Richard Harold Lathrop
Artificial Intelligence Laboratory
Massachusetts Institute of Technology
545 Technology Square
Cambridge, Massachusetts 02139

Parallelism in Manipulator Dynamics

Abstract

This paper addresses the problem of efficiently computing the motor torques required to drive a manipulator arm in free motion, given the desired trajectory—that is the inverse dynamics problem. It analyzes the high degree of parallelism inherent in the computations and presents two "mathematically exact" formulations especially suited to high-speed, highly parallel implementations using VLSI devices. The first method presented is a parallel version of the recent linear Newton-Euler recursive algorithm. The time cost is linear in the number of joints, but the real-time coefficients are reduced by almost two orders of magnitude. The second formulation reports a new parallel algorithm that shows that it is possible to improve on the linear time dependency. The real time required to perform the calculations increases only as the [log_2] of the number of joints. Either formulation is susceptible to a systolic pipelined architecture in which complete sets of joint torques emerge at successive intervals of four floating-point operations. Hardware requirements necessary to support the algorithm are not excessive, and a VLSI implementation architecture is possible.

Keywords: Robots, robotics, industrial robots, manipulator dynamics, inverse dynamics, parallel processing, pipeline processing, large-scale integration.

1. Introduction

In active, articulated mechanisms, the parameters that one typically can directly control are actuator currents and voltages, and thus the joint forces and torques. Unfortunately, the parameters of interest are frequently the linear and rotational accelerations (hence, velocities and positions).

One proposal is to use rigid-body rotational dynamics to calculate the joint forces and torques that would produce the desired motion (that is, to solve the inverse plant problem). If these could be calculated and applied to the actuators in real time, perhaps the resulting accelerations would be sufficiently accurate that relatively simple feedback would suffice. This calculation is of interest to robot simulation. The most efficient simulation algorithms (Orin and McGhee 1981; Walker and Orin 1981; Featherstone 1984) use it to compute a composite-rigid-body inertia matrix, which is then solved to yield the joint accelerations.

This paper presents two inverse dynamics formulations suited to highly parallel implementations. Table 1 shows the improvement over serial implementations. The first formulation is linear in the number of joints and reduces the real-time coefficients by almost two orders of magnitude to

$$(2n + 3) \text{ mults} + (6n + 7) \text{ addns.} \qquad (1)$$

The second formulation exploits a novel parallel algorithm, described below, to attain $\mathcal{O}(\log(n))$, achieving

$$(2[\log_2(n + 1)] + 5) \text{ mults} + (6[\log_2(n + 1)] + 10) \text{ addns.} \qquad (2)$$

Either formulation is susceptible to a systolic pipelined architecture, with a basic time cycle of 1 mult + 3 addns. This yields the ability to rapidly evaluate many alternatives. The principal thrust of this paper lies in these formulations, but one implementation architecture (of many possible) is explored.

2. Inverse Dynamics

The inverse dynamics problem is as follows: Given the joint accelerations desired, find the joint forces and torques necessary. The problem is posed by giving the joint positions and velocities $(\{q_i(t_0), \dot{q}_i(t_0)\})$ that describe the state of the manipulator at a given point in time (t_0), together with the joint accelerations that are desired at that point $(\{\ddot{q}_i(t_0)\})$. The answer expected is the set of n motor torques $(\{\tau_i(t_0)\})$ at each joint that would induce the desired accelerations. Typically, one

The International Journal of Robotics Research,
Vol. 4, No. 2, Summer 1985,
0278-3649/85/020080-22 $05.00/0,

Table 1. Comparison of Dynamics Formulations (Adapted from Hollerbach 1980)

	Comparison of Time Dependencies		Comparison for $n = 6$	
Method	Mults.	Addns.	Mults.	Addns.
Uicker/Kahn (original Lagrangian)	$32\frac{1}{2}n^4 + 86\frac{5}{12}n^3 + 171\frac{1}{4}n^2 + 53\frac{1}{3}n - 128$	$25n^4 + 66\frac{1}{3}n^3 + 129\frac{1}{2}n^2 + 42\frac{1}{3}n - 96$	66,271	51,548
Waters (partially recursive)	$106\frac{1}{2}n^2 + 620\frac{1}{2}n - 512$	$82n^2 + 514n - 384$	7,051	5,652
Hollerbach (4 × 4 Lagrangian)	$830n - 592$	$675n - 464$	4,388	3,586
Hollerbach (3 × 3 Lagrangian)	$412n - 277$	$320n - 201$	2,195	1,719
Luh, Walker, Paul (Newton-Euler)	$150n - 48$	$131n - 48$	852	738
Horn, Raibert (table lookup)	$2n^3 + n^2$	$n^3 + n^2 + 2n$	468	264
Luh, Lin (scheduled parallel Newton-Euler)	$57n - 18$ (estimated; see text)	$50n - 18$ (estimated)	323 (estimated)	280 (estimated)
Lathrop (linear parallel Newton-Euler)	$2n + 3$	$6n + 7$	15	43
Lathrop (logarithmic parallel Newton-Euler)	$2[\log_2(n + 1)] + 5$	$6[\log_2(n + 1)] + 10$	11	28
Lathrop (systolic pipeline)	1 (successive; see text)	3 (successive)	1 (successive)	3 (successive)

NOTE: This table reflects the algorithmically indicated cost for the fully general six-link rotary manipulator, as in Hollerbach (1980). By consideration of special cases, introduction of configuration or workspace assumptions, or tailoring of the computation, additional reductions are possible.
Tables I & II from J. M. Hollerbach, "A Recursive Formulation of Manipulator Dynamics," *IEEE Frame Systems, Man & Cybernetics,* SMC-10, No. 11, pp. 730–736, have been adapted to include new algorithms since 1980.

measures q and $\dot{q}$, while $\ddot{q}$ is supplied by a higher-level planning and control module.

Computing the motor torques is quite complicated, however, because of the high degree of nonlinearity inherent in rigid-body rotational mechanics. The torques supplied must compensate for the inertia of the manipulator, gravitational force, the Coriolis and centrifugal forces, and viscous friction at the joints. Viscous frictional forces often depend only on q_i and $\dot{q}_i$ at joint i; hence they are susceptible to relatively simple correction and will hereafter be ignored. All of the other terms vary in a highly nonlinear fashion depending on the manipulator configuration at a given point in time; additionally, the Coriolis and centrifugal forces depend on all pairwise products ($\dot{q}_i\dot{q}_j$, $1 \leq i$, $j \leq n$) of joint velocities. This complicated computation has until recently posed a bottleneck in on-line control of manipulators, and much effort has been expended in devising more time-efficient methods.

3. Previous Work

Resonant frequencies of many mechanical manipulators are around 10 Hz, so if this approach is to be used, the computation must be repeated at about 60 Hz or faster (Luh, Walker, and Paul 1980*a*). Uicker (1965) and Kahn (1969) derived an early formulation having an $\mathcal{O}(n^4)$ time complexity and requiring 7.9 seconds on a PDP 11/45 for just one trajectory point (Luh, Walker, and Paul 1980). Efforts to improve this time have typically explored other computational algorithms (Paul 1972; Whitney 1972; Vereshchagin 1974; Waters 1979; Hollerbach 1980; Luh, Walker

Table 2. Linear Recursive Newton-Euler Formulation (After Luh, Walker, and Paul 1980)

Newton-Euler backward recursion:

$$\omega_i = \begin{cases} \mathbf{A}_i^T(\omega_{i-1} + \mathbf{z}_{i-1}\dot{q}_i) & \text{if joint } i \text{ rotational;} \\ \mathbf{A}_i^T\omega_{i-1} & \text{if joint } i \text{ translational} \end{cases}$$

$$\dot{\omega}_i = \begin{cases} \mathbf{A}_i^T(\dot{\omega}_{i-1} + \mathbf{z}_{i-1}\ddot{q}_i + \omega_{i-1} \times \mathbf{z}_{i-1}\dot{q}_i) & \text{if joint } i \text{ rotational;} \\ \mathbf{A}_i^T\dot{\omega}_{i-1} & \text{if joint } i \text{ translational} \end{cases}$$

$$\ddot{\mathbf{p}}_i = \begin{cases} \mathbf{A}_i^T\ddot{\mathbf{p}}_{i-1} + \dot{\omega}_i \times \mathbf{p}_i^* + \omega_i \times (\omega_i \times \mathbf{p}_i^*) & \text{if joint } i \text{ rotational;} \\ \mathbf{A}_i^T\ddot{\mathbf{p}}_{i-1} + \dot{\omega}_i \times \mathbf{p}_i^* + \omega_i \times (\omega_i \times \mathbf{p}_i^*) + \mathbf{A}_i^T\mathbf{z}_{i-1}\ddot{q}_i + 2\omega_i \times \mathbf{A}_i^T\mathbf{z}_{i-1}\dot{q}_i & \text{if joint } i \text{ translational} \end{cases}$$

$$\ddot{\mathbf{r}}_i = \omega_i \times (\omega_i \times \mathbf{r}_i^*) + \dot{\omega}_i \times \mathbf{r}_i^* + \ddot{\mathbf{p}}_i$$

$$\mathbf{F}_i = m_i\ddot{\mathbf{r}}_i$$

$$\mathbf{N}_i = \mathbf{J}_i\dot{\omega}_i + \omega_i \times (\mathbf{J}_i\omega_i)$$

Newton-Euler forward recursion:

$$\mathbf{f}_i = \mathbf{F}_i + \mathbf{A}_{i+1}\mathbf{f}_{i+1}$$

$$\mathbf{n}_i = \mathbf{A}_{i+1}\mathbf{n}_{i+1} + \mathbf{N}_i + \mathbf{s}_i^* \times \mathbf{F}_i + \mathbf{p}_i^* \times (\mathbf{A}_{i+1}\mathbf{f}_{i+1})$$

$$\tau_i = \mathbf{z}_{i-1} \cdot \mathbf{n}_i$$

Equations from LVH, Walker & Paul, *ASME Journal of Dynamic Systems Measurement and Control,* June 1980, vol. 102, pp. 69–76, recast into equivalent notation.

and Paul 1980*a*, 1980*b*; Hollerbach and Sahar 1983; Kane and Levinson 1983), made simplifying assumptions (Paul 1972; Bejczy 1974), or substituted table lookup for computation (Albus 1975*a*, 1975*b*; Raibert 1978; Raibert and Horn 1978). Table 1, reproduced from Hollerbach (1980), compares the true dependencies of several dynamics formulations.

3.1. Recursive Formulations

The most successful formulations involve recursive algorithms. In a little-known paper dealing mostly with direct dynamics, Vereshchagin (1974) developed the first partial-recursive $\mathcal{O}(n^2)$ inverse dynamics algorithm from Gauss's principle of least constraint, including a fully linear backward recursion. Waters (1979) first presented an $\mathcal{O}(n^2)$ partial recursive form of the Lagrange equations, which was made fully linear recursive by Hollerbach (1980). Orin and colleagues (1979) first presented a linear recursive form of the Newton-Euler equations, which was refined by Luh, Walker, and Paul (1980*a*). We have considered both the Newton-Euler and the Lagrangian formulations, which Silver (1982) has shown to be fundamentally equivalent. This paper presents only Newton-Euler results, the linear recursive formulas for which are shown in Table 2 (Luh, Walker, and Paul 1980). The Lagrangian attains essentially the same time bounds using substantially more hardware.

In both cases, the motion of the manipulator base and each individual joint is assumed known. Beginning at the base allows the motion of each successive link to be cascaded recursively to the tip. The forces and torques applied by the environment to the last manipulator link are known or measured. By propagating this toward the base through each joint, one can calculate in linear time the force or torque necessary at each joint to drive each link through its desired motion.

3.2. Off-CPU Control

A number of practical attempts have been made to relieve the host computer of some of the computational burden associated with manipulator control. A common strategy is to use a single microprocessor to control the robot and to use the host to control the controller, (Dubowsky 1981; Kopacek 1981; Guo and Koivo 1982). Most attempts to apply parallel processing to manipulator control have involved the use of microprocessors to servo individual joints and have not attempted to include dynamic considerations (Kuo 1981; Rafauli, Sinha, and Tlusty 1981; Shin and Malin 1981; Besant 1982; Carlisle 1982; Cook et al. 1982; Goshorn 1982; Gupta 1982; Klein and Wahawisan 1982; Martin, Satterlee, and Bolfing 1982). Alternatively, the host computer incorporates attached special-purpose processors for real-time numerically intensive computations (Mudge 1981; Lee, Mudge, and Turney 1982; Mudge and Turney 1982; Turner, Craig, and Gruver 1982).

