LBL-27821

Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA

To be presented at the Supercomputing "89" Conference,
Reno, NV, November 13–17, 1989

Macrotasking the Singular Value Decomposition of Block Circulant Matrices on the Cray-2

J.R. Baker

September 1989

Donner Laboratory
Biology & Medicine Division

LOAN COPY
Circulates
for 2 weeks
Bldg. 50 Library.
LBL-27821
Copy 2

Prepared for the U.S. Department of Energy under Contract Number DE-AC03-76SF00098.

DISCLAIMER

This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor the Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or the Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or the Regents of the University of California.

Macrotasking the Singular Value Decomposition of Block Circulant Matrices on the Cray-2

John R. Baker
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
and
Research Medicine and Radiation Biophysics Division
Lawrence Berkeley Laboratory

Abstract

A parallel algorithm to compute the singular value decomposition (SVD) of block circulant matrices on the Cray-2 is described. For a block circulant form described by M blocks with $m \times n$ elements in each block, the computation time using an SVD algorithm for general matrices has a lower bound $\Omega(M^3 \min(m,n)mn)$. Using a combination of fast Fourier transform (FFT) and SVD steps, the computation time for block circulant singular value decomposition (BCSVD) has a lower bound $\Omega(M \min(m,n)mn)$; a relative savings of $\sim M^2$. Memory usage bounds are reduced from $\Theta(M^2mn)$ to $\Theta(Mmn)$; a relative savings of $\sim M$. For $M = m = n = 64$, this decreases the computation time from approximately 12 hours to 30 seconds and memory usage is reduced from 768 megabytes to 12 megabytes. The BCSVD algorithm partitions well into n macrotasks with a granularity of $\Theta(mM \log M)$ for the FFT portion of the algorithm. The SVD portion of the algorithm partitions into M macrotasks with a granularity of $\Omega(\min(m,n)mn)$. Again, for the case where $M = m = n = 64$, the FFT granularity is $29ms$ and the SVD granularity is $428ms$. A speedup of 3.06 was achieved by using a prescheduled partitioning of tasks. The process creation overhead was $2.63ms$. Using a more elaborate self-scheduling method with four synchronizing server processes, a speedup of 3.25 was observed with four processors available. The server synchronization overhead was $0.32ms$. Relative memory overhead in both cases was about 4% for data space and 40% for code space.

1 Introduction

Singular value decomposition is a powerful technique used in image processing for singular value spectral analysis[1][2] of imaging systems and the solution of linear systems of equations using pseudo-inverses[3][4]. However, the computational complexity makes its use impractical for many problems where the linear dimension of the matrix is large.

A special class of matrices have the block circulant structure shown in equation 1. There are $M \times M$ blocks each of dimension $m \times n$. This form of matrix arises quite frequently when a function is invariant under rotation. As an example for the rest of the paper, the case where $M = m = n = 64$ shall be used.

$$A = \begin{bmatrix} A_0 & A_1 & A_2 & \cdot\cdot & A_{M-2} & A_{M-1} \\ A_{M-1} & A_0 & A_1 & \cdot\cdot & A_{M-3} & A_{M-2} \\ A_{M-2} & A_{M-1} & A_0 & \cdot\cdot & A_{M-4} & A_{M-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ A_2 & A_3 & A_4 & \cdot\cdot & A_0 & A_1 \\ A_1 & A_2 & A_3 & \cdot\cdot & A_{M-1} & A_0 \end{bmatrix} \quad (1)$$

A $\Theta(mnM \log M)$ [1] fast Fourier transform (FFT) technique[6][7] and a $\Omega(M \min(m,n)mn)$ singular value decomposition (SVD) algorithm are used to

[1] Let $n, n_0 \in \mathbf{N}$ and $\epsilon \in \mathbf{R}, \epsilon > 0$. Also, $f, g : \mathbf{N} \rightarrow \mathbf{R}$. Then, define[5]

