Proceedings of the IMACS SICE International Symposium on Robotics, Mechatronics and Manufacturing Systems'92 Kobe, Japan, Sep. 16-20, 1992

Application of Parallel Computing to Robot Dynamics

P. Schäfer and W. Schiehlen

Institute B of Mechanics, University of Stuttgart,
Pfaffenwaldring 9, W-7000 Stuttgart 80, Germany

Abstract

In this paper an approach for the application of parallel processing to the dynamic analysis of robots based on the multibody system method is presented. The inherent structure of the symbolic equations of motion is used for partitioning those into independent modules for concurrent evaluation. The applied strategies for parallelization include the parallel evaluation of subsystem equations and the parallel computation of the inertia matrix along with its factorization, and of the force vectors and control inputs. The implementation of the parallel structures is discussed with respect to optimal utilization of transputer resources. The presented approach yields a strong reduction of computing time and supports real-time simulations of controlled robots, which is also shown in an example.

1. INTRODUCTION

For the modeling of robots and walking machines the method of multibody systems is well qualified and it provides reliably the system's state equations, see e.g Schweitzer; Mansour [1]. However, the resulting equations of motion and the equations of reaction are highly nonlinear and real-time solutions cannot be obtained by standard computing technology. Therefore, special real-time hardware such as, for instance, a transputer network has to be used for this type of application, e.g. Schiehlen; Schäfer [2]. Transputers are specifically designed for distributed memory parallel processing. In order to exploit the features of this computer hardware the equations of motion have to be investigated with respect to the potential of parallel computation. The approach to be presented is based on the automatic generation of symbolic equations of motion for robots, Kreuzer; Leister [3]. An efficient parallel implementation has to take into account that communication between processors is costly and coarse grain parallelism has to be achieved. Combining different strategies for parallelizing the robot dynamics equations leads to a considerable reduction of computation time and supports real-time simulation.

2. MODELING OF ROBOT DYNAMICS

For the modeling of the dynamical behavior of a robot the links are assumed to be rigid bodies interconnected by joints. The links are subject to applied forces and torques, e.g. tool forces, or forces and torques due to actively controlled drives. For a system consisting of p bodies subject to q holonomic and scleronomic constraints the position vector $\boldsymbol{r}_i$ of the

center of mass C_i and the rotation tensor S_i of body i, $i=1(1)p$, can be expressed in terms of $f=p-q$ generalized coordinates represented by the $f \times 1$ position vector y as

$$r_i = r_i(y,t), \quad S_i = S_i(y,t), \tag{1}$$

see Figure 1. Differentiation with respect to time yields the linear and angular velocities v_i and ω_i and accelerations a_i and α_i, respectively,

$$\begin{aligned} v_i &= J_{Ti}\,\dot{y} + \bar{v}_i, & \omega_i &= J_{Ti}\,\dot{y} + \bar{\omega}_i, \\ a_i &= J_{Ti}\,\ddot{y} + \bar{a}_i, & \alpha_i &= J_{Ti}\,\ddot{y} + \bar{\alpha}_i \end{aligned} \tag{2}$$

with the Jacobian matrices of translation $\mathbf{J}_{Ti}$ and rotation $\mathbf{J}_{Ri}$. The vectors $\bar{v}_i$, $\bar{\omega}_i$, $\bar{a}_i$ and $\bar{\alpha}_i$ contain the terms that are not linearly dependent on $\dot{y}$ and $\ddot{y}$, respectively. The dynamic equations can be derived by applying Newton's and Euler's law for each free body with mass m_i and inertia tensor I_i. They read as

$$m_i\,a_i = f_i^a + f_i^c, \qquad I_i\alpha_i + \omega_i \times I_i\omega_i = l_i^a + l_i^c, \tag{3}$$

where f_i^a and l_i^a represent the applied forces and torques and f_i^c and l_i^c comprise the constraint forces and torques, respectively. After assembling the masses and inertia tensors as well as the Jacobians to the global matrices

$$\overline{\overline{M}} = diag\{m_1E,\ldots,m_pE,I_1,\ldots,I_p\}, \quad \bar{J} = \left[J_{T1}^T,\ldots,J_{Tp}^T,J_{R1}^T,,\ldots,J_{Rp}^T\right]^T, \tag{4}$$

and assembling the applied forces $\bar{q}^a$ in an analogous manner the Newton–Euler equations of the overall system can be written as a $6p \times 1$ vector equation

