A Parallel Algorithm for Singular Value Decomposition as Applied to Failure Tolerant Manipulators

Tracy D. Braun, Anthony A. Maciejewski, and Howard Jay Siegel

Parallel Processing Laboratory
School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907-1285 USA
{tdbraun, maciejew, hj}@ecn.purdue.edu

Abstract

The system of equations that govern kinematically redundant manipulators is commonly solved by finding the singular value decomposition (SVD) of the corresponding Jacobian matrix. This can require considerable amounts of time to compute, thus a parallel SVD algorithm minimizing execution time is sought. The approach employed here lends itself to parallelization by using Givens rotations and information from previous decompositions. The key contributions of this research include the presentation and implementation of a new variation of a parallel SVD algorithm to compute the SVD for a set of post-fault Jacobians. Results from implementation of the algorithm on a MasPar MP-1 and an IBM SP2 are provided. Specific issues considered for each implementation include how data is mapped to the processing elements, the effect that increasing the number of processing elements has on execution time, and the type of parallel architecture used.

1. Introduction

The singular value decomposition (SVD) of a matrix is a fundamental matrix decomposition that provides information useful for a wide range of applications (e.g., feature extraction, data reduction, and low rank approximation [11]). In many of these applications, rapid computation of the SVD is necessary, e.g., when the SVD is used for solving the systems of equations for real-time motion control of robotic manipulators. Consider the kinematics of a manipulator with $\underline{n}$ joints operating in $\underline{m}$ dimensions. Let $\underline{\dot{x}} \in \mathbb{R}^m$ specify the manipulator's end-effector velocity, $\underline{\dot{\theta}} \in \mathbb{R}^n$ denote the joint velocities of the manipulator, and $\underline{J} \in \mathbb{R}^{m \times n}$ be the manipulator Jacobian matrix. The kinematics of the manipulator can then be represented by the equation $\dot{x} = J\,\dot{\theta}$. The ability to compute this SVD in real time would allow for singularity avoidance and evaluation of dexterity.

In general, techniques for computing the SVD of an arbitrary matrix involve iterating an unknown number of times until a data-dependent convergence criterion is met. Therefore, the number of operations required is not known *a priori* and guaranteeing real-time computation of the SVD is difficult. However, for this application the current Jacobian matrix, $J(t)$, can be regarded as a perturbation of the previous Jacobian matrix, i.e., $J(t) = J(t-\Delta t) + \Delta J(t)$. Knowledge of this previous state can be used to decrease computational complexity during calculation of the current SVD [8, 9].

This article defines a new approximation technique based on research performed in [13] for real-time analysis of the SVD for single locked-joint failures of a manipulator. The performance of this approximation technique is analyzed on two different parallel architectures: an SIMD MasPar MP-1 [1] and an MIMD IBM SP2 [16]. Specific issues considered for each implementation include how data is mapped to the processing elements, the effect that increasing the number of processing elements has on execution time, and the type of parallel architecture used. Case studies, such as the one presented here, are a necessary step in developing software tools for mapping an application task onto a single parallel machine, and for mapping an application task onto a heterogeneous suite of parallel machines, where different types of machines are present [10].

The remainder of this article is arranged as follows. The SVD procedure from [8] is reviewed in Section 2. Section 3 describes the new SVD approximation technique. Section 4 examines different mappings of this

This research was supported in part by Sandia National Laboratories under contract number AL-3001 and by the DARPA/ITO Quorum Program under NPS subcontract numbers N62271-97-M-0900, N62271-98-M-0217, and N62271-98-M-0448. T. D. Braun was also supported by a Purdue Benjamin Meisner Fellowship. The MasPar MP-1 was provided and supported in part by the National Science Foundation under grant number CDA-9015696.

technique onto parallel machines. The results summarized in Section 5 are obtained from the MasPar MP-1 implementations. The results from the IBM SP2 implementations are discussed in Section 6. Finally, Section 7 reviews the approximation technique and the application study results.

