Performance Evaluation of Processor Architectures for Robotics*

Stefano Caselli§, Eugenio Faldella‡, Francesco Zanichelli§

§Dipartimento di Ingegneria dell'Informazione
Università di Parma - Viale delle Scienze
43100 Parma - Italy

‡Dipartimento di Elettronica, Informatica e Sistemistica
Università di Bologna - Viale Risorgimento, 2
40136 Bologna - Italy

Abstract

In this paper several commercially-available processor architectures are evaluated with respect to their suitability for the robotic domain. To this aim, a realistic workload, encompassing all the major computations required for high-performance robot manipulator control, is presented. The C language implementation of the algorithms, developed for evaluation purposes but reflecting an actual robotic system programming scenario, is also discussed. Results achieved by execution of the robotic benchmark suite on a set of advanced microprocessors are contrasted with those of conventional benchmarks, and reveal architectural features and performance issues of interest for designers of computer systems targeted to robotic applications.

1 Introduction

The goal of this work is to investigate the suitability of several commercially-available processor architectures for the robotic domain. The robotic domain is peculiar, since very high computational requirements are coupled with strong real-time constraints. As a matter of fact, robotic computations have always saturated the processing power of the available controllers, so that the implementation of many proposed control schemes has been actually hindered by their computational complexity. Despite the increased performance exhibited by the processor architectures now available, the computational issue still plays a fundamental role. New, highly-complex robotic devices (e.g. dextrous hands, mobile robots and walking machines, redundant manipulators), direct drive actuation schemes (requiring to take into account the full dynamic model of the physical system), and increased task requirements (e.g. accuracy, speed, interaction with the environment, acquisition of massive sensorial information) are placing demanding requirements on current computer architectures for robotics [1, 2].

Custom VLSI devices offer the potential for solving many of the computational problems in robotics, and have been investigated in several research laboratories [2, 3]. Their actual application as processing elements within industrial robot controllers, however, is more difficult than with other embedded systems due to the lower end-product volumes. It is particularly useful, therefore, to evaluate off-the-shelf processors and architectures, especially the novel high-performance families, with respect to their relative suitability for the robotic domain.

Currently, processor architectures are evaluated either by using architecture-dependent ratings as the number of instructions per second (MIPS) or the number of (peak) floating-point operations per second (MFLOPS), or by resorting to general-purpose benchmark programs such as Dhrystone [4] for integer performance, Whetstone [5] and Linpack [6] for floating-point performance, and more complex benchmark suites [7] for overall system performance (i.e. including operating system overhead, mass memory access, and other factors). Since the actual amount of computation performed per instruction is architecture-dependent, the MIPS figure does not represent a consistent metric, and the "equivalent VAX MIPS" rating is sometimes used, usually drawn from the Dhrystone performance with respect to the DEC VAX 11/780. The benchmark approach, on the other hand, is to emulate the average workload of a general-purpose computer system, or to consider its behavior in presence of heavy, numerically-intensive applications. General-purpose benchmark programs are meant to address a wide range of computer architectures as well as of applications, but their usefulness to specific domains as robotics is questionable because of poor matching with the actual workload profile. For example, many robotic algorithms are characterized by demanding computational requirements, but exhibit very low regularity when compared, for example, to the Linpack benchmark. Thus, it is not a priori clear whether Dhrystone and Whetstone measures are representative of the actual capabilities of a processor architecture in a robotic context.

Moreover, in the last few years several novel architectures have been introduced in the market, leading to a diversification of the architectures suitable for real-time, embedded applications, namely, the RISC architectures, the existing CISC architectures, and the DSP-based architectures. DSPs, for example, exhibit excellent floating-point ratings. However, their suitability for robotic applications has been questioned [8] and their use in a uniprocessor solution for robot control appears difficult, due to the lack of system-oriented features like trap, lock or flexible interrupt handling instructions. Nevertheless, one can think of extending the coprocessor concept typical of standard CISC architectures, allocating only decision and scheduling activities on the system CPU, whereas the specific robotic code is executed on the DSP acting as an algorithm-level coprocessor [1, 9, 10].

