PARALLEL ALGORITHMS FOR COMPUTATION OF THE MANIPULATOR INERTIA MATRIX

Masoud Amin-Javaheri and David E. Orin

Department of Electrical Engineering
The Ohio State University
Columbus, Ohio 43210

Abstract

This paper presents the development of an $O(\log_2 N)$ parallel algorithm for the manipulator inertia matrix. It is based on the most efficient serial algorithm which uses the composite rigid body method. Recursive doubling is used to reformulate the linear recurrence equations which are required to compute the diagonal elements of the matrix. It results in $O(\log_2 N)$ levels of computation. Computation of the off-diagonal elements involves N linear recurrences of varying-size and a new method, which avoids redundant computation of position and orientation transforms for the manipulator, is developed. The $O(\log_2 N)$ algorithm is presented in both equation and graphic forms which clearly show the parallelism inherent in the algorithm.

1. Introduction

A major problem in effectively realizing advanced control schemes for robotic systems has been the difficulty of implementing the kinematic and dynamic equations required for coordination, in real time. This problem is accentuated by the increasing structural and task complexities of the next generation of robots under development for space applications. Coordination of multiple-chain systems, with compliant structures operating in higher speeds regimes while making and breaking contact with the environment, places stringent computational demands on the control system.

One approach that has been used in advanced dynamic control schemes to obtain better performance has been to employ the inertia matrix to decouple the dynamics along the several axes of a robot manipulator. This allows either linear or nonlinear control schemes to be more effectively applied [1,2,3]. Specific tasks in which the inertia matrix has been applied in the control include surface tracking and object identification using force control [4] and computation of collision effects [5].

Determination of the inertia matrix involves a considerable amount of computation (approximately equal to that of Inverse Dynamics for a 6 degree-of-freedom manipulator). The most efficient serial algorithm for computing the inertia matrix was first developed by Walker and Orin [6] and requires $O(N^2)$ time on a single processor system. Systolic architectures have been proposed in [7] which reduce the order of the computation to $O(N)$ using N processors. The composite rigid body method developed in [6] was used in [7]. Essentially, sets of links at the end of the manipulator are considered to be fixed with respect to each other so that elementary physics principles may be used to compute their composite mass, center of mass, and moment of inertia. Use of the Newton-Euler dynamic equations on the reduced system results in efficient computation of the inertia matrix components.

This paper presents the development of a parallel algorithm to compute the manipulator inertia matrix in $O(\log_2 N)$ time. Recursive doubling [8], which may be applied to linear recurrence equations to reduce the order of the computation, is used to compute the diagonal elements of the inertia matrix. Computation of the off-diagonal elements involves solution of N sets of linear recurrence equations of size N, $N-1$, etc. for which recursive doubling is not easily applied. Calculation of position and orientation transforms across the links of a varying-size composite rigid body at the base end of the manipulator is required. A new method is developed to compute the off-diagonal elements in $O(\log_2 N)$ time, and it avoids redundant computation of the position and orientation transforms.

Set notation is used to develop the equations for the algorithm in a form which explicitly shows the parallelism available. The notation used was first developed in part in [9] where recursive doubling was used

to compute the Jacobian.

Previously, recursive doubling was applied to solve Inverse Dynamics in $O(\log_2 N)$ time [10]. Prior to this, Lathrop [11] had achieved similar results through a logarithmic recursion method which was derived through a restructuring of the fundamental computational framework for the equations. Parallel computation of the inertia matrix has also been considered in the context of computing robot forward dynamics [12]. Recursive doubling was used to compute the diagonal elements and a modified row-sweeping algorithm was used to compute the off-diagonal elements with a resulting algorithm which was of $O(N)$. The work of this paper differs from [12] in that here quantities are not transformed to base coordinates before applying recursive doubling. Also, the new method for computing the off-diagonal elements results in a parallel algorithm which is of $O(\log_2 N)$. More recently, Fijany and Bejczy [13] have developed two parallel algorithms which achieve the lower bound in the computation time of $O(\log_2 N) + O(1)$. However, when they mapped the algorithms onto arrays of processors, they concluded that an $O(N)$ parallel/pipeline algorithm will have a much improved efficiency with only a slight reduction in speedup.

In the next section, a brief overview of the $O(N)$ parallel algorithm is given. In the section following, the $O(\log_2 N)$ parallel algorithm is developed. The entire parallel algorithm is then summarized in a table in a form which shows much of the parallelism inherent in the algorithm. Finally, the work is summarized, conclusions are made, and several areas in which the work may be extended are discussed.

2. $O(N)$ Parallel Algorithm for the Inertia Matrix

An $O(N)$ parallel algorithm, based upon the determination of the mass, center of mass, and moment of inertia of a series of composite rigid bodies for an N-degree-of-freedom open-chain manipulator, has been previously derived to compute the inertia matrix, $\mathbf{H(q)}$ [7]. It was based on the earlier work of Walker and Orin [6] in which an efficient $O(N^2)$ algorithm was developed for the inertia matrix to further realize efficient dynamic simulation on a single processor. In addition to the $O(N)$ algorithm, various systolic architectures were also proposed in [7] to achieve a real-time response. A complete listing of the algorithm is shown in Table 1.

