A block matrix generalization of the Jacobi rotation method for computing the eigendecomposition of a symmetric matrix is presented. This Blocked Classical Jacobi (BCJ) algorithm selects for block rotation at each step the off-diagonal block(s) of largest mass. The BCJ algorithm exhibits substantially shorter runtimes than another blocked Jacobi method, but is slower than a scalar Jacobi algorithm on random matrices with *i.i.d.* uniform elements. A probabilistic analysis of the BCJ selection method is presented. Timings and other data are presented from experiments on random matrices.

A Blocked Jacobi Method for the Symmetric Eigenproblem

David E. Foulser

Research Report YALEU/DCS/RR-680
February 1989

This research supported by the Office of Naval Research under grant N00014-86-K-0310.

1 Introduction

The class of Jacobi rotation methods [4,7,10,12] for computing the symmetric eigenvalue decomposition

$$A = UDU^T \tag{1}$$

of an $n \times n$ real matrix A, where U is orthogonal and D is diagonal, has generated substantial interest in recent years, particularly in the context of parallel computer architectures. Algorithms have been developed for systolic processor arrays as well as for more general purpose parallel computers. These methods differ principally from the original method of Jacobi in that they choose a fixed sequence of matrix elements for the necessary orthogonal rotations. Jacobi's method performs a rotation to zero out the largest off-diagonal element at each step; the sequence of rotations is data-dependent.

This paper presents a novel block matrix or "hypermatrix" adaptation [2,3,16] of the original algorithm, which we label the Blocked Classical Jacobi (BCJ) algorithm. The matrix A is treated as a smaller $m \times m$ matrix of $b \times b$ submatrices; computations work on entire submatrices rather than on scalars. Furthermore, the sequence of submatrices to be rotated is chosen to locally maximize the reduction of A to diagonal form by selecting the off-diagonal blocks of largest mass. BCJ reduces serial runtimes compared with other blocked Jacobi methods on a particular class of inputs.

For computers with a hierarchical memory system, in which successively larger yet slower memories are located at increasing distances from the arithmetic processor, many numerical calculations are efficiently structured in terms of block algorithms [3,8,15]. Rather than computing with scalar quantities, block algorithms operate on small square or rectangular submatrices of data. The resulting "surface-to-volume" effect of a single block data transfer followed by several computations allows a fast processor with local memory to achieve nearly full utilization even when supplied by a significantly slower bus or main memory.

The blocked organization of BCJ reduces the overhead cost of determining the maximum off-diagonal elements. It also makes BCJ especially well-suited for implementation on multiprocessors with a hierarchical memory system (*e.g.,* [8]). As well, BCJ is suitable for parallel implementation.

However, the significantly shorter runtimes of the sequential Jacobi suggest that it is to be preferred over all blocked methods on this class of inputs.

The organization of the paper is as follows. Section 2 gives a brief review of serial Jacobi methods for the symmetric eigenproblem. Section 3 gives the motivations for BCJ and presents the algorithm as implemented in this study. Section 4 lays out the numerical experiments with BCJ, including timings and numbers of iterations to convergence. Section 5 presents the analysis of the block selection method using the theory of order statistics, and discusses the implications for the experimental data. Concluding remarks and indications for parallel implementations are presented in section 6. Section 7 contains the proofs of two probabilistic results from section 5.

2 Review of Serial Jacobi Methods

The Jacobi method of solving (1) constructs a sequence of orthogonal rotations $U_1 = U(\theta_1, i_1, j_1, A^{(0)})$, $U_2 = U(\theta_2, i_2, j_2, A^{(1)}),\ldots$, such that $U = U_1 U_2 \cdots$ diagonalizes A (that is, $U^T A U$ is diagonal), $0 \leq \theta_i \leq \pi/4$, and $\lim_{i\to\infty} \theta_i = 0$. In practice the computation is terminated after a finite number of rotations, leaving $U = U_1 U_2 \ldots, U_N$. The rotation U_ν is selected to zero out the matrix elements in positions (i_ν, j_ν) and (j_ν, i_ν).

Given $(i,j) \equiv (i_\nu, j_\nu)$, the rotation angle θ_ν is computed so that $A^{(\nu)} = U_\nu^T A^{(\nu-1)} U_\nu$, according to

$$\begin{pmatrix} a_{ii}^{(\nu)} & a_{ij}^{(\nu)} \\ a_{ji}^{(\nu)} & a_{jj}^{(\nu)} \end{pmatrix} = \begin{pmatrix} c_\nu & s_\nu \\ -s_\nu & c_\nu \end{pmatrix}^T \begin{pmatrix} a_{ii}^{(\nu-1)} & a_{ij}^{(\nu-1)} \\ a_{ji}^{(\nu-1)} & a_{jj}^{(\nu-1)} \end{pmatrix} \begin{pmatrix} c_\nu & s_\nu \\ -s_\nu & c_\nu \end{pmatrix}, \qquad (2)$$

with $a_{ij}^{(\nu)} = a_{ji}^{(\nu)} = 0$; here $A^{(0)} = A$. The cosine c_ν and sine s_ν of the angle θ_ν may be calculated by [9]

$$\tau = (a_{ii}^{(\nu-1)} - a_{jj}^{(\nu-1)})/(2a_{ij}^{(\nu-1)}), \qquad a_{ij}^{(\nu-1)} \neq 0, \qquad (3)$$

then solving for t in

$$t^2 + 2t\tau = 1 \qquad \left(t = \frac{\text{sign}(\tau)}{|\tau| + \sqrt{1+\tau^2}}\right) \qquad (4)$$

and substituting in

$$c_\nu = (1+t^2)^{-1/2}, \qquad s_\nu = c_\nu t. \qquad (5)$$

U_ν is set to the identity matrix, except in rows and columns i_ν and j_ν, where it is zero everywhere but in the 2×2 principal submatrix; there it is $\begin{pmatrix} c_\nu & s_\nu \\ -s_\nu & c_\nu \end{pmatrix}$. If $a_{ij}^{(\nu-1)} = 0$ then c_ν is set to 1 and s_ν to 0, for θ_ν is obviously 0.

There are several methods for choosing the rotation index pair (i, j). The classical Jacobi method selects (i, j) at each step to locally minimize the resulting off-diagonal Frobenius norm by choosing (i, j) as the location of the largest off-diagonal element. However, the effort of determining the location of the maximum element ($O(n^2)$ operations) exceeds the work in calculating and applying the orthogonal rotation U_ν (approximately $18n$ operations neglecting symmetry). For this reason the method is rarely used on computers.

