Technical note

Engineering and Scientific Subroutine Library Release 3 for IBM ES/3090 vector multiprocessors

by R. C. Agarwal
F. G. Gustavson
J. McComb
S. Schmidt

This technical note should be read in conjunction with the paper by McComb and Schmidt[1] which describes the Engineering and Scientific Subroutine Library through Release 2. In this technical note, which is an addendum to that paper, we briefly describe some of the new features in Release 3 and indicate some of the techniques used to optimize vector and parallel performance.

The paper by McComb and Schmidt[1] previously published in the *IBM Systems Journal* described the contents and performance features of the Engineering and Scientific Subroutine Library (ESSL) Release 2. In this technical note, we describe several new features in ESSL Release 3 and discuss some of the techniques that were used to optimize vector and parallel performance.

The following new features are provided in Release 3:

- Forty-seven new subroutines, including six parallel processing subroutines
- Seventy-one subroutines modified to improve performance, to add capability, or for migration purposes
- Calling sequence validation using the VS FORTRAN Intercompilation Analysis (ICA)[2-4] feature

The new subroutines include:

- A select group provided for parallel processing using either the VS FORTRAN Multitasking Facility (MTF)[3,4] or Parallel FORTRAN,[5,6] including matrix multiplication, general and positive definite symmetric matrix factorization, and one-, two-, and three-dimensional complex Fourier transforms
- In-place and out-of-place general matrix transpose
- Complex general matrix factorization and solve
- Positive definite symmetric matrix inverse, condition number reciprocal, and determinant
- Real triangular matrix solve (proposed Level 3 BLAS)[7]
- General sparse matrix factorization and solve by direct methods
- General sparse matrix iterative solve
- Extreme eigenvalues and corresponding eigenvectors of real symmetric and complex Hermitian matrices
- Eigenvalues and eigenvectors for both generalized real and generalized real symmetric eigensystems
- Three-dimensional complex, real-to-complex, and complex-to-real Fourier transforms
- Long precision direct method convolution or correlation with decimated output
- Two-dimensional cubic spline interpolation
- Two-dimensional Gauss-Legendre quadrature
- Normally distributed random number generators
- Utility to determine the stride value for optimal performance in certain Fourier transform subroutines
- Utility to set the vector section size of the scalar library

Parallel processing subroutines

IBM ES/3090™ systems presently consist of up to six processors. Each processor has its own cache memory. If data in cache is repeatedly referenced, it becomes cache resident, resulting in a very good performance. For parallel processing, VS FORTRAN Version 2 provides the Multitasking Facility (MTF),[3,4] which uses DSPTCH (a forking routine) and SYNCRO (a joining routine) subroutines. The DSPTCH subroutine creates a subtask by calling a parallel subroutine with its set of parameters. Typically, several subtasks are created with each subtask assigned a part of the data. The operating system assigns these subtasks to available processors. These subtasks work asynchronously; therefore, it is important that they share only "read only" data. All "write" data sets must be disjoint. Each processor has its own cache, and the shared data can reside in "read only" mode on several processors. The SYNCRO subroutine ensures that all subtasks complete at each stage before proceeding further with the next parallel phase of the computation. Parallel FORTRAN[5,6] also provides a similar capability. ESSL parallel routines work in both environments.

In ESSL parallel subroutines, we attempt to multitask the problem at the highest possible level. If this attempt fails, we multitask at the next lower level such that a fairly large amount of computing is done in each subtask. Such an arrangement makes the multitasking overhead small compared to the computational workload. One of our aims is to not significantly increase the total CPU time compared to a uniprocessor time. Attaining this goal requires maintaining the performance obtainable from our vectorized cache-based blocking algorithms. This granularity determines the minimum subtask size to obtain good performance. Thus, even if all the processors are not actually available for the task (often the case in a multiuser environment), multitasking does not significantly increase the total number of processor cycles. Notwithstanding, when the designated number of processors are actually available, we want an elapsed-time speedup to be close to the number of processors available. Meeting this condition requires careful load balancing between each synchronization step and eliminating or minimizing the serial computation. In ESSL, different phases of computation are often overlapped to achieve load balancing. To put these ideas into perspective, we briefly describe how we have used them to implement our uniprocessor and parallel routines for matrix multiplication, general LU factorization, and three-dimensional fast Fourier transform (FFT).