Briefly, Inverse Dynamics is applied to the manipulator N times. Starting with joint N and working toward joint 1, a unit acceleration is applied to a joint with all joint velocities and other joint accelerations equal to zero. This simply divides the manipulator into two sets of composite rigid bodies with one degree of freedom between them. The mass (M_i), center of mass ($\mathbf{c}_i$), and moment of inertia ($\mathbf{E}_i$) for the composite rigid body at the end of the manipulator (links i through N) are first computed recursively using basic physics principles. Then for each composite rigid body, the forces and moments at joint i due to a unit acceleration there ($\mathbf{f}_{i,i}, \mathbf{n}_{i,i}$) may be simply computed by applying the Newton-Euler equations of motion to the composite body. The component of the force along (prismatic) or moment about (revolute) the joint axis (i) is the diagonal component of the inertia matrix, $H_{i,i}$. The required force or moment needed to ensure zero velocity and acceleration at joint j (for $j < i$) is simply the off-diagonal element of the inertia matrix, $H_{j,i}$. These forces and moments ($\mathbf{f}_{j,i}, \mathbf{n}_{j,i}$) are recursively computed for the various joints of the lower composite rigid body by simple resolution of the force and moment at joint i to the required points. Only the diagonal and upper off-diagonal elements of the inertia matrix are computed since the inertia matrix is symmetric.

Noting Table 1, the computation of the composite rigid body parameters and the diagonal elements of the inertia matrix is seen to require $O(N)$ time. The computation of the off-diagonal elements involves N recursions each of which may be computed in parallel, also giving $O(N)$ time.

3. Development of an $O(\log_2 N)$ Parallel Algorithm for the Inertia Matrix

The concept of recursive doubling [8] may be used to develop an $O(\log_2 N)$ parallel algorithm to compute the composite rigid body parameters and diagonal elements of the inertia matrix. It has been previously applied to achieve $O(\log_2 N)$ parallel algorithms for Inverse Dynamics [10,11]. In order to understand the basic concept and develop the notation used, computation of the composite rigid body masses for a manipulator of eight degrees of freedom will first be considered in this section. A parallel algorithm of $O(\log_2 N)$ will be developed for computing these masses, M_i. Then, the approach will be extended to computing all of the composite rigid body parameters, resulting in an $O(\log_2 N)$ parallel algorithm for computing the diagonal elements of the

Table 1: $O(N)$ Parallel Algorithm for Computing the Inertia Matrix.

CONST

$$M_{N+1} = 0$$
$$\mathbf{c}_{N+1} = \mathbf{0}$$
$$\mathbf{E}_{N+1} = \mathbf{0}$$
$$\mathbf{z}_0 = [0\ 0\ 1]^T$$
$$\sigma_i = \begin{cases} 1 & \text{revolute joint} \\ 0 & \text{prismatic joint} \end{cases}$$

BEGIN

{* Computation of composite rigid body parameters and diagonal elements of the inertia matrix *}

FOR i := N TO 1 DO
BEGIN1

$$M_i := M_{i+1} + m_i$$
$$\mathbf{c}_i := \left\{ \frac{1}{M_i} \left[m_i (\mathbf{s}_i^*) + M_{i+1} \left({}^iU_{i+1}\, \mathbf{c}_{i+1} + {}^i\mathbf{p}_{i+1}^* \right) \right] \right\}$$
$$\mathbf{E}_i := {}^iU_{i+1} \mathbf{E}_{i+1}\, {}^{i+1}U_i + M_{i+1} \left[\left({}^iU_{i+1}\, \mathbf{c}_{i+1} + {}^i\mathbf{p}_{i+1}^* - \mathbf{c}_i \right) \cdot \left({}^iU_{i+1}\, \mathbf{c}_{i+1} + {}^i\mathbf{p}_{i+1}^* - \mathbf{c}_i \right) \mathbf{1} \right.$$
$$\left. - \left({}^iU_{i+1}\, \mathbf{c}_{i+1} + {}^i\mathbf{p}_{i+1}^* - \mathbf{c}_i \right) \left({}^iU_{i+1}\, \mathbf{c}_{i+1} + {}^i\mathbf{p}_{i+1}^* - \mathbf{c}_i \right)^T \right]$$
$$+ \mathbf{I}_i + m_i \left[(\mathbf{s}_i^* - \mathbf{c}_i) \cdot (\mathbf{s}_i^* - \mathbf{c}_i) \mathbf{1} - (\mathbf{s}_i^* - \mathbf{c}_i) (\mathbf{s}_i^* - \mathbf{c}_i)^T \right]$$
$$\mathbf{F}_i := \sigma_i (\mathbf{z}_0 \times M_i\, \mathbf{c}_i) + \bar{\sigma}_i (M_i\, \mathbf{z}_0)$$
$$\mathbf{N}_i := \sigma_i (\mathbf{E}_i\, \mathbf{z}_0)$$
$$\mathbf{f}_{i,i} := \mathbf{F}_i$$
$$\mathbf{n}_{i,i} := \mathbf{N}_i + \mathbf{c}_i \times \mathbf{F}_i$$
$$H_{i,i} := \sigma_i (\mathbf{n}_{i,i} \cdot \mathbf{z}_0) + \bar{\sigma}_i (\mathbf{f}_{i,i} \cdot \mathbf{z}_0)$$