In this paper we investigate the characteristics of a sample of representative processor architectures belonging to the aforementioned classes (i860, SPARC, IBM RS/6000, Transputer T800, 80386/387, 68020/68881, DSP32C) by using both a set of benchmark programs representative of the robotic domain, and some of the standard, general-purpose benchmarks. Relationships between specific architectural features of the considered devices and the performance obtained on the robotic benchmark suite are identified and discussed. As a side goal, we also assess the usefulness and relevance of the general-purpose benchmarks in evaluating processor architectures for dedicated applications.

A variety of multiprocessor architectures has also been proposed by researchers (and adopted by industry) to overcome the computational limits of the available processing elements [2]. These architectures can take advantage of the parallelism inherent in many robotic algorithms, even at the expense of resorting to ad hoc techniques to handle concurrency. Although some of the parallel architectures now on the market, especially the transputer-based systems, have reached a maturity that makes them attractive candidates for robotic control systems, in this work parallel architectures are not directly considered. In point of fact, it would not be possible to devise a unique benchmark frame suitable for the many heterogeneous non-uniprocessor solutions, especially when the architecture itself drives the parallel decomposition of the algorithm. Even restricting the study to shared-memory multiprocessors, too much influence would be exerted upon the results by system-level features, like bus-transfer rate and global/local memory size and performance. However, since our work aims at providing

*Work supported by Consiglio Nazionale delle Ricerche, Italy, under "Progetto Finalizzato Robotica".

general insight into the suitability of processor families for robotic applications, its results might be relevant even to those cases where devices belonging to these families are employed in multiprocessor configurations.

2 Choosing a robotic workload: the Rhobstone suite

2.1 Algorithm selection

It is well known that, given a computer system design problem, the best benchmark is represented by the intended application. A set of algorithms reflecting an advanced robotic paradigm, i.e., likely to be performed at a high-rate in a state-of-the-art robotic system, is the following:

- Inverse Kinematics
- Inverse Jacobian
- Full Inverse Dynamics
- Linear Regulators
- Quintic Polynomial Trajectory Interpolation
- Sensorial Processing Task.

These computations (termed by us the "Rhobstone suite") represent all the major blocks within a robot manipulator control loop [11] (Figure 1). The inverse kinematics and inverse Jacobian transformations map, respectively, cartesian positions and cartesian velocities into the equivalent joint-space quantities, hence, they are iteratively computed, for instance, to synthesize a robot motion along a straight-line trajectory or in a constrained environment. Polynomial trajectory interpolation is often used for free space robot motion in joint coordinates. Inverse kinematics, inverse Jacobian, and polynomial trajectory generation represent, altogether, the set-points generation block. The inverse dynamic model of the manipulator and the linear regulators are used as the basic elements of the preeminent techniques for robot control, the computed torque technique and its variations, and must be computed at the sample rate, which for certain advanced manipulators can be in excess of 1 KHz. Finally, a sensorial processing block is inherent in any robot control loop (e.g. a visually or tactile-servoed manipulator), although its characteristics are largely application-dependent.

The complexity of most robotic computations varies widely on the basis of the number of degrees of freedom (d.o.f.) and the mechanical architecture of the manipulator. Six is the standard, non-redundant number of d.o.f. for many manipulators. The computational complexity of the algorithms, which is for a great part related to the joint-to-cartesian space mapping performed by the manipulator, is lowest for cartesian-type arm, as the IBM 7565. The Unimation Puma 560 represents a well-known industrial robot, also familiar to many research laboratories, with documented parameters. It is an anthropomorphic, orthogonal-type arm and has closed-form inverse kinematic solution. Hence, the complexity of the algorithms for the Puma is intermediate. Currently, there is a trend towards mechanical architectures exhibiting Puma-like characteristics. The Puma 560, therefore, has been chosen as a representative manipulator for our benchmarking effort.

Attention has been paid to the selection of efficient and up-to-date versions of the robotic algorithms from the literature [11]. Appropriate representations for the kinematic variables and efficient formulations of the physical laws play a crucial role in robotic computations efficiency, and a great deal of research is currently being devoted to the investigation of these issues. For instance, the Newton-Euler (N-E) formulation of the inverse dynamics (chosen for the benchmark suite) has been shown to be significantly more efficient than the Lagrange-Euler formulation [12, 13]. However, the algorithm efficiency issue is not crucial in this context, as long as the same algorithms are used across all architectures with their distinctly robotic nature preserved. As an example of the computations involved in a robotic control loop, in Table 1 N-E equations for a six rotational-d.o.f. manipulator are detailed. In actual coding of N-E equations, we took advantage of the simplifications in the computation permitted by the specific mechanical structure of the Puma 560.