The vector algorithms for matrix multiplication, $C = AB$, and general $A = LU$ factorization are carefully tuned to take advantage of the memory hierarchy of the IBM 3090 system, especially its cache.[8] For matrix multiplication, DGEMUL, we get cache data reuse by blocking the A matrix into submatrices which fit into cache and by using a matrix update routine, an example of which is the level 3 BLAS, DGEMM.[7] The matrix update routine is used repeatedly to do the entire computation. For matrix factorization, DGEF, we use a recursive block factorization and update algorithm. For large matrices, almost all of the computation in both of these routines is done during the update phase. The update phase runs at close to the peak performance of the machine. This ensures very good overall performance for both of these ESSL vectorized routines, DGEMUL and DGEF.

These two vectorized routines have also been parallelized. The main feature in the parallel matrix multiplication routine, DGEMLP, is the division of the C array into P equal areas that are suitable to exploit the matrix update algorithm. We then assign a processor to each of the P equal pieces of C and call DGEMUL for each piece. This strategy balances the load very well. In the parallel matrix factorization routine, DGEFP, we initially, and outside of the parallel structure, factor a certain number of columns. Then we initiate a recursive parallel computation. On each parallel iteration there is a block update step and the next block factorization step. We estimate the total workload for this computation and try to divide it equally among the available processors. The main task updates a certain number of columns and factors the next block. Since the processes are asynchronous, the main task must update at least those columns which are to be factored next. The remaining columns of the update step are split equally among the remaining processors. Any extra columns assigned to the main task are chosen to achieve load balancing. In principle, this gives close to 100 percent efficiency for large problems.

We give the performance of the parallel routines DGEMLP and DGEFP in Tables 1 and 2. Measurements were made on the IBM ES/3090 model 600S, under Multiple Virtual Storage System Product (MVS/SP™) in a dedicated environment. All measured times are in seconds and are rounded. The third column of each table gives the performance of the corresponding vector uniprocessor routines, DGEMUL and DGEF. For the DGEMUL routine, the performance quickly approaches the asymptotic value of 104 MFLOPS (mil-

lion floating point operations per second), which is 91 percent of the possible peak rate of 114 MFLOPS. For the DGEF routine, as long as the matrix fits in main memory, the performance keeps improving as the matrix size increases. This performance should eventually come close to the matrix multiply performance. For the matrix factorization, the speedup factor improves as the problem gets larger and eventually comes very close to the number of processors used. For the parallel matrix multiplication routine, the speedup is very close to the number of processors used and is achieved even for a small problem. For the matrix factorization routine, for the size 100 problem, we could not efficiently use more than three processors; therefore, internally the number of processors used was limited to three. In summary, for two important linear algebra kernels, we have obtained close to the effective peak rate of the new model ES/3090 600S.

Three-dimensional FFT routines have been provided in ESSL Release 3. In performing multidimensional Fourier transforms, data are accessed with large strides. Because of the memory hierarchy of the 3090, it is very important that these strides are chosen such that blocks of data fit comfortably in various levels of memory. ESSL FFT routines do appropriate blocking of data and, where necessary, internal transpositions so that blocks of data fit in memory. In spite of this, certain strides, for example, powers of two, degrade performance. To help users in selecting good strides, a utility routine, STRIDE, that recommends good strides for any given size problem has been provided.

A parallel three-dimensional FFT routine, SCFT3P, has also been provided.[9] This routine is functionally equivalent to the uniprocessor SCFT3 (three-dimensional complex-to-complex FFT routine) and uses multiple processors when available to speed up the computation. In both of these routines, the first two dimensions are transformed one plane at a time, and the total number of planes is divided among all of the processors. At the end of this step, the Fourier transform is computed along the third dimension. This step is also divided among the available processors.

In Table 3, we give the performance of SCFT3 and SCFT3P. Measurements were made on the IBM ES/3090 model 600S under MVS/SP in a dedicated environment. The timings include the initialization times. The routine was run for three-dimensional arrays of size N, and $L = N^3$ is the total number of points processed. The array was dimensioned for optimal performance, using the ESSL STRIDE routine. All times are actual "wall clock" (elapsed) times. The fourth column gives the uniprocessor time (for SCFT3) normalized by $L \log_2(L)$, in microseconds. These numbers are quite uniform indicating that uniformly good performance can be obtained for large problems in spite of the large strides. The last three columns give the elapsed time speedup of the parallel routine SCFT3P over SCFT3, using two, four, and six processors.