END1

{* Computation of off-diagonal elements of the inertia matrix *}

FOR ALL i := N TO 1 DO
FOR j := i-1 TO 1 DO
BEGIN2

$$\mathbf{f}_{j,i} := {}^jU_{j+1}\, \mathbf{f}_{j+1,i}$$
$$\mathbf{n}_{j,i} := {}^jU_{j+1} \left(\mathbf{n}_{j+1,i} + {}^j\mathbf{p}_{j+1}^* \times \mathbf{f}_{j+1,i} \right)$$
$$H_{j,i} := \sigma_j (\mathbf{n}_{j,i} \cdot \mathbf{z}_0) + \bar{\sigma}_j (\mathbf{f}_{j,i} \cdot \mathbf{z}_0)$$

END2

END

inertia matrix.

Computation of the off-diagonal elements of the inertia matrix involves N independent recursions of varying size for which recursive doubling is not easily applied. The last part of this section gives an $O(\log_2 N)$ algorithm to compute these elements. It effectively uses the position and orientation transforms for the full N-link manipulator, which are computed while obtaining the diagonal elements, to transform forces and moments over a reduced, varying-size composite rigid body at the base end.

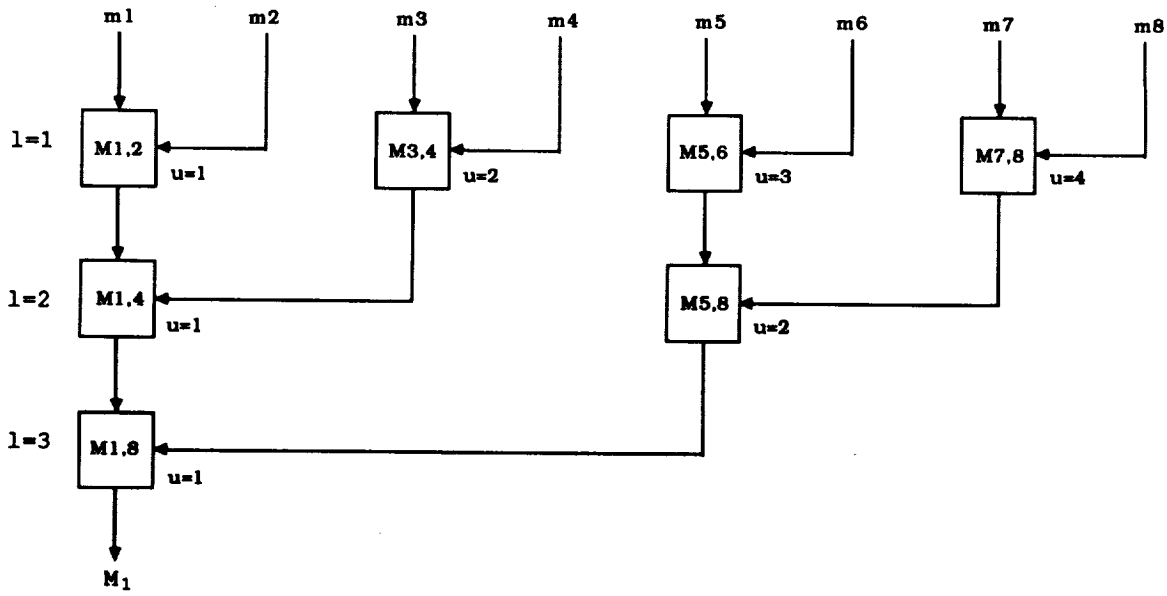

Figure 1: Flow of Data and Computation for Determining M_1 ($M_{1,8}$)

3.1 Parallel Algorithm for M_i

As shown in Table 1, a recursive algorithm for computing M_i is given by

$$M_{N+1} = 0 \tag{1}$$

$$M_i = M_{i+1} + m_i \qquad \text{for } i = N, \cdots, 1. \tag{2}$$

Eq. (2) is an example of a linear recurrence relationship for which recursive doubling [8] may be applied. Consider computation of M_1, the composite rigid body mass for the entire manipulator. The mass across sets of two links may first be computed, all in parallel. For eight links, the links are simply combined as follows:

$$\{1\},\{2\},\{3\},\{4\},\{5\},\{6\},\{7\},\{8\} \longrightarrow \{1,2\},\{3,4\},\{5,6\},\{7,8\} \tag{3}$$

which indicates that the size of the set, for which the mass has been computed, has doubled. Again, doubling the number of links in a set gives

$$\{1,2\},\{3,4\},\{5,6\},\{7,8\} \longrightarrow \{1,2,3,4\},\{5,6,7,8\}, \tag{4}$$

indicating that the mass is now computed for sets of four links. The doubling effect may be recursively applied (recursive doubling):

$$\{1,2,3,4\},\{5,6,7,8\} \longrightarrow \{1,2,3,4,5,6,7,8\}, \tag{5}$$

so that the total mass of all eight links, M_1, is available after 3 steps ($\log_2 8$).