The cyclic-by-rows ordering of elements $((i, j) = (1, 2), (1, 3), \ldots, (1, n), (2, 3), \ldots, (2, n), \ldots, (n-1, n))$ is more amenable to automatic computation. However, the successive index pairs are almost always dependent (sharing a row or column), and thus not suited for parallel computation. Parallel orderings have featured other index pair selections chosen for data locality and utility on a systolic processor. The Brent-Luk and Sameh orderings [4,13] have many desirable features. They preserve data locality and are amenable to systolic or other parallel implementations, they converge faster than the cyclic-by-rows ordering, and they rotate each off-diagonal element exactly once in a "sweep." A particularly useful feature is that at each step, the $n/2$ independent rotations (operating on $n/2$ mutually distinct pairs of rows and columns) may be carried out simultaneously.

3 Algorithm BCJ

We now develop a blocked analogue of the classical Jacobi algorithm for the symmetric eigenproblem (1) that performs more work in selecting the index pairs yet requires less run-time than a blocked Brent-Luk ordering. BCJ also generalizes to computation of the singular value decomposition of a rectangular matrix. The new Blocked Classical Jacobi (BCJ) method selects the largest off-diagonal block(s) for rotation, in order to locally minimize the off-diagonal mass of A. Through a suitable choice of the block size, the extra computations to determine the off-diagonal block of maximum mass are offset by a reduced number of iterations; BCJ is more

efficient on a serial computer than the blocked Brent-Luk Jacobi method.

BCJ is also highly parallel in nature. Where several processors are available to solve a single eigenproblem, the $K > 1$ largest independent off-diagonal blocks may be selected for simultaneous rotations, leading to a straightforward parallel implementation.

At each iteration, BCJ selects an off-diagonal block submatrix (i, j) for rotation and computes a block orthogonal rotation matrix $\hat{U}_\nu$, which it then applies to help reduce A to block diagonal form. The block orthogonal rotation can be chosen as a sequence of scalar Jacobi rotations or from the eigendecomposition of the small block matrix; we use a full scalar Sameh sweep on the small block matrix. (However, there is no restriction that the small block matrix must be diagonalized, only that its off-diagonal mass be reduced. Computations by Bischof [1] on the SVD indicate that the extra effort of completely diagonalizing the block matrix at each step may be wasted.) The method then proceeds by selecting another block element of A to rotate. A final processing step of Sameh sweeps forms the eigenvalues and eigenvectors from the block diagonal elements of A. On an $m \times m$ block matrix, a BCJ "sweep" is $(m^2 - m)/2$ two-block by two-block rotations.

The precise block algorithm for carrying out BCJ to compute the symmetric eigendecomposition (1), with D overwriting A, is as follows. Assume, for ease of exposition, that the block size b divides the matrix size n exactly, so that $n = mb$. $K \geq 1$ independent off-diagonal blocks may be selected for simultaneous rotation. The iterations continue until a tolerance criterion `TOL` is met. The method begins with $U = I$, $\nu = 0$, and continues

1. Compute the squared masses $\{M_{ij}\}_{i,j=1}^{m}$, with

$$M_{ij} = \sum_{r,s=1}^{b} \left(a^{(\nu)}_{(i-1)b+r,(j-1)b+s}\right)^2.$$

2. Select K independent rotation pairs $(i_k, j_k), 1 \leq k \leq K$ with

$$M_{i_k j_k} = \max(M_{ij} | i \notin \{i_1, \ldots, i_{k-1}\}, j \notin \{j_1, \ldots, j_{k-1}\}).$$

3. Compute K block rotations $\{\hat{U}(\theta_k, i_k, j_k, A^{(\nu)})\}_{k=1}^{K}$ to reduce the block off-diagonal mass of A (as indicated below).

4. Apply the block rotations of step 3 to U and to $A^{(\nu)}$, forming $A^{(\nu+1)}$.

5. If the block off-diagonal mass of $A^{(\nu+1)}$ is not less than TOL times the block diagonal mass, then set $\nu := \nu + 1$ and go to step 1.

6. Diagonalize the diagonal blocks of $A^{(\nu)}$ (until the off-diagonal mass is less than TOL times the diagonal mass) and update U.

Step 3 of our BCJ implementation uses a single scalar Sameh sweep to reduce the off-diagonal mass of the two-block by two-block submatrix. This sweep includes $b(2b-1)$ pointwise rotations performed sequentially. Step 6 uses successive Sameh sweeps to diagonalize the block diagonals of $A^{(\nu)}$.

The BCJ algorithm is to be compared against the "block Brent-Luk" algorithm, which omits step 1 and replaces step 2 by selecting $m/2$ block index pairs according to the Brent-Luk ordering. A block Brent-Luk sweep also involves $(m^2-m)/2$ two-block by two-block rotations. It is important to note that the two methods under comparison differ only in their index selection methods. In figure 5 we also display timings for the Sameh ordering of the scalar Jacobi and the Eispack (TRED2/TQL2) methods.

4 Experimental Results

Several numerical experiments were conducted to compare the efficiency of BCJ and blocked Brent-Luk symmetric eigensolvers on matrices of random data. The test matrices were generated as matrices of uniform random deviates from (0,1]; in each case 10 tests were run to give non-parametric error bounds to within 10%. All computations were carried out with a tolerance of TOL $= 10^{-8}$. Table 1 summarizes the run times of the two methods on problems with various values of n, m, b, and k. Comparable average Eispack times from TRED2/TQL2 are 1.46, 0.26, and 0.05 seconds for $n = 64, 32, 16$, respectively. Comparable average times for the Sameh ordering of scalar Jacobi sweeps are 6.34, 0.79, and 0.11 seconds, for $n = 64, 32, 16$, respectively.

Figures 1–4 display iteration counts and relative efficiencies of the two algorithms. Figure 6 shows the average runtimes for the four methods with their optimal b values, for $n = 16, 32, 64, 128$. Table 2 shows the number of scalar rotations required by the Jacobi, BCJ, and BBL methods for the

		BCJ execution times			Blocked Brent-Luk times		
N	B	BCJ MIN	BCJ AVG	BCJ MAX	B-L MIN	B-L AVG	B-L MAX
64	2	26.04	26.93	28.68	44.96	79.39	136.12
64	4	22.11	25.48	29.98	30.02	63.13	146.09
64	8	31.48	39.97	66.47	38.19	145.70	424.32
64	16	58.36	107.90	253.24	97.38	242.68	513.49
32	2	3.60	4.09	4.71	3.70	8.04	15.39
32	4	3.71	5.20	7.66	5.34	9.14	19.00
32	8	5.91	13.50	23.14	9.41	16.55	31.36
16	2	0.51	0.72	0.83	0.45	2.47	14.23
16	4	0.52	1.40	7.11	0.62	1.04	2.36

Table 1: Multiflow Trace/7 (compiler version 1.5.3) execution times (sec.) for BCJ, blocked Brent-Luk, `TOL` $= 10^{-8}$, 10 trials.