3.3. Parallelism in Dynamics

Orin (1984) divides the overall problem of robot control into ten necessary tasks and considers their approximate relative execution times. The longest is the inverse dynamics, requiring eight times as long as most tasks, twice as long as the second longest, and consuming over one-third of the total time required. By executing some tasks concurrently on several CPUs, Orin is able to reduce the total time by nearly one-third and to nearly triple the throughput, but the inverse dynamics task is active for half the total real time of execution. Reducing the computation time of this task is identified as the only way to further increase the throughput, and he discusses the desirability of pipelining the inverse dynamics.

In the work most closely related to this paper, Luh (1981) and Luh and Lin (1982) describe a procedure for scheduling the subtasks of a group of microprocessors computing the Newton-Euler inverse dynamics. One microprocessor is again assigned to each controlled joint axis, and each microprocessor computes the recursion variables that correspond to its joint axis. Since these variables (as well as intermediate partial results) recursively depend on each other in different ways, often some microprocessor(s) will be idle while waiting for others to complete pending subtasks, and idle time must be included in the schedule. A modified branch-and-bound search finds a specific execution sequence that minimizes the global computation time because in general each new manipulator configuration requires a new rescheduling of subtasks. Though by its nature this procedure is not subject to precise analysis of the computational complexity of the resulting schedules, the authors report a concurrency factor of 2.64 on the Stanford arm. This estimated factor is used in Table 1 for the fully general manipulator.

4. Notation

The notation is based on that used by Hollerbach (1980), Luh, Walker, and Paul (1980*a*), and Lathrop (1983) in their analyses of the linear recursive inverse dynamics, which in turn derives from the Denavit-Hartenberg (1955) convention for lower-pair chains. It is summarized in Table 3.

5. Parallelism Within a Node

This section investigates the effect of exploiting parallelism within a node (joint) while remaining within the general linear recursive framework. As a conceptual aid in this section, we assume that there is one group of parallel processors for each joint (node) on the forward and the backward recursion, though an implementation could be constructed using only one processor group by connecting the output back to the input through a buffer. If only one processor group is used, computation of one set of joint torques must be completed before the next can begin. Otherwise, with one processor group for each joint (node) it is possible to systolically pipeline successive sets of joint torques at intervals of 1 mult + 3 addns, or 4 flops.

The linear recursive structure of the Newton-Euler computations is shown as a directed graph in Fig. 1. Acceleration is propagated outward, incorporating at each stage the next joint acceleration, and forces are thereafter propagated inward, allowing calculation of joint torque at each stage. Nodes represent the total processing associated with each joint in the forward or backward recursion, and directed arcs represent data dependencies. Nonsystolic and systolic arrangements are indicated in Figs. 2 and 3, where the nodes represent the associated processing. In the nonsystolic phys-

Table 3. Summary of Notation Used

m_j:	The mass of link j.
$\mathbf{r}_j^*$:	A vector to the center of mass of link j from the origin $\mathbf{O_j}$.
$\mathbf{p}_j^*$:	A vector to the origin $\mathcal{O}_j$ from the origin $\mathcal{O}_{j-1}$.
$\ddot{\mathbf{p}}_j^g$:	Accounts for gravitational acceleration, g, if $j = 0$, or 0.
$\mathbf{s}_j^*$:	$\mathbf{p}_j^* + \mathbf{r}_j^*$, a vector to the center of mass of link j from the origin $\mathcal{O}_{j-1}$.
ω_j:	The angular velocity vector of link j.
$\mathbf{J}_j$:	The inertial tensor of link j.
q_j:	The joint generalized variable for joint j, θ if rotational and s if translational.
τ_j:	The joint generalized actuator force at joint j, torque if rotational and force if translational.
$\mathbf{F}_j$:	The total force (excluding gravity) on link j.
$\mathbf{N}_j$:	The total torque on link j.
$\mathbf{f}_j$:	Constraint force (unknown) exerted on link j by link $j - 1$.
$\mathbf{n}_j$:	Constraint torque (unknown) exerted on link j by link $j - 1$.
$\mathbf{A}_j$:	A pure rotation matrix mapping vectors in $\mathcal{O}_j$ into vectors in $\mathcal{O}_{j-1}$.
${}^i\mathbf{W}_j$:	A pure rotation matrix mapping vectors in $\mathcal{O}_j$ into vectors in $\mathcal{O}_i$.
$\mathbf{W}_{i,j}$:	${}^{i-1}\mathbf{W}_j$.
$\mathbf{u}_i$:	${}^i\mathbf{u}_i$ (referred to link coordinates, $\mathcal{O}_i$).
$\dot{\mathbf{u}}_{a,b}$:	$\mathbf{u}_{a,b}$ differentiated in $\mathcal{O}_{a-1}$ and referred to $\mathcal{O}_b$.

ical structure of Fig. 2, recursion is implemented by connecting each node's output back to its input, perhaps through buffers. In the systolic Fig. 3, recursion is implemented by connecting each node's output to the next processor.

Table 4 shows the detailed internal structure of each node of the directed graph of Fig. 1. The computation times given in Table 4 reflect the times represented by the indicated operation. In general we follow Hollerbach (1980), and Tables 1 and 4 reflect his analysis for the fully general six-link rotary manipulator. By hand-tailoring the computation to account for special cases, one frequently can reduce the computational cost below that shown (Luh and Lin 1982; Hollerbach and Sahar 1983; Kane and Levinson 1983; Lathrop 1983).

The times shown in Tables 1 and 4 reflect, for each variable, the rotational case only. The extension to the translational case follows directly in analogous fashion, and the time bounds stated remain almost exactly the same with only minor variation in the constant term.

While on either the forward or the backward recursion, the linear time coefficient is determined by the longest time required to propagate any single variable across the node. In determining this time only ω_i, $\dot{\omega}_i$, $\ddot{\mathbf{p}}_i$, $\mathbf{f}_i$, and $\mathbf{n}_i$ need be considered, as the other variables are not propagated down the recursive chain. From Table 4 we see that each propagated variable becomes available to the $(i + 1)$st node one matrix-vector multiplication (MV) and one vector addition (VA) after it is made available to the ith node. Thus the time required for the entire dynamics calculation reduces to

$$2n \cdot (MV + VA) + C, \tag{3}$$

where C is a constant that accounts for initiating and finishing the calculation.

This is the case even though each individual variable may take longer than (MV + VA) to compute. Figure 4 illustrates this arrangement graphically for ω_i and $\dot{\omega}_i$. The times when each of the variables becomes available are shown as well as the intermediate partial results. For simplicity of presentation, a timing model is used in which 1 mult = 1 addn = 1 flop.

For example, ω_i depends on nothing but ω_{i-1} and

Fig. 1. Linear recursive graph structure. $Bkwd_i$ = backward recursion processor for joint i; Fwd_i = forward.

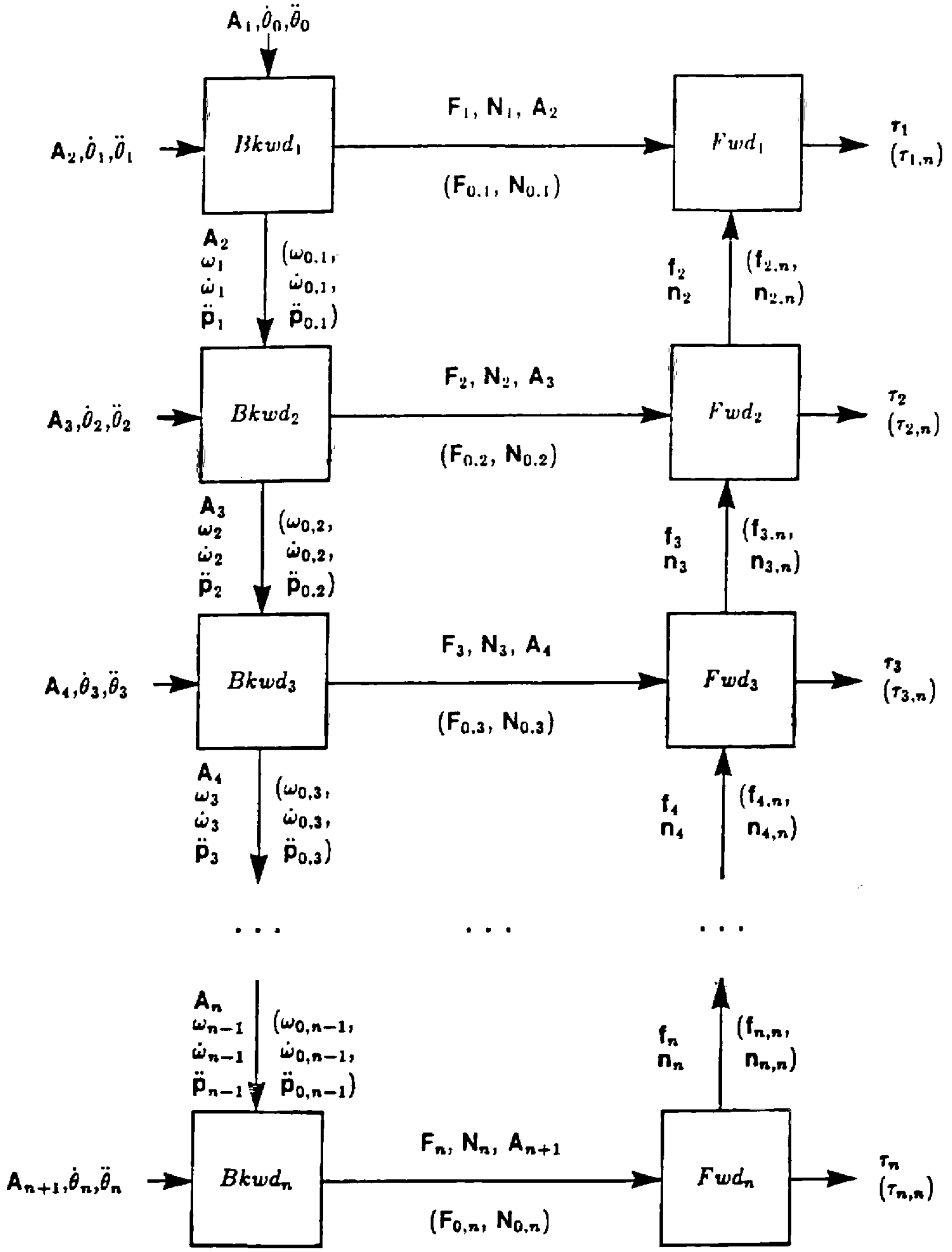


Fig. 2. Nonsystolic pipelined process, $n = 4$. Time intervals $2n(MV + VA) + C$ per complete set of joint torques.