Fig. 1 shows the flow of the data and computation involved in the three steps for computing M_1 ($M_{1,8}$). In the figure, $M_{j,k}$ represents the mass of links j through k. This implies the following mapping relationship:

$$M_i \longrightarrow M_{i,N} \tag{6}$$

and

$$m_i \longrightarrow M_{i,i} \tag{7}$$

to extend the previous notation used.

Equations may be developed for computing M_1. The total number of levels of computation is given by

$$l_T = \log_2 N. \tag{8}$$

The total number of sets of links at each level whose composite mass is to be computed in parallel is given by

$$u_T = 2^{l_T - l} \tag{9}$$

where l is the level number. Also, the variable u is defined and used in Fig. 1 as the number of a set on a given level.

At the first level ($l = 1$), links are combined in groups of two; at the second level ($l = 2$), links are combined in groups of four, etc. Thus, the number of links in a set at each level, the width w, is a function of l and is given by

$$w = 2^l. \tag{10}$$

Each computational step in Fig. 1 then combines two sets of links into one:

$$\{i+1, \cdots, j\}, \{j+1, \cdots, k\} \longrightarrow \{i+1, \cdots, j, j+1, \cdots, k\} \tag{11}$$

where

$$i = w(u-1), \tag{12}$$
$$j = w(u-0.5), \textit{ and} \tag{13}$$
$$k = w\,u. \tag{14}$$

Using the above notation for indexing, the following set of equations formalizes the parallel computation of M_1 ($M_{1,8}$).

$$\left\{ \begin{array}{l} l_T := \log_2 N \\ \text{FOR } l := 1 \text{ TO } l_T \text{ DO} \\ \left\{ \begin{array}{ll} w & := 2^l \\ u_T & := 2^{l_T - l} \end{array} \right\} \\ \quad \text{FOR ALL } u := 1 \text{ TO } u_T \text{ DO} \\ \quad \left\{ \begin{array}{ll} i & := w(u-1) \\ j & := w(u-0.5) \\ k & := w\,u \end{array} \right\} \\ \quad\quad \text{BEGIN} \\ \quad\quad M_{i+1,k} := M_{i+1,j} + M_{j+1,k} \\ \quad\quad \text{END} \end{array} \right\} \tag{15}$$

The above equations only determine the following set of composite rigid body masses:

$$\{M_{1,8}, M_{5,8}, M_{7,8}, M_{8,8}\} \equiv \{M_1, M_5, M_7, M_8\}. \tag{16}$$

To obtain the other composite rigid body masses needed, in general, multiple computations must be performed on each set of links at each level. All necessary computational steps are shown in Fig. 2. Note that the masses for subsets of links, from a link in the first half of a set to the last link in the set (Eq. (11)), must be computed. In so doing, another parameter which gives the total number of computations for each set of links on level l may be defined:

$$v_T = 2^{l-1}. \tag{17}$$

Here, v is also defined to be the computation number for set u for a given level l. Note that

$$u_T * v_T = 2^{l_T - l} * 2^{l-1} = 2^{l_T - 1} = \frac{N}{2} \tag{18}$$

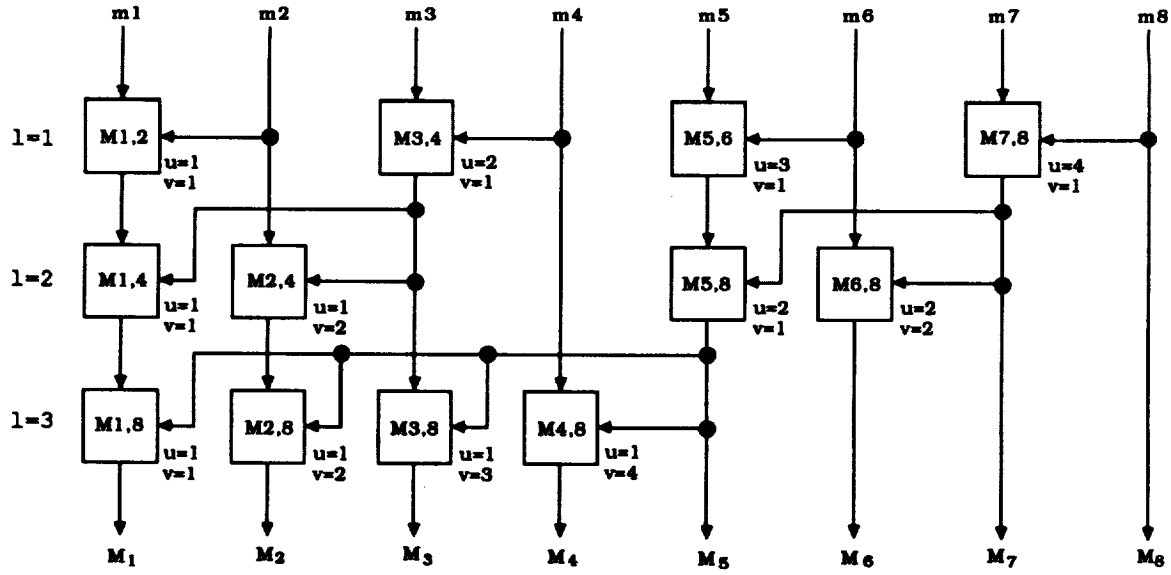

Figure 2: Flow of Data and Computation for Determining M_i ($M_{i,N}$)

