SOLVING EQUATIONS OF MOTION ON A VIRTUAL TREE MACHINE*

W. W. ARMSTRONG†, T. A. MARSLAND†, M. OLAFSSON†, AND J. SCHAEFFER † §

Abstract. The dynamic equations of motion for rigid links connected at hinges to form a tree are amenable to a parallel solution. Such a solution can potentially provide a real-time simulation of the dynamic behavior of a broad class of objects used in animation and robotics even when the number of links is large. However, the design of a parallel algorithm poses problems not encountered in the sequential, one-processor case, such as assignment of links to processes, allocation of processes to physical processors, synchronization of processes, and the reduction of communication losses. The computation time using a parallel algorithm and an unlimited number of processors is shown to grow with the height of the tree rather than with its number of links. Increased communication costs degrade performance by a factor at most equal to the fanout. The facility used to investigate these problems is a Virtual Tree Machine (VTM) multi-computer implemented on a network of autonomous VAX-11/780 computers and SUN-2 workstations, each running a UNIX operating system.‡ Although the VTM is physically connected via a local area network, its performance in this study reflects what would occur in a point-to-point interconnection of processors in a tree configuration which would be much more expensive to build and test.

Key words. dynamics of open chains, equations of motion, animation, robotics, parallel computation, ordinary differential equations

AMS(MOS) subject classifications. 65L99 70-04 70-08 70-09

1. Introduction. Recent papers have discussed the use of dynamics for purposes of graphical simulation or animation. The thesis of Wilhelms gives a general overview of the area [1]. It has been noted that the computation time required to solve the equations of motion of a collection of rigid links, connected at hinges, by techniques such as that of Gibbs-Appel can grow as the cube or fourth power of the number of links in the system [2], making such techniques inappropriate when the number of links is large. Recently, the equations of motion of such tree linkages have been formulated in hinge-centered coordinates, and solved by a method requiring time growing linearly with the number of links [3]. In addition, this new algorithm allows a significant amount of parallelism. Indeed, the exploitation of parallel execution on several processors has the potential to reduce the growth of the computation time to being linear in the height of the tree, rather than in its number of links.

The improved performance due to parallel processing should make possible applications of dynamics to graphical simulation and animation which have heretofore been beyond the capacity of existing general-purpose computers. For example, the use of dynamics to generate frame-to-frame motions of animated characters, under real-time control of an animator, can possibly add realism and reduce costs of animation, as well as allowing some exploration of alternative possibilities by the animator. The addition of numerous "sensor" links to the dynamic system, no longer an impediment to real-time operation, has been suggested as a way to control the shape of deformable surfaces. Again, this could remove considerable burden from the animator. There are applications of the parallel simulation to computer-aided manufacturing. For example, the graphical simulation of a robot manipulator in real-time can be useful during its design as well as for the training of operators. Recently, methods of using parallel processing for manipulator control have been proposed which depend on solving a scheduling problem [4], however that work takes the desired motion as input and produces joint forces and torques to realize the motion as output. In animation, the forces and torques (or control information from which they can be derived) are the inputs, and the motion is the output. In this way, built-in limitations on forces and torques constrain the animator to create a motion which could realistically be produced by muscular action.

---

* Received by the editors November 25, 1985; accepted for publication (in revised form) May 14, 1986.

† Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada, T6G 2H1.

§ Funding for this work was provided by the Natural Sciences and Engineering Research Council of Canada.

‡ VAX is a trademark of Digital Equipment Corporation, SUN is a trademark of Sun Microsystems, Inc., and UNIX is a trademark of AT&T Bell Laboratories.

The main thrust of our paper is to describe a parallel solution for determining the motion of tree linkages. In § 2 the equations are given and a solution is outlined which imposes an order on the computations reflecting the interconnection of links. In § 3 we show how these computations can be mapped to a tree of processes. Details of these computations are deferred to the Appendix. The Virtual Tree Machine multicomputer which supports the parallel implementation is described in § 4. Experimental and analytical results follow.