$$\overline{\overline{M}}\,\bar{J}\,\ddot{y} + \bar{k} = \bar{q}^a + \bar{Q}\,g, \tag{5}$$

where $\bar{k}$ contains Coriolis and centrifugal forces. The constraint forces are represented by the $q \times 1$ vector g of generalized constraint forces and a corresponding $6p \times q$ distribution matrix $\bar{Q}$. According to d'Alembert's principle the constraint forces can now be eliminated by premultiplying (5) with the transpose $\bar{J}^T$ of the global Jacobian matrix to obtain the equations of motion

$$M(y)\ddot{y}(t) + k(y,\dot{y},t) = q(y,\dot{y},u,t) \tag{6}$$

with the $f \times f$ symmetric and positive definite inertia matrix M, the $f \times 1$ vector of generalized Coriolis and centrifugal forces k and the $f \times 1$ vector of generalized applied forces q. The vector u summarizes the control inputs. The equations of reaction, on the other hand, can be found by premultiplying (5) with $\bar{Q}^T\overline{\overline{M}}^{-1}$ which yields

$$N(y)g(t) - \hat{k}(y,\dot{y},t) = -\hat{q}(y,\dot{y},u,t). \tag{7}$$

For numerical integration (6) has to be solved for the generalized accelerations $\ddot{y}$ which can be done by Cholesky decomposition $M = LL^T$ with a lower triangular $f \times f$ matrix L to be determined. The required number of operations is of order $O(f^3)$ and this procedure

becomes computationally expensive if the number of degrees of freedom increases. Therefore, the recursive dynamics approach is widely being applied in robot dynamics analysis, see e.g. Brandl, Johanny, Otter [4]. In this approach the kinematic quantities are determined recursively starting from the base body. Likewise, the forces and torques are determined in a backward recursion starting from the endeffector. The inertia matrices are inverted locally in the Newton–Euler equations (3) for each body. A final recursion solves a set of linear equations with triangular structure to obtain explicit equations of motion in the generalized accelerations,

$$\ddot{y} = f(y,\dot{y},u,t). \tag{8}$$

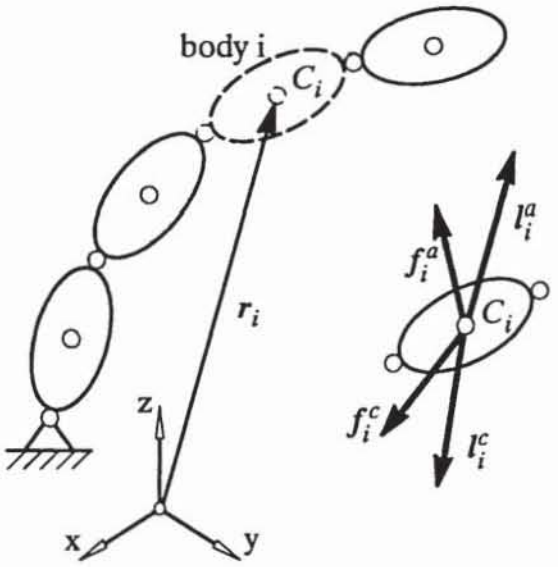

Figure 1. Robot structure and free body

Thus, the inversion of total inertia matrix is avoided. However, the evaluation of the generalized accelerations requires some overhead not necessary in the nonrecursive formulation.

3. PARALLEL EVALUATION OF ROBOT DYNAMICS EQUATIONS

The aim of the parallelization of robot dynamics equations is to implement the computations on transputers, which are distributed memory parallel computers. Therefore, fine grain parallelization is not considered. Instead, it is necessary to partition the equations to be evaluated into blocks that are independent of each other as far as possible. Since function evaluations are by far the most costly part of robot simulations, the discussion of parallelism is restricted to the evaluation of the equations of motion. Furthermore, the investigations are based on symbolic equations of motion of the form (6) and the serial recursive dynamics are not considered.