Table 1 General matrix multiplication performance

			P-way Speedup			
		MFLOPS				MFLOPS
N	*T*	1-way	2	4	6	6-way
100	0.022	89.7	1.93	3.63	5.02	448
200	0.167	101.8	1.97	3.85	5.53	565
500	2.400	104.1	1.98	3.92	5.75	599
1000	19.150	104.4	2.00	3.97	5.91	617
1500	64.600	104.5	1.99	3.98	5.91	617
2000	153.000	104.6	2.00	3.99	5.94	621

Table 2 General matrix factorization performance

			P-way Speedup			
		MFLOPS				MFLOPS
N	*T*	1-way	2	4	6	6-way
100	0.0145	45.9	1.68	1.86	1.85	85
200	0.0787	67.8	1.90	3.08	3.40	230
500	0.9840	84.7	2.00	3.95	5.64	478
1000	7.1800	92.8	2.00	4.00	5.93	551
2000	54.3200	98.2	2.00	3.98	5.94	583
3000	179.7000	100.1	2.00	3.99	5.96	597

Table 3 Three-dimensional FFT performance

			T	*P*-way Parallel Speedup *P*		
N	Size $L = N^3$	*T* (s)	$L*\log_2(L)$ (μs)	2	4	6
60	216000	0.30	0.078	1.95	3.79	5.53
84	592704	0.89	0.078	1.97	3.91	5.66
128	2097152	3.65	0.083	1.99	3.95	5.71
180	5832000	10.9	0.083	1.99	3.98	5.86
240	13824000	28.8	0.088	1.97	3.93	5.88
256	16777216	38.9	0.097	1.99	3.91	5.69
320	32768000	77.0	0.094	1.99	3.93	5.75
360	46656000	101.6	0.086	1.99	3.91	5.65

N is the matrix order or transform length.
T is the elapsed time of the ESSL nonparallel version of the subroutine in seconds.
P is the number of processors used for parallel processing.

Table 4 Performance of some Release 3 vector subroutines

DGSF—Sparse Matrix Factorization (Sparse Matrix Stored by Indices and Rows)

N	*PNZ*	*T*(S)	*T*(E) (indices)	*T*(S)/*T*(E) (indices)	*T*(E) (rows)	*T*(S)/*T*(E) (rows)
1442	0.9	2.127	0.625	3.40	0.607	3.50
3606	0.4	0.781	0.504	1.55	0.418	1.87
4422	0.2	0.776	0.353	2.20	0.290	2.68
4482	0.1	0.171	0.063	2.71	0.037	4.62
7020	0.2	1.511	0.707	2.14	0.568	2.66

DTRSM—Triangular Solve (Lower Triangular Matrix on Left Side)

M	*N*	*T*(S)	*T*(E)	*T*(S)/*T*(E)
100	100	0.1116	0.0222	5.03
500	500	14.2825	1.6760	8.52
1000	1000	114.8365	11.8617	9.68

DSPSV—Largest (10%) Eigenvalues & Eigenvectors (Symmetric Matrix in Upper Packed Storage Mode)

N	*M*	*T*(S)	*T*(E)	*T*(S)/*T*(E)
100	10	0.23901	0.09362	2.55
300	30	5.19810	1.27213	4.09
500	50	22.87726	5.04824	4.53

N = Order of the matrix or the number of columns in solution matrix B
M = Number of eigenvalues and eigenvectors computed or the number of rows in solution matrix B
PNZ = Percentage of nonzero elements of the matrix
T(S) = IBM ES/3090 CPU time in seconds for scalar subroutine
T(E) = IBM ES/3090-VF CPU time in seconds for ESSL subroutine

Vector subroutines

In addition to the parallel processing subroutines, several new vector subroutines were added in Release 3. Among others, a direct method for sparse matrix factorization—subroutine DGSF,[10] a triangular solve—DTRSM, and an eigenvalue-eigenvector code—DSPSV are included. DGSF has two main components: a fast scalar factorization and a vectorized factorization designed mainly for dense matrices.[11] The scalar LU-factorization uses a simple algorithm to identify triangular factors by permutations. The remaining nucleus is factorized by Gaussian elimination using threshold pivoting with an approximate Markowitz count criteria. Several improvements to other known techniques contribute to the speed of DGSF during the search for a pivot.[12] DGSF proceeds with scalar processing if the active submatrix is reasonably sparse and switches over to dense vectorized processing if the density of the active submatrix reaches a certain threshold value. The design of the vectorized part is described in the paper by Suhl, Aittoniemi, and Su.[13] DTRSM is a proposed Level 3 BLAS[7] which solves triangular systems of equations with multiple right-hand sides. DSPSV computes the extreme eigenvalues and eigenvectors of a real symmetric matrix by the rational QR method with Newton corrections. Performance information for these subroutines is given in Table 4.

