	Values of N				
Subroutine	100	200	300	400	500
DGEMV('N', ...	11	12	11	11	11
DGEMM('N', 'N', ...	17	18	19	19	19
DGETRF, M=N †	7	11	13	15	16
DGETRI	5	9	11	13	14
DPOTRF('U', ...	5	11	13	14	15
DPOTRF('L', ...	5	10	11	11	11
DPOTRI('U', ...	4	6	7	7	7
DPOTRI('L', ...	4	7	8	9	10
DSYTRF('U', ...	4	8	10	12	13
DSYTRF('L', ...	4	8	10	12	13
DGEQRF, M=N	6	11	13	14	14
DGEHRD	7	8	10	10	11
DSYTRD	6	8	8	7	7

†– times reported for DLUBR

Table 19: Speed in megaflops, Alliant FX/4, 4 processors
BLAS from FX/Series Linear Algebra Library, ver. 5

	Values of N				
Subroutine	100	200	300	400	500
DGEMV('N', ...	10	12	13	13	12
DGEMM('N', 'N', ...	10	11	11	11	12
DGETRF, M=N †	7	10	11	12	12
DGETRI	7	9	8	10	11
DPOTRF('U', ...	7	9	11	13	13
DPOTRF('L', ...	6	9	10	11	11
DPOTRI('U', ...	6	8	10	11	11
DPOTRI('L', ...	6	9	10	11	11
DSYTRF('U', ...	8	10	11	11	12
DSYTRF('L', ...	7	10	11	12	13
DGEQRF, M=N	10	12	13	14	14
DGEHRD	10	12	13	13	14
DSYTRD	8	9	10	11	11

†– times reported for DLUBR

Table 20: Speed in megaflops, Stardent 3000
3.01 FCS, Fortran BLAS, -O2 -inline (vectorization only)

	Values of N				
Subroutine	100	200	300	400	500
DGEMV('N', ...	24	30	22	21	21
DGEMM('N', 'N', ...	74	76	78	78	84
DGETRF, M=N †	13	30	41	47	53
DGETRI	8	20	28	36	42
DPOTRF('U', ...	11	27	38	49	54
DPOTRF('L', ...	10	20	25	27	29
DPOTRI('U', ...	6	11	14	15	15
DPOTRI('L', ...	7	15	20	23	26
DSYTRF('U', ...	9	15	22	29	32
DSYTRF('L', ...	8	15	23	29	33
DGEQRF, M=N	11	28	39	47	50
DGEHRD	13	20	25	27	27
DSYTRD	13	18	18	19	18

†– times reported for DLUBR

Table 17: Speed in megaflops, Alliant FX/80, 8 processors

	Values of N				
Subroutine	100	200	300	400	500
DGEMV('N', ...	39	48	52	54	56
DGEMM('N', 'N', ...	37	50	56	57	59
DGETRF, M=N †	13	19	25	29	33
DGETRI	18	32	42	47	52
DPOTRF('U', ...	20	35	42	47	49
DPOTRF('L', ...	22	36	44	48	51
DPOTRI('U', ...	18	34	42	48	53
DPOTRI('L', ...	16	31	40	46	50
DSYTRF('U', ...	22	33	38	42	45
DSYTRF('L', ...	22	32	39	42	45
DGEQRF, M=N	27	40	46	51	54
DGEHRD	27	39	43	48	52
DSYTRD	15	24	29	32	34

†– times reported for DLUBR

Table 18: Speed in megaflops, FPS Model 500, 1 processor
Optimized BLAS from FPS Computing

	Values of N				
Subroutine	100	200	300	400	500
DGEMV('N', ...	29	29	32	30	33
DGEMM('N', 'N', ...	67	70	70	71	72
DGETRF, M=N	22	33	39	44	47
DGETRI	44	56	63	64	66
DPOTRF('U', ...	34	54	65	65	67
DPOTRF('L', ...	34	54	56	61	65
DPOTRI('U', ...	34	45	58	59	63
DPOTRI('L', ...	34	54	62	63	65
DSYTRF('U', ...	17	22	27	34	34
DSYTRF('L', ...	17	27	29	36	36
DGEQRF, M=N	27	43	45	49	53
DORGQR, M=N=K	27	41	46	51	53
DGEHRD	24	36	41	43	45
DSYTRD('U', ...	27	28	29	29	30
DSYTRD('L', ...	27	28	28	29	30
DGEBRD	22	24	27	30	32

