INTELLIGENT MOTION CONTROL

Proceedings of the IEEE International Workshop on Intelligent Motion Control

Boğaziçi University, Istanbul, Turkey
20 - 22 August 1990

Cosponsored by: IEEE Industrial Electronics Society
In Cooperation with: IEEE Robotics and Automation Society
Hosted by: Boğaziçi University

Volume II

Editor/Chair
Okyay KAYNAK

Catalog Number: 90 TH0272-5

IEEE International Workshop on Intelligent Motion Control
Boğaziçi University, Istanbul, 20-22 August 1990.
Edited by Okyay KAYNAK. IEEE Cat. No: 90TH0272-5

COMPUTER SYSTEM EVALUATION IN ROBOT CONTROL

J.A. Tenreiro Machado, J.L. Martins de Carvalho and Alexandra M.S.F. Galhano
Faculty of Engineering of the University of Porto
Dept. of Electrical and Computer Engineering
4099 Porto Codex, Portugal

Abstract: The high computational burden posed by modern control algorithms preclude its industrial application using present day microcomputers. In this paper we evaluate the computational load of different logical and arithmetic operations and the capabilities of several computing systems (software & hardware). This study reveals that real-time limitations can be alleviated through the adoption of general techniques associated with the data representation. Such techniques achieve not only a more efficient management of the computational resources but also provide a deeper insight on developments towards future real-time control architectures.

Introduction

Since the advent of robotic technology manipulator control has been an area of active research. These mechanical devices composed of several links connected through revolute or prismatic joints, have complex dynamic phenomena which make difficult the development of efficient controllers. Although linear PID controllers are still used in present day industrial robots they are inappropriate for high performances. In fact, PID systems lead to limited path tracking accuracy and may exhibit vibrations at high speeds. The low efficiency of these systems motivated the appearance of controllers based on different concepts [1-5]. However, the high computational burden posed by many of these algorithms precludes its industrial application using present day microcomputers. While powerful monoprocessor controllers may be non-economical, the alternative of a multi-microprocessor architecture [6-10] is still in a research stage and the total computational efficiency is probably far from desirable. This situation can be overcome through the development of control strategies more adapted to microcomputer-based structures. In fact, the greater the controller complexity the greater the computational requirements. It is then appropriate to question to what extent is it feasible to implement a given algorithm and which are the most adequate techniques to do the job. Furthermore, the technical literature is scarce on the evaluation of the computational load posed by each algorithm and its dependence on the software & hardware structure.

This paper compares the real-time capabilities of several computer systems and introduces techniques capable of render more efficient practical implementations. In order to develop this study the article is organized as follows. In section two we evaluate the computational load with respect to different logical and arithmetic operations of several computing systems (software & hardware). Section three presents general techniques amenable to real-time implementations. Finally, in section four conclusions are drawn.

Evaluation of Computational Load and Software & Hardware Capabilities

The evaluation of the computational load required by an algorithm and the capabilities of a given computational system, are essential steps in any preliminary study regarding its future implementation.

Many of the algorithms suggested in the literature neglect these points, and no assessment is made about computational requirements. Some studies take into account the computational load based on the required number of sums and multiplications, while others only mention the maximum sampling frequencies achieved after the algorithm has been installed in a given system. Obviously such approaches are far from satisfactory, because they do not take into account all the "factors" involved. The difficulty of the problem lies precisely in the large number of factors involved, such as the type of microprocessor, clock frequency, memory wait states, type and version of the compiler, type and accuracy of the operations, etc.

Although not considering all possible combinations of operations/compiler/processor/co- processor, the data displayed in Tables 1 to 4 attempts to clarify these issues [11]. They show the range of variation of the computational time required by each arithmetic or logical operation for a given system (software & hardware). Among the large number of possibilities we have chosen those combinations more relevant in controller implementation. Inspecting the data several conclusions can be drawn:

- Trigonometric operations are the most time consuming.
- Logical and integer arithmetic operations are the fastest.
- The arithmetic coprocessor is essential to speed-up floating point operations.
- Logical and integer arithmetic operations are not affected by the presence or absence of coprocessors.
- For a given hardware, large variations of computing time may occur depending on the compiler being used.
- Theoretically, the speed of calculations increases linearly with the clock frequency. However, for large frequencies wait states may occur when accessing memory. In this case the way memory is organized is the determining factor in the full use of that velocity [12].
- Reduced Instruction Set Computer (RISC) architectures appear to be far more efficient than conventional Complex Instruction Set Computers (CISC).