Acknowledgments

The authors would like to acknowledge the contributions of additional developers, U. Suhl and L. Aittoniemi, for ESSL Release 3.

Cited references and notes

1. J. McComb and S. Schmidt, "Engineering and Scientific Subroutine Library for the IBM 3090 Vector Facility," *IBM Systems Journal* 27, No. 4, 404–415 (1988).
2. The VS FORTRAN Intercompilation Analysis (ICA) feature allows the diagnosis of inconsistencies in call sequence arguments passed to ESSL subroutines at compile time. These inconsistencies can be corrected before running the program, thus saving time in debugging large, complex FORTRAN programs.

3. *VS FORTRAN Version 2 Language and Library Reference*, SC26-4221, IBM Corporation (1988); available through IBM branch offices.
4. *VS FORTRAN Version 2 Programming Guide*, SC26-4222, IBM Corporation (1988); available through IBM branch offices.
5. *Parallel FORTRAN Language and Library Reference*, SC23-0431, IBM Corporation (1988); available through IBM branch offices.
6. L. J. Toomey, E. C. Plachy, R. G. Scarborough, R. J. Sahulka, J. F. Shaw, and A. W. Shannon, "IBM Parallel FORTRAN," *IBM Systems Journal* **27**, No. 4, 416–435 (1988).
7. J. J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling, "A set of Level 3 basic linear algebra subprograms," *Technical Memorandum 88*, Argonne National Laboratory, Argonne, IL (May 1988).
8. R. C. Agarwal and F. G. Gustavson, "A parallel implementation of matrix multiplication and LU factorization on IBM 3090," *IFIP WG 2.5 Working Conference 5*, held at Stanford University (August 22–26, 1988). Published in *Aspects of Computation on Asynchronous Processors*, M. H. Wright, Editor, Elsevier Science Publishers B.V. (North-Holland), Amsterdam (1989).
9. R. C. Agarwal, "A vector and parallel implementation of the FFT algorithm on IBM 3090," *IFIP WG 2.5 Working Conference 5*, held at Stanford University (August 22–26, 1988). Published in *Aspects of Computation on Asynchronous Processors*, M. H. Wright, Editor, Elsevier Science Publishers B.V. (North-Holland), Amsterdam (1989).
10. The DGSF comparison scalar code used in Table 4 is MA28. MA28 is described in: I. S. Duff, *MA28—A Set of Fortran Subroutines for Sparse Unsymmetric Linear Equations*, AERE Harwell Report, R.8730 (1977).
11. F. G. Gustavson, "Aspects of dense/sparse LU-factorizations for vector processing," *SIAM Conference on Sparse Matrices*, Gleneden Beach, Oregon (May 1989).
12. U. H. Suhl and L. Aittoniemi, *Computing Sparse LU-Factorizations for Large-Scale Linear Programming Bases*, Working Paper 58/87, Freie Universität Berlin, FB 10, WE 6.
13. U. H. Suhl, L. Aittoniemi, and J. Su, "Computing LU-factorizations for large sparse general matrices," *SIAM Conference on Sparse Matrices*, Gleneden Beach, Oregon (May 1989).

Ramesh C. Agarwal *IBM Research Division, T. J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Agarwal received a B.Tech. (Hons.) degree from the Indian Institute of Technology (IIT) Bombay, India, and the M.S. and Ph.D. degrees from Rice University, Houston, all in electrical engineering in 1968, 1970, and 1974, respectively. During 1971–72, he was an associate lecturer at the School of Radar Studies, IIT Delhi, India, and from 1974 to 1977, he was with the T. J. Watson Research Center. From 1977 to 1981 he was a principal scientific officer at the Center for Applied Research in Electronics at IIT Delhi, and then he returned to IBM in 1982. His research interests have included network synthesis, information theory and coding, number theoretic transforms, fast algorithms for computing convolution and DFT, application of digital signal processing to structure refinement of large biological molecules using X-ray diffraction data, sonar signal processing, architecture for special-purpose signal processors, digital DTMF/MF receivers, filter structures, analysis of the Kennedy assassination tapes, computation of elementary functions, and vectorization and parallelization of engineering/scientific computation. Dr. Agarwal received the President's Gold Medal from IIT Bombay in 1968, the Best Ph.D. Thesis Award from Sigma Xi Society of Rice University in 1974, the 1974 IEEE ASSP Senior Award for papers on number theoretic transforms, an IBM Outstanding Contribution Award in 1979 for work on cyrstallographic refinement of biological molecules, an IBM Outstanding Technical Achievement Award in 1984 for elementary functions work, and an IBM Outstanding Innovation Award in 1985 for work on vectorizing the FFT algorithm. He is a Fellow of the IEEE.