Table 15: Speed in megaflops, IBM RISC/6000-550

	Values of N									
Subroutine	100	200	300	400	500	600	700	800	900	1000
DGEMV('N', ...	18	19	19	20	20	20	20	20	20	20
DGEMM('N', 'N', ...	42	44	44	43	43	43	43	43	44	43
DGETRF, M=N †	19	26	29	32	34	34	36	36	37	37
DGETRI	22	31	35	36	38	38	39	39	40	40
DPOTRF('U', ...	24	29	34	36	38	39	40	40	41	41
DPOTRF('L', ...	19	25	29	32	34	34	35	36	37	37
DPOTRI('U', ...	17	23	26	27	27	27	28	28	27	28
DPOTRI('L', ...	17	22	25	25	25	25	26	26	26	26
DSYTRF('U', ...	15	20	23	25	27	29	29	29	30	31
DSYTRF('L', ...	13	19	23	24	26	28	28	29	30	30
DGEQRF, M=N	19	26	30	32	34	34	35	36	36	37
DGEHRD	16	21	22	23	24	24	25	25	26	26
DSYTRD	15	16	17	17	17	18	18	18	18	18

†– times reported for DLUBR

Table 16: Speed in megaflops, IBM RISC/6000-530
AIX 3.1, XL FORTRAN, Optimized BLAS from IBM ECSEC

	Values of N				
Subroutine	100	200	300	400	500
DGEMV('N', ...	64	73	66	71	74
DGEMM('N', 'N', ...	87	102	103	103	107
DGETRF, M=N †	25	42	53	59	66
DGETRI	16	24	28	30	32
DPOTRF('U', ...	30	44	56	62	67
DPOTRF('L', ...	48	63	65	66	67
DPOTRI('U', ...	14	22	25	27	28
DPOTRI('L', ...	13	20	26	29	31
DSYTRF('U', ...	37	55	65	71	75
DSYTRF('L', ...	37	55	64	71	75
DGEQRF, M=N	34	60	71	75	80
DGEHRD	37	59	65	70	74
DSYTRD	19	28	32	35	37

†– times reported for DLUBR

Table 13: Speed in megaflops, IBM3090J-6VF, 1 processor
Optimized BLAS from ESSL

	Values of N				
Subroutine	100	200	300	400	500
DGEMV('N', ...	44	45	46	48	51
DGEMM('N', 'N', ...	70	70	70	71	73
DGETRF, M=N †	28	39	47	52	55
DGETRI	18	32	42	47	52
DPOTRF('U', ...	20	35	42	47	49
DPOTRF('L', ...	22	36	44	48	51
DPOTRI('U', ...	18	34	42	48	53
DPOTRI('L', ...	16	31	40	46	50
DSYTRF('U', ...	22	33	38	42	45
DSYTRF('L', ...	22	32	39	42	45
DGEQRF, M=N	27	40	46	51	54
DGEHRD	27	39	43	48	52
DSYTRD	15	24	29	32	34

†– times reported for DLUBR

Table 14: Speed in megaflops, IBM3090-600 E/VF, 1 processor
VM/XASP, VS Fortran ver. 2.4, Optimized BLAS from
ESSL rel. 4, Umeå University, NAGMark 13, and IBMECSEC

	Values of N				
Subroutine	100	200	300	400	500
SGEMV('N', ...	260	268	287	285	291
SGEMM('N', 'N', ...	281	279	290	289	292
SGETRF, M=N †	129	208	241	256	266
SGETRI	200	242	270	274	282
SPOTRF('U', ...	123	208	242	259	268
SPOTRF('L', ...	123	208	243	259	268
SPOTRI('U', ...	136	212	252	263	274
SPOTRI('L', ...	136	216	252	266	274
SSYTRF('U', ...	89	161	199	221	235
SSYTRF('L', ...	94	167	206	227	238
SGEQRF, M=N	162	234	256	266	272
SORGQR, M=N=K	194	250	265	271	276
SGEHRD	195	244	268	271	278
SSYTRD('U', ...	145	219	247	261	269
SSYTRD('L', ...	147	221	248	262	269
SGEBRD	161	234	256	266	272