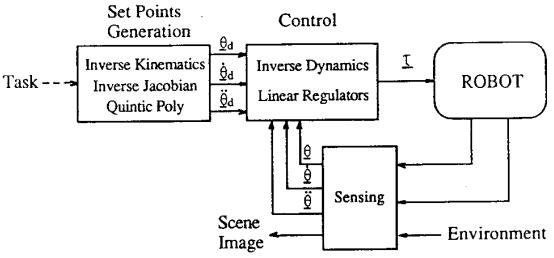

Figure 1: Major blocks for the control of a manipulator

Forward equations ($i = 1, 2, 3, \ldots, n$):

$$
\begin{aligned}
R_i^0\omega_i &= R_i^{i-1}(R_{i-1}^0\omega_{i-1} + z_0\dot{\Phi}_i) \\
R_i^0\dot{\omega}_i &= R_i^{i-1}[R_{i-1}^0\dot{\omega}_{i-1} + z_0\ddot{\Phi}_i + R_{i-1}^0\omega_{i-1} \times z_0\dot{\Phi}_i] \\
R_i^0 v_i &= (R_i^0\dot{\omega}_i) \times (R_i^0 p_i) + (R_i^0\omega_i) \times [(R_i^0\omega_i) \times (R_i^0 p_i)] + \\
&\quad R_i^{i-1}(R_{i-1}^0\dot{v}_{i-1}) \\
R_i^0\bar{\alpha}_i &= (R_i^0\dot{\omega}_i) \times (R_i^0\bar{r}_i) + (R_i^0\omega_i) \times [(R_i^0\omega_i) \times (R_i^0\bar{r}_i)] + R_i^0\dot{v}_i
\end{aligned}
$$

Backward equations ($i = n, n-1, \ldots, 1$):

$$
\begin{aligned}
R_i^0 f_i &= R_i^{i+1}(R_{i+1}^0 f_{i+1}) + m_i R_i^0\bar{\alpha}_i \\
R_i^0 n_i &= R_i^{i+1}[R_{i+1}^0 n_{i+1} + (R_{i+1}^0 p_i) \times (R_{i+1}^0 f_{i+1})] + \\
&\quad (R_i^0 p_i + R_i^0\bar{r}_i) \times (R_i^0 F_i) + (R_i^0 I_i R_0^i)(R_i^0\dot{\omega}_i) + \\
&\quad (R_i^0\omega_i) \times [(R_i^0 I_i R_0^i)(R_i^0\omega_i)] \\
\tau_i &= (R_i^0 n_i)^T(R_i^{i-1} z_0) + b_i\dot{\Phi}_i
\end{aligned}
$$

where:

$z_0 = (0, 0, 1)^T$,

$\dot{\Phi}_i, \ddot{\Phi}_i$ = velocity and acceleration of joint i,

$\omega_i, \dot{\omega}_i$, = angular velocity and acceleration of link i,

$\dot{v}_i, \bar{\alpha}_i$ = linear acceleration of link i and of its center of mass,

$\bar{r}_i, p_i$ = locations of the center of mass and of the origin of link i,

F_i = external forces exerted on link i,

m_i, I_i = mass of link i and inertia about its center of mass,

f_i, n_i = forces and moments exerted on link i by link $i-1$,

b_i = viscous damping coefficient for joint i,

τ_i = required generalized torque for joint i,

R_i^{i-1} = rotation matrix that transforms vectors between consecutive link frames (θ_i time-dependent joint displacement, α_i structure-dependent joint parameter).

$$
R_i^{i-1} = \begin{bmatrix} \cos\theta_i & -\cos\alpha_i \sin\theta_i & \sin\alpha_i \sin\theta_i \\ \sin\theta_i & \cos\alpha_i \cos\theta_i & -\sin\alpha_i \cos\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i \end{bmatrix}
$$

Table 1: Newton-Euler inverse dynamics equations for a six rotational-d.o.f. manipulator