N	B	Jacobi AVG	BBL AVG	BCJ AVG
16	2(4)	640	1381	5668
32	2	3072	4128	11242
64	4	14029	26691	91422
128	8	62669	147510	

Table 2: Multiflow Trace/7 (compiler version 1.5.4): Average number of scalar rotations for Jacobi, BCJ, and BBL, `TOL` $= 10^{-8}$, 10 trials.

same problems as in Figure 6. It is clear that the Eispack and scalar Jacobi methods yield superior speeds and that the execution time is dependent on the number of scalar rotations. Data for figures 1–4 and table 1 were computed using version 1.5.3 of the Multiflow Trace compiler; table 2 and figure 6 rely upon version 1.5.4.

These experiments show that the extra work of finding the largest independent off-diagonal blocks is offset by faster algorithmic convergence of BCJ, which makes the present method competitive with the blocked Brent-Luk technique. However, the block rotations require more scalar work than the scalar method; a block rotation seems to contribute less toward diagonalizing the matrix than an equal number of scalar rotations performed in the Sameh order.

5 Algorithmic Analysis

An important factor in determining the efficiency of the algorithm is the block size. BCJ has the following computational work per iteration ($n = mb$):

Step	Computations
1	$2n^2$
2	$O(m^2 \log m)$
3	$6K(2b)(2b-1)/2$
4	$18nK(2b)(2b-1)/2$
5	$2n^2$
6	$2n^2b + O(nb^2)$

The work for step 1 is actually completed in step 5, where the block masses are computed, so that after the first iteration step 1 contributes no work. Step 2 can be done in $O(Km^2)$ operations, which is an improvement if $K = o(\log m)$. Step 6 is performed once at the end of the calculation and has asymptotically negligible work if $b = o(n)$; for very large b step 6 dominates the total work.

A moderate value of b should be preferable in order to maximize the sum of off-diagonal block masses. Indeed, Figure 1 reflects this behavior. For small b, the overhead of determining the largest block exceeds the work of diagonalizing A. As b increases, the maximal off-diagonal squared block mass will approach the average block mass, reducing the effect of each block rotation, and consequently lengthening BCJ computations. Figure 3 shows that for several matrix sizes n, increasing the block size b increases monotonically the number of sweeps of BCJ, as expected. Furthermore, the example of Figure 5 indicates that with relatively few blocks, two large off-diagonal masses are likely to be dependent.

We now examine BCJ's behavior with a brief review of relevant order statistics theory [6,11], which describes the behavior of sorted random variates in terms of the probability distributions of the individual elements. Given independent and identically distributed random variables X_1, X_2, ..., X_N, the N order statistics $X^*_{1,N}$, $X^*_{2,N}$, ..., $X^*_{N,N}$ are the random variables associated with the lowest ranked to highest ranked X_i.

A particular instance of the theory is instructive with regard to BCJ, which starts with the $(n^2-n)/2$-sized upper triangular array from an $n \times n$

symmetric matrix $A^{(0)}$ of uniform random variates. Let Y_i be one of $(n^2 - n)/2$ *iid* uniform variates on the interval $(0, 1]$, and set $X_i = Y_i^2$. Then an average off-diagonal element of $A^{(0)}$ has squared mass $E[X_i] = 1/3$, while the maximum has mean squared mass $E[X^*_{(n^2-n)/2,(n^2-n)/2}] = 1 - 2/(n^2 - n+2)$. Selecting the maximum off-diagonal element, rather than an average element, increases the reduction to diagonal form of an individual rotation.

Assuming further that each X_i, $1 \leq i \leq (m^2 - m)/2$, is distributed as the sum of b^2 squares of uniform random values from the interval (0, 1], as is M_{ij} in the first step of our blocked experiments, it is clear that the central limit theorem applies to the block mass distributions. For large b, one may represent the off-diagonal squared block mass as a normal random variable with mean $\mu = b^2/3$ and variance $\sigma^2 = 4b^2/45$ (corresponding to the sum of b^2 uniform random variables).

The maximal order statistic for these large blocks tends toward a standard limiting distribution, from which we may determine the moments. Although the example employs sums of uniform variates, the proposition holds for any blocks that have asymptotically normal squared mass.

Proposition 1. *Let* $\{X_i\}_{i=1}^{(m^2-m)/2}$ *be iid normal variates with mean* μ *and variance* σ^2. *In the limit as* $m \to \infty$, *the expected largest variate is*

$$E[X^*_{(m^2-m)/2,(m^2-m)/2}] = \mu + \sigma\sqrt{2\log((m^2 - m)/2)} + O\left(\log\log m/\sqrt{\log m}\right) \tag{6}$$

and the variance is

$$Var[X^*_{(m^2-m)/2,(m^2-m)/2}] = \sigma^2 \frac{\Gamma''(1) - \Gamma'(1)^2}{2\log((m^2 - m)/2)} + O\left(\log\log m/(\log m)^2\right), \tag{7}$$

where $\Gamma'(\cdot)$ *and* $\Gamma''(\cdot)$ *are the first two derivatives of the gamma function, respectively. (Note that* $\Gamma''(1) - \Gamma'(1)^2 \approx 1.64$.*)*

Thus the expected largest squared mass is about $2\sqrt{\log m}$ standard deviations above the mean, with asymptoticlly vanishing variance. Cohen [5] has derived similar results for generalized matrix products. The proof of Proposition 1 is left to section 7.

BCJ operates by maximizing the mass of the selected off-diagonal blocks. This works well when the ratio $(\mu + 2\sigma\sqrt{\log m})/\mu$ is large while the additional cost to determine the largest block is low. Both cost and benefit decrease with increasing blocksize.

For certain values of b, BCJ inherits the fast convergence of the classical Jacobi method without paying a large cost for maximal selection. If b is chosen approximately $b = (\log n)^{1/3}$ and $K = m/2$, the work of computing and selecting the independent maximal blocks is $O\left(n^2(\log n)^{1/3}\right)$ per iteration, as is the rotation work, so that the two are of comparable sizes. For larger block sizes b, the block selection cost is asymptotically negligible. If b grows as $\sqrt{\log n}$, then the largest squared block mass is a constant multiple of the average squared block mass, while the extra cost of determining the maximal off-diagonal blocks is of smaller order.