†– times reported for Crout LU

Table 11: Speed in megaflops, CRAY Y-MP, 6.41 nsec clock, 1 processor
UNICOS 7.0, CFT77 5.0, libsci BLAS

	Values of N				
Subroutine	100	200	300	400	500
DGEMV('N', ...	44	70	102	128	150
DGEMM('N', 'N', ...	151	155	156	154	159
DGETRF, M=N †	39	71	90	101	111
DPOTRF('U', ...	32	63	82	96	103
DPOTRF('L', ...	32	61	82	96	106
DSYTRF('U', ...	24	46	56	66	76
DSYTRF('L', ...	23	45	53	66	76
DGEQRF, M=N	41	65	82	97	106
DGEHRD	45	61	77	89	108
DSYTRD	25	36	41	46	48

†– times reported for DLUBR

Table 12: Speed in megaflops, Convex C240, 4 processors
Optimized BLAS from Convex Computer Corp.

	Values of N				
Subroutine	100	200	300	400	500
DGEMV('N', ...	262	554	791	945	1012
DGEMM('N', 'N', ...	327	633	882	994	1080
DGETRF, M=N †	66	175	303	423	526
DGETRI	40	109	183	255	324
DPOTRF('U', ...	53	145	237	312	369
DPOTRF('L', ...	49	136	222	302	376
DPOTRI('U', ...	38	102	179	253	325
DPOTRI('L', ...	32	90	155	210	264
DSYTRF('U', ...	36	90	140	182	217
DSYTRF('L', ...	32	82	130	171	206
DGEQRF, M=N	101	237	329	388	457
DGEHRD	141	307	411	481	517
DSYTRD	37	89	138	183	223

†- times reported for DLUBR

Table 9: Speed in megaflops, Siemans/Fujitsu VP400-EX, 1 processor
Optimized BLAS from Universität Karlsruhe

	Values of N				
Subroutine	100	200	300	400	500
SGEMV('N', ...	289	324	354	353	365
SGEMM('N', 'N', ...	415	411	446	436	450
SGETRF, M=N †	117	234	300	340	357
SGETRI †	204	296	360	375	399
SPOTRF('U', ... †	105	216	289	317	358
SPOTRF('L', ... †	105	215	288	311	357
SPOTRI('U', ... †	119	241	315	349	379
SPOTRI('L', ... †	117	239	311	347	379
SSYTRF('U', ...	59	131	189	225	254
SSYTRF('L', ...	62	130	186	222	249
SGEQRF, M=N	133	218	269	308	330
SORGQR, M=N=K	152	229	279	312	335
SGEHRD	158	232	290	310	338
SSYTRD('U', ...	123	214	260	280	302
SSYTRD('L', ...	122	210	262	284	300
SGEBRD	132	203	227	258	278

†- libsci routine

Table 10: Speed in megaflops, CRAY-2, 1 processor
UNICOS 7.0, CFT77 5.0, libsci BLAS

	Values of N								
Subroutine	100	250	500	750	1000	1250	1500	1750	2000
SGEMV('N', ...	740	863	891	894	898	897	899	899	900
SGEMM('N', 'N', ...	860	893	897	899	899	900	901	901	901
SGETRF, M=N †	327	650	789	832	851	862	871	875	878
SGETRI †	522	781	855	875	883	887	890	892	894
SPOTRF('U', ... †	272	607	778	829	851	863	871	876	879
SPOTRF('L', ... †	257	596	774	829	852	864	872	876	880
SPOTRI('U', ... †	275	624	797	845	864	875	881	885	888
SPOTRI('L', ... †	270	619	794	845	865	875	882	886	889
SSYTRF('U', ...	149	396	580	655	700	722	735	753	772
SSYTRF('L', ...	161	414	589	651	682	701	714	725	747
SGEQRF, M=N	349	669	781	812	827	834	842	846	844
SORGQR, M=N=K	435	712	797	820	831	836	835	846	846
SGEHRD	456	738	817	833	842	845	844	842	847
SSYTRD('U', ...	276	573	730	788	816	833	846	852	860
SSYTRD('L', ...	278	571	727	783	813	827	838	846	853
SGEBRD	346	671	783	815	828	834	842	844	842

†– libsci LAPACK with best blocksize

Table 7: Speed in megaflops, CRAY Y-MP C90, 1 processor
UNICOS 7.C, CFT77 5.0, libsci BLAS