2. Background Information

For a matrix $J \in \mathbb{R}^{m \times n}$, let $\underline{U} \in \mathbb{R}^{m \times m}$ denote an orthogonal matrix of output singular vectors, $\underline{V} \in \mathbb{R}^{n \times n}$ represent an orthogonal matrix of input singular vectors, and $\underline{\Sigma}$ be a nonnegative diagonal matrix. Then, the SVD of J is defined as the matrix factorization $J = U \Sigma V^T$, where the diagonal elements of Σ, referred to as the singular values and denoted $\underline{\sigma_i}$, are typically ordered so that $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_m \geq 0$. It is assumed for the remainder of this paper that $m < n$.

The most common technique for computing the SVD of a matrix is the Golub-Reinsch algorithm [6]. However, when attempting to parallelize this method, there are two disadvantages. First, it is not straightforward to incorporate information from the SVD of a perturbed matrix to update the SVD. Secondly, the algorithm is relatively sequential in nature.

While there have been several parallel SVD algorithms written and implemented on various machine architectures [3, 5, 7, 14], they make no assumptions about the form of the matrix being decomposed. For this work, the fact that the current Jacobian matrix is a perturbation of the previous Jacobian matrix can be exploited by applying Givens rotations [6] to ultimately orthogonalize the columns of the Jacobian [8].

In particular, consider an orthogonal matrix V, generated by successive Givens rotations, that results in $JV = B$, where $\underline{B} \in \mathbb{R}^{m \times n}$, and the columns of B are orthogonal. The matrix B can be decomposed into two matrices, one orthogonal and one diagonal. Decomposing B in this way, into an orthonormal matrix (U) and a diagonal matrix (Σ) results in $B = U\Sigma$. This is accomplished by letting the columns of U, u_i, be equal to the normalized versions of the columns of B, b_i, and then defining the diagonal elements of Σ to be equal to the norm of the columns of B, i.e.,

$$u_i = \frac{b_i}{\|b_i\|} \text{ and } \sigma_i = \|b_i\|, \quad \text{where} \|b_i\| = \sqrt{b_i^T b_i}. \quad (1)$$

The result is the SVD of J, as given above.

The critical step in the above procedure is determining the orthogonal matrix V that will orthogonalize the columns of J. This V matrix is constructed as a product of Givens rotations, each of which is designed to orthogonalize two columns. Consider the i-th and j-th columns of an arbitrary matrix A. Postmultiplication by a Givens rotation results in two new columns, a_i' and a_j', given by

$$a_i' = a_i \cos(\phi) + a_j \sin(\phi), \quad a_j' = a_j \cos(\phi) - a_i \sin(\phi) \quad (2)$$

The constraint that these columns be orthogonal results in $\cos(\phi)$ and $\sin(\phi)$ terms that are calculated using formulas in [12]. These formulas are based on the following quantities p, q, and c:

$$p = a_i^T a_j, \; q = a_i^T a_i - a_j^T a_j, \text{ and } c = \sqrt{4p^2 + q^2}. \quad (3)$$

For the case when $q \geq 0$,

$$\cos(\phi) = \sqrt{(c+q)/(2c)}, \quad \sin(\phi) = p/(c \cdot \cos(\phi)) \quad (4)$$

and when $q < 0$,

$$\begin{aligned} \sin(\phi) &= \text{sgn}(p)\sqrt{(c-q)/(2c)} \\ \cos(\phi) &= p/(c \cdot \sin(\phi)) \end{aligned} \quad (5)$$

where $\text{sgn}(p) = 1$ if $p \geq 0$, and $\text{sgn}(p) = -1$ if $p < 0$. The two sets of formulas are given so that ill-conditioned equations resulting from the subtraction of nearly equal numbers can always be avoided.