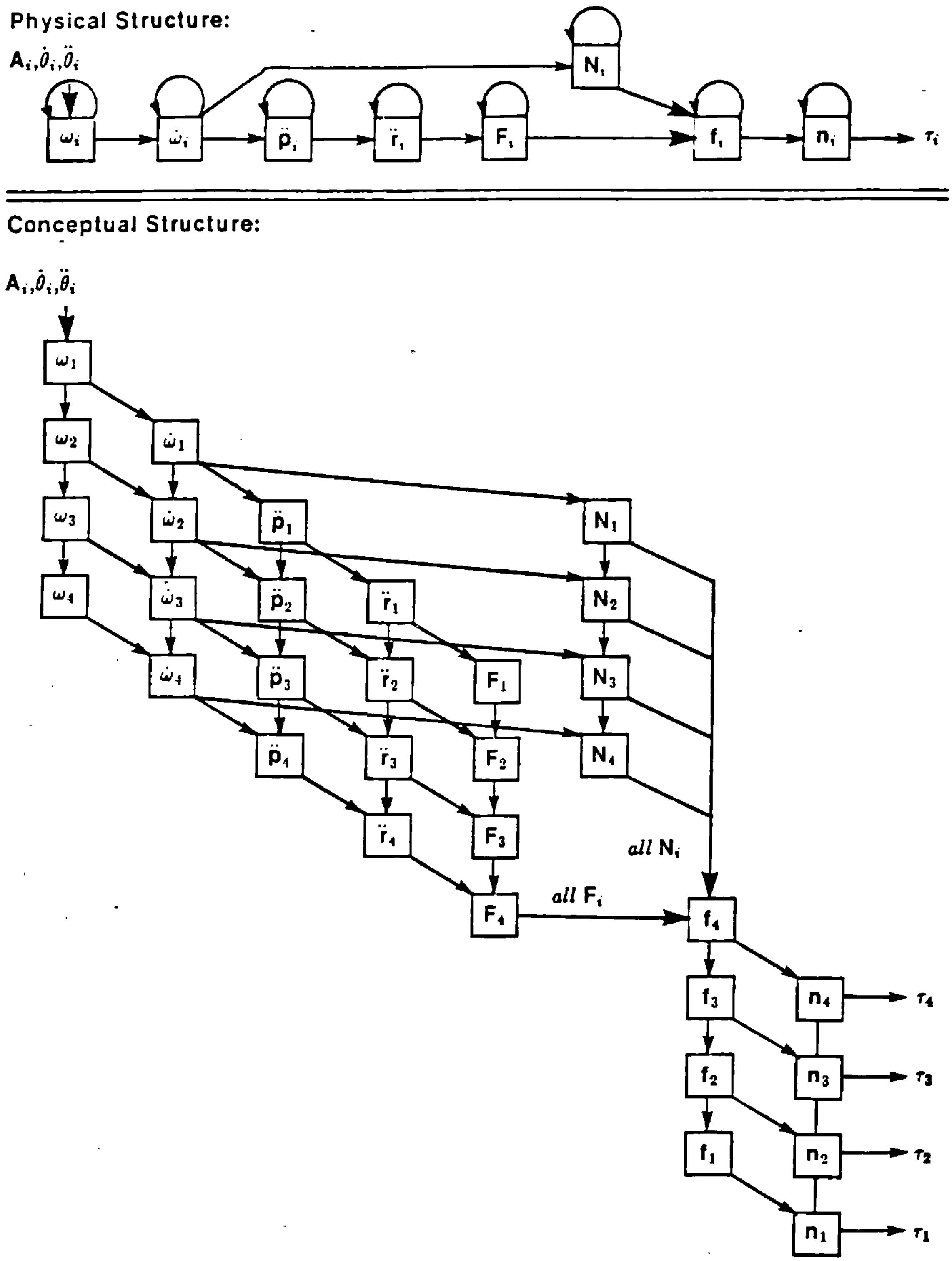


Fig. 3. Systolic pipelined process, $n = 4$. Time intervals $1(MV + VA)$ per complete set of joint torques. $Bkwd_i$ = backward recursion processor for joint i; Fwd_i = forward.

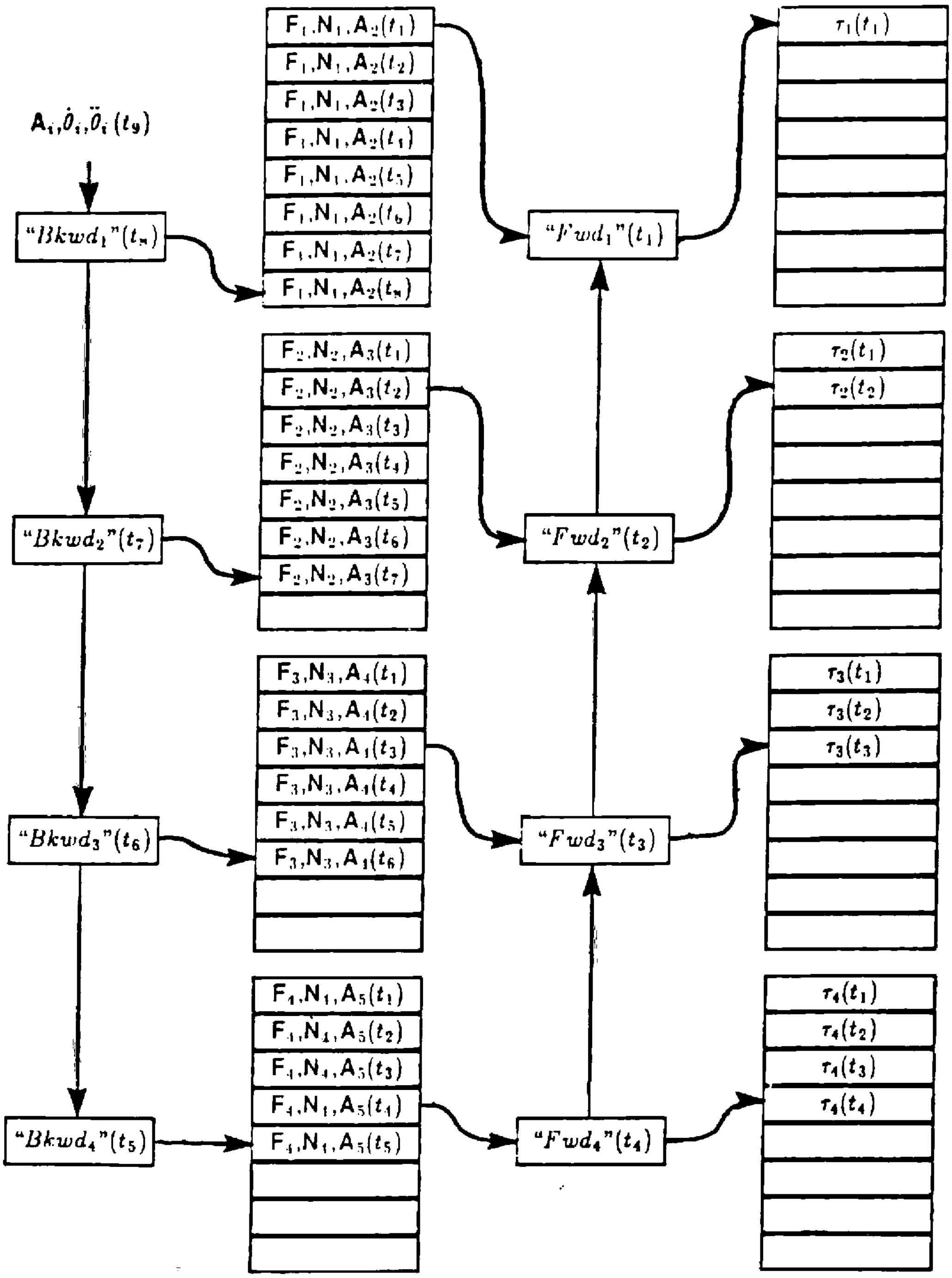


Table 4. Relative Time of Linear Data Dependencies

Linear Backward Newton-Euler Recursion

Variable	Waits On†	Time at Step End	Step	Cost‡
$\boldsymbol{\omega}_i$	$a = \text{Avail}(\omega_{i-1})$	Input	$T_1 = \mathbf{z}_{i-1}\dot{q}_i$	–
		$a + \text{VA}$	$T_2 = \omega_{i-1} + T_1$	VA
		$a + \text{MV} + \text{VA}$	$\omega_i = \mathbf{A}_i^T T_2$	MV
$\dot{\omega}_i$	$a = \text{Avail}(\omega_{i-1})$	Input	$T_3 = \mathbf{z}_{i-1}\ddot{q}_i$	–
	$b = \text{Avail}(\dot{\omega}_{i-1})$	$a + \text{VC}$	$T_4 = \omega_{i-1} \times T_1$	VC
		$a + \text{VC} + \text{VA}$	$T_5 = T_3 + T_4$	VA
		$\max(b, a + \text{VC} + \text{VA}) + \text{VA}$	$T_6 = T_5 + \dot{\omega}_{i-1}$	VA
		$\max(b, a + \text{VC} + \text{VA}) + \text{MV} + \text{VA}$	$\dot{\omega}_i = \mathbf{A}_i^T T_6$	MV
$\ddot{\mathbf{p}}_i$	$c = \text{Avail}(\omega_i)$	$c + \text{VC}$	$T_7 = \omega_i \times \mathbf{p}_i^*$	VC
	$d = \text{Avail}(\dot{\omega}_i)$	$c + 2\text{VC}$	$T_8 = \omega_i \times T_7$	VC
	$e = \text{Avail}(\ddot{\mathbf{p}}_{i-1})$	$d + \text{VC}$	$T_9 = \dot{\omega}_i \times \mathbf{p}_i^*$	VC
		$\max(d + \text{VC}, c + 2\text{VC}) + \text{VA}$	$T_{10} = T_8 + T_9$	VA
		$e + \text{MV}$	$T_{11} = \mathbf{A}_i^T \ddot{\mathbf{p}}_{i-1}$	MV
		$\max(e + \text{MV}, d + \text{VC} + \text{VA}, c + 2\text{VC} + \text{VA}) + \text{VA}$	$\ddot{\mathbf{p}}_i = T_{10} + T_{11}$	VA
$\ddot{\mathbf{r}}_i$	$c = \text{Avail}(\omega_i)$	$c + \text{VC}$	$T_{12} = \omega_i \times \mathbf{r}_i^*$	VC
	$d = \text{Avail}(\dot{\omega}_i)$	$c + 2\text{VC}$	$T_{13} = \omega_i \times T_{12}$	VC
	$f = \text{Avail}(\ddot{\mathbf{p}}_i)$	$d + \text{VC}$	$T_{14} = \dot{\omega}_i \times \mathbf{r}_i^*$	VC
		$\max(d + \text{VC}, c + 2\text{VC}) + \text{VA}$	$T_{15} = T_{13} + T_{14}$	VA
		$\max(f, d + \text{VC} + \text{VA}, c + 2\text{VC} + \text{VA}) + \text{VA}$	$\ddot{\mathbf{r}}_i = T_{15} + \ddot{\mathbf{p}}_i$	VA
$\mathbf{F}_i$	$g = \text{Avail}(\ddot{\mathbf{r}}_i)$	$g + \text{SV}$	$\mathbf{F}_i = m_i \ddot{\mathbf{r}}_i$	SV
$\mathbf{N}_i$	$c = \text{Avail}(\omega_i)$	$c + \text{MV}$	$T_{16} = \mathbf{J}_i \omega_i$	MV
	$d = \text{Avail}(\dot{\omega}_i)$	$c + \text{MV} + \text{VC}$	$T_{17} = \omega_i \times T_{16}$	VC
		$d + \text{MV}$	$T_{18} = \mathbf{J}_i \dot{\omega}_i$	MV
		$\max(c + \text{VC}, d) + \text{MV} + \text{VA}$	$N_i = T_{17} + T_{18}$	VA