In order to provide a better perspective of these properties, we have decided to measure the frequency of calculation for an adequate benchmark. Because ext apolation from generic benchmarks are questionable [12-14], we have decided to select a benchmark capable of reflecting the requirements normally associated with robot control. Thus, we show in Fig. 1 the frequencies of calculation of the inverse dynamics for the six degrees of freedom PUMA 560 manipulator and for the systems mentioned in Tables 1 to 4. The inverse dynamics equations already include the simplifications presented in [15]. Some improvements introduced by the authors have reduced the number of required operations to 5 sines, 6 cosines, 89 sums and 151 multiplications. The results in Fig. 1 show that, by the time the whole control algorithm is implemented, which contains several sub-algorithms such as kinematics, dynamics, control and trajectory planning, the computational load can easily reach levels incompatible with the use of high sampling frequencies. The development of techniques for the real-time implementation of these algorithms is the matter of the next section.

Techniques to Improve the Real-Time Performances of Computer Control Systems

Modern robot control algorithms pose very stringent computational requirements. Although technological progress makes available systems with ever increasing performances, the truth is that their use as robot controllers may not be economically feasible. In the sequel we present a set of techniques to improve the

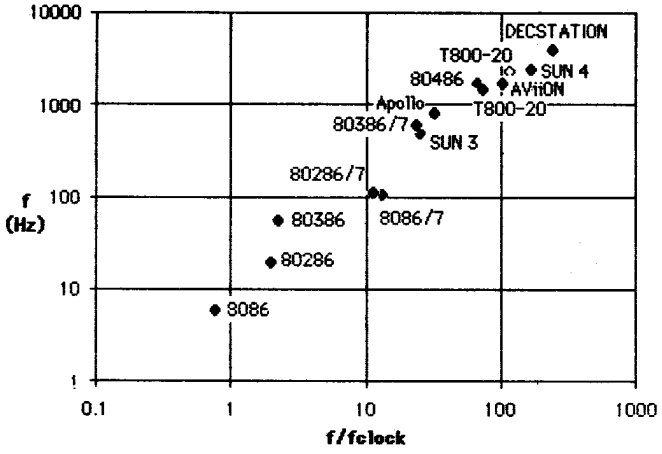

Fig. 1 Chart of the computing frequencies (f) for PUMA 560 dynamic equations versus the index f/f_{clock} using several computing systems considered in Tables 1 to 4. For the systems tested with two alternative compilers, it is depicted only the result of the fastest code. Floating point operations are evaluated using the standard precision of: ◆ 8 bytes, ◇ 4 bytes.

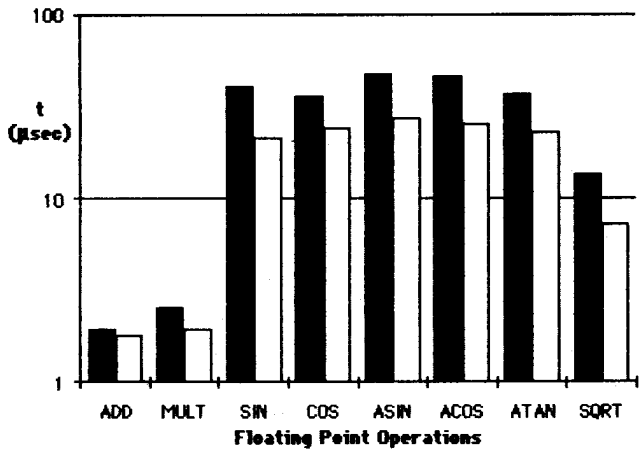

Fig. 2 Chart of the computing time (t) for several floating point operations in transputer T800-20 using:

◆ 8 bytes precision, ◇ 4 bytes precision

(source code programmed in Occam 2).

real-time performances of the control system, which are, to a large extent "hardware independent" [11]. We group these techniques into six categories:

- Assembly language programming.
- Low precision arithmetic calculations.
- Use of memory.
- Multirate schemes.
- Preview techniques.
- Dedicated compilers.

which are discussed in the sequel.

Assembly Language Programming

This is a well known technique and constitutes a natural starting point to improve the real-time performances of any controller. In fact direct programming in assembly language allows considerable optimization of the generated code. However, modern control algorithms are very complex and that makes their programming in assembly very time consuming. This is the reason why only a few researchers have adopted this strategy [16-18] as an alternative to high level languages such as FORTRAN or C.