1. Upper bound
$$O(f(n)) \equiv \{g(n) : g(n) \leq \epsilon f(n) \, \forall n > n_0\}$$
2. Lower bound
$$\Omega(f(n)) \equiv \{g(n) : g(n) \geq \epsilon f(n) \, \forall n > n_0\}$$
3. Combined bound
$$\Theta(f(n)) \equiv O(f(n)) \cap \Omega(f(n))$$
4. Asymptotic
$$f(n) \sim g(n) \Leftrightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 1$$

compute the factorization[8]

$$
\begin{aligned}
A &= (\mathcal{F}_M \otimes I_m)^\dagger \, D \, (\mathcal{F}_M \otimes I_n) && (2)\\
&= (\mathcal{F}_M \otimes I_m)^\dagger \, U_D S_D V_D^\dagger \, (\mathcal{F}_M \otimes I_n) && (3)\\
&= U S_D V^\dagger && (4)\\
&= U S V^\dagger && (5)
\end{aligned}
$$

where $\mathcal{F}_M$ is a normalized $M \times M$ discrete Fourier operator matrix, I_m is an $m \times m$ identity matrix, and I_n is an $n \times n$ identity matrix. U and V are unitary matrices whose columns are respectively the left and right singular vectors of A. S is a generalized diagonal matrix containing the singular values of A. The operator † is conjugate transpose and $\otimes$ is the outer product operation.

2 Implementation

Each of the mn discrete Fourier transforms of equation 2 can be computed independently; i.e., each sum does not need the result or input of another sum. However, the $\Theta(M \log M)$ grain size of this task is extremely small. For the example, it takes about $0.45ms$[9][10]. This is comparable to the $0.31ms$ necessary to synchronize with a server process and is much smaller than the $2.63ms$ necessary to create a new process. It is thus advantageous to increase the grain size of FFT tasks by computing m FFTs per task. The resulting granularity of $\Theta(mM \log M)$ is about $29ms$.

The SVD of the blocks of D also do not have input/output dependencies with other blocks and can be computed without explicit synchronization. The task granularity of this process is $\Omega(\min(m,n)mn)$ which is $428ms$ for the example problem.

2.1 Prescheduling

A prescheduled algorithm was implemented by creating one process for each of the n FFT tasks and another process for each of the M SVD tasks. The parent task starts k processes with either an FFT or an SVD task. As one of the k processes terminates another one is created so k processes are always available for execution. This method is very easy to implement because all synchronization is implicit in the fork and join like paradigm[11].

2.2 Self-scheduling

To overcome the process creation overhead, a self-scheduling algorithm was constructed[12]. This method is more complex than the prescheduled algorithm but has a smaller time overhead. It requires explicit synchronization between server processes and a task manager. k server processes are created and each waits for a start signal after initial setup of local state information. After receiving the start signal from the task manager, a server checks what part of the matrix it is to work on next. When finished the server sends a ready signal to the hibernating manager. The manager then reassigns each of the server processes until the task queue is empty.

3 Results

Figure 1 shows the computation time for different sizes of input matrices. The speedup of the algorithm, shown in figure 2, increases as the size of M, m, and n are increased. The prescheduled algorithm is faster for very small matrix sizes because the self-scheduled algorithm server processes have a larger startup overhead than a process started by the prescheduled algorithm. The self-scheduled algorithm is faster for medium sized problems that have small grain sizes but the prescheduled algorithm again approaches the speedup of self-scheduling as the problem size increases.