Linear Forward Newton-Euler Recursion

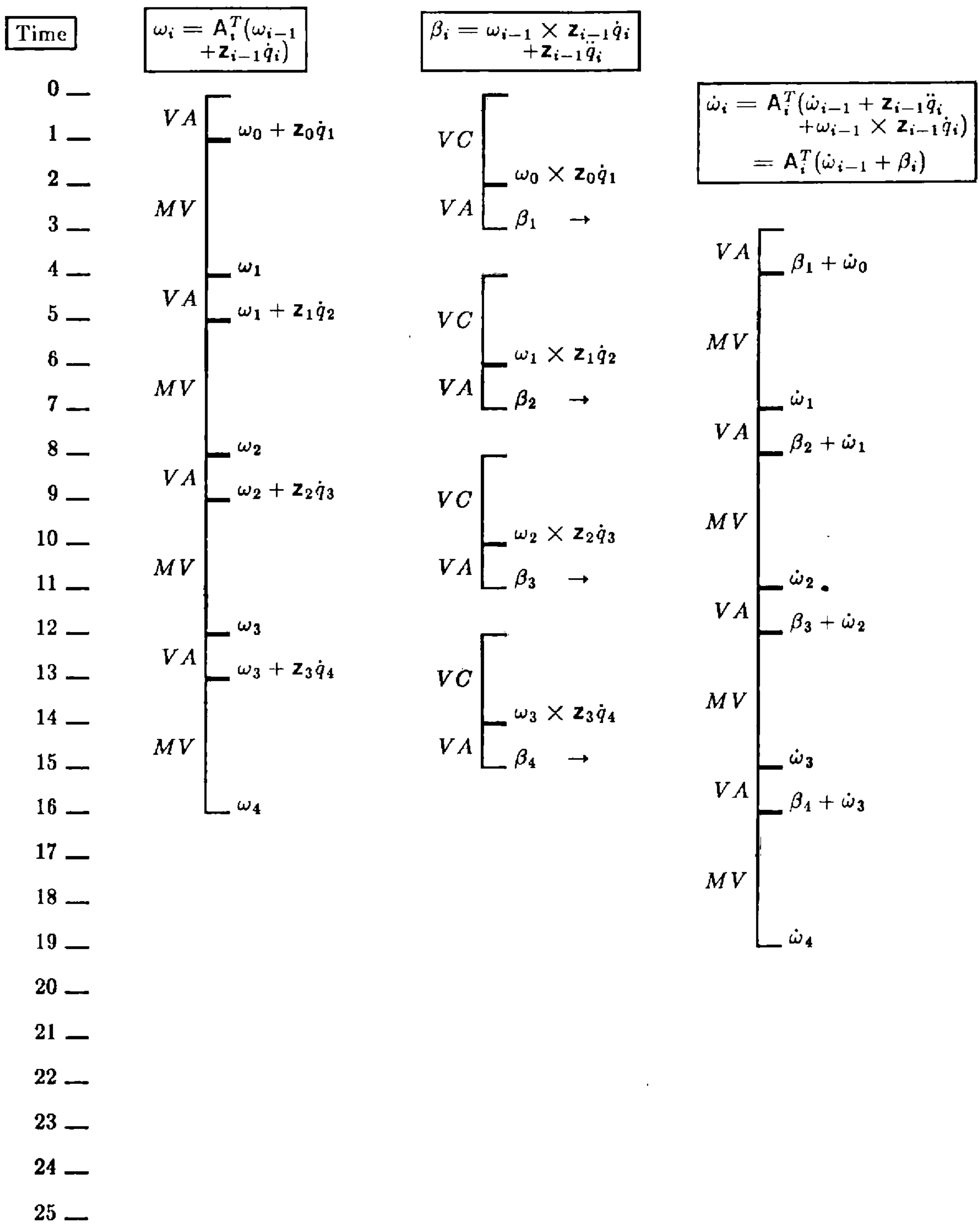
Variable	Waits on†	Time at Step End	Step	Cost‡
$\mathbf{f}_i$	$h = \text{Avail}(\mathbf{f}_{i+1})$	$h + \text{MV}$	$T_{19} = \mathbf{A}_{i+1}\mathbf{f}_{i+1}$	MV
		$h + \text{MV} + \text{VA}$	$\mathbf{f}_i = \mathbf{F}_i + T_{19}$	VA
$\mathbf{n}_i$	$h = \text{Avail}(\mathbf{f}_{i+1})$	Already computer**	$T_{20} = \mathbf{s}_i^* \times \mathbf{F}_i$	(VC**)
	$k = \text{Avail}(\mathbf{n}_{i+1})$	Already computed**	$T_{21} = T_{20} + \mathbf{N}_i$	(VA**)
		$h + \text{MV} + \text{VC}$	$T_{22} = \mathbf{p}_i^* \times T_{19}$	VC
		$h + \text{MV} + \text{VC} + \text{VA}$	$T_{23} = T_{21} + T_{22}$	VA
		$k + \text{MV}$	$T_{24} = \mathbf{A}_{i+1}\mathbf{n}_{i+1}$	MV
		$\max(k, h + \text{VC} + \text{VA}) + \text{MV} + \text{VA}$	$\mathbf{n}_i = T_{23} + T_{24}$	VA

† $\text{Avail}(X_{i-1}) = t$ means that variable X_{i-1} is made available to the ith node at time t on the backward recursion (substitute X_{i+1} on the forward recursion).

‡ VA = Time cost of vector addition
VC = Time cost of Vector cross product
SV = Time cost of scalar multiplication of a vector
MV = Time cost of matrix multiplication of a vector

** That is, computable before the recursion reaches the node, except at the initial node.

Fig. 4. Timing of linear Newton-Euler Backward Recursion. Timing of ω_i and $\dot{\omega}_i$; $n = 4$, rotational joints. $\omega_0 = \omega_{base}$, $\dot{\omega}_0 = \dot{\omega}_{base}$. For simplicity, 1 mult = 1 addn = 1 flop.



$\mathbf{z}_{i-1}\dot{q}_i$ and requires time (MV + VA) to compute after these are available. As shown in Fig. 4, successive values of ω_i will become available at intervals of (MV + VA) because each depends on nothing but its preceding recursive variable value and the input. Also, $\dot{\omega}_i$ depends on ω_{i-1} as well as on $\dot{\omega}_{i-1}$ and $\mathbf{z}_{i-1}\ddot{q}_i$. However, ω_{i-1} and $\mathbf{z}_{i-1}\ddot{q}_i$ are required in the computation well before $\dot{\omega}_{i-1}$. Given the availability of $\{\omega_i\}$ at intervals of (MV + VA), any ω_i-dependent intermediate partial results required by $\dot{\omega}_i$ can be calculated before $\dot{\omega}_{i-1}$ becomes available (hence the offset between the availability of ω_i and $\dot{\omega}_i$ in Fig. 4). These intermediate results already computed, when $\dot{\omega}_{i-1}$ does become available it will be possible to compute $\dot{\omega}_i$ in time (MV + VA). Meanwhile, computational precursors to $\dot{\omega}_{i+1}$ will have been similarly calculated, so that when $\dot{\omega}_i$ becomes available it is then possible to compute $\dot{\omega}_{i+1}$ in time (MV + VA). The availability of $\{\omega_i\}$ at intervals of (MV + VA) implies the availability of $\{\dot{\omega}_i\}$ at similar but offset intervals. Both of these imply the availability of $\{\ddot{\mathbf{p}}_i\}$ at intervals of (MV + VA), and so forth.

VA reduces to 1 addn by adding each coordinate concurrently. The product MV is the vector dot product of each row of the matrix with the vector. These products are done concurrently and take 1 mult + 2 addns (an example is worked below).

The availability offsets and the constant C may both be determined from Table 4.

$$\begin{aligned}\text{Avail}(\dot{\omega}_{i-1}) &\geqslant \text{Avail}(\omega_{i-1}) + \text{VC} + \text{VA},\\ \text{Avail}(\ddot{\mathbf{p}}_{i-1}) &\geqslant \text{Avail}(\omega_{i-1}) + 2\text{VC} + 3\text{VA},\\ \text{Avail}(\ddot{\mathbf{p}}_{i-1}) &\geqslant \text{Avail}(\dot{\omega}_{i-1}) + \text{VC} + 2\text{VA},\\ \text{Avail}(\mathbf{n}_{i+1}) &\geqslant \text{Avail}(\mathbf{f}_{i+1}) + \text{VC} + \text{VA}.\end{aligned} \quad (4)$$

These equations define the relative delays by which the propagation of the Newton-Euler recursion variables should be staggered. The constant C is found by showing when τ_1, the last generalized joint force of the forward recursion, becomes available as output. Assuming that all input values become available simultaneously at time $t = 0$, the time required to calculate the Newton-Euler dynamics exploiting maximal linear parallelism is

$$2n \cdot (1 \text{ mult} + 3 \text{ addns}) + (5 \text{ mults} + 9 \text{ addns}). \quad (5)$$

6. Parallelism Exploiting Logarithmic Recursion

The preceding section showed that parallelism potentially leads to considerable time savings even within a linear recursive framework. This section shows that by a suitable restructuring of the fundamental computational framework, together with a corresponding revision to the recursive forms of the equations, an $\mathcal{O}(\log(n))$ total time may be achieved. For convenient reference, the logarithmic recursive formulas are collected in Table 5.

The basic intuition may be illustrated by considering n consecutive additions. If these are performed serially, then time $\mathcal{O}(n)$ is required: A sum is initialized to zero, and a loop is entered that on the ith iteration adds the ith number to the sum. In parallel, time $\mathcal{O}(\log(n))$ may be achieved. In the first step, every odd-indexed number is added to the next-higher even-indexed number, forming $n/2$ partial sums. At each ith step, every odd-indexed partial sum is added to the next-higher even-indexed partial sum, forming $n/(2^i)$ partial sums (Reverse "even" and "odd" if the indexing is zero-based rather than one-based.)

To exploit this potential in the inverse dynamics case, however, it is necessary to generalize the linear recursive equations to produce something analogous to partial sums. The linear form may be regarded as an operator $\oplus$ that maps a variable X_{i-1} representing the base through $i - 1$ inputs, together with the ith input I_i, into X_i representing the base through the ith input:

$$X_{i-1} \oplus I_i \rightarrow X_i. \quad (6)$$

For logarithmic recursion, an operator $\otimes$ is required that maps a variable $X_{a,k}$ representing the ath through kth inputs, together with $X_{(k+1),b}$ from the $(k + 1)$st through the bth input, into $X_{a,b}$ representing inputs a through b:

$$X_{a,k} \otimes X_{(k+1),b} \rightarrow X_{a,b} \quad (7)$$

The linear recursive equation is a special case of the more general logarithmic form in which $a = 0$, $k = i - 1$, $b = k + 1 = i$, $X_{i,i} = I_i$, and computation proceeds serially, so that

Table 5. Logarithmic Recursive Formulations

Logarithmic backward recursion:

$$\sigma_a = \begin{cases} 1, & \text{joint } a \text{ rotational,} \\ 0, & \text{joint } a \text{ translational} \end{cases}$$

$$\bar{\sigma}_a = 1 - \sigma_a$$

$$\mathbf{W}_{a,b} = \begin{cases} \mathbf{A}_a, & \text{if } a = b, \\ \mathbf{W}_{a,k}\mathbf{W}_{(k+1),b}, & \text{if } a \neq b \end{cases}$$

$$\omega_{a,b} = \begin{cases} \sigma_a \mathbf{A}_a^T \mathbf{z}_{(a-1)}\dot{q}_a, & \text{if } a = b, \\ \mathbf{W}_{(k+1),b}^T \omega_{a,k} + \omega_{(k+1),b}, & \text{if } a \neq b \end{cases}$$

$$\dot{\omega}_{a,b} = \begin{cases} \sigma_a \mathbf{A}_a^T \mathbf{z}_{(a-1)}\ddot{q}_a, & \text{if } a = b, \\ \mathbf{W}_{(k+1),b}^T \dot{\omega}_{a,k} + (\mathbf{W}_{(k+1),b}^T \omega_{a,k}) \times \omega_{(k+1),b} + \dot{\omega}_{(k+1),b}, & \text{if } a \neq b \end{cases}$$

$$\mathbf{Q}_{a,b} = \begin{cases} \bar{\sigma}_a \mathbf{A}_a^T \mathbf{z}_{a-1}\dot{q}_a, & \text{if } a = b, \\ \mathbf{W}_{(k+1),b}^T \mathbf{Q}_{a,k} + \mathbf{Q}_{(k+1),b}, & \text{if } a \neq b \end{cases}$$

$$\mathbf{R}_{a,b} = \begin{cases} \mathbf{p}_a^*, & \text{if } a = b, \\ \mathbf{W}_{(k+1),b}^T \mathbf{R}_{a,k} + \mathbf{R}_{(k+1),b}, & \text{if } a \neq b \end{cases}$$

$$\mathbf{S}_{a,b} = \begin{cases} \omega_{a,a} \times \mathbf{p}_a^*, & \text{if } a = b, \\ \mathbf{W}_{(k+1),b}^T \mathbf{S}_{a,k} + (\mathbf{W}_{(k+1),b}^T \omega_{a,k}) \times \mathbf{R}_{(k+1),b} + \mathbf{S}_{(k+1),b}, & \text{if } a \neq b \end{cases}$$

$$\ddot{\mathbf{p}}_{a,b} = \begin{cases} \ddot{\mathbf{p}}_a^g + \dot{\omega}_{a,a} \times \mathbf{p}_a^* + \omega_{a,a} \times (\omega_{a,a} \times \mathbf{p}_a^*) + \bar{\sigma}_a \mathbf{A}_a^T \mathbf{z}_{a-1}\ddot{q}_a + 2\omega_{a,a} \times \mathbf{Q}_{a,a}, & \text{if } a = b, \\ \mathbf{W}_{(k+1),b}^T \ddot{\mathbf{p}}_{a,k} + \ddot{\mathbf{p}}_{(k+1),b} + (\mathbf{W}_{(k+1),b}^T \dot{\omega}_{a,k}) \times \mathbf{R}_{(k+1),b} & \\ \quad + (\mathbf{W}_{(k+1),b}^T \omega_{a,k}) \times \Big((\mathbf{W}_{(k+1),b}^T \omega_{a,k}) \times \mathbf{R}_{(k+1),b} + 2(\mathbf{S}_{(k+1),b} + \mathbf{Q}_{(k+1),b}) \Big), & \text{if } a \neq b \end{cases}$$