Fred G. Gustavson *IBM Research Division, T. J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Gustavson is manager of Algorithms and Architectures in the Mathematical Sciences department at the Research Center. He received his B.S. in physics in 1957 and his M.S. and Ph.D. in applied mathematics in 1960 and 1963 from Rensselaer Polytechnic Institute. Since joining IBM in 1963, his primary interest has been in developing theory and programming techniques for exploiting the sparseness inherent in large systems of linear equations. He has worked in the areas of nonlinear differential equations, linear algebra, symbolic computation, computer-aided design of networks, design and analysis of algorithms, and programming applications. As a manager, he designed and headed a project to improve IBM's elementary function package and to produce vector algorithms for IBM's first vector computer, the 3090. He recently designed and headed a project to produce high-performance software for matrix algebra, linear systems, signal processing, and fast Fourier transforms. He and his group produced state-of-the-art software which became part of IBM's VS FORTRAN Version 2 Library and Engineering and Scientific Subroutine Library. He also contributed to designing algorithms for IBM's linear programming package, MPSX2, and, with R. K. Brayton and G. D. Hachtel, was responsible for many of the essential algorithms of the IBM circuit analysis package ASTAP. Dr. Gustavson has received an IBM Outstanding Contribution Award for his work in sparse matrices and an IBM Outstanding Invention Award, jointly with Brayton and Hachtel, for the sparse tableau approach to network analysis and design. In 1984, he received an IBM Outstanding Innovation Award, jointly with R. C. Agarwal, J. W. Cooley, and B. Tuckerman, for producing highly accurate and significantly faster elementary function algorithms for System/370 machines. In 1985, he received an IBM Outstanding Technical Achievement Award for his contribution to the design and implementation of novel high-performance algorithms for solving linear equations. In 1988, he received an IBM Outstanding Technical Achievement Award for his prior contributions to the theory, design, and implementation of a new approach which became the embodiment of ASTAP. This work was also recognized with a corporate award.

Joan McComb *IBM Data Systems Division, P.O. Box 100, Kingston, New York 12401.* Ms. McComb is an advisory programmer in the Engineering and Scientific Library Development Group. She received a B.A. and M.S. in mathematics from New York University in 1976 and 1978, respectively, and an M.S. in computer engineering from Syracuse University in 1987. She joined IBM in Owego, New York, in 1978, and worked on a real-time simulator for the LAMPS (Light Airborne Multipurpose System) project and on the CPEXEC microcode for the IBM 3838 Array Processor. Ms. McComb transferred to Poughkeepsie, New York, in 1981 and became involved in design verification of large systems. Since 1984, she has worked on ESSL development in Kingston.

Stanley Schmidt *IBM Data Systems Division, P.O. Box 100, Kingston, New York 12401.* Mr. Schmidt is currently a senior engineering manager in the Engineering and Scientific Program Development department where he is responsible for the ESSL program product. He has B.S. and M.S. degrees in mathematics from the University of Wisconsin. He joined IBM in Poughkeepsie, New York, in 1963 in a scientific computations department as a mathematical analyst. His primary involvement has been with numerical solutions to problems in circuit, device, and packaging analysis as well as algorithms for machine design. He has also participated in the development of APL mathematical functions. In 1972–73, Mr. Schmidt was a visiting Associate Professor at Hampton Institute, teaching numerical analysis and programming. In 1981 he joined Scientific and Engineering Processor Development where he worked on architecture issues. He subsequently initiated the ESSL development effort in which he continues to be involved. Mr. Schmidt has received an IBM Data Systems Division award for his ESSL contributions.

Reprint Order No. G321-5363.