Proceedings of the 1992 IEEE
International Conference on Robotics and Automation
Nice, France - May 1992

A Parallel Algorithm and Architecture for the Control of Kinematically Redundant Manipulators

Anthony A. Maciejewski James M. Reagin

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

Abstract—**This work presents a parallel algorithm for solving the equations of motion for kinematically redundant robotic systems. This algorithm, which relies on the calculation of the Singular Value Decomposition (SVD), is implemented on a simple linear array of processing elements. By taking advantage of the error bounds on the perturbation of the SVD, it is shown that an array of only four AT&T DSP chips can result in control cycle times of less than 3 milliseconds for a seven degree-of-freedom manipulator.**

I. INTRODUCTION

The vast majority of efforts to utilize redundancy in robotic manipulators have been focused on the resolution of redundancy at the kinematic level. The kinematics of manipulators is frequently represented by

$$\dot{\mathbf{x}} = J\dot{\theta} \tag{1}$$

where $\dot{\mathbf{x}}$ is an m-dimensional vector specifying the end effector velocity, $\dot{\theta}$ is an n-dimensional vector denoting the joint velocities, and J is the m by n Jacobian matrix. For redundant manipulators $n > m$ so that the general solution to (1) is typically presented in the form

$$\dot{\theta} = J^{+}\dot{\mathbf{x}} + (I - J^{+}J)\mathbf{z} \tag{2}$$

where $^{+}$ denotes the pseudoinverse and $(I - J^{+}J)\mathbf{z}$ is the projection of an arbitrary vector $\mathbf{z}$ in $\dot{\theta}$ space onto the null space of J. The second term in (2) is the homogeneous solution to (1) since it results in no end effector velocity and will be denoted here by $\dot{\theta}_H$. This homogeneous solution is frequently used to optimize some secondary criterion under the constraint of the specified end effector velocity by choosing $\mathbf{z}$ to be the gradient of some function $g(\theta)$ [4]. The homogeneous solution can also be used to optimize secondary criteria defined in Cartesian space, either to impose a priority to the manipulation variables [8] or to avoid obstacles [5], by using

$$\mathbf{z} = [J_S(I - J^{+}J)]^{+}(\dot{\mathbf{x}}_S - J_S J^{+}\dot{\mathbf{x}}) \tag{3}$$

where the subscript S refers to the secondary criterion. The overall solution is then given by substituting (3) into (2) to obtain

$$\dot{\theta} = J^{+}\dot{\mathbf{x}} + [J_S(I - J^{+}J)]^{+}(\dot{\mathbf{x}}_S - J_S J^{+}\dot{\mathbf{x}}) \tag{4}$$

which has been simplified by taking advantage of the fact that the projection operator is Hermetian and idempotent [5]. An analogous expression at the acceleration level allows the local minimization of joint torque.

An alternative to the formulations of (2) and (4) that also utilizes the available redundancy is to include additional kinematic constraints to the original problem described by (1). If the vector $\dot{\mathbf{x}}$ is augmented with $n - m$ additional kinematic constraints then the resulting Jacobian will be square and traditional inverses can be applied when it is nonsingular [9]. It has also been shown [1] that the secondary criterion $g(\theta)$ can be optimized by including the constraint that the gradient of this function be orthogonal to the null space of J. In both of these cases, the extra constraints will introduce algorithmic singularities [1] in addition to the kinematic singularities of the original manipulator.

One effective method of dealing with singularities, be they kinematic or algorithmic, is to use the damped least squares formulation [7],[11]. The damped least squares solution of an equation such as (1) will be denoted by $\dot{\theta}^{(\lambda)}$ and is defined as the solution that minimizes the quantity

$$\|\dot{\mathbf{x}} - J\dot{\theta}\|^2 + \lambda^2\|\dot{\theta}\|^2 \tag{5}$$

where λ is a weighting factor, sometimes referred to as the damping factor, which is used to set the relative importance of satisfying (1) versus the norm of that solution. It is easy to show that the damped least squares solution is a generalization of the pseudoinverse solution (obtained by setting $\lambda = 0$). Therefore, in the remainder of this work the first term in (2) or (4) will always be calculated as a damped least squares solution. Since the addition of the homogeneous term will always increase the solution norm, it is added only when the joint norm is below its physical constraint thus implying that the damping factor is zero and that the pseudoinverse has been calculated. The advantage of using this more general technique is that the same algorithm and architecture can be used for all types

This work was supported in part by the NEC Corporation, the TRW Foundation, and General Motors.