Processor	Coprocessor	System/model	Ext. Cache (Kbyte)	RAM	Frequency (MHz)	C Compiler
68020	68881	HP 9000/330	-	8Mb	20	HP-UX 7.0 comp.
80386	80387	Compaq Desktop PC	64	1Mb	33	Turbo C compiler 2.0
SPARC	SPARC	SparcStation 1+	64	8Mb	25	Sun 4.1 C comp.
IBM RS/6000	chip set	POWERstation 320	8+32	8Mb	20	AIX XLC comp.
T800	int.	Meiko CS	-	4Mb	20	MCC comp.
i860	int.	Intel EVAT board	-	4Mb	33	Green Hills (beta) comp.
DSP32C	int.	Smis VEC128 board	-	128Kb	50	AT&T comp. 1.1

Table 2: Tested systems configurations

2.2 Algorithm implementation

A critical element of an experimental evaluation procedure is the uniformity in programming environment and style. Caveats about the benchmarking procedures abund in literature [14, 15, 7]. Some of the characteristics of the adopted procedure are as follows:

- all algorithms have been coded in C, with the same source code used as much as possible across all processor architectures,
- similar compilers or optimization levels have been used when possible, with optimized results reported separately,
- no hand code optimization whatsoever has been made.

This evaluation procedure, clearly, only assesses the combined performance of the processor architecture and the C language compiler. However, the high-level language benchmarking choice is aimed at emulating a typical robotic systems programming environment [16]. In fact, although assembly language programming would allow performance improvements and analysis of the processor architecture issue by itself, complexity of robotic algorithms and requirements for shorter development times and for enhanced flexibility have led to almost-universal high-level language coding. Also, other (especially higher-level) languages could be considered, however, the choice of the C language more closely reflects a realistic robotic systems programming scenario.

Numerical algorithms have been coded using (single) 32 bit floating-point precision, for increased flexibility, ease of programming, and dynamic range. A higher number of bits for floating-point representation does not seem to be a requirement for most robotic applications, also because the increased precision would be completely lost in the measurement noise, the major limit to further accuracy.

Only a subset of the algorithms previously mentioned has been actually exploited in the experimental evaluation procedure. In particular, the sensorial processing task is largely application dependent: in some cases only proprioceptive feedback is present, whereas in others extensive amount of information on the environment must be collected in real-time (e.g. tactile and visual feedback). Massive sensorial processing is often carried out by means of dedicated subsystems. DSPs are obvious candidates for many of these processing tasks, but even application-specific architectures, such as SIMD architectures, are being investigated for the most demanding chores [2].

As far as timing procedures are concerned, attention has been paid to provide the most accurate results by employing the same methodology discussed in [4]. Where the Unix `times()` function was available, process times have been gathered, whereas on single-user systems timings have been obtained with the system functions `time()` or `clock()`.

Instead of a single performance index, disaggregated measures for the individual algorithms have been collected. The reason is twofold. First, performance on each of the robotic algorithms is relevant by itself, since there are a number of robotic control schemes exploiting only a subset of the algorithms. Secondly, the relative weight of each algorithm within a single performance index should reflect its relative execution rate, which cannot be stated in general. There are, indeed, only general constraints on the execution rates of both kinematics and dynamics algorithms [11] (e.g. trajectory interpolation rate is obviously not greater than the sample rate and should be at least 10 times the manipulator structural resonant frequency; the sample rate itself is related to the time constants and the maximum speed of the manipulator). Several control schemes explore the effects of various computational tradeoffs, for example totally neglecting the dynamic couplings and relying on high-rate linear servos only (independent joint control), or computing the inverse dynamics at a lower rate than the sample rate of the linear regulators (dual control rate methods).

The choice of individual programs for the Rhobstone suite has led also to smaller executable and data set sizes, calling for a discussion of cache effects. It can be argued that the computational core of robotic control systems typically determines limited memory requirements, especially as far as data are concerned. As a matter of fact, kinematic and dynamic variables are related to the number of joints, and algorithms do not generate large amount of data, as happens, for example, in finite element simulations. Thus, caches (both internal and external) significantly benefit the overall performance of computational architecture for robotics, and 32/64Kbyte external caches can be expected to represent a good price/performance tradeoff. Indeed, not all the systems we have evaluated were provided with external caches. However, in Unix systems in order to minimize caches influence on the benchmark results, algorithms have been run under "enforced" heavy load conditions.

3 Experimental results

This section is organized as follows. First, the main characteristics of processor architecture subject to evaluation are briefly illustrated. Next, the computational requirements for the Rhobstone suite algorithms are discussed with respect to the architectural features of the processors and to what extent they are exploited by their compilers. The configurations of the evaluated systems are summarized in Table 2.