which is constant over all the levels.

3.2 Parallel Algorithm for the Diagonal Elements

The approach taken to compute the composite rigid body masses, M_i, may be extended to develop the $O(\log_2 N)$ parallel algorithm for all of the composite rigid body parameters and to further compute the diagonal elements of the inertia matrix. The algorithm is summarized in the first part of Table 2 and will be discussed in the following paragraphs.

First consider the computation of the composite rigid body parameters: mass (M), center of mass ($\mathbf{c}$), and moment of inertia ($\mathbf{E}$). At a given level l, the main body of the computation is to calculate these in parallel, for any combination of u and v, across the sets of links from $i+v$ to k. Note that the components of $\mathbf{c}$ and $\mathbf{E}$ are determined with respect to the coordinate system of the base link of the set $(i+v)$ so that the orientation and position transforms from links $i+v$ to $j+1$, ${}^{i+v}U_{j+1}$ and $\mathbf{p}_{i+v,j+1}$, are required in the computations. Note also that transform of inertial quantities associated with the links, back to the base of the manipulator, is not required here as has been the case in other work [12]. As in the general case for recursive doubling [8], the transforms needed on level l are computed on level $l-1$. Also, in the equations, note that $\mathbf{p}_{i,i+1} \equiv {}^{i}\mathbf{p}^{*}_{i+1}$, $M_{i,i} \equiv m_i$, $\mathbf{c}_{i,i} \equiv \mathbf{s}^{*}_{i}$, and $\mathbf{E}_{i,i} \equiv \mathbf{I}_i$ which are all initially given for each link i.

The ceiling function $\lceil\ \rceil$ and conditions on the range of the indices j and k have been used so that the equations are appropriate for any number of degrees of freedom, N. Note that the computation is required only if the number of degrees of freedom N is greater than or equal to the number of the first link $(j+1)$ of the second of the sets of links to be combined (Eq. (11)).

The composite rigid body parameters as well as the diagonal components of the inertia matrix may be computed in $O(\log_2 N)$ time and this is graphically depicted as Stage A of the computation in Fig. 3. In the figure, the numbers in the parentheses within a box give the associated links for which the computation is made. A "zeroth" level of computation is also shown in which the position and orientation transforms across each individual link are computed. Note that the angle for the first degree of freedom is not needed and that ${}^{i}U_{i+1}$ and ${}^{i}\mathbf{p}^{*}_{i+1}$ are computed for link i.

Table 2: $O(\log_2 N)$ Parallel Algorithm for Computing the Inertia Matrix.

BEGIN11

$l_T := \lceil \log_2 N \rceil$

FOR $l := 1$ TO l_T DO
BEGIN12

$$\left\{ \begin{array}{ll} w & := 2^l \\ u_T & := 2^{l_T - l} \\ v_T & := 2^{l-1} \end{array} \right\}$$

FOR ALL $u := 1$ TO u_T AND
FOR ALL $v := 1$ TO v_T WITH

$$\left\{ \begin{array}{l} i := w(u-1) \\ j := w(u-0.5) \\ k := w\,u \\ \text{IF } (k > N) \text{ THEN } (k := N) \end{array} \right\}$$

IF $(j+1 \leq N)$ THEN DO
BEGIN13

$$^{i+v}U_{k+1} := {}^{i+v}U_{j+1}\ {}^{j+1}U_{k+1}$$

$$\mathbf{p}_{i+v,k+1} := \mathbf{p}_{i+v,j+1} + {}^{i+v}U_{j+1}\ \mathbf{p}_{j+1,k+1}$$

$$M_{i+v,k} := M_{i+v,j} + M_{j+1,k}$$

$$\mathbf{c}_{i+v,k} := \frac{1}{M_{i+v,k}} \left[M_{i+v,j}\ \mathbf{c}_{i+v,j} + M_{j+1,k} \left({}^{i+v}U_{j+1}\ \mathbf{c}_{j+1,k} + \mathbf{p}_{i+v,j+1} \right) \right]$$