2. The equations of motion and their solution. The linkage trees considered consist of rigid links connected at hinges. The hinges allow three rotational degrees of freedom. This corresponds to a ball and socket joint. The reason for using this type of joint is the greater ease of formulating the equations of motion without the constraints which reduce the rotational degrees of freedom. Prismatic joints, which allow translations, are not considered. A typical linkage is shown in Figure 1.

place figure 1 about here

The following representations are used for physical quantities, whereby vectors, unless otherwise noted, are here represented in a frame attached to link r and moving with it: m_r is the mass of link r; a_G is the acceleration of gravity (in an inertial frame); f_E^r is an external force (inertial frame) acting on link r at the point p_E^r; g_E^r is an external torque (inertial frame) acting on link r; a^r is the acceleration of the proximal (parent) hinge of link r; ω^r is the angular velocity of link r; c^r is the vector from the proximal hinge to the center of mass of link r; f^r and g^r are the force and torque which link r exerts on its parent at the proximal hinge; l^s is the vector from the proximal hinge of link r to the proximal hinge of child s of link r. R^r converts vector representations in the frame of link r to the representations in the frame of the parent link; R_I^r converts to representations in the inertial frame; J^r is the 3×3 moment of inertia matrix of link r about its proximal hinge; S_r is the set of all links having link r as parent.

The first equation of motion, giving the rate of change of ω^r as a result of various torques, is:

$$J^r \dot{\omega}^r = g_\Sigma^r - m_r\, c^r \times a^r + \sum_{s \in S_r} l^s \times R^s\, f^s \tag{1}$$

where

$$g_\Sigma^r = -\omega^r \times (J^r\, \omega^r) - g^r + \sum_{s \in S_r} R^s\, g^s + {R_I^r}^T\, g_E^r \tag{2}$$

$$+ m_r\, c^r \times {R_I^r}^T\, a_G + p_E^r \times {R_I^r}^T\, f_E^r.$$

The next equation of motion gives the constraint force which link r exerts on its parent at the proximal hinge

$$f^r = f_\Sigma^r - m_r\, a^r + m_r\, c^r \times \dot{\omega}^r + \sum_{s \in S_r} R^s\, f^s \tag{3}$$

where

$$f_\Sigma^r = -m_r\, \omega^r \times (\omega^r \times c^r) + {R_I^r}^T\, (f_E^r + m_r\, a_G). \tag{4}$$

The last equation comes from the hinge constraints

$$R^s\, a^s = a_c^s + a^r - l^s \times \dot{\omega}^r \tag{5}$$

where

$$a_c^s = \omega^r \times (\omega^r \times l^s). \tag{6}$$

The hypothesis which aids in solving the equations is that there exist linear relationships

$$\dot{\omega}^r = K^r\ a^r + d^r \tag{7}$$

$$f^r = M^r\ a^r + f'^r. \tag{8}$$

In solving the equations of motion at each time step, we set the external forces f_E^r, the external torques g_E^r and the hinge torques g^r according to the control parameters, then we solve for K^r, d^r, M^r, and f'^r, starting at the leaves of the tree of links and proceeding towards the root. Intuitively, the solution method can be understood as follows. Suppose some agent accelerates a certain point on a rigid body r by an amount a^r. Then there will be a certain change of the angular velocity vector (represented in the frame of link r) given by the derivative $\dot{\omega}^r$. At the same time, there will be a force f^r (represented in frame r) acting upon the agent. Here the agent is the parent link of link r. In the absence of any child links, the equations of motion will suffice to determine $\dot{\omega}^r$ and f^r as a function of a^r. The function will be linear as expressed in equations (7) and (8) through the coefficients K^r, d^r, M^r, and f'^r. If there are child links, then link r will act as an agent to accelerate them. Assuming that the coefficients of (7) and (8) for the child links are known, and without knowing any of the accelerations, the coefficients for link r can be computed [3]. In this manner the coefficients for all links can be computed in an *inbound* pass towards the root link. Then, since the root is not subject to forces and torques from any parent body in the tree, we can solve the equations to get the linear and angular accelerations of all the links in an *outbound* pass toward the leaves.

3. Parallel implementation. The design of a parallel solution to these equations poses problems not encountered in the sequential case. Difficulties such as synchronizing processes, process to processor allocation, and communication overhead must be addressed [5], [6]. Synchronization overhead occurs any time a processor becomes idle, waiting for others to complete their tasks. Processor allocation deals with the problem of minimizing the possibility of two or more processes trying to use the same processor at the same time. Communication overhead is the additional burden incurred by sending messages between processes.

place figure 2 about here

For example, in real-time animation of the human body, the problem can be represented by a tree-like process structure with each process representing a link. In Figure 2, processes are assigned in a one-to-one fashion to each link in the stick-figure. If two processes which are neighbors in the tree are assigned to different processors (*single-link-per-process*), there is overhead involved in their exchange of data, but only slightly more than if they are assigned to the same processor. Alternatively, this overhead can be minimized by assigning several interconnected links to the same process (*multiple-links-per-process*). Both of these approaches are examined, but for simplicity the single-link-per-process model is used to explain the distributed implementation, unless otherwise stated.