Low Precision Arithmetic Calculations

This option has been one of the principal alternatives to the assembly language. The most common technique consists in giving up floating point calculations in favour of fixed point arithmetic [19-20].

At a first sight it would seem more practical to reduce only the accuracy of the floating point calculations. However, this is somewhat deceptive because many of the high level languages implement low accuracy floating point operations on the basis of operations with standard accuracy. Therefore, and contrary to our expectations, there is no improvement of the computation times. Clearly, if the high level language has distinct implementations for different accuracies then computation can be speeded up. This is shown in Fig. 2 for several floating point arithmetic operations, on the IMS T800-20 transputer, programmed in Occam 2.

Use of Memory

This method transfers, in part or entirely, the load of on-line calculations to memory-based evaluations. A common procedure consists of replacing the floating point calculations of trigonometric functions by memory tables [17,20]. Although this method can be generalized to a greater portion of the algorithm, the application of this technique to control algorithms has not received much attention to date with the exception of controllers based on learning strategies [21-26]. Therefore, a large field of research remains open.

Multirate Schemes

The effects of the sampling frequency upon the performances of a digital robot controller are very important. In a control system, made of several feedback and feedforward paths, it is natural to expect different speeds of response along them. Therefore, it is reasonable to allocate different sampling (and computing) rates to such paths; this forms the basis of multirate schemes [27-30]. In this way the computing power is assigned to each loop in accordance to its needs, allowing therefore a more rational management of the system resources.

Preview Techniques

Because microprocessors have limited computational capabilities the maximum allowable sampling frequency of the controller is bounded above. At the same time this gives rise to a time delay between the instant sensors are read and the control command is issued.

Preview techniques attempt to minimize this detrimental effect. Their application to the control of robot manipulators has shown to be simple and efficient [31-33].

Dedicated Compilers

The implementation of a control algorithm implicitly assumes the representation of process variables by means of real numbers. To these real numbers corresponds a computer internal representation by means of floating point structures. This representation, requiring several bytes, is in sharp contrast with the 8 to 16 bit accuracy of standard A/D and D/A converters.

The authors have developed a method for implementing real-time control algorithms [34-35] that eliminate

this discrepancy, by optimizing the chain:

Data collection -> Proccessing -> Control action

In their method, an "uniform" representation for the variables is adopted, that is, real world and computer internal variables have identical accuracies. Once the admissible range of a variable is known, quantization levels can be defined as a function of the accuracy of the A/D and D/A converters; to each of these levels, an integer binary code is assigned. Then the control algorithm can be processed by Boolean Algebra operations upon the bits representing the variables. The Boolean Algebra operations are optimized by means of Binary Decision Diagrams, that are very efficient, once converted to IF_THEN_ELSE structures as shown by Tables 1 to 4.

This method is still in a research stage; however, it has already suggested interesting extensions to RISC computational structures and parallel architectures. In the first case the extension is motivated by the fact that processing by means of BDD's only requires microprocessors with a reduced number of instructions; in the second case the advantage stems from the simplicity of task allocation to parallel processors.

Conclusions

A large number of algorithms for robot control has been proposed so far. However, the validation of these algorithms through practical implementations is still confined to a few examples. Moreover, the high computational burden posed by many of these algorithms precludes its industrial application using present day microcomputers. This situation can be overcome through the development of control strategies more adapted to microcomputer-based structures. The analysis of both the computational requirements and microcomputer capabilities reveals that limitations are alleviated through the adoption of general techniques associated with the data representation. Furthermore, the use of these techniques achieves not only a more efficient management of the computational resources but also provides a deeper insight on developments towards future real-time control architectures.

References

[1] A.K. Bejczy, "Robot Arm Dynamics and Control," Jet Propulsion Lab. Tech. Memo 33-669, 1974.

[2] S. Dubowsky and D.T. DesForges, "The Application of Model-Referenced Adaptive Control to Robotic Manipulators," J. Dynamic Syst. Measurement, Contr., Trans. ASME, vol. 101, pp. 193-200, 1979.

[3] J.Y.S. Luh, M.W. Walker and R.P.C. Paul, "Resolved-Acceleration Control of Mechanical Manipulators," IEEE Trans. Automat. Contr., vol. AC-25, pp. 468-474, 1980.