$$\begin{aligned} \mathbf{E}_{i+v,k} := {}& {}^{i+v}U_{j+1}\ \mathbf{E}_{j+1,k}\ {}^{j+1}U_{i+v} + \mathbf{E}_{i+v,j} \\ & + M_{j+1,k} \left[\left({}^{i+v}U_{j+1}\ \mathbf{c}_{j+1,k} + \mathbf{p}_{i+v,j+1} - \mathbf{c}_{i+v,k} \right) \cdot \left({}^{i+v}U_{j+1}\ \mathbf{c}_{j+1,k} + \mathbf{p}_{i+v,j+1} - \mathbf{c}_{i+v,k} \right) \mathbf{1} \right. \\ & \left. - \left({}^{i+v}U_{j+1}\ \mathbf{c}_{j+1,k} + \mathbf{p}_{i+v,j+1} - \mathbf{c}_{i+v,k} \right) \left({}^{i+v}U_{j+1}\ \mathbf{c}_{j+1,k} + \mathbf{p}_{i+v,j} - \mathbf{c}_{i+v,j} \right)^T \right] \\ & + M_{i+v,j} \left[\left(\mathbf{c}_{i+v,j} - \mathbf{c}_{i+v,k} \right) \cdot \left(\mathbf{c}_{i+v,j} - \mathbf{c}_{i+v,k} \right) \mathbf{1} \right. \\ & \left. - \left(\mathbf{c}_{i+v,j} - \mathbf{c}_{i+v,k} \right) \left(\mathbf{c}_{i+v,j} - \mathbf{c}_{i+v,k} \right)^T \right] \end{aligned}$$

END13
END12
FOR ALL $i := 1$ TO N DO
BEGIN14

$$\mathbf{f}_{i,i} := \sigma_i \left(\mathbf{z}_0 \times M_{i,N}\ \mathbf{c}_{i,N} \right) + \bar{\sigma}_i \left(M_{i,N}\ \mathbf{z}_0 \right)$$

$$\mathbf{n}_{i,i} := \sigma_i \left(\mathbf{E}_{i,N}\ \mathbf{z}_0 \right) + \mathbf{c}_{i,N} \times \mathbf{f}_{i,i}$$

$$H_{i,i} := \sigma_i \left(\mathbf{n}_{i,i} \cdot \mathbf{z}_0 \right) + \bar{\sigma}_i \left(\mathbf{f}_{i,i} \cdot \mathbf{z}_0 \right)$$

END14
END11

3.3 Parallel Algorithm for the Off-Diagonal Elements

To obtain the off-diagonal elements of the inertia matrix, the force and moment at joint i due to a unit acceleration there ($\mathbf{f}_{i,i}$ and $\mathbf{n}_{i,i}$) should be resolved to the previous joints ($\mathbf{f}_{j,i}$ and $\mathbf{n}_{j,i}$) for $j \leq i$. This involves a total of N linear recurrences of size N, $N-1$, etc. Several approaches to parallel computation of the off-diagonal elements are first discussed in this section. Then an efficient approach, which uses the transforms computed in Stage A for the diagonal elements and which achieves an $O(\log_2 N)$ time, is detailed.

First of all, computation of the off-diagonal terms may be computed in $O(N)$ time if the parallel algorithm given in Table 1 is employed. This results in a computation time of $K_1 O(N) + K_2 O(\log_2 N)$ for the entire inertia matrix where the K_1 coefficient is relatively small. This is similar to the modified row-sweeping algorithm

Table 2 (continued)

BEGIN21
FOR ALL $x := 1$ TO N DO
BEGIN22

$l_x := \lceil \log_2 x \rceil$

FOR $l := 1$ TO l_x DO
BEGIN23

$$\left\{ \begin{array}{lll} w & := & 2^l \\ u_T & := & 2^{l_x - l} \\ v_T & := & 2^{l-1} \end{array} \right\}$$

FOR ALL $u := 1$ TO u_T AND
FOR ALL $v := 1$ TO v_T WITH

$$\left\{ \begin{array}{lll} i & := & w(u-1) \\ j & := & w(u-0.5) \\ k & := & w\,u \end{array} \right\}$$

IF $(j+1 \leq x \leq k)$ THEN DO
BEGIN24

$$\mathbf{f}_{i+v,x} := {}^{i+v}U_{j+1}\,\mathbf{f}_{j+1,x}$$

$$\mathbf{n}_{i+v,x} := {}^{i+v}U_{j+1}\,(\mathbf{n}_{j+1,x} + \mathbf{p}_{i+v,j+1} \times \mathbf{f}_{j+1,x})$$

$$H_{i+v,x} := \sigma_{i+v}\,(\mathbf{n}_{i+v,x} \cdot \mathbf{z}_0) + \bar{\sigma}_{i+v}\,(\mathbf{f}_{i+v,x} \cdot \mathbf{z}_0)$$

END24
END23
END22
END21

applied by Lee and Chang [12] when computing the inertia matrix for parallel forward dynamics computation.

The order of the computation may be further reduced if recursive doubling is applied to each of the recurrences. Computation of the largest recurrence, which gives the elements of the last column of the inertia matrix, can then be achieved in $O(\log_2 N)$ time. Since each of the recurrences may be computed in parallel, the overall computation time for the off-diagonal elements is $O(\log_2 N)$. The major problem with this approach, however, is that the position and orientation transforms (U's and p's) across the links of a varying-size composite rigid body at the base end of the manipulator are computed independently. This results in redundant computation both within this stage of computation and with Stage A for the diagonal elements.

