

Parallel Decomposition of Robot Newton-Euler Inverse Dynamics Computation

Giulio Iannello

Istituto di Fisica Matematica e Informatica
Università degli Studi di Salerno
84084 Fisciano (SA), Italy

Bruno Siciliano

Dipartimento di Informatica e Sistemistica
Università degli Studi di Napoli "Federico II"
Via Claudio 21, 80125 Napoli, Italy

ABSTRACT

The parallel decomposition of robot Newton-Euler inverse dynamics recursive computation for real-time computed torque based dynamic controllers is discussed in this paper. A suitable task decomposition of the equations is derived which allows parallel execution on a message passing computer system. A remarkable reduction of the sampling rate is achieved if multiple processors are employed. An implementation of the algorithm on Transputers is presented to validate the analysis.

INTRODUCTION

It is well-known that in order to improve performance of current industrial robots, there is a need of replacing conventional PID independent joint controllers with computed torque based dynamic controllers [1]. These make use of real-time computation of the robot dynamic model, i.e. the so-called *inverse dynamics*; namely, given the time history of the joint displacements, velocities and accelerations, compute the time history of the joint torques to be applied by the actuators in order to follow the specified trajectories. In fact, the inverse dynamics approach is very effective for control purposes, since it allows to compensate for the highly coupled and nonlinear robot dynamics which play a dominant role at higher operational speeds.

For advanced real-time sensor-based applications, the inverse dynamics must be computed at sampling rates as fast as 300 Hz or higher, so as not to excite the resonant frequencies of the mechanical structure (10-30 Hz) [2]. This has stimulated a great deal of research to seeking for computationally efficient algorithms, the most effective of which is the *Newton-Euler (NE)* algorithm based on forward-recursive equations propagating

velocities and accelerations from the base to the end-effector and backward-recursive equations propagating forces from the end-effector to each joint [3].

Efforts have been devoted to taking advantage of particular computation architectures and/or parallel computations. A number of solutions have been proposed which employ special-purpose processing elements [4] or *Digital Signal Processors (DSPs)* [5].

On the other hand, the possibility of resorting to parallel architectures based on general-purpose multiple processors has been investigated, e.g. in [6] with shared memory. In spite of an apparently reduced computational efficiency, there seem to be several advantages, such as higher modularity, scalability* of the hardware parallelism, availability of high-level language compilers, integrability in hierarchical intelligent control systems, which may in turn cause appreciable cost reductions. These advantages become even more evident in message passing architectures [7], which then deserve further investigation.

The issue addressed in this work is how and up to what extent the inverse dynamics computation problem could be decomposed in order to achieve favourable cost/performance ratios on message passing machines based on general-purpose processing elements. The actual hardware considered consists of *Transputers* which provide high processing power combined with easiness of connection through high speed serial links.

A task decomposition is considered which is based on the parallelism inherently present in the NE formulation. This refers to the particular structure of the NE formulation and corresponds to the sequence of steps in the recursive computation and the matrix/vector operations appearing in each mathematical equation, respectively [8].

The above decomposition leads to a pipeline structure which allows the use of a simple model to distribute the computational load among multiple processors. Moreover, time overlapping is realized between forward and backward computation chains, thus exploiting a further level of parallelism. These devices together yield a considerable increase in the sampling rates which largely satisfy the typical requirements stated above.

Analytical results are given which are confirmed by experimental measures for a practical Transputer implementation. The results encourage application of these architectures to more complex computation problems in robotics.

TASK DECOMPOSITION

The NE equations of motion for an articulated manipulator are a set of computationally efficient forward and backward equations. The former propagate velocities and accelerations from the base to the end-effector, while the latter propagate forces from the end-effector to each joint. A task decomposition of the equations is derived in the following for the case of n revolute joints, without loss of generality [8].