0-8186-2720-4/92 $3.00 ©1992 IEEE

of formulations whether they are described by (2), (4), or by an augmented or extended Jacobian.

II. REVIEW OF THE SVD

The SVD of the Jacobian is defined as the matrix factorization

$$J = UDV^T \tag{6}$$

where U is an m by m orthogonal matrix of the output singular vectors, V is an n by n orthogonal matrix of the input singular vectors, and D is a diagonal matrix of the singular values, denoted σ_i, which are typically ordered from largest to smallest. An efficient implementation of the SVD is central to the calculation of (2), (4), and $\dot{\theta}^{(\lambda)}$. In particular, if (6) is written as

$$J = \sum_{i=1}^{r} \sigma_i u_i v_i^T \tag{7}$$

where u_i and v_i are the ith columns of U and V respectively, and r is the rank of J, then the damped least squares solution can be obtained using

$$J^{(\lambda)} = \sum_{i=1}^{r} \frac{\sigma_i}{\sigma_i^2 + \lambda^2} v_i u_i^T \tag{8}$$

and the projection onto the null space by using

$$(I - J^+ J) = \sum_{i=r+1}^{n} v_i v_i^T. \tag{9}$$

Once again, the pseudoinverse is easily obtained from (8) when $\lambda = 0$.

While the Golub-Reinsch algorithm is arguably the best general algorithm for calculating the SVD, for this application one would like to use an algorithm that is more amenable to parallelization [2]. The algorithm used here is based on Givens rotations [6]. Successive Givens rotations are used to generate the orthogonal matrix V that will result in

$$JV = B \tag{10}$$

where the columns of B are orthogonal. A matrix with orthogonal columns can be written as the product of the orthogonal matrix U and the diagonal matrix D

$$B = UD \tag{11}$$

by setting the columns of U to normalized versions of the columns of B

$$\mathbf{u}_i = \frac{\mathbf{b}_i}{\|\mathbf{b}_i\|} \tag{12}$$

and setting the diagonal elements of D to the norms of the columns of B

$$\sigma_i = \|\mathbf{b}_i\| \tag{13}$$

thus resulting in the SVD of J.

The orthogonal matrix V is formed as a product of Givens rotations, each of which is designed to orthogonalize two columns. Considering the current ith and jth columns of J, multiplication by a Givens rotation results in the new columns, $\mathbf{j}_i'$ and $\mathbf{j}_j'$ given by

$$\mathbf{j}_i' = \mathbf{j}_i \cos(\phi) + \mathbf{j}_j \sin(\phi) \tag{14}$$

$$\mathbf{j}_j' = \mathbf{j}_j \cos(\phi) - \mathbf{j}_i \sin(\phi). \tag{15}$$

The terms $\cos(\phi)$ and $\sin(\phi)$ required to achieve orthogonality can be computed by using the formulas given in [6] which are based on the quantities

$$p = \mathbf{j}_i^T \mathbf{j}_j \tag{16}$$

$$q = \mathbf{j}_i^T \mathbf{j}_i - \mathbf{j}_j^T \mathbf{j}_j \tag{17}$$

$$v = \sqrt{4p^2 + q^2} \tag{18}$$

so that for $q \geq 0$

$$\cos(\phi) = \sqrt{\frac{v+q}{2v}} \quad \text{and} \quad \sin(\phi) = \frac{p}{v\cos(\phi)} \tag{19}$$

and for $q < 0$

$$\sin(\phi) = \mathrm{sgn}(p)\sqrt{\frac{v-q}{2v}} \quad \text{and} \quad \cos(\phi) = \frac{p}{v\sin(\phi)} \tag{20}$$

where

$$\mathrm{sgn}(p) = \begin{cases} 1 & \text{if } p \geq 0 \\ -1 & \text{if } p < 0 \end{cases}. \tag{21}$$

If the Givens rotation to orthogonalize columns i and j is denoted by V_{ij} then the matrix V can be computed as the product of a set of $n(n-1)/2$ rotations, referred to as a sweep. While the number of sweeps required to orthogonalize the columns of J is not generally known *a priori*, it has been shown that by using information from the SVD of the previous J one can obtain V in a single sweep [6]. In particular, if one considers θ_k to be the current configuration of the manipulator, then the SVD of J from the previous computation cycle time is known and is given by

$$J(\theta_{k-1}) = U(\theta_{k-1})D(\theta_{k-1})V^T(\theta_{k-1}). \tag{22}$$

If one considers the current manipulator Jacobian to be a perturbation of the previous Jacobian then the matrix $J(\theta_k)V(\theta_{k-1})$ will have nearly orthogonal columns provided that this perturbation is small compared to $J(\theta_{k-1})$.