The preceding discussion describes a single Givens rotation that will orthogonalize two columns of a given matrix. The single Givens rotation to orthogonalize columns i and j is denoted $\underline{Q_{ij}}$. For a matrix with n columns, $n(n-1)/2$ rotations are required to orthogonalize each possible pair of columns. This set of $n(n-1)/2$ rotations is referred to as a sweep [6]. Multiple sweeps are generally required to obtain an orthogonal matrix because subsequent rotations may destroy the orthogonality produced by previous rotations. The V matrix can therefore be computed as the series of sweeps such that $V = \prod_{\#sweeps} \left(\prod_{i=1}^{n-1} \prod_{j=i+1}^{n} Q_{ij} \right)$. The disadvantage to this approach is the fact that the number of sweeps required to orthogonalize the columns of J is usually not known beforehand. However, this problem can be circumvented by considering the current Jacobian matrix to be a perturbation of the previous Jacobian, i.e., $J(t) = J(t - \Delta t) + \Delta J(t)$. Using this information, it was shown that a good approximation for the current SVD could be obtained from the previous SVD information in a single sweep if ΔJ is small [8]. That is, using

$$B(t) \approx J(t) \times V(t - \Delta t) \quad (6)$$

and applying one sweep of Givens rotations, the current SVD can be found. Therefore, in this work, V is calculated using $V(t) = V(t - \Delta t) \times \left(\prod_{i=1}^{n-1} \prod_{j=i+1}^{n} Q_{ij} \right)$.

Thus, the previous V matrix is updated using only a single sweep. Because it is known *a priori* that only one sweep of rotations will be performed, the algorithm does not have to iterate until convergence, and the computational complexity of the algorithm is greatly reduced.

3. Algorithm Descriptions

Jacobians of size $m = 6$ and $n = 7$ were chosen because they represent the minimum size for a redundant manipulator in three dimensions and manipulators of this size are commercially available. However, the algorithms presented are completely general and can be used for arbitrary values of m and n.

The goal of this study is not only to find the SVD of 6×7 Jacobians, but also to find the SVD of these Jacobians after a single joint fault has occurred in the manipulator. It is assumed that this single joint fault can be detected by the manipulator and the joint locked into position. Therefore, the Jacobian for the post-fault manipulator, ${}^{f}J$, becomes ${}^{f}J = [\, j_1\ j_2\ \cdots\ j_{f-1}\ \mathbf{0}\ j_{f+1}\ \cdots\ j_n\,]$, where joint f has failed and the column j_f is replaced with $j_f = \mathbf{0}$. The information from the post-fault Jacobians can then be used in several ways, including trajectory planning for maintaining maximum failure tolerance [13].

The approximation technique described below for finding the SVD of these post-fault Jacobians was implemented on two different parallel machines, the MasPar MP-1 and the IBM SP2. The use of a parallel implementation can be justified by the following reasons. (1) This analysis extends to larger arms and to multiple arm systems. (2) This work can be extended to combinations of multiple joint failures. (3) Some systems might require real-time control and recovery capabilities where large amounts of information would be continually computed and stored.

Figure 1 shows the first algorithm, the basic single sweep approximation [17]. In step 1, the previous V matrix and the current J matrix are used to calculate a good initial estimate for the current B matrix. It is assumed in step 2 that the use of the previous decomposition information allows the algorithm to converge in a single sweep. Step 3 then provides the current singular values through straightforward computations.

Notice that if $B = U\Sigma$ is substituted into (6), the algorithm in Figure 1 can be represented by the equation

$$J(t)\ V(t - \Delta t)\ \approx\ U(t)\ \Sigma(t). \qquad (7)$$

The PTPF, previous time post-fault, approximation technique assumes that the decomposition information available from the previous time period includes post-fault matrices. That is, to compute the current post-fault singular values, ${}^{f}\Sigma(t)$, for the current post-fault Jacobian, ${}^{f}J(t)$, the previous post-fault matrix, ${}^{f}V(t - \Delta t)$, is used in (7). Thus, the PTPF approximation technique can be represented by the equation

$${}^{f}J(t)\ {}^{f}V(t - \Delta t)\ \approx\ {}^{f}U(t)\ {}^{f}\Sigma(t)\ . \qquad (8)$$