Figure 1 clearly indicates the benefits of choosing a moderate blocksize, as the average solution time initially decreases as b grows. However, the use of a large b produces longer runtimes.

The selection of $K > 1$ maximal independent off-diagonal blocks (step 2) forms a more complex sum of conditional order statistics, which we now examine. Let X_i, $1 \le i \le M_1$, be *iid* random variates with density $f(x)$ and distribution $F(x)$. Denote by $X^{*1}_{M_1}$ the maximal order statistic. Now fix a particular subset of size M_2 of the remaining variates (excluding the selected maximum and others), and let $X^{*2}_{M_1,M_2}$ be the maximal variate in the subset. It is clear that $X^{*2}_{M_1,M_2} \le X^{*1}_{M_1}$. Inductively define $X^{*k+1}_{M_1,\ldots,M_{k+1}}$ from $X^{*k}_{M_1,\ldots,M_k}$ as the maximal order statistic in a chosen subset of M_{k+1} variates selected from the previous subset of size M_k (excluding the previous maximum and others). We call $X^{*k}_{M_1,\ldots,M_k}$ the k^{th} *conditional maximal order statistic* of the $\{X_i\}_{i=1}^{M_1}$.

Proposition 2. *For $M_1 > M_2 > \cdots > M_k > 0$, the probability distribution of the k^{th} conditional maximal order statistic is*

$$\Pr\left\{X^{*k}_{M_1,\ldots,M_k} \le x\right\} = \sum_{i=1}^{k} F(x)^{M_i} \prod_{\substack{j=1 \\ j\neq i}}^{k} \left(\frac{M_j}{M_j - M_i}\right). \tag{8}$$

Letting $\mu_{M_i} = E[X^*_{M_i}]$ be the unconditional mean of the maximum on

	2	4	10	6	7
		5	9	2	3
			3	4	4
				8	4
					4

Figure 5: Conditional maximum selection $X_{(1,4)} = X^{*1}_{15} = 10$, $X_{(2,3)} = X^{*2}_{15,6} = 5$,$X_{(5,6)} = X^{*3}_{15,6,1} = 4$.

M_i observations, we have

$$E[X^{*K}_{M_1,\ldots,M_K}] = \sum_{i=1}^{K} \mu_{M_i} \prod_{\substack{j=1 \\ j\neq i}}^{K} \left(\frac{M_j}{M_j - M_i} \right). \tag{9}$$

We briefly indicate the formulation of the first step of BCJ in terms of Proposition 2. In BCJ, K maximal off-diagonal blocks are selected in K stages from an $m \times m$ upper triangular array of $(m^2 - m)/2$ *iid* random variates. Independence of the selected locations requires striking out the row and column of the maximum. At stage i, $1 \leq i \leq K$, the maximal variate will be drawn from a subset of $\binom{m+2-2i}{2}$ blocks in the strict upper triangle of the array and then two rows and columns of the array will be struck out, corresponding to the row and column indices of the selected maximal element.

For instance, in the 6 by 6 example of Figure 5, the first maximum is 10 (row 1, column 4). Thereafter rows and columns 1 and 4 are struck from the array (to preserve independence) and the second conditional maximum is selected; it is 5 (row 2, column 3). Note that larger elements that are dependent upon the first maximum may be ignored in the selection of the second maximum. Finally, rows and columns 2 and 3 are struck from the array and the final maximum of 4 (row 5, column 6) is selected.

The selection of the K maximal independent off-diagonal blocks (which forms the more complex sum of conditional order statistics discussed above), determines on average a smaller sum of off-diagonal masses than K successive iterations choosing the single largest block. However, it is observed

in Figure 4 that the number of sweeps to convergence initially *declines* as K increases. This probably reflects the amortization of step 5 costs over additional blocks. As expected, BCJ requires slightly more iterations to converge as K reaches its upper limit of $m/2$ (*e.g.*, $b = 2, 4$).

Figure 2 presents in graphical form the ratios of the average BCJ and blocked Brent-Luk execution times on a Multiflow Trace/7 computer. The efficiency ratio shows the speedup of BCJ, with improvements up to a factor of 3.6 due entirely to improved index selection. Examination of Table 1 shows that, for almost all cases, BCJ runtimes have lower deviations from the mean.

Asymptotically, $b = \Omega\left((\log n)^{1/3}\right)$ guarantees that the work of selecting the maximal blocks will be at most comparable to the other arithmetic operations. However, assuming normality of initial data and intermediate results, the optimal b so that largest blocks are substantially larger than average ($\hat{b}^2 = O(\hat{b}\sqrt{\log m^2})$, from eq. (6)) is $b = O(\sqrt{\log n})$. For $b \approx (\log n)^{\alpha}$, $1/3 \leq \alpha \leq 1/2$, BCJ should be asymptotically faster than a blocked Brent-Luk method. The numerical experiments show speedups for problems of moderate size.

In general, the distribution of the elements of $A^{(\nu)}$ will be more complex than described here and the order statistic argument must be specialized to include the distributions of intermediate results, in order to rigorously prove rates of convergence. However, the improved performance of the new algorithm is consonant with the analysis performed here.

In cases where the matrix has few large elements or is close to diagonal, one expects BCJ to acheive shorter runtimes than indicated by these experiments on uniform random data. For instance, the method may prove useful in adaptive signal processing algorithms that rely on eigenvalue decompositions [14].

6 Conclusions

The improved index selection process of BCJ produces a substantial overall reduction in the program running time, compared to a blocked Brent-Luk algorithm. In particular, the extra work of determining the largest off-diagonal blocks is offset by fewer iterations needed for convergence; the overhead is negligible for large problems. Furthermore, because the algo-

rithm employs blocked data concepts, it is appropriate for computers with a hierarchical memory system. The concentration of work on the relatively small and numerous block elements is advantageous for parallelization of the algorithm. However, it seems that the block rotation strategy of BCJ and BBL reduces the convergence rate of both methods, leading to significantly longer run-times. A block rotation contributes less toward diagonalizing the matrix than an equal number of scalar rotations performed in the Sameh order.

The selection of parameters b and K is important to the efficiency of BCJ. A moderate value of b gives the lowest run times (though not the lowest number of block sweeps). The extra benefit of increasing K falls off rapidly for small n.