	Values of N				
Subroutine	100	200	300	400	500
DGEMV('N', ...	706	1152	917	1202	1263
DGEMM('N', 'N', ...	784	1206	927	1216	1272
DGETRF, M=N †	150	292	356	423	495
DGETRI	40	99	156	198	232
DPOTRF('U', ...	155	387	589	719	819
DPOTRF('L', ...	102	224	334	406	463
DPOTRI('U', ...	47	110	169	216	252
DPOTRI('L', ...	49	114	173	211	243
DSYTRF('U', ...	104	235	346	417	471
DSYTRF('L', ...	100	221	325	394	444
DGEQRF, M=N	217	498	617	690	768
DGEHRD	338	662	664	787	862
DSYTRD	117	244	335	392	430

†– times reported for DLUBR

Table 8: Speed in megaflops, NEC SX2-400, 1 processor
SXOS Ver. R4.11, FORTRAN77SX rev. 041
Optimized BLAS from HNSX Supercomputers

	Values of N								
Subroutine	100	250	500	750	1000	1250	1500	1750	2000
SGEMV('N', ..	434	1398	5229	6422	6757	6610	6776	6874	6940
SGEMM('N', 'N', ...	5954	6768	7039	7085	7077	7091	7125	7123	7119
SGETRF, M=N †	372	1506	3216	4192	4871	5196	5501	5680	5838
SGETRI †	902	2656	4308	5121	5375	5860	6058	6216	6366
SPOTRF('U', ... †	361	1463	3078	4162	4863	5269	5612	5866	6052
SPOTRF('L', ... †	355	1389	3074	4101	4814	5292	5625	5855	6038
SPOTRI('U', .. †	355	1441	3073	4176	4877	5283	5619	5846	6023
SPOTRI('L', .. †	356	1456	3112	4183	4852	5320	5629	5862	6034
SSYTRF('U', ...	134	584	1502	2321	2829	3300	3658	3890	4118
SSYTRF('L', ...	136	647	1701	2585	3229	3695	4006	4275	4468
SGEQRF, M=N	322	1361	3239	4368	5297	5635	5906	6080	6226
SORGQR, M=N=K	377	1588	3525	4755	5480	5748	6003	6157	6298
SGEHRD	422	1757	3915	5098	5735	5919	6181	6300	6425
SSYTRD('U', ...	267	1120	2693	3907	4689	5088	5414	5664	5846
SSYTRD('L', ...	258	1127	2716	3924	4705	5095	5421	5654	5848
SGEBRD	320	1359	3224	4488	5278	5616	5861	6063	6214

†– libsci LAPACK with best blocksize

Table 5: Speed in megaflops, CRAY Y-MP C90, 8 processors, dedicated
UNICOS 7.C, CFT77 5.0, libsci BLAS

	Values of N									
Subroutine	100	200	300	400	500	600	700	800	900	1000
DGEMV('N', ...	1043	2316	3329	3839	4315	4217	4566	4416	4441	4589
DGEMM('N', 'N', ...	1567	2871	3545	4096	4550	4417	4743	4504	4554	4710
DGETRF, M=N †	217	648	1123	1603	2036	2323	2667	2837	3010	3179
DGETRI	134	391	663	925	1167	1392	1605	1762	1928	2073
DPOTRF('U', ...	164	454	753	1030	1263	1454	1607	1725	1827	1966
DPOTRF('L', ...	148	351	627	901	1158	1398	1615	1797	1955	2108
DPOTRI('U', ...	129	332	562	791	1026	1232	1430	1609	1771	1918
DPOTRI('L', ...	109	305	536	746	945	1128	1277	1406	1509	1615
DSYTRF('U', ...	126	335	521	671	787	874	945	1001	1089	1174
DSYTRF('L', ...	123	303	485	632	751	841	917	978	1027	1109
DGEQRF, M=N	309	779	1165	1455	1653	1849	2075	2267	2404	2554
DGEHRD	470	1094	1558	1859	2091	2197	2336	2449	2585	2734
DSYTRD	116	268	413	547	668	777	876	962	1045	1120

†– times reported for DLUBR

Table 6: Speed in megaflops, Siemans/Fujitsu VP 2600-EX, 1 processor
Optimized BLAS from Universität Karlsruhe