[4] E. Freund, "Fast Nonlinear Control with Arbitrary Pole-Placement for Industrial Robots and Manipulators," The Int. J. Robotics Research, vol. 1, pp. 65-78, 1982.

[5] R. Horowitz and M. Tomizuka, "An Adaptive Control Scheme for Mechanical Manipulators - Compensation of Nonlinearity and Decoupling Control," J. Dynamic Syst. Measurement, Contr., Trans. ASME, vol. 108, pp. 127-135, 1986.

[6] J.Y.S. Luh and C.S. Lin "Scheduling of Parallel Computation for a Computer-Controlled Mechanical Manipulator," IEEE Trans. Syst., Man, Cybern., vol. SMC-12, pp. 214-234, 1982.

[7] R. Nigam and C.S.G. Lee "A Multiprocessor-Based Controller for the Control of Mechanical Manipulators," IEEE J. Robotics and Automation, vol. RA-1, pp. 173-182, 1985.

[8] E.E. Binder and J.H. Herzog, "Distributed Computer Architecture and Fast Parallel Algorithms in Real-Time Robot Control," IEEE Trans. Syst., Man, Cybern., vol. SMC-16, pp. 543-549, 1986.

[9] T. Watanabe, M. Kametani, K. Kawata and K. Tetsuya, "Improvement in the Computing Time of Robot Manipulators Using a Multimicroprocessor," J. Dynamic Syst. Measurement, Contr., Trans. ASME, vol. 108, pp. 190-197, 1986.

[10] C.-H. Liu and Y.-M. Chen "Multi-Microprocessor-Based Cartesian-Space Control Techniques for a Mechanical Manipulator," IEEE J. Robotics and Automation, vol. RA-2, pp. 110-115, 1986.

[11] J.A. Tenreiro Machado and J.L. Martins de Carvalho, "Microprocessor-Based Controllers for Robotic Manipulators," in Microprocessors in Robotic and Manufacturing Systems (to be published) Kluwer Academic Publishers (Spyro Tzafestas, editor).

[12] K.T. Lua, "Relative Performance Measurement of 80386, 80286 and 8088 Personal Computer Systems," Microprocessing and Microprogramming, vol. 26, pp. 85-95, 1989.

[13] W.J. Price, "A Benchmark Tutorial," IEEE Micro, vol. 9, pp. 28-43, 1989.

[14] "Lies, damned lies and benchmarks' in The Transputer Applications Notebook: Systems and Performance, INMOS, pp. 258-279, 1989.

[15] B. Armstrong, O. Khatib and J. Burdick, "The Explicit Dynamic Model and Inertial Parameters of the PUMA 560 Arm," in Proc. 1986 IEEE Int. Conf. on Robotics and Automation, San Francisco, CA, 1986.

[16] E. Freund and H.J. Klein, "Practical Results With Nonlinear Control Strategy for Computer-Controlled Industrial Robots," in Proc. 14th ISIR, Gothenburg, Sweden, 1984.

[17] M.W. Spong, J.S. Thorp and J.M. Kleinwaks, "Robust Microprocessor Control of Robot Manipulators," Automatica, vol. 23, pp. 373-379, 1987.

[18] S.H. Murphy and G.N. Saridis, "Experimental Evaluation of Two Forms of Manipulator Adaptive Control," in Proc. 26th IEEE Conf. on Decision and Control, Los Angeles, CA, 1987.

[19] C.H. An, C.G. Atkeson, J.D. Griffiths and J.M. Hollerbach, "Experimental Evaluation of Feedforward and Computed Torque Control," in Proc. 1987 IEEE Int. Conf. on Robotics and Automation, Raleigh, NC, 1987.

[20] T. Suehiro and K. Takase, "A Manipulation System Based on Direct-Computational Task-Coordinate Servoing," Robotics Research: The Second International Symposium, MIT Press, 1985.

[21] J.S. Albus, "A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC)," J. Dynamic Syst. Measurement, Contr., Trans. ASME, vol. 97, pp. 220-227, 1975.

[22] J.S. Albus, "Data Storage in the Cerebellar Model Articulation Controller (CMAC)," J. Dynamic Syst. Measurement, Contr., Trans. ASME, vol. 97, pp. 228-233, 1975.

[23] M.H. Raibert, "A Model for Sensorimotor Control and Learning," Biological Cybernetics, vol. 29, pp. 29-36, 1978.