Logarithmic forward recursion:

$$\mathbf{f}_{a,b} = \begin{cases} \mathbf{F}_a, & \text{if } a = b, \\ \mathbf{W}_{a+1,k+1}\mathbf{f}_{(k+1),b} + \mathbf{f}_{a,k}, & \text{if } a \neq b \end{cases}$$

$$\mathbf{n}_{a,b} = \begin{cases} \mathbf{N}_a + \mathbf{s}_a^* \times \mathbf{F}_a, & \text{if } a = b, \\ \mathbf{n}_{a,k} + \mathbf{W}_{a+1,k+1}(\mathbf{n}_{(k+1),b} + (\mathbf{A}_{k+1}^T \mathbf{R}_{a,k}) \times \mathbf{f}_{(k+1),b}), & \text{if } a \neq b \end{cases}$$

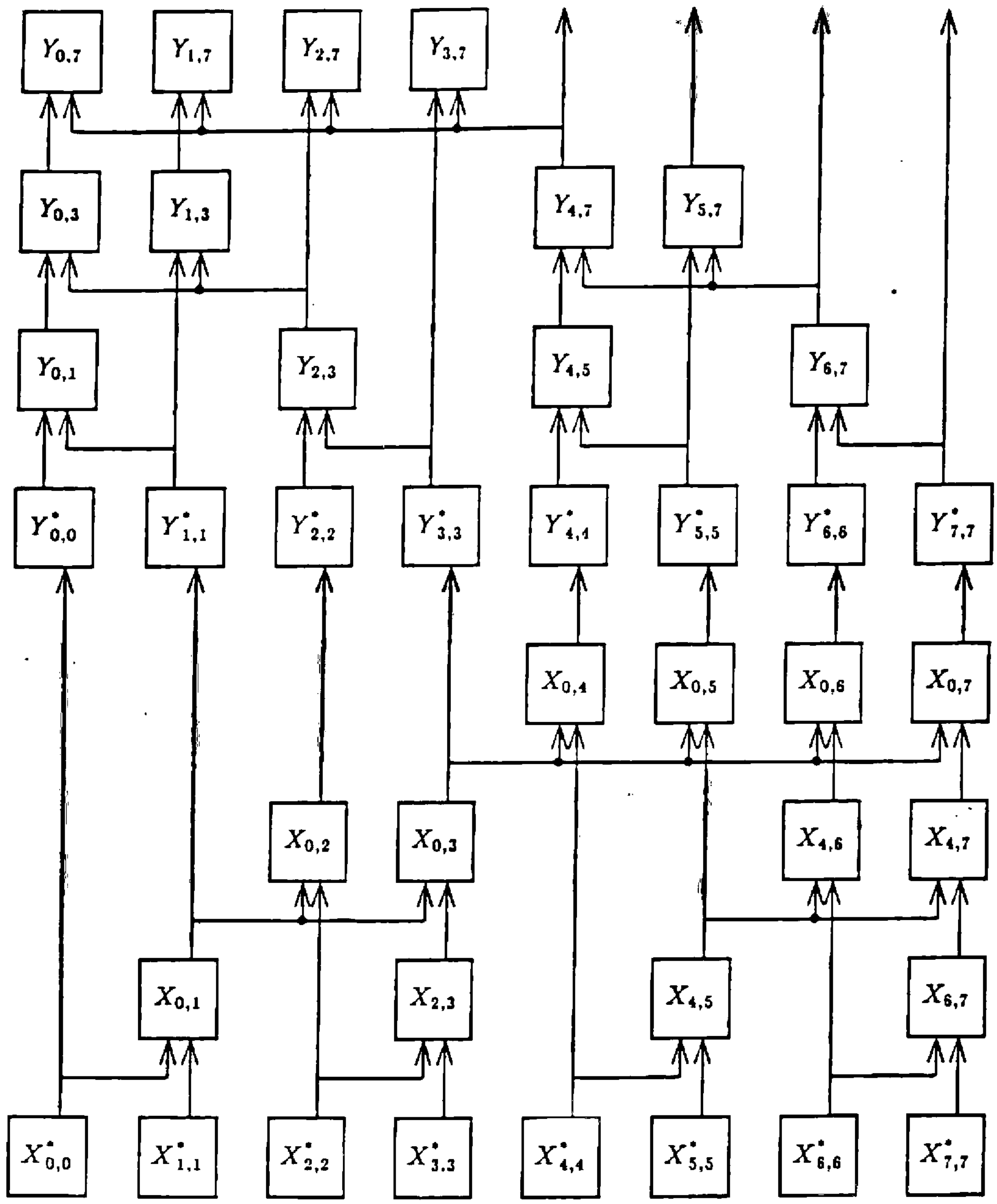
$$\begin{aligned} X_i &= X_{0,i} \\ &= X_{0,(i-1)} \otimes X_{i,i} \\ &= X_{i-1} \oplus I_i. \end{aligned} \tag{8}$$

This is the meaning of the double subscripts in Fig. 1.

What physical meaning can be assigned to this? The recursive forms (both linear and logarithmic) may be thought of as mechanisms for relating physical parameters at one coordinate system (or link or joint) to physical parameters at another, by abstracting away the intervening links of the physical manipulator. The goal in both cases is to relate the acceleration of each link to the acceleration of the base by abstracting away the intervening joint accelerations and then to relate the distal forces and torques acting on each link to the distal forces and torques acting on the tip by abstracting away the intervening joint forces and torques. This done, the joint forces and torques necessary to sustain the desired acceleration may be found from purely local applications of Newtonian mechanics.

The linear and the logarithmic recursive forms differ principally in how they approach the problem of relating physical parameters between coordinate systems. The linear form relates the base (backward recursion) parameters to the first link to obtain parameters of the form X_1 ($\equiv X_{0,1}$). These in turn are related to the second link to obtain parameters of the form X_2 ($\equiv X_{0,2}$), which relate the second link to the base. In sequence,

*Fig. 5. Logarithmic recursive graph structure. 7-link manipulator with moving, spinning base. $X_{a,b}$ represents backward, and $Y_{a,b}$ backward, recursion variables; * shows a = b case.*



the parameters $X_{0,3}, X_{0,4}, \ldots, X_{0,n}$ are formed, which relate respectively the third, the fourth, and the nth links to the base. The process may be viewed as one that, at each step, "glues" the next link parameter onto the current parameters, thereby relating the base to the next successive link.

In contrast, the logarithmic recursive form may be viewed as a mechanism for "gluing" together any two adjacent parameters. Parameters of the form $X_{i,i}$ reflect only the input values at link (joint) i; those of the form $X_{i,j}$ reflect (abstract away) links i through j. Thus on the first step we relate alternating pairs of adjacent links to form the parameters $X_{0,1}, X_{2,3}, X_{4,5}, \ldots, X_{n-1,n}$ (if n is odd). On the second step, $X_{0,2}, X_{0,3}, X_{4,6}, X_{4,7}, \ldots, X_{n-3,n-1}, X_{n-3,n}$ are additionally formed; on the third step we also pick up $X_{0,4}, X_{0,5}, X_{0,6}, X_{0,7}, X_{8,12}, \ldots, X_{n-7,n-1}, X_{n-7,n}$, and so forth. This process is illustrated in schematic form in Fig. 5. There, $n = 7$, so the backward recursion would take three steps as shown. In $[\log_2 (n + 1)]$ steps, all (backward recursion) parameters of the form $X_{0,i}$, $0 \leq i \leq n$, may be formed. As noted above, these are just the linear recursive variables X_i. The same is true of the forward recursion, hence of the inverse dynamics computation.

As before, it is necessary to consider only those variables that are propagated the length of the recursive path. Suppose we wished to find the value of nonpropagated values, for example to find $\ddot{\mathbf{r}}_m$ for node m on the forward recursion path. We first compute $\omega_{0,m}$, $\dot{\omega}_{0,m}$, and $\ddot{\mathbf{p}}_{0,m}$ in time $\mathcal{O}(\log (m))$ using the logarithmic forms given in Table 5. These are exactly ω_m, $\dot{\omega}_m$, and $\ddot{\mathbf{p}}_m$ of Table 2; from these, $\ddot{\mathbf{r}}_m$ may be computed in constant time $\mathcal{O}(c)$.

To cover both the rotational and translational cases, it is convenient to introduce

$$\sigma_i = \begin{cases} 1, & \text{joint } i \text{ rotational}, \\ 0, & \text{joint } i \text{ translational}, \end{cases} \qquad (9)$$
$$\bar{\sigma}_i = 1 - \sigma_i.$$

6.1. Newton-Euler Backward Recursion Variables

We present here only enough of the derivations to illustrate the approach; the rest are given in Appendix A.

First introduce the auxiliary variable $\mathbf{W}_{a,b}$, which represents products of the coordinate transform matrices $\mathbf{A}_i$. Notation sometimes used elsewhere in the literature is ${}^{a-1}\mathbf{W}_b$, but we write $\mathbf{W}_{a,b}$ here for consistency with other variables. Let

$$\begin{aligned} \mathbf{W}_{a,b} &\equiv \prod_{j=a}^{b} \mathbf{A}_j \\ &= \left(\prod_{j=a}^{k} A_j\right)\left(\prod_{j=(k+1)}^{b} \mathbf{A}_j\right) \\ &= \mathbf{W}_{a,k}\mathbf{W}_{(k+1),b}. \end{aligned} \qquad (10)$$

$\mathbf{W}_{a,b}$ maps the coordinates of a vector expressed in the system $\mathcal{O}_b$ into coordinates expressed in $\mathcal{O}_{a-1}$. The physical significance of the combining form is to compose a mapping that relates $\mathcal{O}_b$ and $\mathcal{O}_k$ with a mapping that relates $\mathcal{O}_k$ and $\mathcal{O}_{a-1}$ to produce a mapping that relates the coordinate systems $\mathcal{O}_b$ and $\mathcal{O}_{a-1}$. Thus $\mathbf{W}_{a,a} = \mathbf{A}_a$, and $\mathbf{W}_{a,a-1} = \mathbf{I}$ (the identity matrix).

Next consider the angular velocity ω_i. We define $\mathbf{z}_{(-1)}$ to be the axis of rotation of the base system and $\dot{q}_0$ to be its magnitude, so that $\omega_0 = \mathbf{A}_0^T\mathbf{z}_{(-1)}\dot{q}_0$. This will be nonzero if, for example, it is desired to include the earth's rotation in the calculations (perhaps for satellite applications). If one creates a fictitious frame $\mathcal{O}_{(-1)}$ at the earth's center, then $\mathbf{z}_{(-1)}$ points through the North Pole and $\dot{q}_0$ is equal to the earth's angular velocity. Alternatively, nonzero quantities would arise if a spinning base were assumed.

We show that ω_i satisfies the closed-form nonrecursive formula

$$\omega_i = \sum_{j=0}^{i} \mathbf{W}_{j,i}^T \sigma_j \mathbf{z}_{j-1} \dot{q}_j \qquad (11)$$

because this is a fixed point of the recursive formula for ω_i given in Table 2:

$$\begin{aligned} \omega_i &= \mathbf{A}_i^T \left(\sum_{j=0}^{i-1} \mathbf{W}_{j,i-1}^T \sigma_j \mathbf{z}_{j-1} \dot{q}_j + \sigma_i \mathbf{z}_{i-1} \dot{q}_i\right) \\ &= \mathbf{A}_i^T(\omega_{i-1} + \sigma_i \mathbf{z}_{i-1} \dot{q}_i). \end{aligned} \qquad (12)$$

To produce an identical form at $a = 0$ and $b = i$, we define

$$
\begin{aligned}
\omega_{a,b} &\equiv \sum_{j=a}^{b} \mathbf{W}_{j,b}^{T} \sigma_j \mathbf{z}_{j-1} \dot{q}_j \\
&= \sum_{j=a}^{k} (\mathbf{W}_{j,k} \mathbf{W}_{(k+1),b})^{T} \sigma_j \mathbf{z}_{j-1} \dot{q}_j \\
&\quad + \sum_{j=k+1}^{b} \mathbf{W}_{j,b} \sigma_j \mathbf{z}_{j-1} \dot{q}_j \\
&= \mathbf{W}_{(k+1),b}^{T} \omega_{a,k} + \omega_{(k+1),b}.
\end{aligned} \tag{13}
$$

This is the combining form for $\omega_{a,b}$.