There are several possibilities for obtaining partially independent computations in the symbolic equations of motion (6). The first strategy is motivated by the partitioning of the overall system into subsystems as shown in Figure 2. The local coordinates are represented by the vectors y_1 and y_2, respectively, and the local equations of motion read as

$$M_1 \ddot{y}_1 + k_1 = q_1 + Q_1 g, \qquad M_2 \ddot{y}_2 + k_2 = q_2 + Q_2 g \tag{9}$$

with matrices M_i, and vectors k_i and q_i, $i=1(1)2$, corresponding to (6). Here, g represents the constraint forces and torques f_{12}^c and l_{12}^c introduced by cutting the system and Q_1, Q_2 are the corresponding distribution matrices. The coupling of the subsystems can be performed by relating the local coordinates y_1 and y_2 to the global generalized coordinates,

$$\dot{y}_1 = I_1 \dot{y}, \quad \dot{y}_2 = I_2 \dot{y} \tag{10}$$

with the subsystem Jacobian matrices I_1 and I_2. The global equations of motion (6) follow by substituting the accelerations $\ddot{y}_1$ and $\ddot{y}_2$ derived from (10) into (9), premultiplying the resulting equations by $I_1^T$and I_2^T, respectively, and adding them up in order to eliminate the constraint forces g. The coupling of subsystems can also be achieved for local equations of the form (8), Schiehlen [5]. This method avoids the inversion of the global inertia matrix M, but the constraint forces have to be computed instead using (7). This approach may be beneficial if systems with kinematic loops are treated.

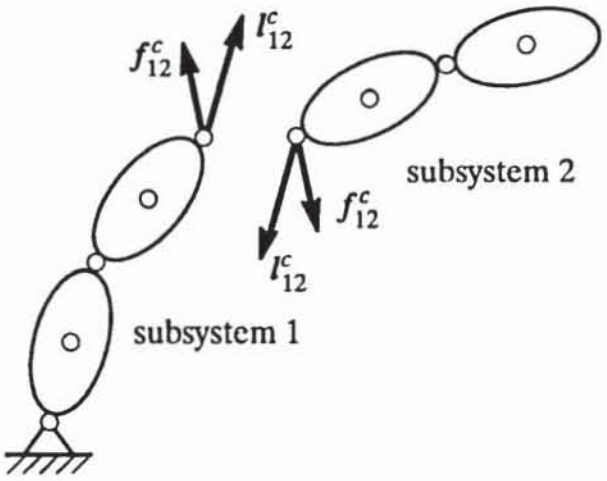

Figure 2. Subsytem partitioning

Equations (9) can be implemented for concurrent evaluation and the coupling is done subsequently, as shown in Figure 3. However, the inherent dependency between the subsystem kinematics exploited by the recursive dynamics approach leads to a strong data dependency. Consequently, the parallel computation is serialized by the communication of interface variables and only minor benefits of a concurrent evaluation can be achieved. For tree structured systems, on the other hand, subsytem equations of independent branches can be evaluated fully in parallel. In vehicle dynamics, e.g., this strategy proves to be most appropriate, Knaupp [6].

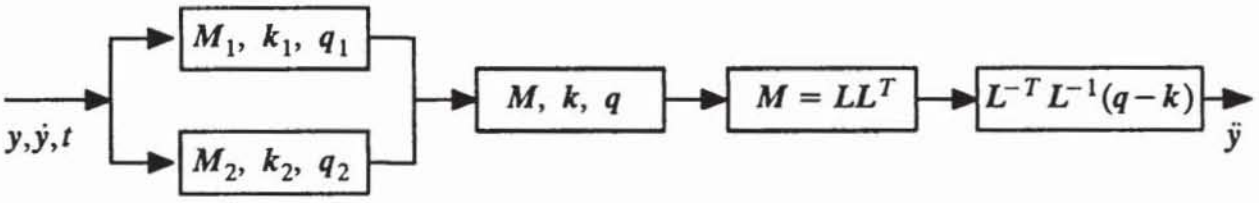

Figure 3. Parallel evaluation of subsystem equations, strategy 1

The second strategy for parallelization follows from the independence of J, $\bar{k}$ and $\bar{q}^a$ in (5) which can be computed completely concurrently. The basic structure of the computational evaluations is shown in Figure 4. This approach is particularly interesting for extensive control law calculations which again can be performed on separate processors, as indicated with dashed lines in Figure 4. In some cases it may be of advantage either to transfer selected elements of the Jacobian matrix to the neighboring processors or to compute them locally, thus enabling these processors to compute $k = J^T\bar{k}$ or $q = J^T\bar{q}$, respectively, instead of evaluating $\bar{k}$ or $\bar{q}$ only.