Machine	DGEMM	DGEMV	DGETRF	Ratio
CRAY Y-MP C90 (8 proc)	7039	5229	3216	0.46
Fujitsu VP2600	4550	4315	2036	0.45
CRAY Y-MP (8 proc)	2449	2244	1422	0.58
CRAY-2 (4 proc)	1797	1276	822	0.46
NEC SX2	1272	1263	495	0.39
Fujitsu VP-400 EX	1080	1012	526	0.49
CRAY-2 (1 proc)	451	364	357	0.79
CRAY Y-MP (1 proc)	312	311	284	0.91
Convex C240 (4 proc)	159	150	111	0.70
IBM3090J	107	74	66	0.62
Alliant FX/80	84	21	53	0.63
IBM3090-600E	73	51	56	0.77
FPS Model 500	59	56	33	0.56
IBMRISC/6000-530	43	20	34	0.79
Alliant FX/4	19	11	16	0.80

Table 3: Speeds in megaflops and the ratio DGETRF/DGEMM, N = 500

Machine	DGEMM	DGEMV	DGETRF	Ratio
CRAY Y-MP C90 (8 proc)	7077	6757	4871	0.69
Fujitsu VP2600	4710	4589	3179	0.67
CRAY Y-MP (8 proc)	2448	2399	1926	0.79
CRAY 2 (4 proc)	1810	1386	1245	0.69
IBMRISC/6000-530	43	20	37	0.86

Table 4: Speeds in megaflops and the ratio DGETRF/DGEMM, N = 1000

Machine	DLUBL (Left)	DLUBC (Crout)	DLUBR (Right)	NB
Fujitsu VP2600	1238	1572	2036	64
CRAY Y-MP (8 proc)	1101	1261	1422	128
CRAY-2 (4 proc)	696	796	822	48
NEC SX2	423	577	495	1
Fujitsu VP-400 EX	344	398	526	64
CRAY-2 (1 proc)	340	361	331	64
CRAY Y-MP (1 proc)	278	284	286	16
Convex C240 (4 proc)	116	112	111	32
IBM3090J	63	64	66	32
Alliant FX/80	35	52	53	32
IBM3090-600E	53	50	56	32
FPS Model 500	20	47	33	1
IBM RISC/6000-530	32	34	34	48
Alliant FX/4	13	15	16	32

Table 1: Speeds in megaflops and block size of best variant, N = 500

Machine	DGETRF (BLAS 3)	DGETF2 (BLAS 2)	DGEFA (BLAS 1)	Speedup
Fujitsu VP2600	2036	1199	229	8.9
CRAY Y-MP (8 proc)	1422	1400	173	8.2
CRAY-2 (4 proc)	822	584	93	8.8
NEC SX2	495	406	258	1.9
Fujitsu VP-400 EX	526	350	85	6.2
CRAY-2 (1 proc)	357	183	93	3.8
CRAY Y-MP (1 proc)	284	283	170	1.7
Convex C240 (4 proc)	111	68	14	7.9
IBM3090J	66	25	24	2.8
Alliant FX/80	53	8	5.6	9.5
IBM3090-600E	56	17	20	2.8
FPS Model 500	33	23	12	2.7
IBM RISC/6000-530	34	13	11	3.1
Alliant FX/4	16	4	3.2	4.9

Table 2: LAPACK vs. LINPACK, LU factorization, N = 500

[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide*. SIAM, Philadelphia, 1992.

[3] E. Anderson and J. Dongarra. Results from the initial release of LAPACK. LAPACK Working Note 16, Technical Report CS-89-89, University of Tennessee, Nov. 1989.

[4] E. Anderson and J. Dongarra. Evaluating block algorithm variants in LAPACK. In J. Dongarra et al., editors, *Proceedings of the Fourth SIAM Conference on Parallel Processing for Scientific Computing*, pages 3–8. SIAM, Philadelphia, 1990. (also LAPACK Working Note 19).

[5] C. Bischof. Adaptive blocking in the QR factorization. *J. Supercomputing*, 3(3):193–208, 1989.

[6] C. Bischof and P. Lacroute. An adaptive blocking strategy for matrix factorizations. In H. Burkhart, editor, *Lecture Notes in Computer Science 457*, pages 210–221, New York, NY, 1990. Springer Verlag.

[7] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart. *LINPACK Users' Guide*. SIAM, Philadelphia, 1979.