Forward equations: $i = 1, 2, \dots, n$

- (1/i) $\omega_i = R_i^{i-1}(\omega_{i-1} + z_0 \dot{\theta}_i)$
 - (2/i) $\text{vec}_{1i} = \omega_{i-1} \times z_0 \theta_i$
 - (3/i) $\dot{\omega}_i = R_i^{i-1}(\dot{\omega}_{i-1} + z_0 \ddot{\theta}_i + \text{vec}_{1i})$
 - (4/i) $\text{vec}_{2i} = R_i^{i-1} \dot{v}_{i-1}$
 - (5/i) $\text{vec}_{3i} = \dot{\omega}_i \times p_i$
 - (6/i) $\text{vec}_{4i} = \omega_i \times (\omega_i \times p_i)$
 - (7/i) $\dot{v}_i = \text{vec}_{2i} + \text{vec}_{3i} + \text{vec}_{4i}$
 - (8/i) $\text{vec}_{5i} = \dot{\omega}_i \times a_i$
 - (9/i) $\text{vec}_{6i} = \omega_i \times (\omega_i \times a_i)$
 - (10/i) $a_i = \text{vec}_{5i} + \text{vec}_{6i} + \dot{v}_i$
-

Backward equations: $i = n, n-1, \dots, 1$

- (11/i) $\text{vec}_{7i} = I_i \dot{\omega}_i$
 - (12/i) $\text{vec}_{8i} = \omega_i \times I_i \omega_i$
 - (13/i) $f_i = R_i^{i+1} f_{i+1} + m_i a_i$
 - (14/i) $\text{vec}_{9i} = p_i \times f_{i+1}$
 - (15/i) $\text{vec}_{10i} = R_i^{i+1} (n_{i+1} + \text{vec}_{9i})$
 - (16/i) $\text{vec}_{11i} = (p_i + a_i) \times m_i a_i$
 - (17/i) $n_i = \text{vec}_{7i} + \text{vec}_{8i} + \text{vec}_{10i} + \text{vec}_{11i}$
 - (18/i) $\tau_i = n_i^T (R_i^{i-1} z_0)$
-

In the above equations appear the scalar variables defined independently of the particular coordinate frame:

- θ_i : displacement of joint i ,
- $\dot{\theta}_i$: velocity of joint i ,
- $\ddot{\theta}_i$: acceleration of joint i ,
- m_i : mass of link i ,
- τ_i : torque at joint i ,

the (3×1) vector variables all defined with respect to the base frame:

- ω_i : angular velocity of link i ,
- $\dot{\omega}_i$: angular acceleration of link i ,
- p_i : origin of frame i ,
- v_i : linear velocity of link i ,
- \dot{v}_i : linear acceleration of link i ,
- a_i : location of center of mass of link i ,
- \ddot{a}_i : linear acceleration of center of mass of link i ,
- f_i : force exerted on link i by link $i-1$,
- n_i : moment exerted on link i by link $i-1$,

and the (3×3) matrix variables:

- I_i : inertia about center of mass of link i with respect to base frame,
- R_i^{i-1} : rotation transforming any vector from frame i to frame $i-1$.

Also, the following initial conditions hold:

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

$$\omega_0 = \dot{\omega}_0 = 0,$$

$$v_0 = g z_0,$$

$$g = 9.8062 \text{ m/s}^2.$$

To implement the algorithm, it is assumed that the desired position, velocity, and acceleration for each joint are generated at each sample by a suitable trajectory generator. Also, the actual position and velocity are measured by suitable sensors. The trigonometric functions of the joint displacements are readily computed, e.g. a table look-up technique, to be used in the computation of the rotation matrices.

MESSAGE PASSING ARCHITECTURES

We consider here how and to what extent the inverse dynamics problem can be decomposed for parallel execution on a message passing computer system [7].

Message passing parallel architectures are *Multiple Instructions Multiple Data* (MIMD) machines, characterized by the lack of shared memory, among the *Processing Elements* (PEs) forming the system. Cooperation between PEs is achieved through the exchange of messages among the processors across a communication subsystem. Message passing systems can be therefore thought of as networks of general purpose microprocessors, each working on a local private memory.