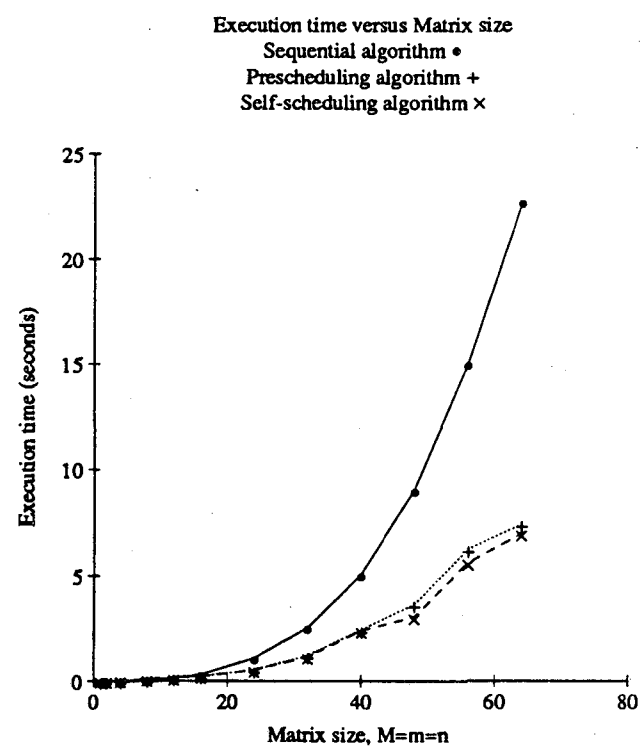

Figure 1: Computation time versus problem size with four tasks and four processors available to service tasks. Each point represents the average of eight trials.

The efficiency, shown in figure 3, does not approach unity as quickly as expected. This might be attributed to the timesharing scheduling algorithm used by the CTSS operating system and not to synchronization overhead because the overhead, shown in table 1, is less than 1.0% for M, m, and n larger than 64[13][14].

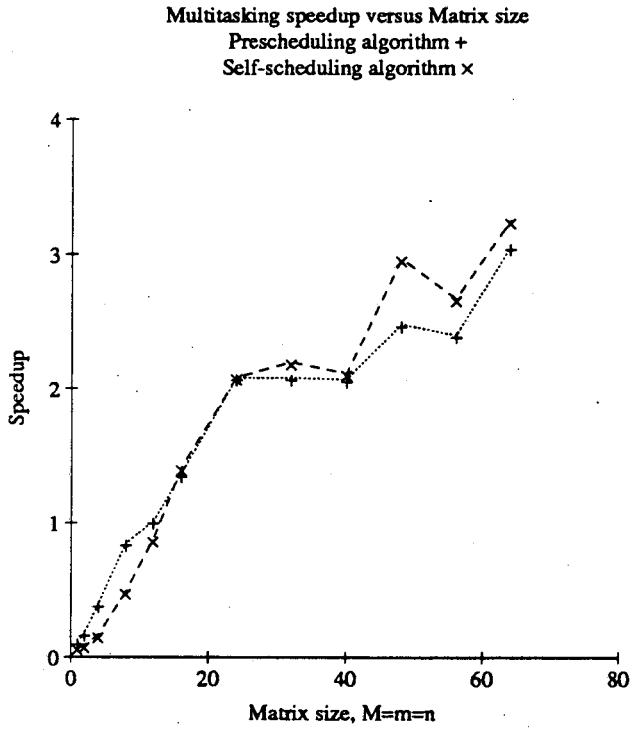

Figure 2: Speedup versus problem size with four tasks and four processors available to service tasks.

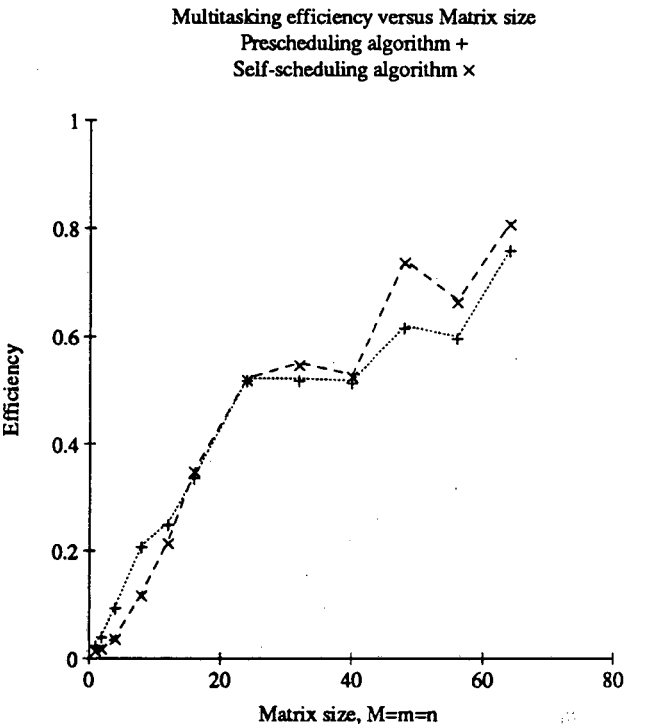

Figure 3: Efficiency versus problem size with four tasks and four processors available to service tasks.

