCI	5.2	6.6	3.3	16.9	3.0	2.8
	LDOI	6.6	9.4	3.9	15.4	3.5	5.4
	MPI	22.6	74.8	14.3	99.2	21.5	34.2
	T_2	34.4	90.8	21.5	131.5	28.0	42.4
Update	LDOI	6.6	9.4	3.9	15.3	3.5	5.4
	SROI	6.6	8.8	4.0	12.6	4.0	5.6
	T_3	13.2	18.2	7.9	27.9	7.5	11.0
Procedure Call	ENT	6.2	8.4	3.3	12.0	2.5	1.2
	RET	17.4	27.3	9.4	29.6	6.2	9.8
	T_4	23.6	35.7	12.7	41.6	8.7	11.0

In order to carry out a multiplication, the P-code instructions LDCI, LDOI, and MPI are required. Addition is carried out by the instruction ADI. To update a variable value in memory, the P-instructions LDOI and SROI are needed. The execution time of each P-code instruction for a specified microprocessor is obtained by expanding the P-code instruction to a set of machine instructions needed for that processor. The P-code instructions' execution times for a variety of 16-bit microprocessors are given in [12]. Table II gives the execution times for addition, multiplication, update, and procedure call. Using these times, the following formula for the execution time bound for simple control algorithms given in Table I may be written as

$$\text{Execution Time Bound} = nT_1 + mT_2 + kT_3 + T_4 \quad (7)$$

where n is the number of additions, m is the number of multiplications, k is the number of updates, T_1 is the addition time, T_2 the multiplication time, T_3 update time, and T_4 is the time to call the control procedure.

Using (7), Table I, and Table II, the execution times of all the control algorithms in Table I for all the processors listed in Table II may be obtained. By inspection of Table II, it may be seen (based on the published data on Z8000) that Z8000 would perform almost as well as PDP 11/40, a 16-bit minicomputer.

III. ON-LINE IDENTIFICATION ALGORITHM

Sinha *et al.* [13] gave two methods for on-line identification. These are the recursive least-squares method and the boot-strap method. As the recurrence relations for the two algorithms have the same structure, we discuss below the recurrence formula for the least-squares method. We obtain a closed-form formula for the execution time based on an analysis of this algorithm.

The recurrence formulas are given by

$$\underset{\sim}{\phi}_{k+1} = \underset{\sim}{\phi}_k + \frac{P_k \underset{\sim}{a}_{k+1}[y_{k+2} - \underset{\sim}{a}_{k+1}{}^T \underset{\sim}{\phi}_k]}{1 + \underset{\sim}{a}_{k+1}{}^T P_k \underset{\sim}{a}_{k+1}} \tag{8}$$

and

$$P_{k+1} = P_k - \frac{P_k \underset{\sim}{a}_{k+1}[P_k \underset{\sim}{a}_{k+1}]^T}{1 + \underset{\sim}{a}_{k+1}{}^T P_k \underset{\sim}{a}_{k+1}} \tag{9}$$

where $\underset{\sim}{\phi}_k$ and $\underset{\sim}{a}_{k+1}$ are column vectors with n components, n is the order of the system, and P_k is a $n \times n$ matrix. From the above two recurrence relations, we see that $P_k \underset{\sim}{a}_{k+1}$ occurs in both the expressions, so that it can be evaluated and stored in a column vector. The complexity of computation of $P_k \underset{\sim}{a}_{k+1}$ may be determined by inspecting the nature of the algorithm which would compute it. The algorithm would be of the type

```
for i: = 1 to n do
    for j: = 1 to n do
        b[i]: = b[i] + P[i,j] * a[j];
```

For the above algorithm, we conclude that we would perform n^2 additions, n^2 multiplications, n^2 calculations of double subscripts of $P(i,j)$, n^2 single subscript computations of a_j and n single subscript computations of b_i, n updates of b_i, and the overheads of nested for loops. Thus, the number of operations may be estimated as

$$n^2 \underline{p} + n^2 \underline{a} + n^2 \underline{ds} + (n^2 + n) \underline{ss} + n \underline{u} + \underline{fn} \tag{10}$$

where $\underline{p}$ stands for multiplication, $\underline{a}$ for addition, $\underline{ds}$ for double subscript computation, $\underline{ss}$ for single subscript computation, $\underline{u}$ for updates, and $\underline{fn}$ for a nested for loop overhead.