[8] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. Duff. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Soft.*, 16(1):1–17, March 1990.

[9] J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson. An extended set of FORTRAN basic linear algebra subprograms. *ACM Trans. Math. Soft.*, 14(1):1–17, March 1988.

[10] J. J. Dongarra and E. Grosse. Distribution of mathematical software via electronic mail. *Communications of the ACM*, 30(5):403–407, July 1987.

[11] J. J. Dongarra, S. J. Hammarling, and D. C. Sorensen. Block reduction of matrices to condensed forms for eigenvalue computations. *Journal of Computational and Applied Mathematics*, 27, 1989. (also LAPACK Working Note 2).

[12] J. Du Croz and M. Pont. The development of a floating-point validation package. In M. J. Irwin and R. Stefanelli, editors, *Proceedings of the 8th Symposium on Computer Arithmetic*, Como, Italy, May 19-21, 1987. IEEE Computer Society Press.

[13] B. S. Garbow, J. M. Boyle, J. J. Dongarra, and C. B. Moler. *Matrix Eigensystem Routines – EISPACK Guide Extension*. Lecture Notes in Computer Science 51. Springer-Verlag, New York, 1977.

[14] B. T. Smith et al. *Matrix Eigensystem Routines – EISPACK Guide*. Lecture Notes in Computer Science 6. Springer-Verlag, New York, second edition, 1976.

which is generally the sustainable peak speed of these machines. We see that the speed of DGETRF is typically around 80% for the smaller machines, and an impressive 91% for one processor of a CRAY Y-MP, but the efficiency declines for the larger supercomputers. In part this is because N = 500 is not a very big problem, so Table 4 extends the problem size to N = 1000 for a few selected machines, with better results. For sufficiently large N, the speed of DGETRF and many other block algorithms should approach that of DGEMM.

We conclude this report by listing in Tables 5–20 the best megaflop rates for a selection of LAPACK routines on the computers in this study. We include data for the matrix factorizations DGETRF, DPOTRF, DSYTRF, and DGEQRF, the matrix inversion routines DGETRI and DPOTRI, the reduction routines DGEHRD, DSYTRD, and DGEBRD, and, if available, the orthogonal transformation routine DORGQR. All of these are blocked routines, and we use the best blocksize for each routine. This assumes that the routine to set the block size, ILAENV, will be optimized for each environment, and in fact a block size other than the five choices tested in the standard LAPACK timing suite may be the optimal one. The purpose here is not to benchmark the different computers, since most of this data was obtained under less than ideal conditions on a busy machine, but simply to demonstrate the performance of these block algorithms and, where the performance is low, identify areas for improvement in the BLAS or LAPACK routines. We specify "optimized BLAS" if anything other than the Fortran BLAS are used, but in many cases only some of the BLAS have been optimized and further improvements in the BLAS could be made (and may have been made since these timings were obtained).

Several changes that became effective with the August 1991 test release of LAPACK do not appear in the older data from the April 1990 test release. The LU factorization subroutine DGETRF was changed to a right-looking variant after the August 1991 test release, so the data reported for DGETRF is generally taken from DLUBR (which is no longer provided with LAPACK). The LU inverse routine DGETRI was changed for the August 1991 release to a faster variant which does more of its work in the Level 3 BLAS routines DGEMM. Timings for the orthogonal transformation routine DORGQR and for the band reduction routine DGEBRD are available only in the August 1991 and later test releases. Also, an UPLO parameter was added to DSYTRD; results reported for DSYTRD with no indication of UPLO='U' or UPLO='L' are from the April 1990 version which assumed lower triangular storage.

Most of the data uses the standard LAPACK data sets, which specifies matrices of order 100, 200, 300, 400, and 500, but in a few cases we have data for larger problems as well. For the CRAY Y-MP C90, some of the LAPACK routines are taken from CRAY's scientific library (libsci), but the block size was varied as in the other examples and the times for the best block size are shown.

References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. LAPACK: A portable linear algebra library for high-performance computers. In *Proceedings Supercomputing '90*, page 11. IEEE Computer Society Press, Los Alamitos, California, 1990. (also LAPACK Working Note 20).

method (a block Crout LU factorization calling a right-looking algorithm within the block, for example) may give the best performance on a particular machine.