No matter whether a single-link-per-process or multiple-links-per-process model is used, each cycle (time step) of the computation proceeds in three phases. In a first outbound phase, a process waits for control information from its parent, processes it, passes it on to all its descendants, and then waits. In the following inbound phase, a process must receive data from all its descendants before it can perform its computations and send the results back to the parent. In particular, the computation of the quantities K^r, d^r, M^r, and f'^r is done in this phase. The second outbound phase calculates the linear and angular accelerations of the links and performs part of their integration. Since not all nodes can be computing at the same time, the processes should be allocated to processors so that two processes that may be active at the same time are not on the same processor.

During initialization, the data from a non-distributed version of the dynamics program are read in and converted into the corresponding data for the distributed version. Following initialization, the dynamic equations are solved using a time step δt, whereby certain quantities are updated only every ν time steps. The functions which update the latter quantities

belong to what we shall call the "slowband" computations, and the rest will be said to belong to the "fastband." The state of the simulation is graphically displayed every μ cycles.

It is worth noting at this point that the difference of execution times during fast- and slowband cycles means that an assignment of links to processes and processes to processors which is optimal for execution of the fastband may cause synchronization problems for the slowband, and vice versa. This would considerably complicate an attempt to formulate and solve the mathematical performance optimization problem.

A complete simulation cycle during the step δt, including the slowband functions and a display call, which are really only done periodically, consists of the following sequence of calls by each link process:

Outbound control pass	Inbound pass	Outbound pass
slowband_control_out	receive_data_in	receive_data_out
fastband_control_out	slowband_integration_in	fastband_out
send_control_out	slowband_in	fastband_integration_out
	fastband_in	send_data_out
	send_data_in	

An optional pass can be done during initial testing runs to verify the correctness of the solution of the equations by using the fastband_check procedure.

The control functions transmit data from a control process through the root to the various links. Again some "control" quantities may be updated in the slowband: *slowband_control_out*. For example there can be certain external forces and torques on links which vary slowly (like wind pressure). Other "control" quantities vary quickly (relative to δt), such as the forces of contact with the earth or other objects, and the torques generated by muscular action. These are updated in *fastband_control_out*. We can model contacts with the earth or with fixed or moving objects using springs and dampers. In this way the contact does not need to be considered as a constraint, which would require reformulation of the equations of motion, and perhaps loss of the efficient solution technique. In order to compute contact forces and to display the links, the root must have the orientations of all links as determined by the rotation matrices. The latter can be used to compute accurate positions and orientations of all links with respect to the root link. Passing such information inward is done by *receive_data_in*.

place figure 3 about here

The three functions *slowband_in*, *fastband_in* and *fastband_out* perform the algebraic solution of the equations of motion. Given control forces and torques applied externally and the torques at the hinges generated by the muscles or motors, the linear accelerations of the hinges and the rates of change of the components of the angular velocity vectors of the links are determined. The *fastband_check* function determines whether the solution performed by those three functions is correct or not. The current simple integration technique in *fastband_integration_out* assumes that the previously mentioned linear and angular accelerations will be constant over the next time step. The accelerations are multiplied by δt and added to the current angular and linear velocities to get the new ones, and then the velocities are integrated again to get infinitesimal rotation vectors and positions. The *slowband_integration* call converts the infinitesimal rotation vectors into rotation matrices. To prevent accumulation of errors, the positions of the links are all derived from the root position using the rotation matrices. However, if positions are needed in the fastband for control purposes, this can be done using the results of *fastband_integration_out*. Figure 3 is a schematic view of the communications between an arbitrary interior node in the process tree, and one of its children.