Nearly all stages of the algorithm are amenable to efficient parallel computation. Step 1 can be computed independently on m^2 processors; step 2 on various combinations of processors and interconnections; step 3 on K large-grained or Kb fine-grained processors, depending on whether the block rotation is parallelized or not; step 4 on up to bKn processors; step 5 on m^2 or more processors; and step 6 on K or more processors.

This investigation of BCJ was prompted by the use of a blocked Brent-Luk method in the Saxpy Computer Corp. mathematical subroutine library. It appears that a BCJ method could be more efficient than the BBL approach chosen there. It is possible that BCJ would show improved performance on more nearly diagonal or sparse matrix problems.

7 Calculation of Distributions of the Maximal and k^{th} Conditional Maximal Order Statistics

Proof of Proposition 1. David [6] presents the limiting distributional behavior of the maximal order statistic $X^*_{n,n}$, which depends upon the well-known distribution

$$\Lambda(x) = \exp(-e^{-x}) \qquad -\infty < x < \infty \tag{10}$$

in the case of *iid* 0-1 normal variates. We now carry through the analysis for general *iid* normal variates.

Let $\phi_{\mu\sigma^2}(x) = (\sigma\sqrt{2\pi})^{-1}\exp(-(x-\mu)^2/2\sigma^2)$ be the normal density function and let $\Phi_{\mu\sigma^2}(x) = \int_{-\infty}^{x}\phi_{\mu\sigma^2}(y)dy$ be the normal distribution function

corresponding to mean μ and variance σ^2, respectively. For large x,

$$\frac{1-\Phi_{\mu\sigma^2}(x)}{\phi_{\mu\sigma^2}(x)} \approx \frac{\sigma^2}{x-\mu}, \tag{11}$$

based on a change of variables from the case $\mu = 0$ and $\sigma = 1$. Thus Theorem 9.3.5 [6] applies and

$$\lim_{n\to\infty} \Pr\left\{(X^*_{n,n} - l_n)n\phi_{\mu\sigma^2}(l_n) \le x\right\} = \Lambda(x) \tag{12}$$

holds uniformly for every $x \in (-\infty, \infty)$, where l_n is selected so that $\Phi_{\mu\sigma^2}(l_n) = 1 - 1/n$.

According to (11),

$$\frac{1}{n} = 1 - \Phi_{\mu\sigma^2}(l_n) \approx \frac{\sigma^2}{l_n - \mu}\frac{1}{\sigma\sqrt{2\pi}}\exp\left(\frac{-(l_n-\mu)^2}{2\sigma^2}\right). \tag{13}$$

The asymptotic form of l_n is then

$$l_n = \mu + \sigma\left(\sqrt{2\log n} - \frac{(\log\log n + \log 4\pi)}{\sqrt{8\log n}}\right) + O(1/\log n). \tag{14}$$

Using the relation $n\phi_{\mu\sigma^2}(l_n) = (l_n - \mu)/\sigma^2 + O(l_n^{-1})$, we see that

$$\lim_{n\to\infty} \Pr\left\{(X^*_n - l_n)(l_n - \mu)/\sigma^2 \le x\right\} = \Lambda(x). \tag{15}$$

It follows directly from (15) that

$$E[X^*_n] = l_n + \frac{\Gamma'(1)\sigma^2}{l_n - \mu} + o(l_n^{-1}) \tag{16}$$

$$= \mu + \sigma\sqrt{2\log n} + O\left(\log\log n/\sqrt{\log n}\right) \tag{17}$$

where $\Gamma'(1)$ (Euler's constant) is the mean associated with $\Lambda(x)$. The variance vanishes asymptotically according to

$$Var[X^*_n] = \sigma^4\frac{\Gamma''(1) - \Gamma'(1)^2}{(l_n-\mu)^2} \tag{18}$$

$$= \sigma^2\frac{\Gamma''(1) - \Gamma'(1)^2}{2\log n} + O\left(\log\log n/(\log n)^2\right), \tag{19}$$

where $\Gamma''(1) - \Gamma'(1)^2$ is the variance associated with $\Lambda(x)$. (Here we have used $\Gamma'(\cdot)$ and $\Gamma''(\cdot)$ to represent the first two derivatives of the gamma function, respectively.) □

Proof of Proposition 2. Let $\left\{X^*_{M_k,M_k} = x \,|\, X_i \leq y\right\}$ denote the event that the maximal order statistic on M_k observations is x, conditioned on all observations X_i in the subset of size M_k being bounded above by y. Then the density of the k^{th} conditional maximal order statistic obeys the relation

$$\Pr\left\{X^{*k}_{M_1,\ldots,M_k} = x\right\} \tag{20}$$

$$= \int_x^\infty \Pr\left\{X^{*k-1}_{M_1,\ldots,M_{k-1}} = y\right\} \Pr\left\{X^*_{M_k,M_k} = x \,|\, X_i \leq y\right\} dy,$$

where

$$\Pr\left\{X^*_{M_k,M_k} = x \,|\, X_i \leq y\right\} = \begin{cases} \frac{d}{dx}\left(\frac{F(x)}{F(y)}\right)^{M_k}, & -\infty < x \leq y < \infty \\ 0, & -\infty < y < x < \infty \end{cases} \tag{21}$$

is the probability distribution of the maximal order statistic on M_k bounded observations.

Define $V_k(x) = \Pr\left\{X^{*k}_{M_1,\ldots,M_k} \leq x\right\}$. Then $V_1(x) = F(x)^{M_1}$. Inductively assuming that $V_k(x) = \sum_{i=1}^k a_{i,k} F(x)^{M_i}$ gives a recurrence relation on the a_{ik} of

$$a_{ik} = a_{i\,k-1}\frac{M_k}{M_k - M_i}, \qquad 1 \leq i < k, \tag{22}$$

and

$$a_{kk} = \sum_{i=1}^k a_{i\,k-1}\frac{M_i}{M_i - M_k}, \tag{23}$$

where $a_{11} = 1$. Consideration of the $k-1$ degree Lagrangian polynomial interpolating the points $(M_i, 1)$, $1 \leq i \leq k$, establishes that

$$a_{ik} = \prod_{\substack{j=1 \\ j\neq i}}^k \left(\frac{M_j}{M_j - M_i}\right). \tag{24}$$

The distribution is thus

$$\Pr\left\{X^{*k}_{M_1,\ldots,M_k} \leq x\right\} = \sum_{i=1}^k F(x)^{M_i} \prod_{\substack{j=1 \\ j\neq i}}^k \left(\frac{M_j}{M_j - M_i}\right). \quad \square \tag{25}$$

References

[1] C. Bischof. *Computing the Singular Value Decomposition on a Distributed System of Vector Processors.* Technical Report TR–87–869, Department of Computer Science, Cornell University, Ithaca, New York, September 1987.