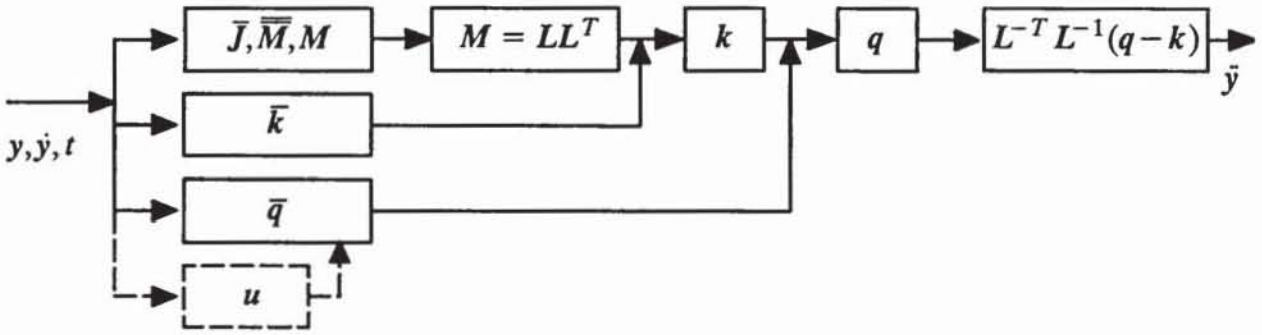

Figure 4. Parallel evaluation of inertia matrix, force vectors and control law, strategy 2

A drawback of the first two strategies presented is that the cost of the Cholesky decomposition fully adds to the execution time required for the preceding computations. In order to avoid this, a separate processor can be assigned to this task running in parallel to the computation of the force vectors k and q, Figure 5. This third strategy pays off in any case and contributes strongly to overcome the disadvantage of the nonrecursive approach due to matrix factorization. The recursive formulation, on the other hand, leads to a coupling of all computations and has less potential for parallelization.

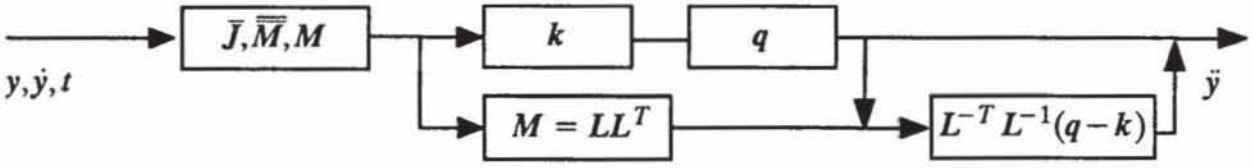

Figure 5. Parallel evaluation of force vectors and matrix factorization, strategy 3

The number of operations in each of the different strategies is strongly problem dependent. In general, optimal processor utilization and speed–up can only be achieved by a combination of the different parallelization strategies. For this, the timing of the separate threads has to be monitored and processor assignment has to be done accordingly.

In the above, the equations of reaction (7) were not considered, since they are completely independent of the solution of the equations of motion (6). However, the computation of the constraint forces can be evaluated in parallel to the equations of motion and the parallelization strategies may be applied accordingly.

4. IMPLEMENTATION ON TRANSPUTER NETWORK

The transputer is a powerful microprocessor designed for real–time parallel processing. It has local memory and four high–speed serial links for interconnection with neighboring transputers. A software controlled link switch allows to establish problem specific processor topologies. For real–time applications, two on–chip timers with 1 μs and 64 μs resolution, respectively, can be accessed. Also, a wide variety of peripheral devices such as, for

instance, analog interfaces are available for control applications. The root–transputer communicates with the host–computer, as shown for a problem specific processor topology in Figure 6. Software development can be done with standard programming languages. The parallel processing language OCCAM, however, allows the most efficient utilization of the transputer resources. For details about transputers and OCCAM, see e.g. INMOS [7].