The most efficient method for computing the off-diagonal elements is to use the transforms computed in Stage A while yet completing the computation in $O(\log_2 N)$ time. But it should be understood that these transforms were basically computed for a manipulator of size N only. However if the computational structure of Stage A is considered in Stage B for the pff-diagonal elements, then the more efficient algorithm is achieved. Noting Fig. 3, computation of the off-diagonal elements of column 6 is shown. Judicious elimination of many of the computational blocks (shown with dashed boxes) results in effective use of the transforms available from Stage A while still achieving $O(\log_2 N)$ time.

Note that the computation in the left part of Stage B is generally not needed. This is shown implemented in the equations, in the last part of Table 2, through appropriate conditions on the indices j and k. Essentially, the computation is not required unless the column number x falls within the range of the second of the sets of links to be combined (Eq. (11)). In Fig. 3, the numbers in parentheses within a box give the indices for the $\mathbf{f}$'s and $\mathbf{n}$'s which are calculated. Not explicitly shown in the figure is the flow of the position and orientation transforms from Stage A to Stage B which are required for the computation of the off-diagonal terms.

The $O(\log_2 N)$ parallel algorithm is shown in its entirety in Table 2. When compared with Table 1, it may be noted that the total number of primitive operations has increased with the parallel algorithm. In general,

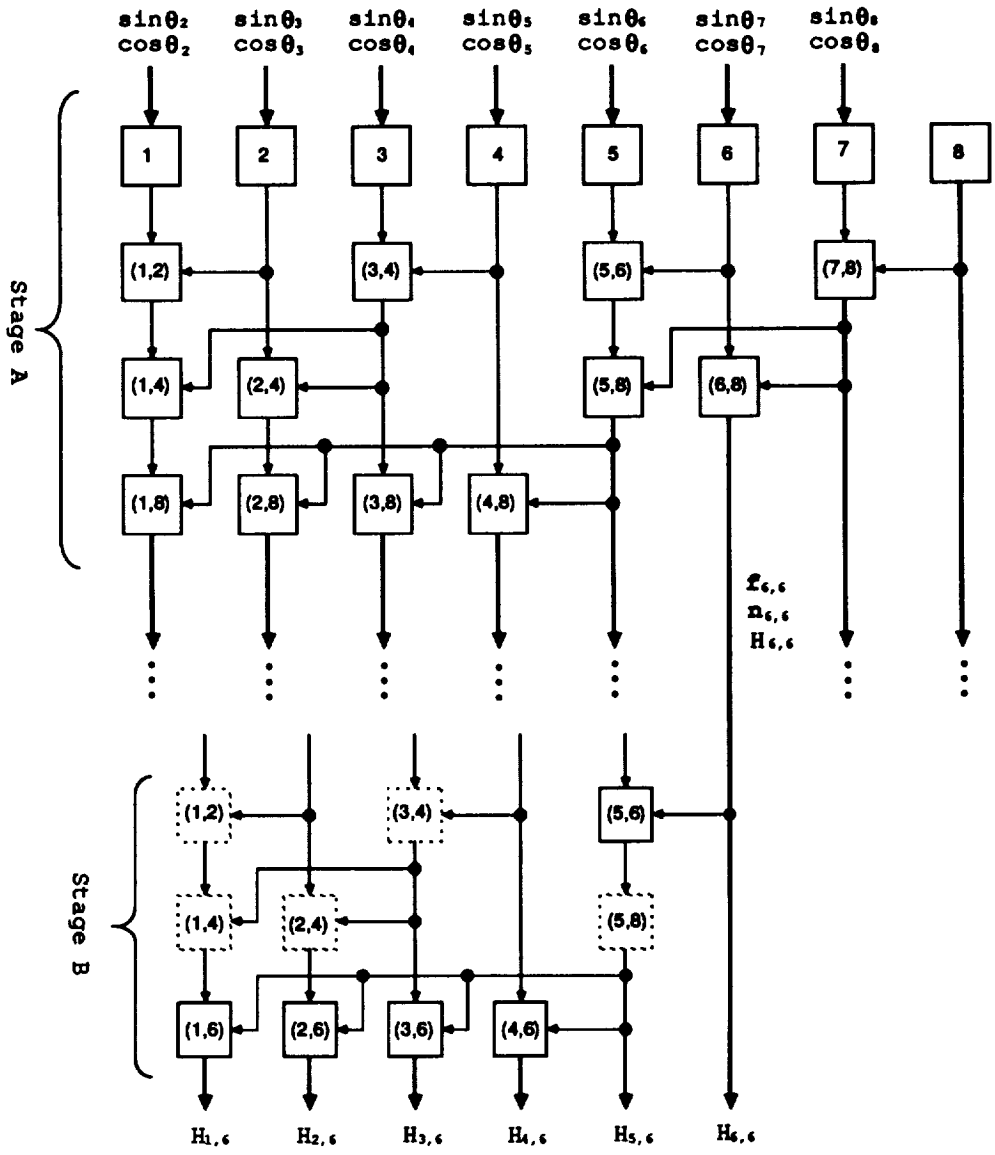

Figure 3: Flow of Data and Computation for the Parallel Algorithm ($N = 8$).

the number of primitive operations increases as one attempts to decrease the order of the computation. In fact if the total number of operations decreased when going from a serial algorithm to a parallel algorithm, then the parallel algorithm should also be used on a serial processor since it becomes the most efficient serial algorithm as well.