3.1 Evaluated processor architectures

The i860 [17] features an Harvard-type architecture with a 4K-byte 64-bit wide instruction cache and a 8K-byte, 128-bit wide data cache. The register file includes thirty-two 32-bit integer registers and thirty-two 32-bit floating-point registers. The adder is a three-stage pipeline, as is the multiplier. Two forms of superscalar parallelism are available: in dual-operation mode, the adder and multiplier work in concert, whereas in dual-instruction mode, the RISC core loads floating-point registers, while the FPU runs in parallel.

The RS/6000 processor architecture [18] is organized into a chip set, including a fixed-point unit, a floating-point unit, a two-way set-associative 8K-byte istruction-cache unit, and two or four 16K-byte data-cache units. The RS/6000 emphasizes the superscalar concept, since, in a single cycle, four instructions can be executed simultaneously, namely, a branch, a condition register instruction, a fixed-point instruction and a floating-point instruction. The register set consists of thirty-two 32-bit integer registers and thirty-two 64-bit floating-point registers. The floating-point unit generates one double-precision result per cycle, with an execution pipeline latency of two cycles.

The architecture of the DSP32C [19] includes a control arithmetic

CPU	Dhrystone 2.1 [D/s]	Whetstone [MW/s]
68020/68881	2812	0.75
80386/387	8030	1.7
SPARC	12217	4.27
IBM RS/6000	14950	6.16
T800	4545	1.7
i860	45030	10.5
DSP32C	7250	4.2

Table 3: Synthetic benchmarks results

unit (CAU), a data arithmetic unit (DAU), a 6K-byte on-chip RAM and two I/O ports. The DAU employs a four-stage pipeline, with a fetch-multiply, accumulate-write cycle. The data bus is 32-bit wide and supports four memory accesses during an instruction cycle. There is a single register set, comprised of nineteen 32-bit registers, with four accumulators provided for the operations of the DAU.

The SPARC architecture [20] features an integer unit (IU) and an external floating-point unit (FPU). The latter can be realized in a number of floating-point arithmetic units, thus specifying the minimum number of concurrent instructions. The IU registers are 32-bit wide and divided into global registers and overlapping windows for efficient parameter passing between procedures. The FPU has thirty-two 32-bit floating-point registers which can be grouped into pairs for double-precision operands. FPU instructions operate only on these registers, with memory load/store executed by the IU.

The T800 transputer [21] consists of a 4K memory, a 32-bit processor (including a microcoded FPU) and a communications system, connected via a 32-bit bus. The processor integer-unit contains three registers organized into a stack, used for integer and address arithmetic and a workspace pointer. Similarly, the FPU has three floating-point registers, also forming a stack. The transputer provides efficient support for the OCCAM model of concurrency and communication, through its four high-speed serial links and the internal microcoded scheduler.

The well-known CISC architectures (80X86, 680X0) have found broad application in robotic control systems [22]. Unfortunately, we have not had the chance yet to test the newer members of the families, namely the i486 or the M68040, to assess the performance improvement in this particular domain.

3.2 Architectural requirements of robotic algorithms

The inverse kinematics algorithm represents a scalar, not-vectorizable, numeric problem and stresses the efficiency of trascendentals calculation. Moreover, a large register set appears to be very useful in order to provide fast access to the several intermediate values. The common RISC approach envisions a rich floating-point register file in order to minimize memory traffic (load/store execution model) and software emulation of complex instructions such as trigonometric functions (as opposed to the 80387 and the 68881). Architectures with a reduced floating-point register set but with on-chip RAM, like the T800 or the DSP32C, can use this memory in lieu of an extended register set with minimal performance penalty. As a general remark, extensive trigonometric functions evaluation is a peculiar aspect of robotic computations, called in action by every cartesian-to-joint trasformation in manipulator control. This emphasis on efficiency of trascendentals, not found in other scientific computations, forced in the past researchers and developers to resort to table look-up techniques.