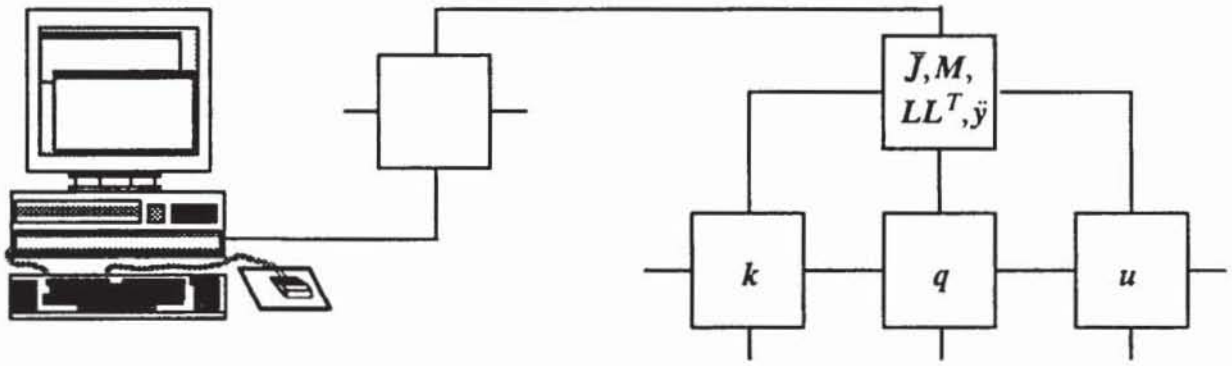

Figure 6. Transputer network for robot simulation

The parallelization strategies shown above lead to coarse grain parallelism and are therefore well suited for the implementation on transputer networks. The implementation of parallel robot dynamics simulation starts with modeling by the program NEWEUL, Kreuzer, Leister [3]. The formalism NEWEUL generates the symbolic equations (5) and (6) in FORTRAN–compatible form. A post–processor analyzes these equations and performs the partitioning into separate modules for concurrent evaluation according to a configuration description provided by the user, Rükgauer [8]. The composition of subsystems can be specified by using symbolic names for the rigid body reference frames. Also, processor numbers can be assigned to the various parallel threads according to Figures 3–5. Thus, arbitrary combinations of the different parallelization strategies can be achieved. A subsequent analysis of data dependencies provides information for setting up communication vectors that contain a minimum number of data to be exchanged between processors. It also allows to separate computation blocks that are independent of the interface variables and, therefore, can be computed in advance of the data exchange, thus minimizing idling times of the processors. At this point the advantages of symbolic equations with respect to code optimization become particularly obvious. Since communication times may be crucial for the resulting efficiency of the implementation, it is important to use the ability of the transputer to communicate through the links and compute in parallel due to its internal hardware structure. The post–processor finally generates OCCAM source code ready to be compiled and linked with a real–time simulation environment also written in OCCAM.

5. EXAMPLE: STANFORD MANIPULATOR

The application of the parallelization methodology presented above is shown for the so-called Stanford manipulator, Kane, Levinson [9]. This robot represents a kinematic chain with five revolute and one prismatic joints and six degrees of freedom. The nonlinear equations of motion of this system are quite large and real-time evaluation is a nontrivial task. The timing of the different subtasks for the implementation on one processor is shown in Figure 7. The shaded areas indicate pure communication without any computations. The times are given in microseconds and refer to single precision (32 bit real) computations. The minimum cycle time turns out to be 1596 μs. Figure 8 shows the corresponding timing diagram for the parallel implementation on three processors. This implementation is based on a combination of the strategies 2 and 3. As already indicated, the subsystem approach is not beneficial due to the chain topology. The result shows that the cycle time could be reduced to 1141 μs and a speed-up of 1.4 could be achieved. This allows the real-time simulation by a second order integration scheme, for instance, the Adams-Moulton method which requires two function evaluations per time-step. It is important to note that certain communications run in parallel to computations, thus reducing the communication overhead considerably, Figure 8.