Figure 2 shows the modifications required to the algorithm from Figure 1 to compute the PTPF approximation. Using this notation, $f = 0$ represents the calculation of the pre-fault SVD, i.e. the SVD for the Jacobian with no columns zeroed out, and $f = 1, \ldots, n$ represent the n post-fault Jacobians with column $1, \ldots, n$ zeroed out, respectively. Given that computing ${}^{f}J(t)$ is simply replacing j_f with $j_f = \mathbf{0}$, and the assumption that ${}^{f}V(t - \Delta t)$ is available from the previous time period, each iteration of the `for all` f loop is independent of previous iterations. Thus, on a parallel machine with enough processors, the entire algorithm could be performed concurrently where each processor would only have to perform the (shorter) algorithm from Figure 1.

The disadvantage to the PTPF technique is the time dependency. If the time interval Δt is increased between decompositions, the performance of the PTPF approximation deteriorates. If Δt is very large it may be preferable to use $V(t)$ as an approximation for ${}^{f}V(t)$. The trade-off between these two approximations is examined in [2].

4. Parallel Mappings Considered

Parallel architectures generally consist of several processing elements (PEs). A PE is a combination of a processor and a memory module, allowing the processor fast access to the local memory. This section presents methods for efficiently distributing the data and computations of the single sweep SVD algorithm among the PEs of a parallel architecture.

For the remainder of this paper, it may be assumed that $n = 8$. This is accomplished by appending a column of zeros to the Jacobian matrix, and will not change the resulting singular values or increase execution time (because of symmetry and concurrent operations).

The PTPF algorithm is based on the single sweep SVD algorithm from Figure 1. Therefore, this section focuses on methods to efficiently implement that algorithm on a parallel architecture, with the goal of decreasing overall execution time. Specifically, three data mapping techniques are summarized: 1CPP, 2CPP, and column segmentation (details in [2]).

One column per PE (1CPP). From the algorithm in Figure 1, notice that no inter-PE communication is required in step 1 if each PE has one column of V, a copy of the entire Jacobian matrix, and one column of B. In step 3, single-column operations can also be performed simultaneously on n PEs. The columns of U and the singular values of Σ are computed from the corresponding columns of B. Again, no inter-PE communications are required.

Step 2, however, presents a disadvantage for this one column per PE (1CPP) distribution. Let the parallel execution of a Givens rotation on all $n/2$ column pairs by the PEs be defined as a rotation step. Then, a minimum of $n-1$ rotation steps must be performed to generate all $n(n-1)/2$ possible column pairings. The 1CPP approach used here performed inter-PE communications for two data items during each rotation step (step 2): one to exchange columns to form a new column pair and compute p, and another to exchange $a_j^T a_j$ to calculate q. The 1CPP communication pattern used formed all possible column pairings using only $n-1$ column transfers [17].

Two column per PE (2CPP). A two column per PE (2CPP) approach that distributes pairs of columns of V and B to $n/2$ PEs was also implemented. This 2CPP approach reduces the frequency and complexity of the inter-PE communications. For the 2CPP approach, step 1 and step 3 are performed concurrently without any inter-PE communications, similar to the 1CPP approach.

Using the 2CPP approach, the first rotation step of step 2 can also be performed without any inter-PE communications. In contrast to the 1CPP method, the 2CPP method only requires one inter-PE communication in between each rotation step so that each PE can obtain a new column pair for orthogonalization. Thus, the $n-1$ rotation steps of step 2 can be performed with only $n-2$ inter-PE communications, where each communication is a new column b_i.

Although it is obvious that step 1 and step 3 of the single sweep SVD algorithm will take twice as long to compute using the 2CPP distribution versus the 1CPP distribution, the 2CPP implementation does not require twice as much total time to execute. The highest percentage of the total execution time for the single sweep SVD algorithm is spent performing step 2, where 2CPP has the advantage of fewer communications. The 2CPP technique implemented is based on a procedure from [17]. Different 2CPP procedures can be found in [5, 9, 14].