4. The virtual tree machine. The facility used to solve the equations of motion is a *multi-computer* called the Virtual Tree Machine [7]. It is implemented on a network of autonomous VAX-11/780's and SUN-2 processors each running the 4.2BSD UNIX

operating system [8]. The Virtual Tree Machine consists of processes under operating system control and communication paths implemented as virtual connections between processes over a local area network. The experimenter views the machine as a collection of processors (interconnected to form a tree) - each with its own local memory and peripherals. In reality, a VTM is a collection of procedures, callable from application programs, and a collection of servers, responsible for the creation of the nodes in the tree-machine according to the description provided by the user. From this description the environment is created automatically with no intervention by the user. No restriction is placed on the mapping between virtual processing elements and physical processors. During development, the whole machine might reside on one physical processor before being distributed over the selected physical machines for productive use.

The Virtual Tree Machine makes it possible to investigate issues such as synchronization, allocation and communication overhead and thus can provide valuable insight into the behavior of unavailable real machines. For example, one such machine has tree structure and high parallelism and may prove suitable for the dynamics computations [9]. Algorithms for execution on these architectures can be developed, tested and debugged using this facility, and their synchronization and communication delay properties can be studied.

5. Experimental results. Several 10-second simulations of the man-like stick figure depicted in Figure 2 were carried out. Both the *single-link-per-process* and the *multiple-link-per-process* implementations were tried. The single-link-per-process model assigns each link to its own process and then assigns processes on a path to a leaf to the same processor P_i. One such assignment is shown in Figure 4. The advantage of this method is that it ensures that information necessary for calculations within each link is made available as soon as possible. Thus, the flow of data up and down the longest path from root to leaf of the process tree determines the length of the computation cycle. The disadvantage is that the cost of communicating all the needed information between processes is high.

place figure 4 about here

The solid line of Figure 5 depicts the time needed for simulation of the 12 link figure. The figure's "torso" is selected as the root of the process tree to minimize the depth of the tree and execution starts with the root reading the control information and passing it on down the tree. The children are idle until this information arrives. When the control information has propagated down to the leaves (the head and lower arms and legs of the figure) the inbound pass is started. Calculations for the inbound pass are made first in the leaf nodes and the results passed up to their parents which begin their inbound calculations only after receiving the inbound data from all their children. Once the root link has received all the inbound data from its subtrees and performed its own inbound calculations the outbound pass is started by calculating the outbound data and sending it to all the child links. Refer to Figure 3 for a graphical representation of the communication and synchronization characteristics of this implementation.

In our simulation the time-step is set to 0.01 second. The execution time is plotted against various degree of parallelism, from all processes assigned to one physical machine to the highest degree of useful parallelism (5 processors). For the optimal configuration of 5 processors, the execution time is reduced to 148 seconds from 227 seconds in the sequential case. It is not clear from these results whether it is the synchronization overhead or the communication overhead that plays the major role in the total overhead experienced. (The total overhead is over 100 seconds, comparing the sequential case to the single-link-per-process case when all processes run on a single machine.)

place figure 5 about here

The multiple-links-per-process model eliminates the need for communication between links on the same branch by allowing each process to handle more than one link. Several

links are assigned to each process, and each process is assigned to a different processor. Figure 6 shows one such assignment. Links are grouped into five processes and each process is assigned to a separate processor P_1 through P_5. Thus, only the solid lines in Figure 6 represent actual communication links. Here, calculations are done for all links assigned to a process before any information is sent on to its child processes or its parent. Although this reduces the communication overhead, the synchronization overhead increases as each link must wait slightly longer for the data from its parent or its children (if assigned to a different processor). The results from the multiple-links-per-process implementation are shown as the dotted line in Figure 5. The reason for the small drop in speed going from one process to two, is the fact that calculations for all links that share a process with the root link are done before any results are sent to the process handling the remaining links, resulting in almost sequential execution. For the 3 process case, the root link is given to a separate process and the remaining links are divided between the other two processes, resulting in close to parallel execution for all but one link.

place figure 6 about here

Comparison of the results from these two implementations shows that the synchronization overhead rather than communication overhead is the principal cause for the lack of performance of the parallel implementations. (Eliminating much of the communication does not result in significantly better performance.) We expect that simulating several similar stick figures at once, as would be required for certain animated sequences, would reduce the effect of the synchronization overhead and permit better speedup with nearly full processor utilization. In the next section we will analytically derive formulas for estimating the maximum speedup. This will lead to a better understanding of how to use parallelism.