Now $\omega_{a,b}$ expresses the angular velocity of $\mathcal{O}_b$ relative to $\mathcal{O}_{a-1}$, referred to $\mathcal{O}_b$. Thus $\omega_{1,i}$ is the angular velocity of $\mathcal{O}_i$ relative to the base frame, and $\omega_{0,i}$ also accounts for the rotation of the base frame (if $\dot{q}_0 = 0$, then these are the same). The combining form for $\omega_{a,b}$ is a means of composing the angular velocity $\omega_{k+1,b}$ of $\mathcal{O}_b$ relative to $\mathcal{O}_k$ with $\omega_{a,k}$, that of $\mathcal{O}_k$ relative to $\mathcal{O}_{a-1}$, to obtain the angular velocity of $\mathcal{O}_b$ relative to $\mathcal{O}_{a-1}$. In particular, $\omega_{a,a-1} = 0$. The rotation matrix $\mathbf{W}_{k+1,b}^{T}$ in the formula transforms $\omega_{a,k}$ from $\mathcal{O}_k$ into $\mathcal{O}_b$, the system to which $\omega_{k+1,b}$ is referred.

Similar remarks apply to the physical significance of $\dot{\omega}_{a,b}$, $\ddot{\mathbf{p}}_{a,b}$, and so on. Derivations of the other propagated recursion formulas, shown in Table 5, are given in Appendix A. Note that ω_0 is the angular velocity of the base, possibly nonzero. Also, $\ddot{\mathbf{p}}_0$ is the acceleration of the base. Typically this is the acceleration due to gravity at the site. If one took $\omega_0 \neq 0$, then $\ddot{\mathbf{p}}_0$ may also include a term for $\omega_0 \times (\omega_0 \times \mathbf{p}_0^*)$, where $\mathbf{p}_0^*$ might be a vector from the earth's center $\mathcal{O}_{(-1)}$ to the site; this accounts for the centripetal acceleration arising from the rotation of the earth. One may account for gravitational acceleration by taking $\ddot{\mathbf{p}}_0^g = \mathbf{g}$, or $\ddot{\mathbf{p}}_i^g = 0$ for $i \neq 0$. The term involving $\ddot{\mathbf{p}}_j^g$ is a technical artifice to account for $\ddot{\mathbf{p}}_0$ cleanly, and it vanishes in the combining form.

We slightly abuse the dot notation for time differentiation (such as $\dot{\mathbf{u}}_{a,b}$) to indicate the time differentiation of $\mathbf{u}_{a,b}$ from within the coordinate system $\mathcal{O}_{a-1}$. This is equivalent to the standard notation at $a = 0$ (the case of interest) and results in a less bulky notation. To illustrate this usage, we consider an alternative derivation of $\dot{\omega}_{a,b}$, where d^c/dt denotes time differentiation in $\mathcal{O}_c$, and ${}^c\mathbf{u}_{a,b}$ denotes $\mathbf{u}_{a,b}$ referred to $\mathcal{O}_c$:

$$
\begin{aligned}
\dot{\omega}_{a,b} &= \frac{d^{(a-1)}}{dt} {}^b\omega_{a,b} \\
&= \frac{d^{(a-1)}}{dt} ({}^b\omega_{a,k} + {}^b\omega_{k+1,b}) \\
&= {}^b\dot{\omega}_{a,k} + \frac{d^{(a-1)}}{dt} {}^b\omega_{k+1,b} \\
&= \mathbf{W}_{k+1,b}^{T} \dot{\omega}_{a,k} + {}^b\dot{\omega}_{a,k} \times {}^b\omega_{k+1,b} + \frac{d^k}{dt} {}^b\omega_{k+1,b} \\
&= \mathbf{W}_{k+1,b}^{T} \dot{\omega}_{a,k} + (\mathbf{W}_{k+1,b}^{T} \omega_{a,k}) \\
&\quad \times \omega_{k+1,b} + \dot{\omega}_{k+1,b},
\end{aligned} \tag{14}
$$

which is our combining form.

6.2. Newton-Euler Forward Recursion Variables

On the Newton-Euler forward recursion, the coordinate matrix products of interest will be $\mathbf{W}_{a+1,k+1}$ instead of $\mathbf{W}_{k+1,b}^{T}$. This is because we wish to transform from $\mathcal{O}_{k+1}$ to $\mathcal{O}_a$ instead of from $\mathcal{O}_k$ to $\mathcal{O}_b$, as we are now working from the tip of the manipulator back toward the base. Hence we want to transform some parameter $X_{k+1,b}$ that abstracts the subchain from coordinate system (or link or joint) $k + 1$ to b and is expressed in $\mathcal{O}_{k+1}$ into the equivalent quantity expressed in $\mathcal{O}_a$. The $X_{k+1,b}$ so transformed can be combined with $X_{a,k}$, representing the subchain between a and k expressed in $\mathcal{O}_a$, to yield $X_{a,b}$ expressed in $\mathcal{O}_a$. We will here assume that the necessary products $\mathbf{W}_{a+1,k+1}$ are generated on the forward recursion in accordance with the formulas above.

6.3. Optimized Logarithmic Recursion

As before, as a conceptual aid we assume that there is one group of parallel processors for each node shown in Fig. 5. Again, an implementation could be constructed using only one tier (only one physical device for each pair of joints) by connecting the outputs back to the inputs through a buffer. The comments about the 4-flop systolic pipeline interval also apply here.

Since it is possible to satisfy the minimum delay conditions, propagation occurs at a rate of (MV + VA) per node. Assuming that all input becomes available simultaneously, we have

$$\text{Avail}(X_{a,b}) = \text{Avail}(X_{a,a}) + [\log_2(b - a + 1)](\text{MV} + \text{VA}). \quad (15)$$

Thus, in particular, if X_i is the linear recursive variable corresponding to $X_{a,b}$, then $X_i = X_{0,i}$, so

$$\begin{aligned}\text{Avail}(X_i) &= \text{Avail}(X_{0,i}) \\ &= \text{Avail}(X_{a,a}) \\ &\quad + [\log_2(i + 1)](\text{MV} + \text{VA}).\end{aligned} \quad (16)$$

By analyzing the time dependencies as we did before and assuming a maximally parallel implementation as before, we have for the total time of the calculations

$$2[\log_2(n + 1)](1 \text{ mult} + 3 \text{ addns}) + 5 \text{ mults} + 10 \text{ addns}. \quad (17)$$

7. Implementation Considerations

The principal thrust of this paper is the preceding analysis, and here we only assert that a physical implementation is feasible and sketch the form that it might take. We imagine that any actual implementation will differ in many particular details from those presented here. This section summarizes results developed in more detail in Lathrop (1983).

7.1. Global Requirements

Except perhaps for some variation in the constant term, the hardware sketched in this section is intended to capture the maximally parallel nature of the algorithms and to physically attain the stated time bounds.

We worry about three main concerns that commonly dog parallel systems:

1. internal buffering and storage of intermediate results,
2. communication and bussing of intermediate results, and
3. number of processors.

We consider requirements for internal buffering and storage. This will lead us to the communication and bussing necessary to get the data to the appropriate storage locations. Finally we consider the processing required.

The processor nodes could be hooked up in a regular, regularly extensible structure very similar to that in Figs. 1 and 5. This would permit global systolic pipelining of successive sets of joint torques at intervals of 4 flops. However, we could map our full conceptual structure into a much smaller nonsystolic one that buffers intermediate results so as to use only one physical processor node (linear) or tier of nodes (logarithmic).

7.1.1. Buffering

From input/output considerations we can bound the amount of buffering external to a node that may be required (this is input, internode, and backward-forward recursion buffering). The potentially massive internal buffering required is actually very tightly bounded, since most intermediate results are used immediately and need not be retained.

7.1.2. Communication

No matrix-vector operator has more than two operands. By counting operations, one may see that no single group of the linear forward or backward recursion, the logarithmic forward or backward recursion, or the nonpropagated variables has more than 50 point-to-point wires. The external communication must carry externally buffered data back and forth between the process nodes and the buffers. The communication required to support those requirements is not excessive.

7.1.3. Number of Processors

An implementation might involve a special-purpose VLSI chip capable of handling general vector arithmetic up to and including matrix-vector multiplication. "Single device" (single-node, linear case or single-tier, logarithmic case) is a nonsystolic configuration. Assuming a stationary base and $n = 6$, "systolic pipeline" in the logarithmic case requires only a total of seven nodes on each of the forward and backward recursions, plus six more each if the $a = b$ case is handled as $a \neq b$ (less hardware than indicated here would be required

Table 6. Matrix-Vector Processors Required by Algorithms

	Matrix-Vector Processor ($n = 6$)			
Algorithm	Per Node	Tier, $n = 6$	Single Device	Systolic Pipeline
Linear Bkwd.	22	*	22	132
Linear Fwd.	8	*	8	48
Logarithmic Bkwd.	36	108	108	468
Logarithmic Fwd.	13	39	39	169

if it were instead handled as a special case). The algorithms would require the number of matrix-vector processors (MVPs) shown in Table 6.

7.2. Robot Chip

We require a chip capable of performing general matrix-vector arithmetic, and we sketch just one architecture suitable for our purposes, though of course there are many.

7.2.1. Modular Processor

A primitive module capable of computing one coordinate of the result might be implemented as a datapath (Barrett et al. 1983), an easily customizable microprocessor-like architecture composed of stacked computational elements, busses running through them, and a centralized control. For datapath elements we assume registers, adders/subtractors, and multipliers. Each element can communicate with the busses and with the adjacent elements above and below. A floor plan of a primitive module that computes one coordinate of the result is shown as a block diagram in Fig. 6. By grouping three such primitive modules together, along with suitable control, we can implement the necessary matrix-vector operations.

A vector dot product, for example, could be computed in three floating-point steps. In the first step OP-A(3) is written to BUS-1 and OP-B(3) is written to BUS-2 as the operands of MULT-1, and MULT-2 takes its operands from the adjacent OP-A(2) and OP-B(2). In the second step the results from the multipliers are written to the busses as the ADDER operands, and MULT-1 takes its operands from OP-A(1) and OP-B(1). In the third step the ADDER operands are the results from the previous step of the ADDER and of MULT-1. In turn, a matrix-vector multiplication may be implemented as three dot products, each row of the matrix with the vector. By appropriate sequencing, the other vector operations required may also be effected.

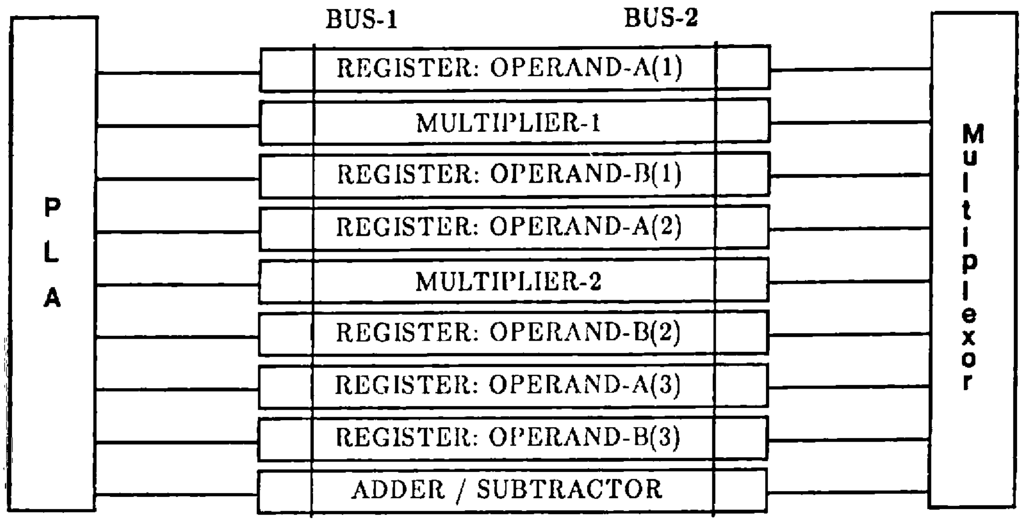
The control must specify when the operands are read and when the result is computed and written. The MVP input operands are wired (perhaps through intermediate delay buffers) to the output result of the MVP computing those operands. Synchronizing the respective dump and load between the source and destination MVPs allows intermediate results to be passed to successive operators. By ensuring that cycle start occurs synchronously on all chips, communication timing between operand sources and destinations can be coordinated using counters and comparators.