It was not possible to verify this conjecture by using the machine without other users present.

The process creation time was found to be $2.63ms$. Task synchronization in the self-scheduling algorithm was $0.31ms$. A typical procedure call was measured to take $4.7\mu s$. Self-scheduling has less time overhead than prescheduling but is still 66 times more expensive than a procedure invocation.

Data memory usage and overhead is shown in table 2. Very little memory is necessary for the synchronization of tasks. Each of the processes needs some local working storage for computing FFTs and SVDs. Code memory usage and overhead is shown in table 3. The code space sharing was small due to a problem in the Fortran compiler that made code replication necessary.

Dynamic memory allocation costs are basically independent of the block size being allocated for small blocks. The cost depends almost entirely on the number of blocks being allocated. Each block takes approximately $0.68ms$ to allocate. The server processes of the self-scheduling algorithm avoid this overhead by reusing their local storage during each activation. The prescheduling algorithm originally allocated local storage blocks within each child process. This was deemed to be unsatisfactory and another parameter with working storage was passed to each child to avoid the overhead of dynamic memory allocation.

4 Discussion

The BCSVD algorithm provides orders of magnitude speedup by utilizing the circulant structure of matrices. A further speedup was obtained using macrotasking. This does not reduce central processing unit charges because time on all processors is billed to the job[14]. However, a substantial savings in memory charges is achieved because the program memory residency time is reduced by the multiprocessor speedup[15][16]. For typical problems M, m, and n are approximately 256. This requires approximately 800 megabytes of memory which can be quite costly to use.

Self-scheduling is useful when the task granularity is small. As the task granularity increases, prescheduling overhead becomes less important. Prescheduling is much easier to implement and debug. There are no explicit synchronizations to consider since the operating system handles the process allocation and scheduling. The parent only has to wait for the operating system to signal that the child has finished. Self-scheduling needs explicit synchronization with the server tasks and is therefore more difficult to implement and debug with the tools available.

The Fortran compiler used does not allocate lo-

$M = m = n$	prescheduled		self-scheduled	
	overhead (ms)	% overhead	overhead (ms)	% overhead
4	25.7		29.9	
8	46.8	42.77	32.3	5.09
16	80.2	10.89	37.3	1.29
32	155.7	2.66	47.2	0.32
64	306.8	0.61	67.1	0.07

Table 1: Synchronization overhead versus problem size.

$M = m = n$	sequential	prescheduled		self-scheduled	
	usage (KB)	usage (KB)	% overhead	usage (KB)	% overhead
8	112	409	265.2	475	324.1
16	240	533	130.4	604	151.7
32	1648	1946	18.1	1948	18.2
64	12400	12698	2.4	12888	3.9

Table 2: Data memory usage and overhead versus problem size with four tasks.

cal variables on the stack properly. It puts some local variables into static storage. Thus, code sharing is not possible for Fortran subroutines. Each process must have a separate copy of the code and local data space. This was done by creating copies of the subroutines and giving each copy a unique name space by appending the process number to the name of the subroutine and all of its descendants.

5 Conclusions

The orthogonality properties of multidimensional fast Fourier transforms (FFT) allows the FFT portion of the block circulant singular value decomposition (BCSVD) algorithm to partition into n macrotasks. Each singular value decomposition (SVD) of the blocks of the reduced matrix can be computed independently using M macrotasks. For an $M = m = n = 64$ example, a speedup of 3.06 was achieved for prescheduling and for self-scheduling a speedup of 3.25 was observed using four processors on the Cray-2. Relative time overhead was 0.5% for the prescheduled algorithm and 0.07% for the self-scheduled algorithm. Relative memory overhead was 4% for both cases. The prescheduled algorithm is satisfactory for most problems because M, m, and n are large; thus, the task granularity will be large when compared to the synchronization overhead.