In the following, we will consider the case of direct networks, i.e. message passing systems whose processing elements are connected by point-to-point high speed channels, according to a proper topology [7]. This class of systems is particularly interesting, for they can be set up by basic components integrated on a single VLSI chip and containing both processing and communication facilities. Hence, networks of different size can be easily assembled giving rise to highly scalable concurrent systems.

An example of such systems are Transputer-based networks. Since our target architecture is a network of Transputers, we will discuss in more detail the features of direct message passing systems with reference to this specific hardware component. Nevertheless, most of our considerations can be easily applied to other systems, based on different hardware, provided that they retain all the essential features outlined in this section.

A Transputer [9] is a microprocessor which has, on the same chip, the CPU, a limited amount of memory and a number of communication links which provide a point-to-point bidirectional connection between Transputers. External memory can be added up to 4 Gbytes. The CPU architecture has been especially designed to be programmed in most high level languages and special support is provided, at the hardware level, for efficient multiprocessing and communication.

Internal parallelism is allowed between the CPU and the communication links, which can access the memory in DMA. In the version of the chip we have used (T800), it is included also a 64-bit floating point unit capable of a peak of 1.5 Mflops (single precision). The speed of the links is about 2 Mbyte/s.

Even though the Transputer can be programmed in assembly language, its architecture has been designed to be programmed in Occam [10], a concurrent language based on the *Communicating Sequential Processes* (CSPs) computational model [11]. Several high level languages, such as Fortran, Pascal and C, provided with libraries for managing concurrency, are also available. We have used the *Parallel Fortran 2.0* by 3L Ltd. to write the software used to perform the measures presented below.

Incidentally, we wish to point out a relevant difference between message passing systems, such as Transputer networks, and shared memory multiprocessors, used so far to exploit parallelism in the computations requested by the inverse dynamics, e.g. [5-6]. In message passing systems, inter-processor communications have a higher cost than in memory shared architectures. In fact, in the latter, the access to data produced by an

other processor requires only the examination of a flag signalling the data are ready, whereas in message passing systems more time has to be spent in transferring data through serial links between the processors.

PARALLEL DECOMPOSITION

The formulation of the NE equations given in the preceding section contains inherent parallelism among the computations to be performed. Several authors agree in leaving out the parallelism existing at the basic arithmetic operation level, for it implies excessive hardware requirements. Instead, the matrix/vector operations occurring in the above equations are usually considered suitable as basic units for carrying out the scheduling of a set of subtasks for parallel execution on the target machine.

In [6], the Stanford manipulator is considered as an example, and a set of 88 subtasks is obtained. Given the assumed execution time for each subtask and the precedence relations among them, a scheduling is determined for execution on a shared memory multiprocessor. The approach was successful and a fairly good agreement between theoretical scheduling and implementation results was found. In fact, the assumption that subtask execution times are independent on the task scheduling is largely verified because shared memory is used in the multiprocessor implementation.

The same approach cannot be followed for scheduling on a direct network of Transputers, because of the higher cost of communications between tasks. Two factors should be taken into account in the determination of the optimal scheduling:

- 1) subtask execution times would increase if previous or next subtasks (or both) were allocated on different processors because of the time needed to start input and output operations between processors;
- 2) communication delays between subtasks allocated on different processors cannot be neglected any more in the evaluation of total completion time.

This prevented us from using simple scheduling algorithms, and a more sophisticated model would be needed. We therefore decided to resort to a different approach that allowed communication overhead to be properly accounted for without considerably increasing the complexity of the model.

We chose to take advantage only of the parallelism inherent to the recursive nature of the above NE formulation. To explain it better, consider, for instance, steps 3 and 5 in the recursive equations. Even though step 5/ i must follow in time step 3/ i , the former can be performed in parallel with step 3/($i + 1$). The same

argument can be applied to any pair of steps x and y , provided that step y/i does not depend on step $x/(i+1)$. This is true for all the steps belonging to the forward chain, as well as for all the steps belonging to the backward chain. In other words, the two chains can be easily pipelined, so that a part of computation can be overlapped in time.