[2] C. Bischof and C. Van Loan. Computing the singular value decomposition on a ring of array processors. In J. Cullum and R. Willoughby, editors, *Large Scale Eigenvalue Problems*, pages 51–66, Elsevier, 1986.

[3] K. A. Braun and Th. Lunde Johnsen. Hypermatrix generalization of the Jacobi and Eberlein method for computing eigenvalues and eigenvectors of Hermitian or non-Hermitian matrices. *Computer Methods in Applied Mechanics and Engineering*, 4:1–18, 1974.

[4] R.P. Brent and F.T. Luk. The solution of singular-value and symmetric eigenvalue problems on multiprocessor arrays. *SIAM J. Sci. Stat. Comput.*, 6(1):69–84, 1985.

[5] J. E. Cohen. Subadditivity, generalized products of random matrices and operations research. *SIAM Review*, 30(1):69–86, 1988.

[6] H.A. David. *Order Statistics.* J. Wiley and Sons, 2nd edition, 1981.

[7] P.J. Eberlein. On the diagonalization of complex symmetric matrices. *J. Inst. Math. Applic.*, 7:377–383, 1971.

[8] D.E. Foulser and R. Schreiber. The Saxpy Matrix-1: A general-purpose systolic computer. *IEEE Computer*, 20(7):35–43, 1987.

[9] G.H. Golub and C.F. Van Loan. *Matrix Computations.* Johns Hopkins Press, 1983.

[10] C.G.J. Jacobi. Uber ein leichtes verfahren die in der theorie der sacularstorungen vorkommendern gleichungen numerisch aufzulosen. *Crelle's Journal*, 30:51–94, 1846.

[11] S. Karlin and H. Taylor. *A Second Course in Stochastic Processes.* Academic Press, 1981.

[12] D.J. Kuck and A.H. Sameh. Parallel computation of eigenvalues of real matrices. In *Information Processing 1971*, pages 1266–1272, North-Holland, 1972.

[13] A.H. Sameh. On Jacobi and Jacobi-like algorithms for a parallel computer. *Math. Comp.*, 25:579–590, 1971.

[14] R.O. Schmidt. *A Signal Subspace Approach to Multiple Emitter Location and Spectral Estimation.* PhD thesis, Stanford University, November 1981.

[15] R. Schreiber and B. Parlett. Block reflector computation and applications. In R. Glowinski and J.L. Lions, editors, *Computing Methods in Applied Sciences and Engineering*, North Holland, 1986.

[16] C. Van Loan. The block Jacobi method for computing the singular value decomposition. In C. Byrnes and A. Lindquist, editors, *Computational and combinatorial methods in systems theory*, pages 245–256, North-Holland, 1986.

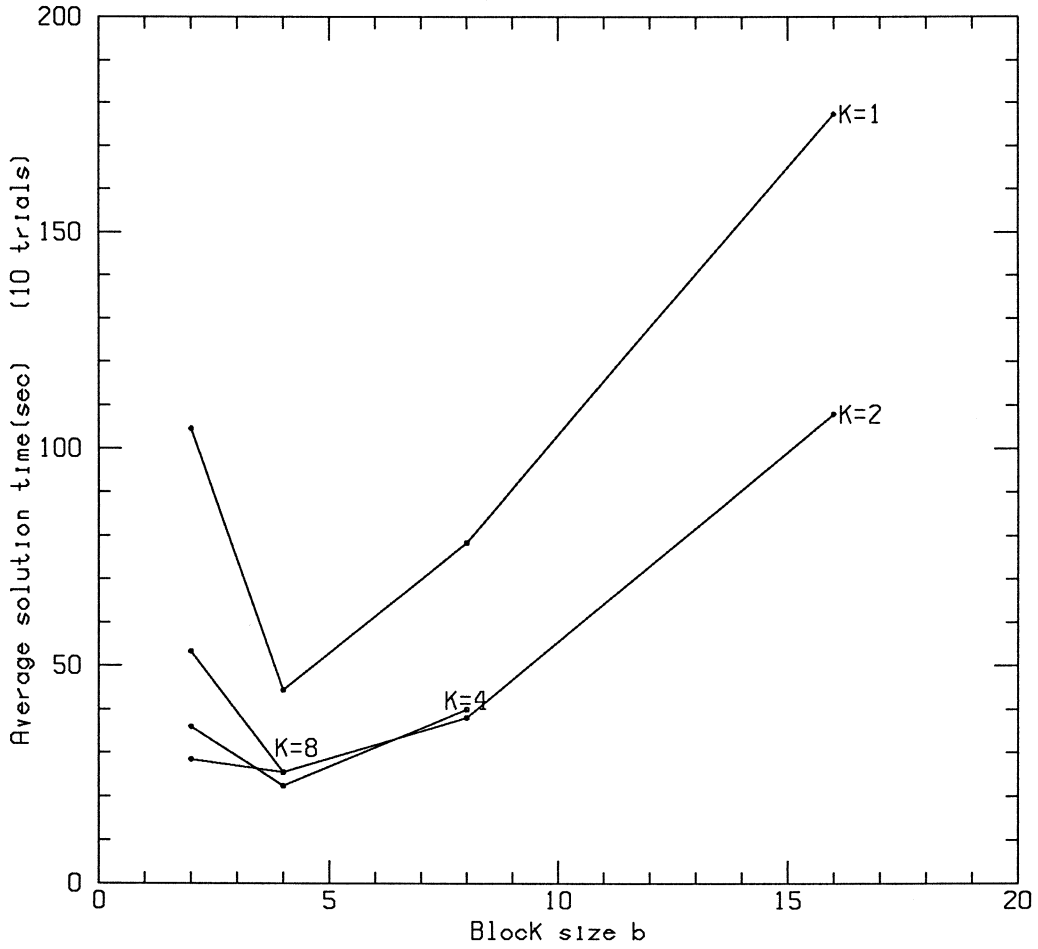

Figure 1. Average BCJ solution time for N=64, TØL = 10D-8
on a Multiflow Trace/7 computer