Therefore, in this work V is calculated as

$$V(\theta_k) = V(\theta_{k-1}) \left(\prod_{i=1}^{n-1} \prod_{j=i+1}^{n} V_{ij} \right) \qquad (23)$$

where only a single sweep is performed to update the value of $V(\theta_{k-1})$. A discussion of the types of configurations which will introduce errors into this approximation is found in [6].

III. A PARALLEL ARCHITECTURE AND ALGORITHM

This section discusses a simple parallel architecture and the implementation of the above algorithm on that architecture so that solutions in the form of (2), (4), or $\theta^{(\lambda)}$ can be calculated in real time. The architecture to solve these kinematic equations of motion consists of a host processor and a linear array of $n/2$ processing elements (PE's). Each PE in the linear array can exchange data with the PE on its right or left as well as with the host processor. This architecture is quite similar to that proposed in [10] for the control of redundant manipulators, however, it does not require the specialized VLSI implementation of a CORDIC SVD processor.

Obtaining solutions in the form of (2) or (4) involves three distinct steps. The first step is the calculation of the end effector Jacobian, J which is computed locally on each of the PE's with the host sending only the current manipulator configuration θ_k. The second step involves the calculation of the SVD of the Jacobian, and the third step involves forming either (2) or (4) using the SVD to form the damped least squares inverse of J, the projection operator $(I - J^+J)$, and for (4) the damped least squares inverse of $[J_S(I - J^+J)]$. The parallel execution of these steps will now be considered in detail.

When the current end effector Jacobian $J(\theta_k)$ is being calculated in the four PE's, the SVD of the previous Jacobian is available in these PE's with each of the PE's containing two singular values and the input and output singular vectors associated with these singular values. The first step in calculating the SVD of the current Jacobian, identified as step 2.1 in the Fig. 1, is to multiply the current Jacobian $J(\theta_k)$ by $V(\theta_{k-1})$, the V matrix associated with the SVD of the previous Jacobian.

The second step in calculating the SVD, identified as 2.2 in Fig. 1, is the performance of a single sweep of Given's rotations. In this figure v_i denotes the columns of the V matrix which are a product of all the Givens rotations performed up to that point, and b_i denotes the columns of the current matrix B which is the Jacobian $J(\theta_k)$ multiplied by the matrix V. The sweep begins at step 2.2.1 with each PE performing a Given's rotation on

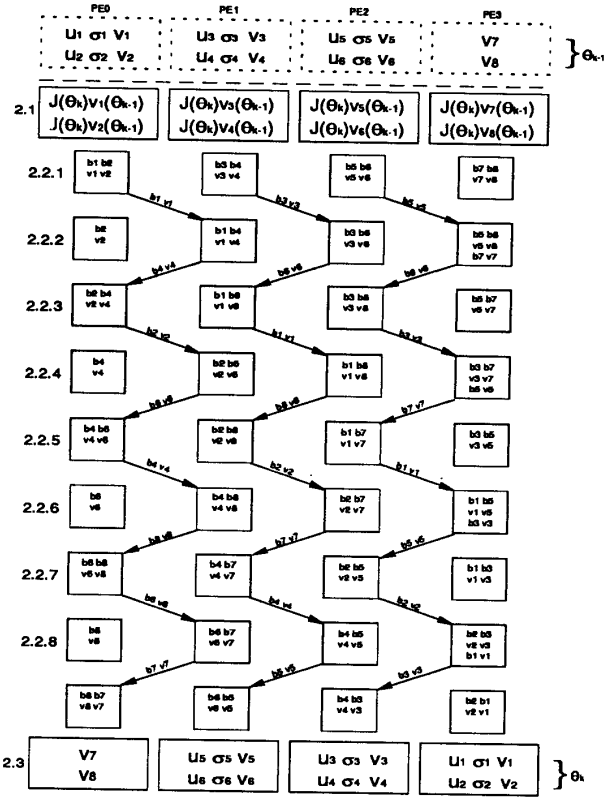

Fig. 1 Step 2 of the algorithm consists of calculating the SVD of the Jacobian. This is done by first multiplying the Jacobian by the previous matrix V in step 2.1, then performing a single sweep of Givens rotations in steps 2.2.1-2.2.8, and finally normalizing the resulting columns of the matrix B to determine the matrix U and the singular values.