Multitasking the block circulant singular value decomposition algorithm decreases overall computation costs by reducing the time large sections of memory are in use. Little or no gain comes from reduced central processing unit charges since processing time on all processors is charged to a job.

6 Acknowledgements

The author would like to thank TF Budinger, RH Huesman, MR Kessler, R Marr, MS Roos, EM Salmeron, and STS Wong for helpful discussions and the consulting staff at NMFECC for answering *many* programming questions. This work was supported by the Office of Energy Research, Office of Health and Environmental Research, of the U.S. Department of Energy under contract No DE-AC03-76SF00098, by the National Institute of Health, National Heart, Lung, and Blood Institute under grants No HL07367 and HL25840, National Institute of Aging under grant AG05890, National Cancer Institute under grant CA38086, by Cray Research Inc., and by IBM Inc.

7 References

[1] Golub GH and CF Van Loan. *Matrix Computations.* Volume 3 of *Johns Hopkins Series in the Mathematical Sciences*, Johns Hopkins University Press, Baltimore, MD, 1983.

[2] Rice JR. *Numerical Methods, Software, and Analysis.* McGraw-Hill, New York, NY, 1983.

[3] Andrews HC and BC Hunt. *Digital Image Restoration.* Prentice-Hall, Englewood Cliffs, NJ, 1977.

[4] Strang G. *Linear Algebra and Its Applications.* Academic Press, Orlando, FL, 1980.

[5] Knuth DE. *The Art of Computer Programming.* Volume 1, Addison Wesley, Reading, MA, second edition, 1981.

k	sequential usage (KB)	prescheduled usage (KB)	prescheduled % overhead	self-scheduled usage (KB)	self-scheduled % overhead
1	404	450	11.4	447	10.6
2	404	489	21.0	492	21.8
3	404	530	31.1	537	32.9
4	404	570	41.1	582	44.1

Table 3: Code memory usage and overhead versus number of active tasks.

[6] Brigham EO. *The Fast Fourier Transform.* Prentice-Hall, Englewood Cliffs, NJ, 1974.

[7] Bracewell RN. Fourier techniques in two dimensions. In Price JR, editor, *Fourier Techniques and Applications*, pages 45–71, Plenum Press, New York, 1985.

[8] Baker JR, RH Huesman, and TF Budinger. *Singular Value Decomposition of Block Circulant Matrices.* Technical Report LBL-27697, LBL, 1989.

[9] Buneman O. Vector FFT for the Cray-2. *NMFECC Buffer*, **10**(11):10–11, 1986.

[10] Despain AM. Very fast Fourier transform algorithms for hardware implementation. *IEEE Transactions on Computers*, **C-28**(5):333–341, 1979.

[11] Mundie DA and DA Fisher. Parallel processing in Ada. *Computer*, **19**(8):20–25, 1986.

[12] Quinn MJ. *Designing Efficient Algorithms for Parallel Computers.* McGraw-Hill, New York, NY, 1987.

[13] Mirin A. Parallelization of a 3-D MHD code, part I: Methodology and results. *NMFECC Buffer*, **11**(7):14–16, 1987.

[14] Mirin A. Parallelization of a 3-D MHD code, part II: Analysis of multiprocessing efficiency on the Cray-2. *NMFECC Buffer*, **11**(8):11–13, 1987.

[15] Patton PC. Multiprocessors: Architectures and applications. *Computer*, **18**(6):29–40, 1985.

[16] Gelernter D. Domesticating parallelism. *Computer*, **19**(8):12–16, 1986.

LAWRENCE BERKELEY LABORATORY
TECHNICAL INFORMATION DEPARTMENT
1 CYCLOTRON ROAD
BERKELEY, CALIFORNIA 94720