8. Conclusions

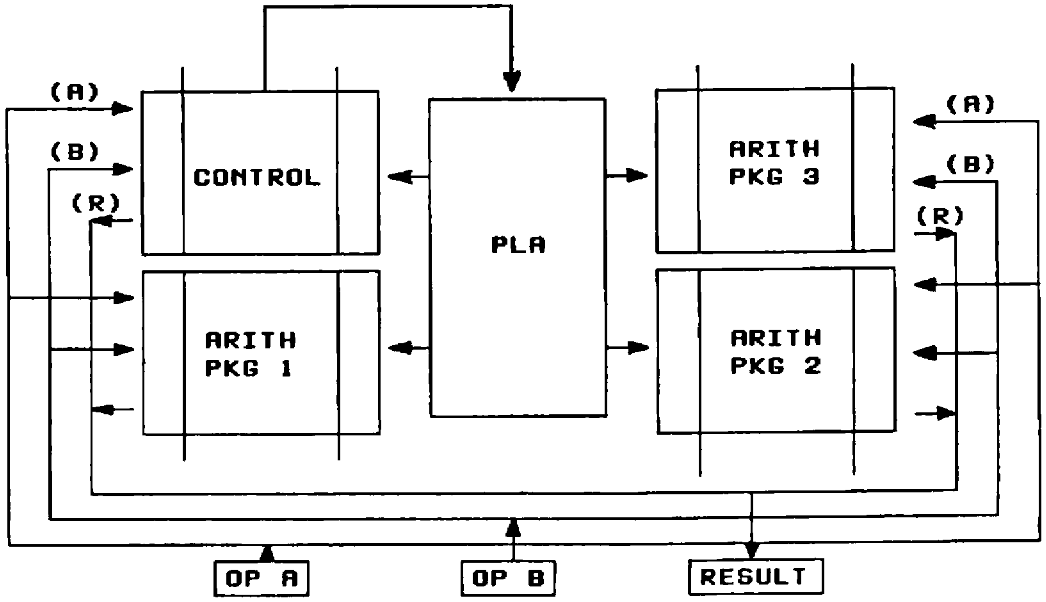
We have shown that considerable time saving accrues from embedding the inverse dynamics calculation in a parallel computation. A parallel-time algorithm with time complexity only logarithmic in the number of joints has been derived. Hardware necessary to implement such parallel algorithms is substantial but not excessive. Using the concepts developed, it should be possible to construct a device capable of implementing the calculation at a speed primarily bounded by the input/output requirements that the algorithm imposes on the host. Much of the parallelism arises from the inherent nature of the matrix-vector operators, from the straight-line nature of the algorithm (no conditional branching), from the unidirectional data depen-

Fig. 6. A. Primitive module block diagram. To enable vector cross product (VC), the two multipliers are placed between the two register pairs not corresponding to the result component calculated by the primitive module. Thus this example calculates the third scalar component of the result. B. Matrix vector processor (MVP).

A



B



dencies, and from operator associativity in the logarithmic combining forms. Many robotics algorithms share these properties, and there is every indication that the techniques presented could yield similarly substantial time saving in such cases.

Acknowledgments

This paper summarizes an S.M. thesis supervised by Prof. Marvin Minsky at the M.I.T. Artificial Intelligence Laboratory, a revised version of which was issued as M.I.T. Technical Report No. 754. Thanks to Bill Barrett, Mike Brady, Bob Giansiracusa, and Bob Rogers for valuable discussion, and particular thanks to John Hollerbach for several clarifying suggestions. Personal support for the author was provided by a graduate fellowship from the National Science Foundation.

Appendix A: Derivation of the Logarithmic Recursive Formulas

In the derivations below, for each linear recursive variable, our general strategy will be as follows:

1. Propose a closed-form, nonrecursive formula equivalent to the linear recursive formula of Table 2.
2. Show that step 1 is correct by showing that it is a fixed point of the linear recursive formula considered and that it holds for the zero term (this is equivalent to an inductive proof).
3. Propose a logarithmic recursive formula equivalent to that in step 1 at $a = 0$ and $b = i$.
4. Show that the resulting combining operator is correct by showing that it preserves the form of the formula in step 3.

The reader may verify that for $a = 0$, $k = i - 1$, $b = k + 1 = i$ on the backward recursion, or $a = k = i$, $b = n$ on the forward recursion, the formulas reduce to Table 2. Alternatively, a derivation may be produced by differentiation as indicated in the text.

It will be necessary to introduce several auxiliary variables not directly corresponding to any variable in the linear recursive formalism. There we will be concerned to show only that the asserted combining operator is correct.

In the following we assume $a \leq k < b$ throughout, the case of $a = b$ being found as a special case of step 3 above.

A1. Newton-Euler Backward Recursion Variables

The derivation of the logarithmic combining form for ω_i has been developed in the text. We show that $\dot{\omega}_i$ satisfies the closed-form formula

$$\dot{\omega}_i = \sum_{j=0}^{i} \mathbf{W}_{j,i}^T \sigma_j (\mathbf{z}_{j-1} \ddot{q}_j + \omega_{j-1} \times \mathbf{z}_{j-1} \dot{q}_j), \tag{A1}$$

as it is a fixed point of the recursive formula for $\dot{\omega}_i$ in Table 2:

$$\begin{aligned} \dot{\omega}_i &= \mathbf{A}_i^T \left(\sum_{j=0}^{i-1} \mathbf{W}_{j,i-1}^T \sigma_j (\mathbf{z}_{j-1} \ddot{q}_j + \omega_{j-1} \times \mathbf{z}_{j-1} \dot{q}_j) \right. \\ &\qquad \left. + \sigma_i (\mathbf{z}_{i-1} \ddot{q}_i + \omega_{i-1} \times \mathbf{z}_{i-1} \dot{q}_i) \right) \\ &= \mathbf{A}_i^T \left(\dot{\omega}_{i-1} + \sigma_i (\mathbf{z}_{i-1} \ddot{q}_i + \omega_{i-1} \times \mathbf{z}_{i-1} \dot{q}_i) \right). \end{aligned} \tag{A2}$$

As in the case of ω_i, we take $\dot{\omega}_0 = \mathbf{A}_0^T \mathbf{z}_{(-1)} \ddot{q}_0$. Most applications of interest will have $\dot{\omega}_0 = \ddot{q}_0 = 0$.

To match at $a = 0$ and $b = i$,

$$\begin{aligned} \dot{\omega}_{a,b} &\equiv \sum_{j=a}^{b} \mathbf{W}_{j,b}^T \sigma_j (\mathbf{z}_{j-1} \ddot{q}_j + \omega_{a,(j-1)} \times \mathbf{z}_{j-1} \dot{q}_j) \\ &= \sum_{j=a}^{k} (\mathbf{W}_{j,k} \mathbf{W}_{(k+1),b})^T \sigma_j (\mathbf{z}_{j-1} \ddot{q}_j \\ &\quad + \omega_{a,(j-1)} \times \mathbf{z}_{j-1} \dot{q}_j) \\ &\quad + \sum_{j=k+1}^{b} \mathbf{W}_{j,b}^T \sigma_j (\mathbf{z}_{j-1} \ddot{q}_j \\ &\quad + (\mathbf{W}_{(k+1),(j-1)}^T \omega_{a,k} + \omega_{(k+1),(j-1)}) \\ &\quad \times \mathbf{z}_{j-1} \dot{q}_j) \\ &= \mathbf{W}_{(k+1),b}^T \dot{\omega}_{a,k} + \dot{\omega}_{(k+1),b} \\ &\quad + \sum_{j=k+1}^{b} \mathbf{W}_{j,b}^T \sigma_j ((\mathbf{W}_{(k+1),(j-1)}^T \omega_{a,k}) \times \mathbf{z}_{j-1} \dot{q}_j) \\ &= \mathbf{W}_{(k+1),b}^T \dot{\omega}_{a,k} + (\mathbf{W}_{(k+1),b}^T \omega_{a,k}) \\ &\quad \times \omega_{(k+1),b} + \dot{\omega}_{(k+1),b}, \end{aligned} \tag{A3}$$

which is the combining form for $\dot{\omega}_{a,b}$.

To derive the combining form for $\ddot{\mathbf{p}}_{a,b}$, it is necessary to create the auxiliary variables

$$\begin{aligned}
\mathbf{Q}_{a,b} &\equiv \sum_{j=a}^{b} \mathbf{W}_{j,b}^T \bar{\sigma}_j \mathbf{z}_{j-1} \dot{q}_j \\
&= \mathbf{W}_{k+1,b}^T \mathbf{Q}_{a,k} + \mathbf{Q}_{(k+1),b}, \\
\mathbf{R}_{a,b} &\equiv \sum_{j=a}^{b} \mathbf{W}_{j+1,b}^T \mathbf{p}_j^* \\
&= \mathbf{W}_{k+1,b}^T \mathbf{R}_{a,k} + \mathbf{R}_{(k+1),b}, \\
\mathbf{S}_{a,b} &\equiv \sum_{j=a}^{b} \mathbf{W}_{j+1,b}^T (\omega_{a,j} \times \mathbf{p}_j^*) \\
&= \mathbf{W}_{k+1,b}^T \mathbf{S}_{a,k} + \sum_{j=k+1}^{b} \mathbf{W}_{j+1,b}^T ((\mathbf{W}_{k+1,j}^T \omega_{a,k} \\
&\quad + \omega_{k+1,j}) \times \mathbf{p}_j^*) \\
&= \mathbf{W}_{k+1,b}^T \mathbf{S}_{a,k} + (\mathbf{W}_{k+1,b}^T \omega_{a,k}) \\
&\quad \times \mathbf{R}_{(k+1),b} + \mathbf{S}_{(k+1),b}.
\end{aligned} \tag{A4}$$

Next we show that $\ddot{\mathbf{p}}_i$ satisfies the closed-form expression

$$\begin{aligned}
\ddot{\mathbf{p}}_i &= \mathbf{W}_{1,i}^T \ddot{\mathbf{p}}_0 \\
&\quad + \sum_{j=1}^{i} \mathbf{W}_{j+1,i}^T (\dot{\omega}_j \times \mathbf{p}_j^* + \omega_j \times (\omega_j \times \mathbf{p}_j^*) \\
&\quad + \bar{\sigma}_j (\mathbf{A}_j^T \mathbf{z}_{j-1} \ddot{q}_j + 2\omega_j \times \mathbf{A}_j^T \mathbf{z}_{j-1} \dot{q}_j)).
\end{aligned} \tag{A5}$$