its two columns of B. As discussed before, the angle of the plane rotation is selected to make these two columns orthogonal. The PE's then shift their left column of B and V to the processor on the right. Now at step 2.2.2 each PE, with the exception of the left-most one, performs a Given's rotation on its two columns. Each processor then shifts its right column of B and V to the PE on its left. This process of left and right shifts continues until step 2.2.8 when eight levels of Given's rotations (a single sweep) have been performed. If two data paths are provided between PE's then the column shift ordering used in [2] can be used to eliminate the idle PE in each level. The last step in determining the SVD, identified as 2.3 in Fig. 1 involves calculating the matrix $U(\theta_k)$ using (12) and the singular values using (13).

Once the SVD of J has been computed the formation of (2) or (4), denoted as the third step, is greatly simplified. The first term of either equation is considered as a damped least squares solution

$$\dot{\theta}_P = \dot{\theta}^{(\lambda)} = J^{(\lambda)}\dot{\mathbf{x}} \tag{24}$$

which is easily computed using the SVD of J by applying (8). Note once again that this provides the pseudoinverse solution if $\lambda = 0$. The second term of (2) is also easily computed once the SVD is available by using (9). For example, when $m = 6$ and $n = 8$ the PE's that have columns 1 to 6 of U,D, and V compute their contribution to $\dot{\theta}_P$ as

$$\dot{\theta}_P(i, i+1) = \frac{\sigma_i}{\sigma_i^2 + \lambda^2} v_i u_i^T \dot{\mathbf{x}} + \frac{\sigma_{i+1}}{\sigma_{i+1}^2 + \lambda^2} v_{i+1} u_{i+1}^T \dot{\mathbf{x}} \tag{25}$$

while the PE with columns 7 and 8 computes the term

$$\dot{\theta}_H = (I - J^+ J)z = v_7 v_7^T z + v_8 v_8^T z \tag{26}$$

which is the second term of (2). If the value of the damping factor was zero then the intermediate terms are summed to form $\dot{\theta}$, as defined by (2).

The computation of (4) is more difficult than (2) due to the calculation of the projection

$$A^+ = [J_S(I - J^+ J)]^+, \tag{27}$$

however, it can be shown that the calculation of the SVD of A^+ is not as expensive as it might first seem. It has been previously shown [5] that

$$(I - J^+ J)[J_S(I - J^+ J)]^+ = [J_S(I - J^+ J)]^+ \tag{28}$$

so that the range of A^+ is known to be a subset of the null space of J. Therefore, to compute its SVD the Givens rotations can be restricted to the $(n-r)$-dimensional space described by the singular vectors v_{r+1} to v_n rather than the full n-dimensional space. As an example, consider the case of a seven degree-of-freedom manipulator with a one-dimensional null space so that the SVD of A is given by

$$A = \sigma_{A_1} u_{A_1} v_7^T \tag{29}$$

where v_7 describes the null space of J. It is easy to show that the only possibly non-zero singular value of A can be computed by using

$$\sigma_{A_1} = \|J_S v_7\|. \tag{30}$$

If $\sigma_{A_1} \neq 0$ then the singular vector u_{A_1} associated with this singular value is computed using

$$u_{A_1} = \frac{J_S v_7}{\sigma_{A_1}}. \tag{31}$$

Similarly for a two-dimensional null space the SVD of A, given by

$$A = \sigma_{A_1} u_{A_1} v_{A_1}^T + \sigma_{A_2} u_{A_2} v_{A_2}^T, \tag{32}$$

can be computed using a greatly simplified procedure. First, the matrix J_S is multiplied by the singular vectors that form the null space of J:

$$b_{A_1} = J_S v_7 \tag{33}$$

$$b_{A_2} = J_S v_8 \tag{34}$$

which results in a basis for the range of A^+. Next, a single Givens rotations is performed to orthogonalize these columns and determine the two input singular vectors:

$$b'_{A_1} = b_{A_1}\cos(\phi) + b_{A_2}\sin(\phi) \tag{35}$$

$$b'_{A_2} = b_{A_2}\cos(\phi) - b_{A_1}\sin(\phi) \tag{36}$$

$$v_{A_1} = v_7\cos(\phi) + v_8\sin(\phi) \tag{37}$$

$$v_{A_2} = v_8\cos(\phi) - v_7\sin(\phi). \tag{38}$$

The singular values and output singular vectors of A are then computed using (12) and (13) on b'_{A_1} and b'_{A_2}. For higher degrees of redundancy the vectors b_{A_1} thru $b_{A_{n-r}}$ can be orthogonalized by using sweeps of Givens rotation that are restricted to this $n - r$ dimensional subspace, a task for which this architecture has been optimized.

Fig. 2 illustrates the computations required to obtain a solution in the form of (4) on the parallel architecture for the case where $m = 6$ and $n = 8$. As in the computation of (2), the PE's with columns 1 to 6 compute their contribution to $\dot{\theta}_P$ using (8) in step 3.1. The PE with columns 7 and 8 is responsible for computing the SVD of $[J_S(I-J^+J)]^+$ using the method described above. In step 3.2 the results from the three PE's are combined to form $\dot{\theta}_P$ which is then distributed back to each of the three PE's in step 3.3 so they can each compute two elements of the intermediate term

$$\dot{\mathbf{x}}_H = \dot{\mathbf{x}}_S - J_S\dot{\theta}_P \tag{39}$$

which are then combined in step 3.4. Once $\dot{\mathbf{x}}_H$ and A^+ are available the second term of (4) is formed using

$$\dot{\theta}_H = \frac{\sigma_{A_1}}{\sigma_{A_1}^2 + \lambda^2} v_{A_1} u_{A_1}^T \dot{\mathbf{x}}_H + \frac{\sigma_{A_2}}{\sigma_{A_2}^2 + \lambda^2} v_{A_2} u_{A_2}^T \dot{\mathbf{x}}_H. \tag{40}$$

The value sent to the joint controller is then obtained by adding $\dot{\theta}_P$ and $\dot{\theta}_H$.

IV. IMPLEMENTATION

The algorithm described in the previous section was implemented on a linear array of eight DSP32 processors (rated at 5 MFLOPS) that were part of an AT&T Pixel

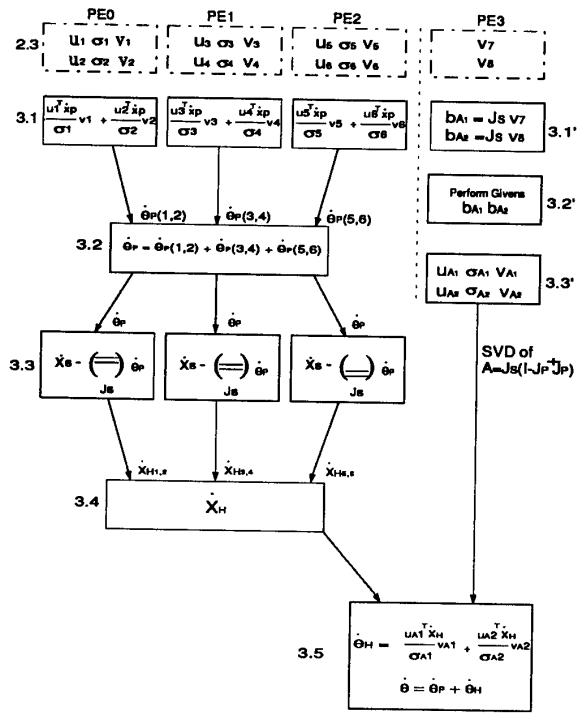

Fig. 2 After the SVD of J_P has been formed, the calculation of $\dot{\theta} = J_P^+ \dot{x}_P + [J_S(I - J_P^+ J_P)]^+ (\dot{x}_S - J_S J_P^+ \dot{x}_P)$ can also be done in parallel.

machine with a Sun workstation serving as the host processor. This configuration allowed timing evaluations for manipulator systems with up to sixteen degrees of freedom. The algorithm was implemented in C and then compiled for the DSP32 PE's. While there was an effort to efficiently utilize register variables to improve performance, the execution times are by no means to be considered optimal.