[24] G. Hirzinger, "Robot Systems Completely Based on Sensory Feedback," IEEE Trans. Indust. Electron., vol. IE-33, pp. 105-109, 1986.

[25] S. Kawamura, F. Miyazaki and S. Arimoto, "Realization of Robot Motion Based on a Learning Method," IEEE Trans. Syst., Man, Cybern., vol. 18, pp. 126-134, 1988.

[26] W.T. Miller III, "Real-Time Application of Neural Networks for Sensor-Based Control of Robots with Vision," IEEE Trans. Syst., Man, Cybern., vol. 19, pp. 825-831, 1989.

[27] D.P. Glasson, "Development and Applications of Multirate Digital Control," IEEE Control Systems Magazine, vol. 3, pp. 2-8, 1983.

[28] M.C. Berg, N. Amit and J.D. Powell, "Multirate Digital Control System Design," IEEE Trans. Automat. Contr., vol. 33, pp. 1139-1150, 1988.
[29] J.A.T. Machado and J.L.M. de Carvalho, "Engineering Design of a Multirate Nonlinear Controller for Robot Manipulators," J. Robotic Systems, vol. 6, pp. 1-17, 1989.
[30] M. Kircanski, M. Vukobratovic, N. Kircanski and T. Timcenko, "A New Program Package for the Generation of Efficient Manipulator Kinematic and Dynamic Equations in Symbolic Form," Robotica, vol. 6, pp. 311-318, 1988.
[31] C.D. Vinante, C. Bermudez and F. Tarre, "Predictive Compensation Implemented with a Microprocessor," IEEE Control Systems Magazine, vol. 6, pp. 40-43, 1986.
[32] K. Yoshimoto and K. Wakatsuki, "Application on the Preview Tracking Control Algorithm to Servoing a Robot Manipulator," Robotics Research: The First International Symposium, MIT Press, 1984.
[33] K. Yoshimoto and H. Sugiuchi, "Trajectory Control of Robot Manipulator Based on the Preview Tracking Control Algorithm," Robotics Research: The Second International Symposium, MIT Press, 1985.
[34] J.A.T. Machado, J.L.M. de Carvalho, J.A.S. Matos and A.M.C. Costa, "An Efficient Computational Scheme for Robot Manipulators," in Proc. IEEE Int. Symp. on Intelligent Control, Arlington, Virginia, 1988.
[35] J.A.T. Machado, J.L.M. de Carvalho, J.A.S. Matos and A.M.C. Costa, "Robot Manipulator Dynamics - Towards Better Computational Algorithms," in Proc. IFAC Symp. on Robot Control, Karlsruhe, FRG, 1988.

TABLE 1. Computation times for several systems

Computer	PS/2-8530-021		PS/2-8560-061		PS/2-8570-A21	
OS	MSDOS 3.3		MSDOS 3.3		MSDOS 3.3	
Clock	8 MHz		10 MHz		25 MHz	
Processor	8086		80286		80386	
Prog.Lang	TC V2.0	µC V4.0	TC V2.0	µC V4.0	TC V2.0	µC V4.0
Operation	Computation Time (µsec)					
ADD real	540-660	210-280	223-251	88-108	83-93	31-40
MULT real	870-940	540-610	293-299	157-161	110-114	55-59
SIN	3500-4300	6600-7450	939-1175	1939-2241	335-418	692-807
COS	3400-4400	7100-7700	907-1181	2236-2329	324-418	802-846
ASIN	6580-9560	7650-10700	1856-2802	2307-3394	664-1016	840-1231
ACOS	7240-9780	7950-10450	2131-2906	2428-3213	763-1049	889-1176
ATAN	4600-6700	6300-7450	1241-1813	1917-2284	434-643	697-835
SQRT	1220-1540	700-1100	392-432	251-388	148-163	93-139
ADD int	8.3	7.5	4.3	4.0	1.2	1.2
MULT int	22.5	24.0	5.2	5.4	1.5	1.3
IF	9.1	12.0	4.5	6.0	1.1	1.5