4. Summary and Conclusions

This paper has outlined the development of an $O(\log_2 N)$ parallel algorithm for computing the $N \times N$ inertia matrix for a robot manipulator. A listing of the algorithm is given in Table 2, and its flow of computation and data is shown in Fig. 3. In each case, the parallelism inherent in the algorithm is explicitly shown.

A recursive doubling technique [8] was used to achieve computational reduction over the $O(N)$ parallel algorithm listed in Table 1 in computing the diagonal elements of the inertia matrix. It avoids transformation of inertial quantities, associated with each link, to base coordinates. An $O(\log_2 N)$ algorithm, which uses

the position and orientation transforms computed for the entire manipulator when determining the diagonal elements, was then formulated to calculate the upper off-diagonal elements of the inertia matrix. Additional computation to determine the transforms over a varying-size composite rigid body (fixed set of links) at the base end of the manipulator is also avoided.

Investigations have also been made in associated work to determine the relationship between the order of the computation and the number of processors required for implementation. As expected, as the order of computation decreases, the total number of processors required increases.

Work is also under way to efficiently map the $O(\log_2 N)$ algorithm into a parallel architecture structure while accounting for communications (I/O) overhead. The basic objective is to minimize the computational latency while maximizing CPU utilization. Further, work is under way to increase concurrent task processing on multiple processors by developing a new computational model which includes effective use of prediction algorithms. Hopefully, the work of this paper will provide the foundation for parallel implementations of the inertia matrix which will facilitate effective realizations of dynamic control schemes for space telerobotic systems.

Acknowledgments

This work was supported by the National Science Foundation under Computational Engineering Grant No. EET-8718434.

REFERENCES

[1] J. R. Hewit and J. Padovan, "Decoupled feedback control of a robot and manipulator arms," in *Proc. of the 3rd CISM-IFToMM Symposium on the Theory and Practice of Robots and Manipulators*, pp. 251–266, New York: Elsevier, 1979.

[2] M. Leborgne, R. Dumas, J. J. Borrelly, C. Samson, and B. Espiau, "Nonlinear control of robot manipulators, Part 2: simulation and implementation of a robust control method," *IRISA/INRIA Report*, Rennes, France, 1986.

[3] O. Khatib, "A unified approach for motion and force control of robot manipulators: the operational space formulation," *IEEE Journal of Robotics and Automation*, vol. RA-3, no. 1, pp. 43–53, February 1987.

[4] J. Bay and H. Hemami, "Hybrid control of a robotic manipulator over an unknown constraint surface," To be Published in *IEEE Journal of Robotics and Automation*, 1988.

[5] Y. Zheng and H. Hemami, "Mathematical modeling of a robot collision with its environment," *Journal of Robotics Systems*, vol. 2, no. 3, pp. 289–307, 1985.

[6] M. W. Walker and D. E. Orin, "Efficient dynamic computer simulation of robotic mechanisms," *Journal of Dynamic Systems, Measurement, and Control*, vol. 104, pp. 205–211, September 1982.

[7] M. Amin-Javaheri and D. E. Orin, "A systolic architecture for computation of the manipulator inertia matrix," in *Proc. of IEEE International Conference on Robotics and Automation*, pp. 647–653, Raleigh, North Carolina, April 1987.

[8] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Transactions on Computer*, vol. C-22, pp. 786–793, August 1973.

[9] D. E. Orin, K. W. Olson, and H. H. Chao, "Systolic architectures for computation of the Jacobian for robot manipulators," in *Computer Architectures for Robotics and Automation*, J. H. Graham, Ed., pp. 39–67, New York: Gordon and Breach Science Publishers, 1987.

[10] C. S. G. Lee and P. R. Chang, "Efficient parallel algorithm for robot Inverse Dynamics computation," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-16, no. 4, pp. 532–542, July/August 1986.

[11] R. H. Lathrop, "Parallelism in manipulator dynamics," *The International Journal of Robotics Research*, vol. 4, no. 2, pp. 80–102, Summer 1985.

[12] C. S. G. Lee and P. R. Chang, "Efficient parallel algorithms for robot forward dynamics computation," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. SMC-18, no. 2, pp. 238–251, March/April 1988.

[13] A. Fijany and A. K. Bejczy, "Parallel algorithms for computation of manipulator inertia matrix," Engineering Memorandum, No. 347-88-249, Jet Propulsion Laboratory, July 1988.

SPATIAL REPRESENTATIONS AND REASONING