The time required for the calculation of $\dot{\theta}$, denoted by $T_{CALC\_\dot{\theta}}$, is given by

$$T_{CALC\_\dot{\theta}} = T_{JACOBIAN} + T_{SVD} + T_{SOLVE} \qquad (41)$$

where $T_{JACOBIAN}$ is the time to acquire the joint positions and calculate the current Jacobian, T_{SVD} is the time to calculate the SVD of the end effector Jacobian J, and T_{SOLVE} is the time required to form the solution of (2) or (4) using the SVD of J. Timing information for both serial and parallel implementations of equations (2) and (4) on the AT&T pixel machine is presented in Table 1. Note that the value of $T_{JACOBIAN}$ is left as a variable parameter since it is manipulator dependent. The equations for calculating the Jacobian of the CESAR manipulator [3], which required approximately 0.38 milliseconds were implemented in order to provide some comparison with the other execution times. Table 1 shows that once the SVD of the manipulator Jacobian has been calculated, the solution to either (2) or (4) can be computed in a relatively short period of time. This is a major motivation for using the SVD in the computation of these equations. Since the calculation of the SVD accounts for the majority of the computation time, approximately 1.7 milliseconds, the speedup ratio (3.6) associated with the SVD has the greatest effect on the overall performance improvement for the calculation of (2) or (4).

The time required to calculate the SVD is given by

$$T_{SVD} = T_{MULT\ V} + T_{SWEEP} + T_{NORM\ B} \qquad (42)$$

where $T_{MULT\ V}$ denotes the time required to multiply $J(\theta_k)$ by $V(\theta_{k-1})$ on the PE's, T_{SWEEP} is the time to perform a single sweep of Givens rotations, and $T_{NORM\ B}$ is the time required to calculate the singular values and columns of U from the columns of B at the end of the sweep. From Table 1 one can see that there is a factor of four speedup for $T_{MULT\ V}$ and $T_{NORM\ B}$ and a 3.5 times speedup for T_{SWEEP} illustrating the extremely parallel nature of this particular algorithm. The theoretical maximum speedup of four could by achieved for T_{SWEEP} as well by using the column ordering used in [2] thus removing the idle PE in every other step as discussed previously.

The computation of (2) in parallel is 3.5 times faster than the calculation of (2) in serial for the case where $n = 8$. The two components involved in the computation of (2) are T_{SVD} and $T_{SOLVE\ (2)}$. The most important thing to note is the relatively insignificant amount of time, 0.16 milliseconds, required to solve (2) once the SVD of the manipulator Jacobian has been calculated. The overall speedup factor for the component $T_{SOLVE\ (2)}$ is 3.3. This is mostly due to step 3.1 of the calculations for $T_{SOLVE\ (2)}$ where there is almost a factor of four speedup except that there are two fewer divisions on PE3 since it is calculating $\dot{\theta}_H$ rather than contributing to $\dot{\theta}_P$. In this timing example, z was calculated as the gradient of the squares of the joints angles in order to utilize the redundancy for joint range availability [4].

Fig. 2 shows the computation of (4) on the parallel architecture. The time $T_{SOLVE\ (4)}$ can be broken down into $T_{J_S\ MULT}$ which is the time required to multiply J_S times v_7 and v_8, T_{GIVEN} which is the time required to perform the single Givens rotation, $T_{NORM\ B}$ which is the time required to find σ_{A1}, σ_{A2}, u_{A1} and u_{A2} from b'_{A1} and b'_{A2}, and $T_{CALC\_\dot{x}_H}$ which is the time required to compute the intermediate term $\dot{\mathbf{x}}_H$. Table 1 shows that there is a 3.2 times overall speedup in the calculation of (4) on the parallel architecture and a 1.9 times speedup

TABLE 1
TIMING STATISTICS

SVD

	Step	Serial	Parallel	S/P
$T_{MULT\ V}$	2.1	0.88 ms	0.22 ms	4.0
T_{SWEEP}	2.2	4.50 ms	1.29 ms	3.5
$T_{NORM\ B}$	2.3	0.88 ms	0.22 ms	4.0

EQ. (2): $\dot{\theta} = J^{+}\dot{x} + (I - J^{+}J)z$

	Serial	Parallel	S/P
T_{SVD}	6.2 ms	1.7 ms	3.6
$T_{SOLVE\ (2)}$	0.5 ms	0.2 ms	3.3
TOTAL	6.7 ms + T_J	1.9 ms + T_J	3.5