Column segmentation. A goal of this study is to extract as much parallelism as possible from both the algorithms and the target machines to reduce execution time. The technique described here divides each column vector of the B and V matrices into r segments, where r is a power of two. This variable r represents the column segmentation of the data (and operations) among PEs and increases the total number of PEs used by a factor of r. Values of $r \in \{1,\ 2,\ 4,\ 8\}$ were implemented. (Some column segments will therefore contain zeros as padding because $m = 6$ is not a power of two. This does not hinder performance because all the operations performed for the $r = 8$ case would still be required if $r = 6$.)

This segmentation of the column data does not interfere with the inter-PE column transfers for the 1CPP and 2CPP methods. Column transfers simply take place between PEs containing the same segment number. Inter-PE communication also occurs among PEs containing different segments of the same column. More details concerning the implementation of column segmentation are in [2].

5. SIMD Architecture Experiments

This section presents the parallel implementations of the PTPF algorithm on an SIMD (single instruction stream, multiple data stream) architecture. The SIMD machine used in this study was a MasPar MP-1 system [1] with 16,384 PEs located at Purdue University.

The MP-1 provides two different high-speed PE interconnection networks, the X-Net and the global router. The X-Net connects a PE to its eight nearest neighbors and provides fast communications for PEs in close proximity. The global router is a multistage interconnection network that connects clusters of PEs and is faster for communications between PEs that are further apart.

There are four different implementations of the PTPF algorithm for the MP-1, based on two different design options: data distribution (1CPP or 2CPP) and interconnection network selection (X-Net or global router). Each implementation also utilized column segmentation for $r \in \{1,\ 2,\ 4,\ 8\}$.

To take advantage of as much parallelism as possible in the global router implementations, only one PE per global router cluster was used to reduce contention. In contrast, implementations using the X-Net selected a collection of PEs that were all adjacent, to keep inter-PE distances short and inter-PE communications fast.

For step 1, each PE contains the entire J matrix, and only a segment (for $r > 1$) of each column of V, so matrix multiplies are performed as concurrent vector–vector multiplies. This creates an $m \times 1$ vector of partial sums on each PE which is then combined to generate the desired result. A recursive doubling scheme

could be used for adding these partial sums together. However, a slightly more efficient method for this case (where PEs only transfer and operate on column segments) called segment combining [2] was used.

Step 2 performs one sweep of rotations on the columns of B and V. To do this, all possible combinations of pairs of columns of B and V must be formed. This makes step 2 the most communication-intensive step, especially for the 1CPP distribution. The 1CPP method requires $n-1$ column transfers and $n-1$ scalar transfers per sweep. The 2CPP approach only requires $n-2$ column transfers per sweep, one after each rotation step. Both approaches also required additional communications for column combining, performed here by full recursive doubling because only the final sum (and not each segment) was important.

Step 3 of the SVD algorithm normalizes the columns of the B matrix to obtain the singular values, as well as the columns of the U matrix, according to (1). Full recursive doubling was repeated for obtaining the final results.

Experimental timing results for the 1CPP, 2CPP, X-net, and global router implementations were examined. Only instructions directly relating to computation or communication of the algorithms were timed. Procedures such as file I/O were not timed to reduce possible disruption by events beyond the control of the programmer, e.g., operating system interrupts.

The timings presented represent the average time to calculate the SVDs of all eight ${}^{f}J$ matrices corresponding to one 6×7 Jacobian matrix. This average is taken over 1000 different randomly generated matrices. (The generation of these matrices is discussed in [2].) Even though the MP-1 is an SIMD machine, meaning it operates synchronously and there should be no variation in timings, averages were still taken because of data conditional execution of statements within the code.