This kind of decomposition allows the use of a simple model for determining how many matrix/vector operations have to be assigned to each processor to attain the desired speed-up. We will see in the next section how it is also easy to evaluate the efficiency obtained.

A serious drawback of this approach is that we renounce to exploit further parallelism inherent to the NE formulation. For instance, the potential ability to compute in parallel steps $1/i$ and $3/i$ cannot be exploited any more. Moreover, for typical value of n (e.g. 6), the speed-up attainable through this approach is limited and even scarcer is the efficiency. Nevertheless, even though we have not carried out the evaluation of the optimal scheduling, it is likely that the greater cost in communications implies that a smaller degree of inherent parallelism can be effectively exploited.

Interestingly, the particular application nature of inverse dynamics computation gives rise to a further opportunity of exploiting parallelism. In fact, the inverse dynamics have to be repeated to update the inputs to actuators in order to follow the desired trajectories. Hence, the computation of the joint torques to be applied at time $T + \Delta T$ might be started when the computation of torques at time T is not finished yet.

To actually exploit this further level of parallelism, we organized the distributed computation as follows:

- a) The processes corresponding to forward and backward chains, though sequential in time, are allocated on different groups of processors.
- b) Communications between forward and backward chains are implemented using high priority channel drivers, able to carry out communication asynchronously with respect to normal processing activity.
- c) Backward chain calculations at time T are overlapped in time to forward chain calculations at time $T + \Delta T$ (Fig. 1).
- d) According to this strategy, the overall sampling rate (i.e. the rate to which joint torques are updated) is the minimum between the sampling rates of the two chains.

As it will be clear later, this greatly increases the processor utilisation factor and the overall sampling rate — also called throughput in the following — of the system.

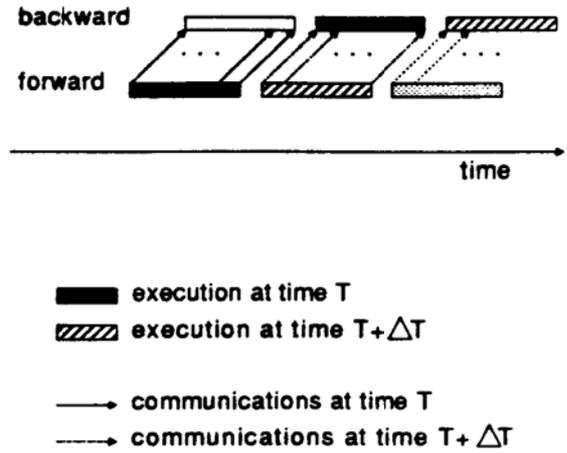


Fig. 1. Time overlap of forward and backward chains.

PERFORMANCE EVALUATION

If forward and backward computations are both implemented as a pipeline of processes running on distinct processors, the total completion time and the throughput can be easily expressed in terms of the amount of operations, the speed of the hardware and the number of processors used.

In Fig. 2 a typical pipeline is shown. Each circle represents a process — a stage of the pipeline — running on a different processor. Each stage executes a cycle containing three basic steps: input of data from previous stage, processing of data, output of data to the next stage. The first stage produces n groups of data which cause the next stages to perform n cycles. The overall computation terminates when the last stage has in turn executed n cycles and produced its final output.

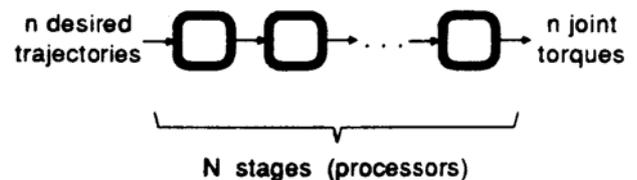


Fig. 2. Structure of the pipeline.