Next, we determine the number of operations necessary to compute $(1 + \underset{\sim}{a}_{k+1}{}^T P_k \underset{\sim}{a}_{k+1})$. This expression occurs in both (8) and (9), but we will compute it only once. Further, $P_k \underset{\sim}{a}_{k+1}$ calculated already will be available as a column vector and would not be recomputed. Thus, $\underset{\sim}{a}_{k+1}{}^T P_k \underset{\sim}{a}_{k+1}$ would be computed with a for loop:

```
for i: = 1 to n do B: = B + b[i] * a[i];
```

From this, we can estimate the number of operations as

$$n\underline{p} + n\underline{a} + 2n\underline{ss} + \underline{u} + \underline{f} \tag{11}$$

where $\underline{f}$ is the overhead for a single for loop.

Having already found and stored $P_k \underset{\sim}{a}_{k+1}$ and $(1 + \underset{\sim}{a}_{k+1}{}^T P_k \underset{\sim}{a}_{k+1})$, $\underset{\sim}{\phi}_{k+1}$ is computed by computing $\underset{\sim}{a}_{k+1}{}^T \underset{\sim}{\phi}_k$ first which would involve n multiplications, n additions, $2n$ subscript computations, and one for loop overhead. Next, we subtract $\underset{\sim}{a}_{k+1}{}^T \underset{\sim}{\phi}_k$ from y_{k+2} and divide by the denominator and store the result in a variable C. Lastly, we multiply the (already stored) values of $P_k \underset{\sim}{a}_{k+1}$ by the value obtained in the previous step, and add this vector to $\underset{\sim}{\phi}_k$. This would be done by the loop

```
for i: = to n do φnew[i]: = φ[i] + b[i] * C;
```

in which the number of operations performed is n multiplications, n additions, $3n$ subscript computations, n updates, and one for loop. Adding all the operations in each of the steps we get for computing $\underset{\sim}{\phi}_{k+1}$ the number of operations as

$$2n\underline{p} + 2n\underline{a} + \underline{s} + \underline{d} + 5n\underline{ss} + n\underline{u} + 2\underline{f} \tag{12}$$

where $\underline{s}$ is the subtraction and $\underline{d}$ is the division operation.

Having already stored $P_k \underset{\sim}{a}_{k+1}$ and the denominator of (9), the number of additional operations in computing P_{k+1} in (9) is given by

$$2n^2 \underline{p} + n^2 \underline{s} + \underline{d} + 2n^2 \underline{ss} + 2n^2 \underline{ds} + n^2 \underline{u} + \underline{fn}. \tag{13}$$

Adding expression (10), (11), (12), and (13), we obtain

$$(3n^2 + 3n)\underline{p} + (n^2 + 3n)\underline{a} + (n^2 + 1)\underline{s} + 2\underline{d} + 3n^2 \underline{ds} + (3n^2 + 8n)\underline{ss} + (n^2 + 2n + 1)\underline{u} + 3\underline{f} + 2\underline{fn}. \tag{14}$$

The next step is to find the execution time estimates for each of the operations $\underline{p}, \underline{a}, \underline{s}, \underline{d}, \underline{ds}, \underline{ss}, \underline{u}, \underline{f}$, and $\underline{fn}$. The times for floating point arithmetic operations and subscript computations are directly obtained from the characteristics of a specified microprocessor. These are listed in Table III.

The times for the for loop and nested for loop overheads are obtained below.