N-E inverse dynamics, illustrated in Table 1, can be characterized as repeated computations on 3D vectors, and stresses efficiency in address calculation capabilities. In this computation, compilers should be able to exploit pipelining where available and, at a smaller degree, vectorization. Particularly, extensions of the math libraries with functions operating on vectors and matrices, capable of taking full advantage of the underlying architecture, would be extremely effective. Evaluation of the transformation matrices R_i^{i-1} must be accomplished at the sample rate as well, though only 1 sine and 1 cosine per joint

CPU	Inv. Kinematics [μs]	N-E Inv. Dynamics [ms]	Quintic Poly [ms]	PID [μs]
68030/68881	1681	16.82	92.3	51.3
80386/387	703	6.65	40.9	28.1
SPARC	396	1.03	8.4	4.7
IBM RS/6000	290	0.92	7.43	5.3
T800	660	3.56	21	12
i860	150	1.05	4	3.3
DSP32C	158	7.3	11.5	8.2

Table 4: Rhobstone 1.0 results (not optimized)

CPU	Inv. Kinematics [μs]	N-E Inv. Dynamics [ms]	Quintic Poly [ms]	PID [μs]
68020/68881	1610	15.45	74.3	49.3
80386/387	702	6.52	40.3	25.2
SPARC	381	0.87	5.5	4.2
IBM RS/6000	276	0.43	1.8	2.7
T800	648	3.48	21	12
i860	-	0.73	3.56	2.39
DSP32C	-	-	-	-

Table 5: Rhobstone 1.0 results (optimized)

are required.

Linear regulators and polinomial trajectory interpolation algorithms represent small numeric computations, which are iterated for each joint of the manipulator and could be sped-up by pipelining and vectorization. Since the computations involved are fairly simple, and efficiency in operand access from memory is predominant, we can expect architectures including off-chip coprocessors to be intrinsecally weaker than those with integrated floating-point support.

3.3 Discussion

Results of the experimental evaluation procedure are reported in Tables 3-5. Table 3, as a reference, presents results from two classical synthetic benchmark programs (single precision, no optimization). These programs are formally and computationally characterized in terms of type and frequency of instructions [4, 5], whereas results in Tables 4-5 refer to direct and efficient implementations of the robotics algorithms. Since we are evaluating both conventional and state-of-the-art processors with substantially different technological implementations, no general ranking can be estabilished among the architectures. Indeed, it is remarkable how the latest breakthroughs in processor technology have permitted execution times for uniprocessor N-E computations better than those reported for some parallel systems in literature [23]. Compiler technology advancements also play a significant role. Table 5 shows how the XLC compiler for the RS/6000 processor achieves dramatic performance improvements through optimization.

The inverse dynamics computation is the dominant cost in high-performance manipulator control, where the N-E equations must be solved at the sample rate. Availability of general and efficient addressing modes plays a crucial role in N-E computation. Although high-end DSP chips exhibit almost general-purpose architectural features, the performance of the DSP32C on the N-E inverse dynamics appears disappointing, especially when compared to other results obtained for the same device. The AT&T C compiler for the DSP32C, by careful analysis of the code generated, does not seem to take full advantage of the architectural features of this device, even though the poor performance on this benchmark is also related to the basic addressing modes available, targeted to signal processing, and to the lack of an integer multiplication istruction, useful in address calculations. As a matter of fact, researchers have resorted to assembly language programming to attain high rates in DSP-based computation of N-E equations [24].

Widespread architectures usually are supported by development tools provided by multiple sources. Thus, users can often take advantage of more efficient optimizing compilers, than in the case of

dedicated processors. As already mentioned, substantial performance gains could be achieved by customizing the 3D vector math library for the specific architecture. From our experience, the "same source code" approach is particularly unfair to DSPs, whose performance would be significantly sped-up by exploiting different arrangements as far as data structures and operand access are concerned.

A final remark stemming from the observation of Tables 3-5 (e.g. the DSP32C and SPARC entries) refers to the limited usefulness of general purpose benchmarking measures (and the underlying instruction count statistics) in a peculiar domain as robotics.

References

[1] S. Caselli, E. Faldella, S. Mariani, F. Zanichelli, "Computational Architecture of a High-Dexterity Robotic Hand", Int. Symp. on Robot Measurement and Control, Houston, TX, June 1990.

[2] J. H. Graham, "Special Computer Architectures for Robotics: Tutorial and Survey", IEEE Trans. on Robotics and Automation, Vol. 5, No. 5, pp. 543-554, October 1989.

[3] P. Sadayappan, Y. C. Ling, K. W. Olson, D. E. Orin, "A Restructurable VLSI Robotics Vector Processor Architecture for Real-Time Control", IEEE Trans. on Robotics and Automation, Vol. 5, No. 5, pp. 583-599, October 1989.