In the general case, the total completion time of a pipeline like the one shown in Fig. 1 is given by the formula:

$$(N - 1)(Dt_e + T_c + kl) + n(Dt_e + T_c) \quad (1)$$

and the throughput by the formula:

$$n(Dt_e + T_c) \quad (2)$$

where:

D : number of basic operations to be performed in each stage of the pipeline to produce output data from the input ones,

t_e : elementary time required to perform one basic operation,

T_c : time required to start input and output operations in each stage,

kl : time required to pass l elementary data produced by a stage to the next stage of the pipeline,

n : number of cycles performed by each stage of the pipeline to produce all the data required (i.e. the number of joints in our case),

N : number of stages (i.e. processors) composing the pipeline.

Formulas (1) and (2) hold if all the stages of the pipeline are given approximately the same amount of calculations. Under this assumption, if C is the total number of basic operations to be performed per joint, we have

$$D = C/N. \quad (3)$$

Now, C depends only on the NE formulation given above and can be easily estimated if the grain of what we called basic operation is established. In what follows, we assume the double precision floating point operation as the basic operation. Substituting (3) in (1), we have the total completion time expressed only in terms of N , n and a number of values corresponding to architectural parameters that can be easily measured. Therefore, it is possible to solve the problem of dimensioning the hardware (e.g. find the value of N that maximises the speed-up).

Another assumption underlying the formulas given above is that, after it has been started, inter-stage communication can proceed in parallel with the CPUs of the communicating processors. Once more, this behaviour is achievable on the Transputer if a dedicated process is used to drive the two communication links involved (input and output).

The measured values for the parameters contained in (1) are given in Tab. 1. These values have then been employed to compute the values reported in Tabs. 2 and 3. The formula actually used is not (1), but a somewhat more sophisticated version which takes into account the effects produced by some simplifications in the NE formulation (e.g. the forward chain can be actually computed only for joint 2 to n because of the constant initial conditions).

Symbol	Value
t_e	1.81 μ s
T_c	54 μ s
k	1.72 Mbyte/s
l	72 byte

Tab. 1. Architectural parameters.

Number of processors*	Execution time (ms)**	Sampling rate (Hz)**
1	2.100 (2.100)	476 (476)
3 (2+1)	2.039	1111
4 (3+1)	1.957 (1.990)	1111 (1098)
5 (3+2)	1.746	1428
6 (4+2)	1.753	1428

* in parentheses the distribution between forward and backward chains

** in parentheses the value actually measured

Tab. 2. Pipelined computation of inverse dynamics.

# Processors*	$n = 6$	$n = 12$	$n = 18$
3 (2+1)	1111	544	352
4 (3+1)	1111	555	370
5 (3+2)	1428	704	456
6 (4+2)	1428	721	481

* in parentheses the distribution between forward and backward chains

Tab. 3. Sampling rates (Hz) for different size problems.

The analytic results show a good agreement with the experimental data measured on a network of four Transputers. The difference (a few percentage units) can be due essentially to the unavoidably non-perfect load balancing among the different stages of the pipeline, and to the difficulty of accurately modelling the context switching of processes in each Transputer. We have also verified that little modifications to the amount of computations in a given stage or to the amount of data to be exchanged between stages has a negligible impact on total completion time and overall sampling rate. This further strengthens the validity of the data reported in the tables.

As for the actual performance attained by the implementation described above, the tables show that the speed-up achievable is limited and so is the efficiency, also for larger dimension problems. On the contrary,

the throughput is remarkably increased and the parallelism actually exploited is sufficient to guarantee a sampling rate matching the requirements stated in [6], even for problems with dimension larger than 18. Substituting (3) in (2) shows that sampling rate can be further increased by adding other processors and decreasing the amount of calculations per stage. However, this cannot be done indefinitely because, besides a given number of processors, the total completion time increases and the delay between the input (desired trajectory) and the output (joint torques) may become unacceptable.