EQ. (4): $\dot{\theta} = J_P^{+}\dot{x}_P + [J_S(I - J_P^{+}J_P)]^{+}(\dot{x}_S - J_S J_P^{+}\dot{x}_P)$

	Serial	Parallel	S/P
T_{SVD}	6.2 ms	1.7 ms	3.6
$T_{SOLVE\ (4)}$	1.4 ms	0.7 ms	1.9
TOTAL	7.6 ms + T_J	2.4 ms + T_J	3.2

for $T_{SOLVE\ (4)}$. As mentioned before, the speedup associated with T_{SVD} has the largest effect since the computation time associated with the SVD of the Jacobian is a major portion of the total computation time. In Fig. 2, steps 3.1′ to 3.3′, the computation of the SVD of A, are executed in 0.6 milliseconds. Note that for the case where $n = 8$, these calculations are performed on a single processor which is the reason that there is no speedup for these individual calculations. However, for manipulators with higher degrees of redundancy step 3.2′ will not be a single Givens rotation, but will instead be several sweeps of Givens rotations. This means that the parallel architecture will be of even greater benefit in the higher order cases since the processors with columns $r+1$ to n will be performing the sweeps of Givens rotations to compute the SVD of A rather than a single processor.

V. CONCLUSIONS

This work has presented a parallel algorithm and architecture for solving the equations of motion for kinematically redundant robotic systems. It has been shown that the various desirable forms of the solutions to these equations can all be efficiently obtained when the SVD of the Jacobian is available. The implementation of this algorithm on an array of AT&T DSP32 processors has illustrated that computation cycle times of less than three milliseconds can be obtained even when using the most complicated form of the solution.

REFERENCES

[1] J. Baillieul, "Kinematic programming alternatives for redundant manipulators," in *IEEE Int. Conf. Robotics Automat.*, (St. Louis, Mar. 25-28, 1985), pp. 722–728.

[2] R.P. Brent, F.T. Luk, and C. Van Loan, "Computation of the singular value decomposition using mesh-connected processors," *J. VLSI Comput. Syst.*, vol. 1, no. 3, pp. 242–271, 1985.

[3] R.V. Dubey, J.A. Euler, S.M. Babcock, "An efficient gradient projection optimization scheme for a seven-degree-of-freedom redundant robot with spherical wrist," in *1988 IEEE Int. Conf. Robotics Automat.*, (Philadelphia, Apr. 24-29, 1988), pp. 28–36.

[4] A. Liégeois, "Automatic supervisory control of the configuration and behavior of multibody mechanisms," *IEEE Trans. Syst., Man, Cyber.*, vol. SMC-7, no. 12, pp. 868–871, Dec. 1977.

[5] A.A. Maciejewski and C.A. Klein, "Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments," *Int. J. Robotics Res.*, vol. 4, no. 3, pp. 109–117, Fall 1985.

[6] A.A. Maciejewski and C.A. Klein, "The singular value decomposition: Computation and applications to robotics," *Int. J. Robotics Res.*, vol. 8, no. 6, pp. 63–79, Dec. 1989.

[7] Y. Nakamura and H. Hanafusa, "Inverse kinematic solutions with singularity robustness for robot manipulator control," *ASME J. Dynamic Systems, Measurement, Control*, vol. 108, no. 3, pp. 163–171, Sept. 1986.

[8] Y. Nakamura, H. Hanafusa, and T. Yoshikawa, "Task-priority based redundancy control of robot manipulators," *Int. J. Robotics Res.*, vol. 6, no. 2, pp. 3–15, Summer 1987.

[9] H. Seraji, "Configuration control of redundant manipulators: Theory and implementation," *IEEE Trans. Robotics Automat.*, vol. 5, no. 4, pp. 472-490, Aug. 1989.

[10] I.D. Walker and J.R. Cavallaro, "Parallel VLSI architectures for real-time control of redundant robots," in *Proc. Fourth American Nuclear Society Topical Meeting on Robotics and Remote Systems*, pp. 299–309, Feb. 24-28, 1991.

[11] C.W. Wampler II, "Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods," *IEEE Trans. Syst., Man, Cyber.*, vol. SMC-16, no. 1, pp. 93–101, Jan./Feb. 1986.