The time for the operation of for loop is given by the formula

$$\underline{f} = A + Bn \tag{15}$$

where A and B are constants depending upon the processor, and n is the number of iterations of the for loop. The constant A is a fixed overhead incurred in initiating a for loop and B is the overhead incurred in each iteration. Similarly, the time for executing nested for loops is given by

$$\underline{fn} = A + (A + B)n + Bmn \tag{16}$$

where m is the number of iterations of the outer for loop and n that of the inner for loop.

TABLE III
EXECUTION TIMES FOR FLOATING POINT ARITHMETIC, SUBSCRIPT EVALUATION, AND LOOP OVERHEADS

Processor	MULT	ADD	SUB	DIV	DS	SS	UPDATE	A	B
TDC-316	1330.7	591.0	835.4	2158.3	110.4	68.2	20.8	39.2	94.0
LSI-11	2650.7	1419.4	2009.4	4037.7	244.5	140.3	28.0	54.6	137.2
LSI-11+	79.1	47.0	47.3	155.9	244.5	140.3	28.0	54.6	137.2
PDP-11/40	31.4	21.22	21.2	49.2	67.7	41.6	12.4	23.8	56.7
TI9900	1707.6	1645.0	2345.0	6348.1	212.4	135.5	37.3	87.2	187.5
Z8000	376.1	232.0	353.6	420.0	93.0	53.2	10.5	25.5	62.5
8086	654.4	534.0	731.0	980.0	112.0	63.8	12.2	22.4	61.8

+LSI-11 with floating point option

All times are in microseconds.

A for loop is implemented using the P-code instructions LDCI, STRI, LDCI, and STRI before the initiation of the loop, and the instructions LODI, LODI, LEQI, and FJP at the beginning of each iteration of the loop and instructions LODI, INC, STRI, and UJP at the end of the loop body in each iteration. (The meaning of these P-code mnemonics is given in the Appendix.) Thus, the values of A and B of (15) are obtained as

$$A = 2T\ (\text{LDCI}) + 2T\ (\text{STRI}) \tag{17}$$

where T(LDCI) is the time to execute the instruction LDCI and T(STRI) that to execute the instruction STRI. Similarly, we obtain for B

$$B = 3T\ (\text{LODI}) + T\ (\text{LEQI}) + T\ (\text{FJP}) + T\ (\text{INC}) + T\ (\text{STRI}) + T\ (\text{UJP}). \tag{18}$$

The times for A and B for various processors is given in Table III. These values are also used in computing the execution time for fn of (16).

IV. VERIFICATION OF RESULTS

Another method of computing the execution time bound of control algorithms would be to write the Pascal application program corresponding to the algorithms and estimate the execution time from this. As our aim is to obtain the time bounds for a variety of processors, the time computation should be obtained without actually executing the program. A methodology to do this is suggested in [12]. The method suggested is to add features to the Pascal P-compiler, which would give, along with the P-code, a linear list with embedded words of the Pascal program. This list is designed to enable computation of the execution time bound. This list, along with the loop parameters of the Pascal program and P-code instruction execution times on a specified microprocessor, is used by an analysis program to compute the execution time bound. This analysis program was implemented on the DEC 1090 system and is reported in [12]. This analysis program was run after coding in Pascal all the control algorithms examined in this paper. The deviation between the execution time as given by the closed-form formulas of this paper and that obtained from the analysis program was less than 5 percent.

V. CONCLUSIONS

A class of controller algorithms and digital filters have the same structure and do not involve iterative calculations or transcendental functions. In these cases, we can find the number of arithmetic operations and the number of variables to be updated by inspection of the controller expression. Thus, it is possible to obtain a simple formula to estimate the execution times of these algorithms on a set of mini- and microcomputers.

In the case of the recursive least-square type identification algorithm, execution time formulas are obtained keeping in view the way these algorithms would be programmed in a higher level language. These algorithms involve recurrence relations, and thus would be programmed using for and nested for loops. Use of such loops contributes significantly to execution time. In this case, also, detailed programs for the algorithms need not be written to estimate execution time bounds. The methodology of this paper is applicable for control algorithms in which arithmetic operations and matrix manipulations dominate.