[4] R. P. Weicker, "Dhrystone: A Synthetic Systems Programming Benchmark", Communications of the ACM, Vol. 27, No. 10, pp. 1013-1030, October 1984.

[5] H. J. Curnow, B. A. Wichmann, "A Synthetic Benchmark", The Computer Journal, Vol. 19, No. 1, pp. 43-49, February 1976.

[6] J. J. Dongarra, "Performance of Various Computers Using Standard Linear Equations Software", Technical Report, CS-89-85, February 1990.

[7] W. J. Price, "A Benchmark Tutorial", IEEE Micro, Vol. 9, No. 5, pp. 28-43, October 1989.

[8] R. L. Anderson, "Computer Architectures for Robot Control: A Comparison and a New Processor Delivering 20 Real Mflops", Proc. IEEE Int. Conf. on Robotics and Automation, Scottsdale, AZ, pp. 1162-1167, May 1989.

[9] J. Ish-Shalon, P. Kazanzides, "SPARTA: Multiple Signal Processors for High Performance Robotic Control", Proc. IEEE Int. Conf. on Robotics and Automation, Scottsdale, AZ, pp. 284-290, May 1989.

[10] I. Radivojevic, J. Herath, "DSP Architectural Features for Intelligence and Control Application", Proc. 5th IEEE Int. Symp. on Intelligent Control, Philadelphia, PA, pp. 273-278, September 1990.

[11] J. J. Craig, *Introduction to Robotics Mechanics and Control*, 2nd edition, Addison-Wesley Pub. Co., Reading, MA, 1989.

[12] J.Y.S. Luh, M.W. Walker, R.P.C. Paul, "On-Line Computational Scheme for Mechanical Manipulators", Transaction of ASME, J. of Dynamic Systems, Measurement and Control, Volume 102, pp. 69-76, June 1980.

[13] P.K. Koshla, S. Ramos, "A Comparative Analysis of the Hardware Requirements for the Lagrange-Euler and Newton-Euler Dynamics Formulations", Proc. IEEE Int. Conf. on Robotics and Automation, Philadelphia, PA, pp. 291-296, April 1988.

[14] J. J. Dongarra, J. L. Martin, J. Worlton, "Computer Benchmarking: Paths and Pitfalls", IEEE Spectrum, Vol. 24, No. 7, pp. 38-43, July 1987.

[15] D. F. Hinnant, "Accurate Unix Benchmarking: Art, Science, or Black Magic?", IEEE Micro, Vol. 8, No. 5, pp. 64-75, October 1988.

[16] V. Hayward, R. P. Paul, "Robot Manipulator Control Under Unix RCCL: A Robot Control 'C' Library", Int. J. of Robotics Research, Vol. 5, No. 4, pp. 94-111, 1986.

[17] L. Kohn, N. Margulis, "Introducing Intel i860 64-Bit Microprocessor", IEEE Micro, pp. 15-30 , Vol. 11, No. 13, 1989.

[18] H.B. Bakoglu, G.F. Grohoski, R.K. Montoye, "The IBM RISC SYSTEM/6000 processor: Hardware overview", IBM J. Res. Develop., pp. 12-21, Vol. 34, No. 1, 1990.

[19] "DSP32C Manual", AT&T, 1988.

[20] "The SPARC Architecture Manual", Version 7, Sun Microsystems, 1987.

[21] "The Transputer Databook", INMOS, 2nd. edition, 1989.

[22] S. Narasimhan , D.M. Siegel, J.M. Hollerbach, "Condor: A Revised Architecture for Controlling the UTAH-MIT Hand", IEEE Int. Conf. on Robotics and Automation, Philadelphia, pp. 446-449, April 1988.

[23] W.-S. Wang, K.-K. Chen, Y.-S. Lai, C.-H. Liu, "Implementation of a Multiprocessor System for Real-Time Inverse Dynamics Computation", Proc. IEEE Int. Conf. on Robotics and Automation, Scottsdale, AZ, pp. 1174-1179, May 1989.

[24] M. Kabuka, R. Escoto, "Real-Time implementation of the Newton-Euler Equations of Motion on the NEC μPD77239 DSP", IEEE Micro, pp. 66-76, Vol. 11, No. 7, 1989.