6. A simple model for estimating maximum speedup. To obtain an upper bound on the possible relative improvement of the VTM implementation over the sequential implementation (the "speedup"), given arbitrarily many processors, we shall consider a simple case: balanced trees of height h and fanout f. We suppose that the first control pass is nonexistent, and that the inbound pass mirrors the outbound pass as far as time is concerned, so that we can limit consideration to just the outbound pass. The speedups for the two passes are different in our case, but dealing with the synchronization of communications in both passes at once is more complex, and so we shall use the more expensive of the two passes for our estimate. We assume further that the execution time for one link is equal for all links and is t_E. Similarly all communications from a link to a child take time $t_C \le t_E$.

Computation in the outbound pass of a tree of height $h+1$ starts at the root link, and takes time t_E. Then the root successively communicates to $f-1$ of its f children the information they require, taking time $(f-1)\,t_C$ to do it. The same processor goes on to compute one of the subtrees of height h, while the other subtrees of height h either have started, or are just starting. If an upper bound on the total time to complete the computation for a tree of height h and fanout f is denoted by $T(h, f)$, then we have

$$T(h+1, f) = t_E + (f-1)\,t_C + T(h, f).$$

Since $T(1, f) = t_E$, we get the general formula

$$T(h, f) = t_E + (h-1)(t_E + (f-1)\,t_C).$$

The number of nodes in the tree is $(f^h - 1) / (f - 1)$, and in a sequential version the computation time would be that number times t_E. Thus the speedup resulting from parallel computation is given by

$$\frac{f^h - 1}{(f-1)\,[\,1 + (h-1)(1 + (f-1)\,t_C / t_E\,)]}.$$

For zero communication time, this becomes

$$\frac{(f^h - 1)}{(f - 1)\, h}.$$

When $t_C = t_E$, we get the maximum speedup

$$\frac{(f^h - 1)}{(f - 1)[\, 1 + (h - 1)f\,]}.$$

If $h = 2$, there is no speedup at all in the latter case.

The computation time using the parallel algorithm and an unlimited supply of processors is thus theoretically shown to grow with the height of a balanced linkage tree rather than with its number of links. Considering communication times between links even as large as the execution times for the links worsens this improvement by a factor at most equal to the fanout.

place figure 7 about here

Experiments were done on balanced trees using the VTM. The execution time was varied by adjusting a waiting time t_E, while t_C was measured. The observed speedups shown in Figures 7 and 8 are close to the above theoretical predictions. If we examine the speedup which could be obtained with the tree representing our stick-figure (Figure 2) by using parallelism, we get $12t_E$ compared to $4(t_E + t_C)$, for a speedup of $3/(1 + t_C/t_E)$. Averaged over both the in- and the outbound pass for the stick figure, the ratio t_C/t_E is about 0.7 which, according to the above formula, gives maximum speedup of 1.76 with five processors. This is also in close conformity to our observations.

place figure 8 about here

For animation purposes, we may want to simulate several stick-figures at once, say four of them. Then t_C/t_E would be reduced by a factor of four, assuming that the cost of communication were unchanged. Then the speedup, by the above, would be 2.55, much closer to the theoretical limit of three. The synchronization overheads causing the nonlinear speedup (of three using five processors) remain, and could be dealt with by running several stick figures out of phase, so that the idle times of one stick-figure are used by others. For example, for two-stick figures on six processors, the synchronization overhead disappears if the link execution times are equal.

7. Conclusions. The parallel solution of the equations of motion of a tree-linkage using the VTM has been successful in providing a significant speedup. The speedup is a function of the tree size and configuration. The computation time using the parallel algorithm was shown to grow with the height of a balanced linkage tree rather than with its number of links. Considering communication times between links even as large as the execution times for the links worsens this exponential speedup by a factor at most equal to the fanout.

Two steps were taken which subsequently proved to be beneficial in reducing the effects of communication overheads: 1) several links were handled by a single Unix process, avoiding interprocess as well as interprocessor communications, and 2) information was not sent as soon as it became available within a process but was accumulated and sent in a single communication at the end of an iteration cycle (even though this runs counter to a dataflow philosophy).

The example of the 12-link human stick-figure which has low fanout, a height of four, and communication times which are comparable to the execution times, illustrates a case of poor speedup due to both communication and synchronization overheads. The simultaneous, in-phase simulation of several stick-figures at once would give a speedup near the maximum of three using five processors, since this reduces the ratio of communication time to execution time. To reduce synchronization overhead and approach linear speedup, several figures could be simulated out of phase so the idle times for one are used by the others.