CONCLUSIONS

The use of general purpose message passing systems for the computation of inverse dynamics has been explored. A simple pipelined decomposition of the NE formulation has been used and the performance achievable on a Transputer network investigated, both analytically and experimentally. The tests carried out have shown satisfactory performance which favourably compares with the typical requirements for real-time inverse dynamics robot control. However, the speed-up attainable is limited and so is the efficiency. The main reasons for this are the fairly small dimension of the problem for normal value of n , and the high performance reached by execution on a single Transputer with respect to previous results [6]. In fact, we have measured the speed-up with respect to a purely sequential version of the algorithm, cleared of all the software and communication overhead [12]. Nonetheless, results have revealed that a remarkable improvement in throughput can be obtained.

Moreover, the approach followed seems to have a number of advantages with respect to cost/performance ratio. On one hand, general purpose processors can be employed and their number scaled according to the performance to be obtained. On the other hand, high level languages can be used to write the software needed with a few calls to already available subroutines to manage concurrency. The implementation is therefore highly modifiable, and a few changes are necessary if further processors are added, or simplifications in the calculations can be introduced, to take into account the actual physical structure of the robot.

The power of the processors can be also exploited to reduce the costs of other components of the controller, or to achieve acceptable performance in larger size inverse dynamics problems, such as highly redundant robots or cooperating multi-arm systems.

We are currently engaged in deeply studying the use of message passing systems in robot applications.

We intend to apply more sophisticated techniques [13] to the problem of optimal scheduling of inverse dynamics, in order to better assess the performance achieved in the pipelined approach. Moreover, we are trying to study how other components of the robot controller can be implemented on architectures similar to those described above.

REFERENCES

- [1] H. Asada and J.-J.E. Slotine, *Robot Analysis and Control*, John Wiley & Sons, New York, 1986.
- [2] P.K. Khosla and T. Kanade, "Experimental evaluation of nonlinear feedback and feedforward control schemes for manipulators," *Int. J. Rob. Res.*, vol. 7, no. 1, pp. 18-28, 1988.
- [3] J.Y.S. Luh, M.W. Walker, and R.P.C. Paul, "On-line computational scheme for mechanical manipulators," *Trans. ASME J. Dynam., Syst., Meas., Contr.*, vol. 102, no. 2, pp. 69-76, June 1980.
- [4] C.S.G. Lee, "Efficient parallel algorithm for robot inverse dynamics computation," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-16, no. 4, pp. 532-542, 1986.
- [5] M. Kabuka and R. Escoto, "Real-time implementation of the Newton-Euler equations of motion on the NEC μ PD77230 DSP," *IEEE Micro*, pp. 66-76, Feb. 1989.
- [6] W.-S. Wang, K.-K. Chen, and C.-H. Liu, "Implementation of a multiprocessor system for real-time inverse dynamics computation," *1989 IEEE Int. Conf. Rob., Autom.*, Scottsdale, AZ, May 1989.
- [7] C.L. Seitz, "Concurrent VLSI architectures," *IEEE Trans. Comp.*, vol. C-33, no. 12, pp. 1247-1265, 1984.
- [8] 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, no. 2, pp. 214-234, 1982.
- [9] INMOS Ltd., *Transputer Reference Manual*, INMOS, 1985.
- [10] INMOS Ltd., *Occam 2 Reference Manual*, Prentice Hall, Cambridge, UK, 1988.
- [11] C.A.R. Hoare, "Communicating sequential processes," *Comm. ACM*, vol. 21, no. 8, pp. 666-676, 1978.
- [12] G.C. Fox et al., *Solving Problems on Concurrent Processors*, Prentice Hall, Englewood Cliffs, NJ, 1988.
- [13] V. Sarkar and J. Hennessy, "Compile-time partitioning and scheduling of parallel programs," *SIGplan Symp. Compiler Construction*, ACM SIGplan Notices vol. 21, no. 7, pp. 17-26, 1986.