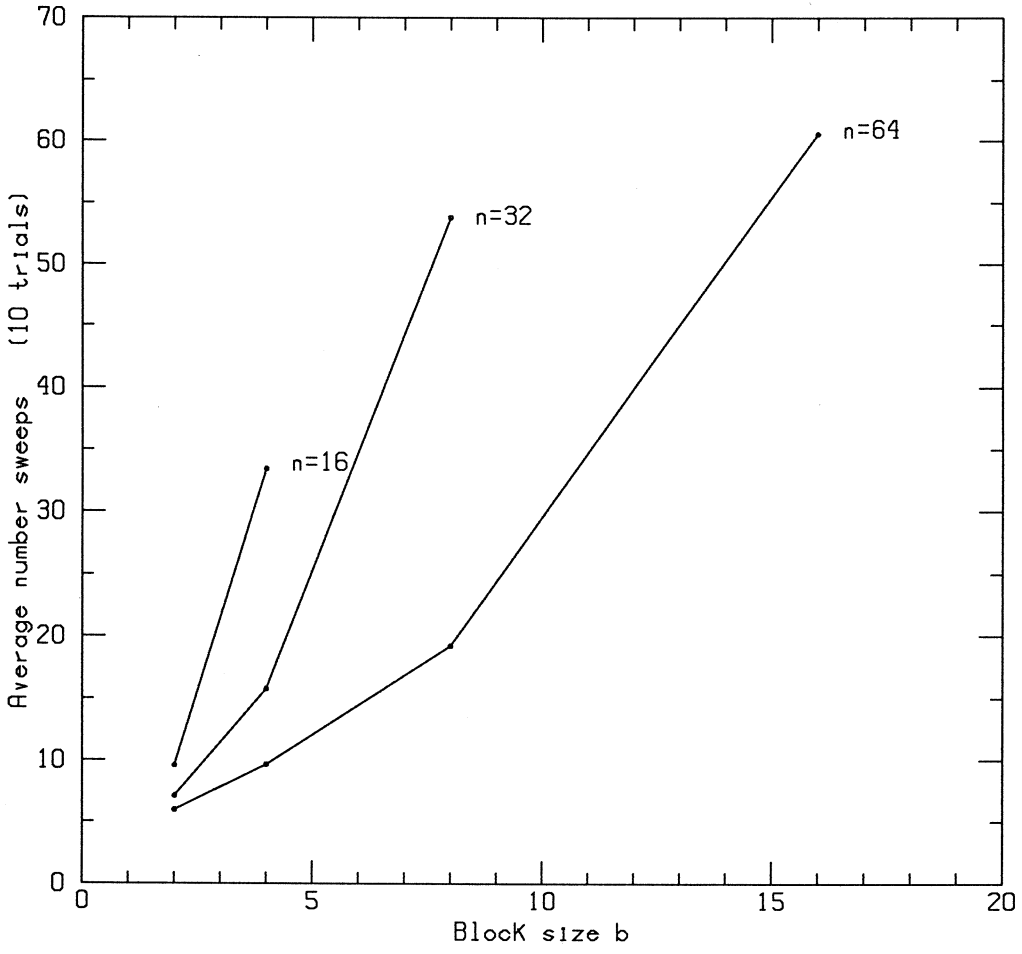

Figure 2. Average BCJ sweeps, TØL = 10D-8, K=n/2b

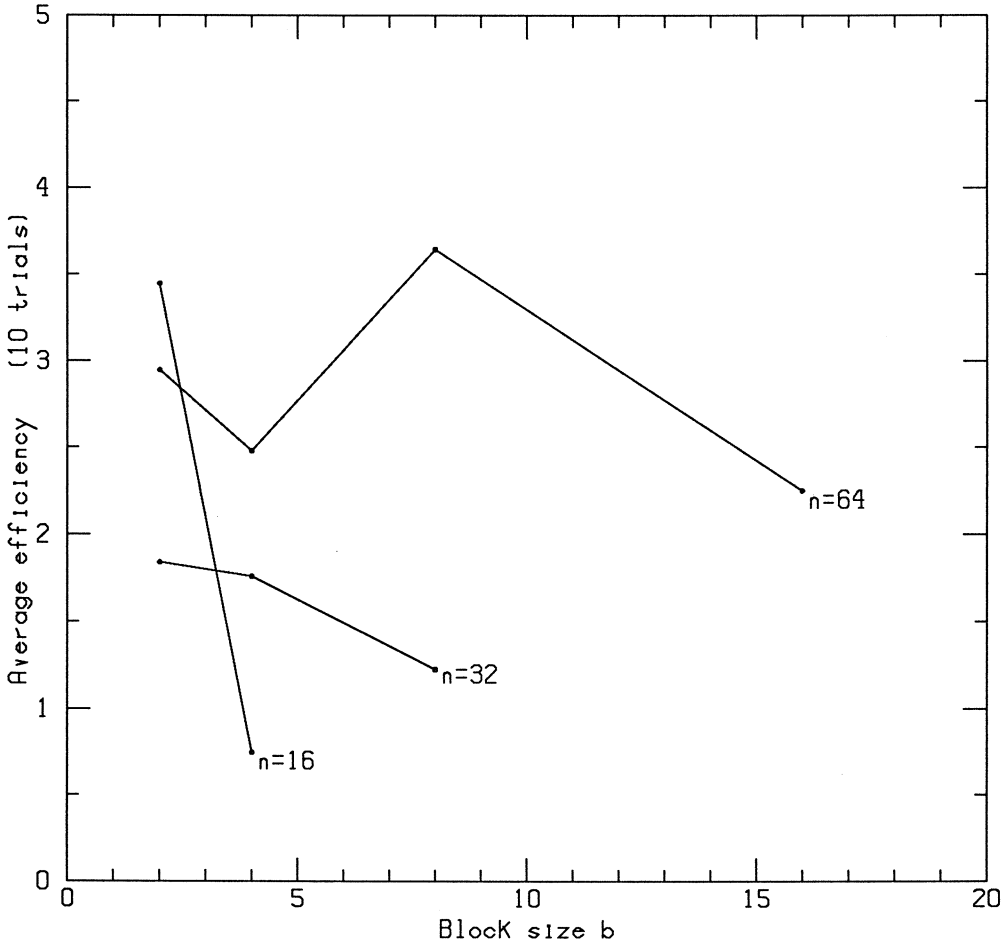

Figure 3. (Avg. Time Block Brent-Luk)/(Avg. Time BCJ)
TØL = 10D-8, K=n/2b

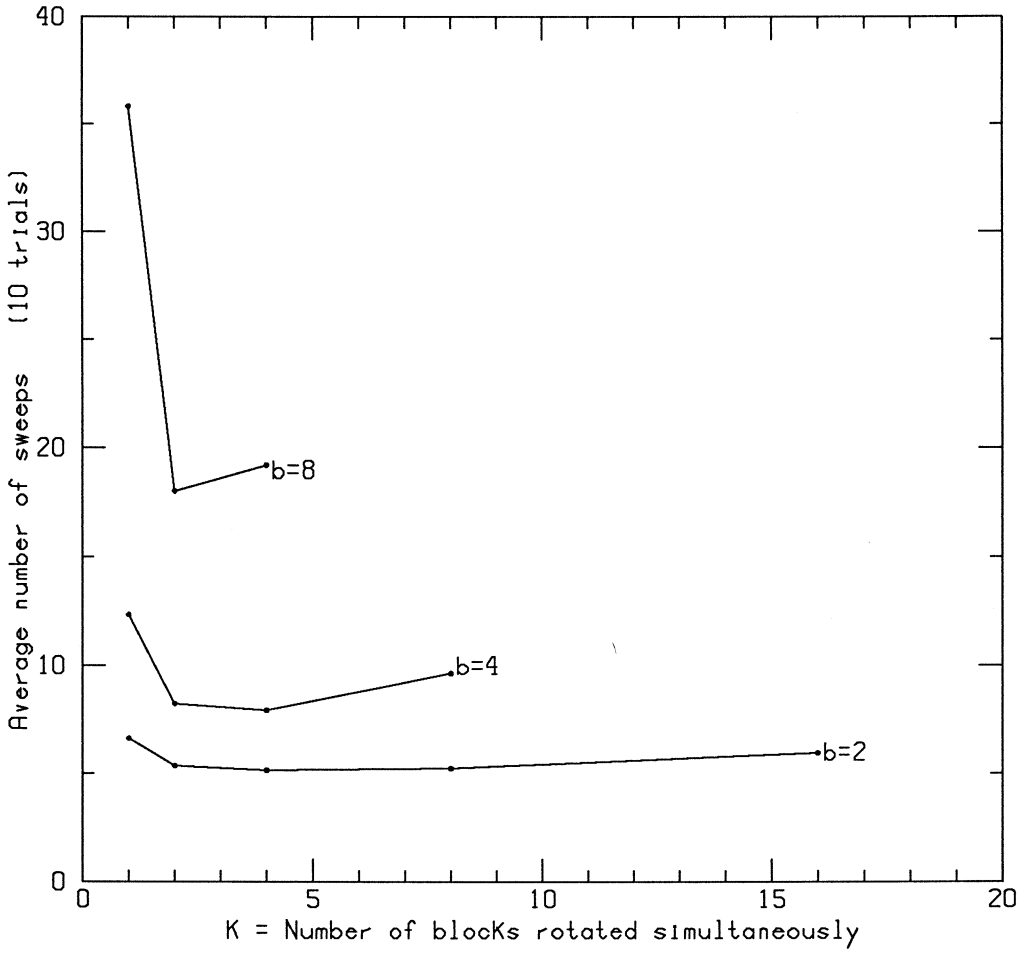