The VTM implementation was, of course, much easier to implement and test than a physical network of processors would have been; yet the kinds of communication and synchronization problems closely reflected what could be expected in a "real" tree machine. Because there is negligible contention in the local area network while solving this problem, the VTM emulates a multiprocessor machine with point-to-point connections except for the time to form packets in the processes and ship them off serially. With more voluminous computations in the processes, and communications which better utilize the packet size, the approximation to point-to-point communications would be much closer than has been observed in the case of the stick-figure.

Many computing facilities currently use a local area network similar to the one on which the VTM is based, and hence, with the aid of VTM software, or its equivalent, it becomes possible to study parallel algorithms on networks of processors without resorting to actual construction of hardware.

REFERENCES

[1] J. WILHELMS, *Graphical simulation of the motion of articulated bodies such as humans and robots, with particular emphasis on the use of dynamic analysis*, Ph.D. Thesis, Computer Science Division, Department of Electrical Engineering and Computer Sciences, Univ. California, Berkeley, 1985.

[2] J. WILHELMS and B. BARSKY, *Using dynamic analysis to animate articulated bodies such as humans and robots*, Proc. Graphics Interface '85, Montreal (May 1985), pp. 97-104.

[3] W.W. ARMSTRONG and M.W. GREEN, *The Dynamics of articulated rigid bodies for purposes of animation*, The Visual Computer, 1 (1985), pp. 231-240 .

[4] H. KASAHARA and S. NARITA, *Parallel processing of robot-arm control computation on a multimicroprocessor system*, IEEE Journal of Robotics and Automation, RA-1 (1985), pp. 104-113 .

[5] J. MOHAN, *A Study of parallel computations - the traveling salesman problem*, Tech. Rep. CMU-CS-82-136, Dept. Comput. Sci. Carnegie Mellon Univ. (1982).

[6] T.A. MARSLAND and F. POPOWICH, *Parallel game-tree search*, IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-7 (1985), pp. 442-452 .

[7] M. OLAFSSON and T.A. MARSLAND, *A* UNIX *based virtual tree machine*, CIPS Congress '85, Montreal (June 1985), pp. 176-181.

[8] D. M. RITCHIE and K. THOMPSON, *The* UNIX *time-sharing system*, Bell Sys. Tech. J., 57 (1978), pp. 1905-1929 .

[9] *Myrias* 4000 *System Description*, Myrias Research Corporation, Edmonton, February 1986.

Appendix: Details of the dynamics computations. The details of the dynamics computations for the VTM implementation are outlined here. No attention is paid to the control pass, since that could vary considerably depending on how control is to be achieved. In the following, the unit 3-by-3 matrix will be denoted by **I**. The tilde operation on a 3-vector v, with components $v1$, $v2$, $v3$, produces a 3-by-3 matrix V such that for any 3-vector w, the vector $v \times w$ is equal to the product of V and the column-vector w.

$$\tilde{v} = V = \begin{bmatrix} 0 & -v3 & v2 \\ v3 & 0 & -v1 \\ -v2 & v1 & 0 \end{bmatrix}.$$

The following variables are also introduced in the solution since they appear as common subexpressions or are computed in the slowband and used in the fastband: W^r, T^r, Q^r, a_c^r, f''^r.

The *receive_data_in* function gets f'^s, δu^s (which is an infinitesimal rotation vector), and ω^s from all children s of link r. In addition, during a slowband pass, *receive_data_in* gets M^s, and R_I^s. The force f'^s is the part of the force exerted by the child s on the proximal hinge which is independent of the child's acceleration. The difference between δu^s and δu^r (converted to frame s) can be used in determining a torque due to rapid changes in the angles at hinge s deriving from hinge stiffness. The difference between the omegas (the parent one converted) is similarly used for computing a frictional torque at that hinge. These quantities are treated as fast-varying, although ω, whose rate of change depends on a physical inertia, should vary slowly and could probably be placed in the slowband. (N. B. This points out a need for careful study of the rates of change of various quantities which could lead to

improved simulations involving more than two bands. In such simulations, one might also determine, either beforehand or adaptively, that certain terms in our solution can be entirely neglected.)

The *slowband_integration_in* computation takes the infinitesimal rotation vector δu^r and uses it to update the rotation matrix R_I^r, resulting in a new link orientation. The result must be orthonormalized to prevent accumulation of errors.