APPENDIX
SOME SELECTED P-CODE INSTRUCTIONS

mnemonic	*description*
ADI	Integer addition.
ADR	Real addition.
CSP	Call standard procedure.
CUP	Call user procedure.
DEC	Decrement address.
DVI	Integer division.
DVR	Real division.
ENT	Enter block.
EQU	Compare on equal.
FJP	False jump.
GEQ	Greater or equal.
GRT	Greater than.
INC	Increment address.
IND	Indexed fetch.
IXA	Computer indexed address.
LAD	Load base-level address.
LCA	Load address of constant.
LDA	Load address.
LDCI	Load constant (integer).
LDOI	Load contents of base-level address (integer).
LEQI	Less than or equal (integer).
LES	Less than.
LODI	Load contents of address (integer).
MOV	Move.
MPI	Integer multiplication.
MPR	Real multiplication.
RET	Return from block.
SBI	Integer subtraction.
SBR	Real subtraction.
SROI	Store (integer).
STOI	Store at base-level address.
STP	Stop.

STRI	Store at address (integer).
UJP	Unconditional jump.
XJP	Indexed jump.

REFERENCES

[1] D. M. Auslander, T. Yasundo, and T. Masayoshi, "Direct digital process control: Practice and algorithms for microprocessor applications," *Proc. IEEE,* vol. 66, pp. 199–208, Feb. 1978.

[2] K. L. Bowels, "A brief description of the UCSD PASCAL software system," *Newsletter, Inst. Inform. Syst.*, Univ. California, San Diego, pp. 1–8, June 1978.

[3] J. A. Cadzow, *Discrete-Time Systems: An Introduction with Interdisciplinary Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1973.

[4] G. K. L. Chien, "Computer control in process industries," *Contr. Syst. Tech. Rep. No. 1,* IBM General Products Division, Nov. 1960.

[5] K. C. Chiu, A. B. Corripion, and C. L. Smith, "Digital control algorithms, Part I, Dahlin algorithms," *Instrum. Contr. Syst.*, pp. 55–58, Nov. 1973.

[6] K. C. Chiu, A. B. Corripio, and C. L. Smith, "Digital control algorithms, Part II, Kalman algorithms," *Instrum. Contr. Syst.*, pp. 55–58, Nov. 1973.

[7] H. A. Davis, "Comparing architectures of three 16-bit microprocessors," *Comput. Des.*, pp. 91–100, July 1979.

[8] F. A. Farrar and R. A. Eidens, "Microprocessor requirements for implementing modern control logic," *IEEE Trans. Automat. Contr.*, vol. AC-25, no. 3, pp. 461–468, June 1980.

[9] C. A. Irvine, "UCSD system makes programs portable," *Electron. Des.*, pp. 113–118, Aug. 1980.

[10] K. V. Nori, U. Ammann, K. Jensen, and N. H. Nageli, "The Pascal P-compiler implementation notes," in *Pascal—The Language and its Implementation,* D. W. Barron, Ed. New York: Wiley, 1981, ch. 9.

[11] B. K. Penny, "The implications of microprocessor architecture on speed, programming and memory size," *Radio Electron. Eng.*, vol. 47, no. 11, pp. 522–528, Nov. 1977.

[12] R. G. Rajulu and V. Rajaraman, "Execution-time analysis of process control algorithms for microprocessors," *IEEE Trans. Ind. Electron.*, vol. IE-29, no. 4, pp. 312–319, Nov. 1982.

[13] N. K. Sinha, A. Sen, and J. D. Wright, "On-line identification of a chemical process using a minicomputer," *IEEE Trans. Ind. Electron. Contr. Instrum.*, vol. IECI-23, no. 3, pp. 314–317, Aug. 1976.

[14] C. L. Smith, *Digital Computer Process Control*. Intext Educational Publishers, 1972.