Figure 3 shows a direct comparison between the 1CPP and 2CPP distribution execution times on the MP-1 for the global router PTPF approximation. The execution times are grouped in terms of the number of PEs available. Recall that in some cases data was padded with zeros, because m was not a power of two.

From Figure 3, when comparing cases that have an equal number of PEs available, the 2CPP method has a higher degree of column segmentation so there are more combining operations. Thus, the communication time for the 2CPP method is greater than the 1CPP method. The 2CPP method requires more PE enabling and disabling statements than the 1CPP method.

Comparing computation times (without communication times), the 2CPP technique should be faster than the 1CPP method. The 2CPP method is more conducive to the pair-wise operations of the rotation steps and avoids some of the redundant calculations the 1CPP method must perform. The exception to this observation occurs for 128 PEs because the data was padded with zeros.

There is a decrease in total execution times as a result of increasing the number of PEs. However, execution times improved less as more PEs were added. This implies that the use of column segmentation was beneficial, but provides diminishing returns as the columns were segmented into smaller and smaller pieces. For the 64, 128, and 256 PE cases, the differences in computation times were not enough to overcome the differences in communication times, and the 1CPP technique had the faster total execution times.

A comparison of the X-Net and global router results revealed that the global router implementations achieve the faster execution times. The computation times between the X-Net and global router implementations were nearly equivalent, as one would expect. However, given the communication patterns and matrix sizes of this application, and the ability to select the enabled PEs, the global router implementations provided better performance than the X-Net implementations in each case.

In all of the cases examined on the MasPar MP-1, increasing the number of processors improved execution times. The amount of improvement varied among the different techniques examined. More detailed results for each case can be found in [2].

6. MIMD Architecture Experiments

The parallel implementation of the PTPF algorithm on a MIMD (multiple instruction stream, multiple data stream) architecture. Each PE in a MIMD machine stores its own set of instructions and data in its local memory module. This allows for asynchronous, multiple threads of control, because PEs may contain unique sets of instructions.

The IBM SP2 is a scalable distributed memory MIMD parallel supercomputer [16]. The interconnection network in the SP2 is a multistage interconnection network based on the SP2 High-Performance Switch [16]. Message passing for this study used a C-based implementation of the Message Passing Interface (MPI) [15]. Simulation results were obtained using only thin nodes [16], with submachine sizes of one, two, four, eight, and 16 PEs.

Only two implementations on the SP2 were examined: 1CPP and 2CPP. The biggest difference between the MP-1 and the SP2 implementations was the use of MPI on the SP2. Another difference between the MP-

1 and SP2 implementations was the number of PEs available. The SP2 has fewer than $(n \times r \times (n+1))$ PEs available, so for most cases, the outer PTPF loop cannot be performed concurrently with all values of f, as it was on the MP-1.

For the sections of the algorithms that were based on the single sweep SVD algorithm, the MP-1 and SP2 implementations were very similar. Because of the limited number of PEs available, and the high MPI overheads observed, two additional techniques were added to the MIMD portion of the study, namely an eight column per PE (8CPP) and a four column per PE (4CPP) method. The 8CPP method executes entirely on one PE with no inter-PE communications. The 4CPP method executes on two PEs, each holding four columns, by performing two exchanges, each containing two columns. The 8CPP and 4CPP techniques did not use column segmentation.

Because of their asynchronous operation, there is usually a large variance in timing information from MIMD machines. The timings recorded once again represented the time to calculate the SVDs of all eight ${}^{f}J$ matrices corresponding to one matrix, taken as the average time over 1000 different matrices.

Comparing the 8CPP, 4CPP, 2CPP, and 1CPP distributions of the PTPF algorithm, the communication times dominated the total execution time of the algorithms on the SP2. This is because of the large overhead associated with MPI communications [18]. When using MPI, the time required for setup and initialization of each communication is relatively large. If only small sets of data are being transferred, the overhead can easily require more time than the actual transfer of data. The best case on the IBM SP2 turned out to be the 8CPP case, which used just one matrix per PE, and only half of the available PEs.