The *slowband_in* computation for link r proceeds as follows:

Compute $a_c^s = \omega^r \times (\omega^r \times l^s)$ for $s \in S_r$. (9)

Compute $Q^s = R^s \; M^s \; R^{s\,T}$ for $s \in S_r$. (10)

Compute $W^s = \tilde{l}^s \; Q^s$ for $s \in S_r$. (11)

Compute $T^r = (J^r + \sum_{s \in S_r} W^s \; \tilde{l}^s)^{-1}$. (12)

Compute $K^r = T^r (\sum_{s \in S_r} W^s - m_r \; \tilde{c}^r)$. (13)

Compute $M^r = (m_r \; \tilde{c}^r) \; K^r - m_r \; \mathbf{I} + \sum_{s \in S_r} Q^s (\mathbf{I} - \tilde{l}^s \; K^r)$. (14)

Compute $g_\Sigma^{1r} = - \omega^r \times (J^r \; \omega^r) + R_I^{r\,T} \; g_E^r$ (15)

$$+ p_E^r \times R_I^{r\,T} \; f_E^r + (m_r \; c^r) \times R_I^{r\,T} \; a_G$$

(assuming f_E^r and g_E^r are slowly-varying).

Compute $f_\Sigma^r = - \omega^r \times (\omega^r \times (m_r \; c^r)) + R_I^{r\,T} (f_E^r + m_r \; a_G)$ (16)

(assuming f_E^r is slowly-varying).

The *fastband_in* computation proceeds as follows:

Compute $f''^s = R^s \; f'^s$ for $s \in S_r$. (17)

Compute $g_\Sigma^r = g_\Sigma^{1r} - g^r + \sum_{s \in S_r} R^s \; g^s$. (18)

Compute $d^r = T^r (g_\Sigma^r + \sum_{s \in S_r} l^s \times (f''^s + Q^s \; a_c^s))$. (19)

Compute $f'^r = f_\Sigma^r + (m_r \; c^r) \times d^r$ (20)

$$+ \sum_{s \in S_r} (f''^s + Q^s (a_c^s - l^s \times d^r))$$

The *send_data_in* function, which occurs last in the inbound pass, provides the data to the parent for its *receive_data_in* function. In the multiple-links-per-process version, this data is, of course, not sent or received, but merely written to and accessed from the shared memory possible when only one process is involved.

Now we turn our attention to the outbound pass. The first function there is *receive_data_out*. All it does for link r is get a^r from the parent. If r is the root, this quantity must instead be computed, which is done in the following computation (equation 21). The *fastband_out* computation for link r proceeds as follows:

If r is the root then compute $a^r = - (M^r)^{-1} \; f'^r$. (21)

Compute $\dot{\omega}^r = K^r \; a^r + d^r$. (22)

Compute $a^s = R^{s\,T} (a_c^s + a^r - l^s \times \dot{\omega}^r)$ (23)

for all children of link r.

In *fastband_integration_out*, $\dot{\omega}^r$ is multiplied by δt and the result added to ω. Then δu^r is obtained from a further integration step. The velocity and position of the root link are similarly determined by integration starting from a^1. Recall that other links have their positions determined by the root's position and link orientations. This integration technique is not stable, and development of some efficient "implicit" technique would be very useful. For animated figures, accuracy is not the goal, only realism, and the present method should be satisfactory. In an interactive system, the animator will correct for inaccuracies. Applications in robotics or biomechanics require better accuracy and stability. To the best of our knowledge, how to achieve this is an open question.

The *send_data_out* procedure merely sends a^s to all children of r. The *fastband_check* procedure goes as follows:

Receive f^s from all children.

Compute $f^r = M^r\ a^r + f'^r$. (24)

Compute $\dot{\omega}^r$ from equations 1 and 2, and check that it agrees with the solution value.

Compute f^r from equations 3 and 4, check that it agrees with the value computed in (24), and send it to the parent link.

SIAM EDITOR: FIGURE CAPTIONS

FIG. 1. *Quantities associated with link* r.

FIG. 2. *Stick figure and the corresponding process tree*.

FIG. 3. *Quantities communicated between links*.

FIG. 4. *Assignment of links to twelve processes on five processors*.

FIG. 5. *Execution time vs. number of physical processors*.

FIG. 6. *Assignment of links to five processes on five processors*.

FIG. 7. *Observed speedup vs. communication to execution time ratio* (t_C/t_E).

FIG. 8. *Observed speedup vs. communication to execution time ratio* (t_C/t_E).