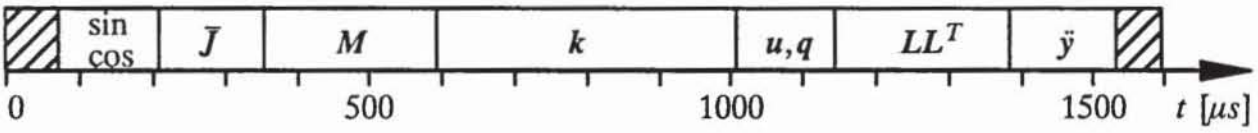

Figure 7. Task timing for Stanford manipulator implemented on one transputer

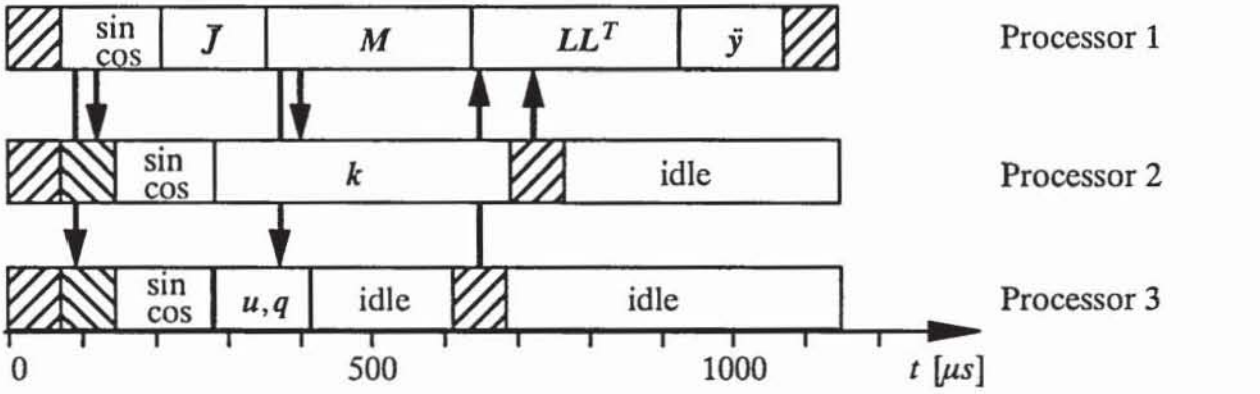

Figure 8. Task timing for Stanford manipulator implemented on three transputers

The transputer implementation in Figure 8 becomes particularly interesting if models for robot drives and sophisticated controllers are to be coupled with the robot simulation. In this case one or more transputers can perform these tasks in parallel to the transputers evaluating the pure multibody model. In the implementation shown in Figure 8, for instance, processor 3 has about 500 μs left for additional control computations that would not affect the total cycle time at all, which raises the achievable speed-up to 1.8.

6. CONCLUSIONS

The symbolic equations of motion of robots were investigated with respect to parallelization for transputer implementation. The evaluation of different matrices and vectors of the equations of motion on separate processors results in a considerable reduction of computing time. This approach is particularly advantageous for the simulation of robots with sophisticated control algorithms. The achievable cycle times allow real–time simulations with simple integration schemes. The next transputer generation promises a further reduction of cycle times for the presented implementation, thus allowing high–speed real–time simulations with more sophisticated integration codes, too.

7. REFERENCES

1 Schweitzer, G.; Mansour, M. (eds): Dynamics of Controlled Mechanical Systems. Berlin/...:Springer, 1989.

2 Schiehlen, W., Schäfer, P.: Modeling of Vehicles with Controlled Components. In: Anderson, R. (ed.): The Dynamics of Vehicles on Roads and Tracks. Proceedings of the 11th IAVSD Symposium, Kingston, Canada, 1989. Lisse: Swets & Zeitlinger, 1990.

3 Kreuzer, E.; Leister, G.: Programmsystem NEWEUL'90. Stuttgart: University of Stuttgart, Institute B of Mechanics, 1991.

4 Brandl, H.; Johanni, R.; Otter, M.: A Very Efficient Algorithm for the Simulation of Robots and Similar Multibody Systems Without Inversion of the Mass Matrix. In: Proc. of the IFAC, IFIP, IMACS International Symposium on Theory of Robots, Vienna, (1986), pp.95–100.

5 Schiehlen, W.: Computational Aspects in Multibody System Dynamics. In: Computer Methods in Applied Mechanics and Engineering, Vol. 90, Nos. 1–3, pp. 562–582. Amsterdam: North–Holland, 1991.

6 Knaupp, E.: Simulation eines räumlichen Fahrzeugmodells mit einem Transputernetzwerk. Stuttgart: University of Stuttgart, Institute B of Mechanics, Student Thesis STUD–82, 1992.

7 INMOS Ltd.: The Transputer Applications Notebook – Applications and Software. Bristol: INMOS Ltd., 1989.

8 Rükgauer, A.: Parallele Implementierung von Bewegungsgleichungen auf Transputern. Stuttgart: University of Stuttgart, Institute B of Mechanics, Student Thesis STUD–83, 1992.

9 Kane, T.; Levinson, D.: The Use of Kane's Dynamical Equations in Robotics. Massachusetts: International Journal of Robotics Research Vol.2, No.3. 1983, pp. 3–21.