TABLE 2. Computation times for several systems

Computer	PS/2-8530-021		PS/2-8560-061		PS/2-8570-A21		PS/2-486/25 Platf	
OS	MSDOS 3.3		MSDOS 3.3		MSDOS 3.3		MSDOS 3.3	
Clock	8 MHz		10 MHz		25 MHz		25 MHz	
Proc/Copr	8086/8087		80286/80287		80386/80387		80486	
Prog.Lang	TC V2.0	µC V4.0	TC V2.0	µC V4.0	TC V2.0	µC V4.0	TC V2.0	µC V4.0
Operation	Computation Time (µsec)							
ADD real	52-55	54-66	46-50	46-50	9-11	9-11	1.7-3.7	1.7-3.7
MULT real	57-63	60-66	51-53	51-53	9-11	9-11	1.7-3.7	1.7-3.7
SIN	300-360	440-550	236-269	363-401	27-39	104-110	11-22	38-50
COS	300-360	430-550	231-269	374-401	27-39	98-110	11-17	38-44
ASIN	220-305	320-390	181-253	247-291	43-66	65-77	21-28	27-33
ACOS	245-330	320-440	192-259	258-303	44-66	65-83	21-28	27-33
ATAN	220-275	220-330	159-198	186-214	39-55	50-61	21-28	21-28
SQRT	76-80	159-165	58-61	104-106	14-17	31-33	5-7	13-15
ADD int	8.3	7.5	4.3	4.0	1.2	1.2	0.5	0.5
MULT int	22.5	24.0	5.2	5.4	1.5	1.3	1.1	1.1
IF	9.1	12.0	4.5	6.0	1.1	1.5	0.47	0.6

TC: Turbo C, µC: Microsoft C
Operation Precision- Real: 8 bytes, Integer: 4 bytes

TABLE 3. Computation times for several systems

Computer	VAX 11-750	μVAX-II Q3	VAXSTATION	DECSTATION 3100
OS	VMS 4.7	Ultrix 2.2-1	VMS 5.1	Ultrix V3.1
Clock				16.7 MHz
Proc/Copr	Proprietary*	Proprietary	Proprietary	MIPS 2000/2010
Prog.Lang	VAX C V2.4	System C	System C	System C
Operation	Computation Time (μsec)			
ADD real	9-10	10-11	2.9-3.3	0.16-0.33
MULT real	9-10	10-11	3.0-3.3	0.16-0.33
SIN	64-73	35-40	19-22	5.0-8.3
COS	72-90	133-138	19-22	6.7-8.3
ASIN	72-89	37-40	20-23	8.3-10
ACOS	82-100	70-73	22-25	8.3-10
ATAN	63-80	167-173	17-20	6.7-10
SQRT	57-67	30-33	15-18	1.6-1.7
ADD int	2.0	3.5	0.5	0.024
MULT int	1.9	7.7	0.5	0.024
IF	5.3	4.3	1.1	0.055

TABLE 4. Computation times for several systems

Computer	Apollo 3500	SUN 3-60		SUN 4-110	AViiON AVX 300		INMOS T800-20	
OS	BSD4.2 DoIX	Unix 4.2		Unix 4.3.2	DG/UX 4.2			
Clock	25 MHz	20 MHz		14.3 MHz	16.7 MHz		20 MHz	
Proc/Copr	68030/68882	68020/68881		Sys4 SPARC	88100		IMS T800-20	
Prog.Lang	System C	System C	GNU V1.25	System C	System C	GNU V1.35	PC V2.0	Occam 2
Operation	Computation Time (μsec)							
ADD real	7.3-7.5	18-19	11-13	3-4	2.2-2.5	1.8-2.5	4-5	1.8-2.0
MULT real	7.3-7.5	15-16	11-12	3-4	2.3-2.8	2.3-2.8	4-5	2.5-2.6
SIN	23-25	17-20	20-25	5-7	22-25	20-25	30-40	38.8-40.7
COS	25-27	18-23	22-28	5-8	3.3-5.0	3.3-5.0	10-20	34.8-36.0
ASIN	143-145	22-25	23-27	27-32	42-48	42-48	10-20	30.6-47.3
ACOS	138-140	20-25	23-30	25-33	43-48	42-47	10-20	31.1-46.6
ATAN	25-27	17-20	20-27	3-7	10-15	10-15	10-20	36.8-36.9
SQRT	27-28	15-22	23-27	3-7	3-7	3-7	10-20	13.0-13.8
ADD int	0.4	1.2	0.5	0.9	2.7	2.7	1.75	0.66
MULT int	0.6	3.3	0.5	2.8	2.7	2.7	1.75	2.54
IF	1.3	1.6	1.6	1.2	0.67	0.67	2.08	0.922

PC: 3L Parallel C, *With FPA L00001
Operation Precision- Real: 8 bytes, Integer: 4 bytes