Comparing timing results between the IBM SP2 and the MasPar MP-1, the relative strengths of each machine become apparent. The MP-1 is a well balanced machine with computation and communication instructions requiring about the same amount of time to execute. In contrast, the SP2 has superior computational speed but relatively slow communications when using MPI. In most cases, communication times on the MP-1 using the global router were equal to or less than the corresponding SP2 implementations using MPI. However, the SP2 communication times were generally less than X-Net communication times. The advantage in computation time goes to the SP2 which defeated the MP-1 in every instance. This can be attributed in part to the SP2 being a newer machine and having better processor technology available at its time of design and construction. In general, the total execution times for the SP2 were faster than the MP-1.

7. Summary

The system of equations used for the kinematic control of robotic manipulators is frequently represented by a Jacobian matrix. One method for solving this system of equations is based on computing the SVD of the Jacobian matrix. This study uses a technique developed in [8] that exploits the well-behaved nature of the SVD and finds approximations to the SVD in a single sweep. This technique has been extended to calculate approximations for the SVD of the full, pre-fault Jacobian matrix and the set of single locked-joint, post-fault Jacobian matrices. These procedures can provide a basis for the real-time control of kinematically redundant manipulators and also provide fault tolerance information useful for real-time singularity avoidance and error recovery.

Experiments were conducted for the PTPF approximations on commercial SIMD and MIMD architectures, the MasPar MP-1 and IBM SP2. For these experiments, data layout and network selection were compared. Timing results from the MP-1 revealed that the 1CPP, global router method provided the fastest overall execution times. The 8CPP method provided the fastest results on the SP2 because of the high overhead involved with communications. Increased column segmentation was an effective method for reducing computation times on the MP-1 but not on the SP2. All of the methods studied here can be extended to larger analyses, including multiple joint failures, systems of multiple arms, or computation of several fault tolerance measures, all of which would require high levels of parallelism to accomplish in real time.

Acknowledgments - The authors thank Muthucumaru Maheswaran, Mitchell D. Theys, and Bill Whitson for their valuable comments.

References

[1] T. Blank, "The MasPar MP-1 architecture," *IEEE Compcon*, Feb. 1990, pp. 20–24.

[2] T. D. Braun, *Parallel Algorithms for Singular Value Decomposition as Applied to Failure Tolerant Manipulators*, Thesis, School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, Dec. 1997.

[3] R. P. Brent, F. T. Luk, and C. Van Loan, "Computation of the singular value decomposition using mesh-connected processors," *Journal of VLSI and Computer Systems*, Vol. 1, No. 3, 1985, pp. 242–270.

[4] B. Champagne, "SVD-updating via constrained perturbations with applications to subspace tracking,"

30th Asilomar Conference on Signals, Systems and Computers, Nov. 1996, pp. 1379–1385.

[5] H. Chuang and L. Chen, "Efficient computation of the singular value decomposition on a cube connected SIMD machine," *Supercomputing '89*, Nov. 1989, pp. 276–282.

[6] G. H. Golub and C. F. Van Loan, *Matrix Computations, Second Edition*, Johns Hopkins University Press, Baltimore, MD, 1989.

[7] F. T. Luk, "A triangular processor array for computing singular values," *Linear Algebra and its Applications*, Vol. 77, No. 5, 1986, pp. 259–273.

[8] A. A. Maciejewski and C. A. Klein, "The singular value decomposition: Computation and applications to robotics," *The International Journal of Robotics Research*, Vol. 8, No. 6, Dec. 1989, pp. 63–79.

[9] A. A. Maciejewski and J. M. Reagin, "A parallel algorithm and architecture for the control of kinematically redundant manipulators," *IEEE Transactions on Robotics and Automation*, Vol. 10, No. 4, Aug. 1994, pp. 405–414.