Figure 4. Average number of BCJ sweeps, n = 64, TØL = 10D-8

Figure 5 appears on page 11.

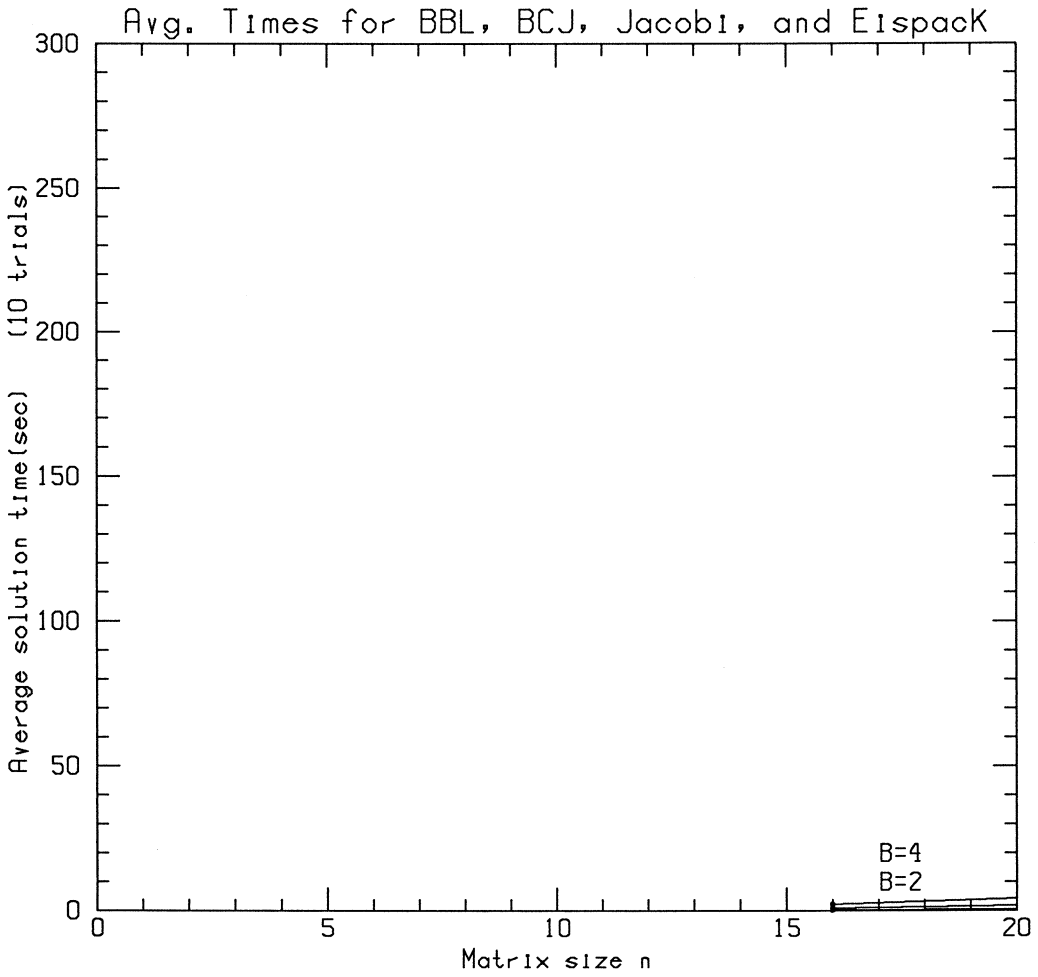

Figure 6. Average solution time for N=16..128, TØL = 10D-8
on a Multiflow Trace/7 computer