This is a fixed point of the recursive formula for $\ddot{\mathbf{p}}_i$ as shown:

$$\begin{aligned}
\ddot{\mathbf{p}}_i &= \mathbf{A}_i^T \Bigg(\mathbf{W}_{1,i-1}^T \ddot{\mathbf{p}}_0 \\
&\quad + \sum_{j=1}^{i-1} \mathbf{W}_{j+1,i-1}^T (\dot{\omega}_j \times \mathbf{p}_j^* + \omega_j \times (\omega_j \times \mathbf{p}_j^*) \\
&\quad + \bar{\sigma}_j (\mathbf{A}_j^T \mathbf{z}_{j-1} \ddot{q}_j + 2\omega_j \times \mathbf{A}_j^T \mathbf{z}_{j-1} \dot{q}_j)\Big) \\
&\quad + \mathbf{W}_{i+1,i}^T (\dot{\omega}_i \times \mathbf{p}_i^* + \omega_i \times (\omega_i \times \mathbf{p}_i^*) \\
&\quad + \bar{\sigma}_i (\mathbf{A}_i^T \mathbf{z}_{i-1} \ddot{q}_i + 2\omega_i \times \mathbf{A}_i^T \mathbf{z}_{i-1} \dot{q}_i)\Bigg) \\
&= \mathbf{A}_i^T \ddot{\mathbf{p}}_{i-1} + \dot{\omega}_i \times \mathbf{p}_i^* + \omega_i \times (\omega_i \times \mathbf{p}_i^*) \\
&\quad + \bar{\sigma}_i (\mathbf{A}_i^T \mathbf{z}_{i-1} \ddot{q}_i + 2\omega_i \times \mathbf{A}_i^T \mathbf{z}_{i-1} \dot{q}_i),
\end{aligned} \tag{A6}$$

where $\ddot{\mathbf{p}}_0$ is the acceleration of the base. Typically this is the acceleration due to gravity at the site. If one took $\omega_0 \neq 0$, then $\ddot{\mathbf{p}}_0$ may also include a term for $\omega_0 \times (\omega_0 \times \mathbf{p}_0^*)$, where $\mathbf{p}_0^*$ is a vector from the earth's center $\mathcal{O}_{(-1)}$ to the site; this accounts for the centripetal acceleration arising from the rotation of the earth (or any spinning base). If one accounts for the gravitational acceleration ($\mathbf{g}$) by taking $\ddot{\mathbf{p}}_0^g = \mathbf{g}$ at the base, or $\ddot{\mathbf{p}}_i^g = 0$ for $i \neq 0$, then the formula may be equivalently rewritten with greater clarity (covering both cases $\omega_0 = 0$ and $\omega_0 \neq 0$) as

$$\begin{aligned}
\ddot{\mathbf{p}}_i &= \sum_{j=0}^{i} \mathbf{W}_{j+1,i}^T (\ddot{\mathbf{p}}_j^g + \dot{\omega}_j \times \mathbf{p}_j^* + \omega_j \times (\omega_j \times \mathbf{p}_j^*) \\
&\quad + \bar{\sigma}_j (\mathbf{A}_j^T \mathbf{z}_{j-1} \ddot{q}_j + 2\omega_j \times \mathbf{A}_j^T \mathbf{z}_{j-1} \dot{q}_j)).
\end{aligned} \tag{A7}$$

This we will take to be the defining closed-form nonrecursive formula for $\ddot{\mathbf{p}}_i$. The term involving $\ddot{\mathbf{p}}_j^g$ is a technical artifice to account for $\ddot{\mathbf{p}}_0$ cleanly and will vanish in the combining form.

The demonstration of the combining form of $\ddot{\mathbf{p}}_{a,b}$ will require the vector identity $\mathbf{a} \times (\mathbf{b} \times \mathbf{c}) + (\mathbf{b} \times \mathbf{a}) \times \mathbf{c} = \mathbf{b} \times (\mathbf{a} \times \mathbf{c})$. To match at $a = 0$ and $b = i$, take

$$\begin{aligned}
\ddot{\mathbf{p}}_{a,b} &\equiv \sum_{j=a}^{b} \mathbf{W}_{j+1,b}^T \Big(\ddot{\mathbf{p}}_j^g + \dot{\omega}_{a,j} \times \mathbf{p}_j^* + \omega_{a,j} \\
&\quad \times (\omega_{a,j} \times \mathbf{p}_j^*) + \bar{\sigma}_j (\mathbf{A}_j^T \mathbf{z}_{j-1} \ddot{q}_j + 2\omega_{a,j} \\
&\quad \times \mathbf{A}_j^T \mathbf{z}_{j-1} \dot{q}_j) \Big) \\
&= \mathbf{W}_{k+1,b}^T \sum_{j=a}^{k} \mathbf{W}_{j+1,k}^T \Big(\ddot{\mathbf{p}}_j^g + \dot{\omega}_{a,j} \\
&\quad \times \mathbf{p}_j^* + \omega_{a,j} \times (\omega_{a,j} \times \mathbf{p}_j^*) \\
&\quad + \bar{\sigma}_j (\mathbf{A}_j^T \mathbf{z}_{j-1} \ddot{q}_j + 2\omega_{a,j} \times \mathbf{A}_j^T \mathbf{z}_{j-1} \dot{q}_j) \Big) \\
&\quad + \sum_{j=k+1}^{b} \mathbf{W}_{j+1,b}^T \Bigg(\ddot{\mathbf{p}}_j^g + \Big(\mathbf{W}_{k+1,j}^T \dot{\omega}_{a,k} \\
&\quad + (\mathbf{W}_{k+1,j}^T \omega_{a,k}) \times \omega_{(k+1),j} + \dot{\omega}_{(k+1),j} \Big) \times \mathbf{p}_j^* \\
&\quad + (\mathbf{W}_{k+1,j}^T \omega_{a,k} + \omega_{(k+1),j}) \\
&\quad \times \Big((\mathbf{W}_{k+1,j}^T \omega_{a,k} + \omega_{(k+1),j}) \times \mathbf{p}_j^* \Big) \\
&\quad + \bar{\sigma}_j (\mathbf{A}_j^T \mathbf{z}_{j-1} \ddot{q}_j + 2(\mathbf{W}_{k+1,j}^T \omega_{a,k} + \omega_{k+1,j}) \\
&\quad \times \mathbf{A}_j^T \mathbf{z}_{j-1} \dot{q}_j) \Bigg)
\end{aligned} \tag{A8}$$

$$
\begin{aligned}
&= \mathbf{W}_{k+1,b}^T \ddot{\mathbf{p}}_{a,k} + \ddot{\mathbf{p}}_{(k+1),b} + (\mathbf{W}_{k+1,b}^T \dot{\omega}_{a,k}) \\
&\quad \times \sum_{j=k+1}^{b} \mathbf{W}_{j+1,b}^T \mathbf{p}_j^* + (\mathbf{W}_{k+1,b}^T \omega_{a,k}) \\
&\quad \times \Bigg((\mathbf{W}_{k+1,b}^T \omega_{a,k}) \times \sum_{j=k+1}^{b} \mathbf{W}_{j+1,b}^T \mathbf{p}_j^* \\
&\quad + 2 \sum_{j=k+1}^{b} (\mathbf{W}_{j+1,b}^T (\omega_{(k+1),j} \times \mathbf{p}_j^*) \\
&\quad + \bar{\sigma}_j \mathbf{W}_{j+1,b}^T (\mathbf{A}_j^T \mathbf{z}_{j-1} \dot{q}_j) \Bigg) \Bigg),
\end{aligned}
$$

$$
\begin{aligned}
\ddot{\mathbf{p}}_{a,b} &= (\mathbf{W}_{k+1,b}^T \ddot{\mathbf{p}}_{a,k}) + \ddot{\mathbf{p}}_{(k+1),b} + (\mathbf{W}_{k+1,b}^T \dot{\omega}_{a,k}) \\
&\quad \times \mathbf{R}_{(k+1),b} + (\mathbf{W}_{k+1,b}^T \omega_{a,k}) \times \Big((\mathbf{W}_{k+1,b}^T \omega_{a,k}) \\
&\quad \times \mathbf{R}_{(k+1),b} + 2(\mathbf{S}_{(k+1),b} + \mathbf{Q}_{k+1,b}) \Big).
\end{aligned}
$$

A2. Newton-Euler Forward Recursion Variables

As noted in the text, on the Newton-Euler forward recursion the coordinate matrix products of interest will be $\mathbf{W}_{a+1,k+1}$ instead of $\mathbf{W}_{k+1,b}^T$.

Noting that the numbering runs backward, we see that f_i satisfies the nonrecursive formula

$$
\begin{aligned}
\mathbf{f}_i &= \sum_{j=i}^{n} \mathbf{W}_{i+1,j} \mathbf{F}_j \\
&= \mathbf{A}_{i+1} \sum_{j=i+1}^{n} \mathbf{W}_{i+2,j} \mathbf{F}_j + \mathbf{F}_i \\
&= \mathbf{A}_{i+1} \mathbf{f}_{i+1} + \mathbf{F}_i .
\end{aligned} \tag{A9}
$$

To match at $a = i$ and $b = n$, we have

$$
\begin{aligned}
\mathbf{f}_{a,b} &\equiv \sum_{j=a}^{b} \mathbf{W}_{a+1,j} \mathbf{F}_j \\
&= \sum_{j=a}^{k} \mathbf{W}_{a+1,j} \mathbf{F}_j + \mathbf{W}_{a+1,k+1} \sum_{j=k+1}^{b} \mathbf{W}_{k+2,j} \mathbf{F}_j \\
&= \mathbf{f}_{a,k} + \mathbf{W}_{a+1,k+1} \mathbf{f}_{(k+1),b}
\end{aligned} \tag{A10}
$$

If desired, forces and torques applied by the environment to the manipulator tip may be incorporated in a fashion similar to incorporating the acceleration of the base in the discussion of $\ddot{\mathbf{p}}_{a,b}$. This will not change the displayed combining forms.

Similarly, $\mathbf{n}_i$ satisfies the nonrecursive formula

$$
\begin{aligned}
\mathbf{n}_i &= \sum_{j=i}^{n} \mathbf{W}_{i+1,j} \Big(\mathbf{N}_j + \mathbf{s}_j^* \times \mathbf{F}_j + \mathbf{p}_j^* \times (\mathbf{A}_{j+1} \mathbf{f}_{j+1}) \Big) \\
&= \mathbf{N}_i + \mathbf{s}_i^* \times \mathbf{F}_i + \mathbf{p}_i^* \times (\mathbf{A}_{i+1} \mathbf{f}_{i+1}) \\
&\quad + \mathbf{A}_{i+1} \sum_{j=i+1}^{n} \mathbf{W}_{i+2,j} \Big(\mathbf{N}_j + \mathbf{s}_j^* \times \mathbf{F}_j + \mathbf{p}_j^* \\
&\quad \times (\mathbf{A}_{j+1} \mathbf{f}_{j+1}) \Big) \\
&= \mathbf{N}_i + \mathbf{s}_i^* \times \mathbf{F}_i + \mathbf{p}_i^* \times (\mathbf{A}_{i+1} \mathbf{f}_{i+1}) \\
&\quad + \mathbf{A}_{i+1} \mathbf{n}_{i+1} .
\end{aligned} \tag{A11}
$$

To match at $a = i$ and $b = n$, we must have

$$
\begin{aligned}
\mathbf{n}_{a,b} &\equiv \sum_{j=a}^{b} \mathbf{W}_{a+1,j} \Big(\mathbf{N}_j + \mathbf{s}_j^* \times \mathbf{F}_j + \mathbf{p}_j^* \\
&\quad \times (\mathbf{A}_{j+1} \mathbf{f}_{(j+1),b}) \Big) \\
&= \sum_{j=a}^{k} \mathbf{W}_{a+1,j} \Big(\mathbf{N}_j + \mathbf{s}_j^* \times \mathbf{F}_j + \mathbf{p}_j^* \\
&\quad \times (\mathbf{A}_{j+1} (\mathbf{f}_{j+1,k} + \mathbf{W}_{j+2,k+1} \mathbf{f}_{k+1,b})) \Big) \\
&\quad + \mathbf{W}_{a+1,k+1} \sum_{j=k+1}^{b} \mathbf{W}_{k+2,j} (\mathbf{N}_j + \mathbf{s}_j^* \times \mathbf{F}_j \\
&\quad + \mathbf{p}_j^* \times (\mathbf{A}_{j+1} \mathbf{f}_{(j+1),b}) \Big) \\
&= \mathbf{n}_{a,k} + \mathbf{W}_{a+1,k+1} \mathbf{n}_{k+1,b} \\
&\quad + \sum_{j=a}^{k} \mathbf{W}_{a+1,j} \Big(\mathbf{p}_j^* \times (\mathbf{W}_{j+1,k+1} \mathbf{f}_{k+1,b}) \Big) \\
&= \mathbf{n}_{a,k} + \mathbf{W}_{a+1,k+1} ((\mathbf{A}_{k+1}^T \mathbf{R}_{a,k}) \\
&\quad \times \mathbf{f}_{k+1,b} + \mathbf{n}_{k+1,b})
\end{aligned} \tag{A12}
$$

Appendix B: Unification of Logarithmic $a = b$ and $a \neq b$ Cases

By the following technical artifice we can make the $a = b$ case look like $a \neq b$.

$$
\mathbf{W}_{a,k} = \mathbf{I}
$$