[10] M. Maheswaran, T. D. Braun, and H. J. Siegel, "Heterogeneous distributed computing," To appear in *Encyclopedia of Electrical and Electronics Engineering*, J. G. Webster, ed., John Wiley & Sons, New York, NY, 1999.

[11] M. S. Moonen and B. R. L. de Moor, ed., *SVD and Signal Processing, III: Algorithms, Architectures, and Applications*, Elsevier Science, New York, NY, 1995.

[12] J. C. Nash, *Compact Numberical Methods for Computers: Linear Algebra and Function Minimisation*, A. Hilger, Bristol, UK, 1979.

[13] R. G. Roberts and A. A. Maciejewski "A local measure of fault tolerance for kinematically redundant manipulators," *IEEE Transactions on Robotics and Automotion*, Vol. 12, No. 4, Aug. 1996, pp. 543–552.

[14] D. E. Schimmel and F. T. Luk, "A new systolic array for the singular value decomposition," in *Advanced Research in VLSI*, C. E. Leiserson, ed., MIT Press, Cambridge, MA, 1986, pp. 205–217.

[15] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI: The Complete Reference*, MIT Press, Cambridge, MA, 1995.

[16] C. B. Stunkel, D. G. Shea, B. Abali, M. G. Atkins, C. A. Bender, D. G. Grice, P. Hochschild, D. J. Joseph, B. J. Nathanson, R. A. Swetz, R. F. Stuckel, M. Tsao, and P. R. Varker, "The SP2 high-performance switch," *IBM Systems Journal*, Vol. 34, No. 2, 1995, pp. 185–204.

[17] R. R. Ulrey, A. A. Maciejewski, and H. J. Siegel, "Parallel algorithms for singular value decomposition," *Eighth International Parallel Processing Symposium (IPPS '94)*, Apr. 1994, pp. 524–533.

[18] Z. Xu and K. Hwang, "Modeling communication overhead: MPI and MPL performance on the IBM SP2," *IEEE Parallel and Distributed Technology*, Vol. 4, No. 1, Spring 1996, pp. 9–23.

step 1: calculate initial estimate for B from J and previous V, using (6)

step 2: for all column pairs (i,j) of B do

/* one sweep */

calculate p, q, and c using (3)

calculate $\cos(\phi)$ and $\sin(\phi)$, using (4) or (5)

perform rotation on columns i and j of B, similar to (2)

perform rotation on columns i and j of V, similar to (2)

end for

step 3: calculate Σ from B, using (1)

calculate U from B and Σ, using (1)

Figure 1. High-level single sweep SVD algorithm using Givens rotations to find the fault-free SVD, based on [17].

for all $f \in \{0, 1, \ldots, n\}$ do

/* failure in joint f */

/* $f = 0$ represents no fault */

perform single sweep SVD algorithm on ${}^{f}J$

end for

Figure 2. High-level algorithm for finding the post-fault SVD using the PTPF approximation technique.

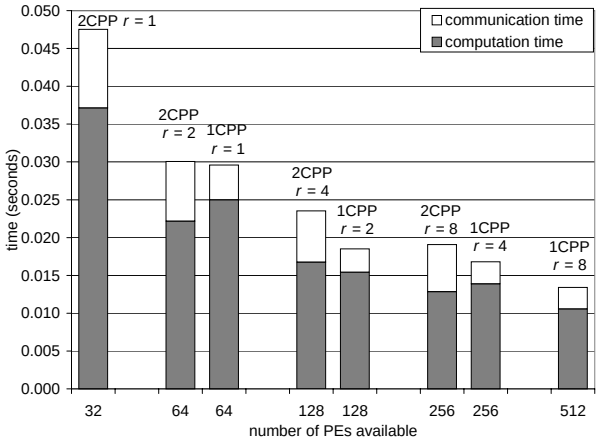

Figure 3. Comparison between 1CPP and 2CPP average execution times for the MasPar MP-1 global router PTPF approximation, in terms of number of PEs available (not necessarily used).