The choice of block size can have a significant effect on performance, but similar results are often observed for a range of block sizes. For example, on one processor of a CRAY Y-MP C90, the difference between the performance of a block algorithm for the worst block size is usually within about 10% of the performance with the best block size, and results within 5% of the best block size are common, so careful optimizations for each problem size may not be necessary.

Precise performance tuning is a difficult task. In principal, the optimal block sizes could depend on the machine configuration, problem dimensions, and user-controllable parameters such as the leading matrix dimension. In some environments, the machine configuration can change dynamically, further complicating this process. We used brute force during beta testing of LAPACK, running exhaustive tests on different machines, with ranges of block sizes and problem dimensions. This has produced a large volume of test results, too large for thorough human inspection and evaluation.

There appear to be at least three ways to choose block parameters. First, we could take the exhaustive tests we have done, find the optimal block sizes, and store them in tables in subroutine ILAENV; each machine would require its own special tables. Second, we could devise an automatic installation procedure which could run just a few benchmarks and automatically produce the necessary tables. Third, we could devise algorithms which tuned themselves at run-time, choosing parameters automatically. The choice of method depends on the degree of portability we desire.

4 Timing and Performance Results

Over the course of the LAPACK project we have tested and tuned various algorithms and software in a number of different platforms. This was done with the help of test sites, including researchers from universities, research centers, and industry at over 50 locations in the United States, Canada, and 10 other countries. We are grateful to our friends and colleagues who have so generously contributed their time and computing resources to this project. We report some of their results in the tables that follow.

Table 1 compares the performance of the block variants of the LU factorization for $N = 500$ on a number of different machines. We observe that the right-looking variant DLUBR gives the best performance in 10 of the 14 cases, and this was the variant finally chosen for the LAPACK routine DGETRF.

Table 2 compares the performance of the block LU factorization routine DGETRF (using Level 3 BLAS) with its best blocksize to the unblocked routine DGETF2 (using Level 2 BLAS) and to the LINPACK routine DGEFA, which uses only Level 1 BLAS. We also compute the speedup over LINPACK to show the actual improvement of LAPACK's DGETRF over DGEFA. The speedups range from around 2 on single processors of a CRAY Y-MP and NEC SX2 to 10 on a single processor of an Alliant FX/80. In particular, we see considerable improvements for the multiprocessors in this study.

Table 3 gives a measure of the efficiency of the LAPACK routine DGETRF at $N = 500$. The efficiency is measured against the matrix multiply DGEMM from the Level 3 BLAS,

processed by the Level 3 BLAS. For example, if the block size is 32 for the Gaussian Elimination routine on a particular machine, then the matrix will be processed in groups of 32 columns at a time. All of the tuning parameters in LAPACK are set via the integer function subprogram ILAENV, which can be modified for the local environment. Details of the memory hierarchy determine the block size that optimizes performance.

3 Performance Tuning

Performance tuning may not be of interest to users who wish to regard LAPACK as mail-order software. For those users, the Fortran BLAS, standard LAPACK, and the default blocking parameters in the auxiliary routine ILAENV are always an option. However, optimization of one or all three of these pieces may be necessary to achieve the best algorithm.

Thanks to strong support of the BLAS standard, the LAPACK approach of using the BLAS as building blocks has turned out to be a satisfactory mechanism for producing fast transportable code for dense linear algebra computations on *shared memory* machines. Gaussian elimination and its variants, QR decomposition, and the reductions to Hessenberg, tridiagonal and bidiagonal forms for eigenvalue or singular value computations all admit efficient block implementations using Level 3 BLAS [1, 4]. Such codes are often nearly as fast as full assembly language implementations for sufficiently large matrices, although assembly language versions are typically better for small problems. Parallelism embedded in the BLAS, is generally useful only on sufficiently large problems, and can in fact slow down processing on small problems. This means that the number of processors exercised should ideally be a function of the problem size, something not always taken into account by existing BLAS implementations.

If a library of optimized BLAS exists and an LAPACK routine has been selected, the installer may wish to experiment with tuning parameters such as the block size. The most important issues affecting the choice of block size are

- A full set of optimized BLAS: Sometimes there isn't an advantage to using a blocked (Level 3 BLAS) algorithm over an unblocked (Level 2 BLAS) algorithm because some of the necessary BLAS have not been optimized.

- Level 3 BLAS vs. Level 2 BLAS: On some machines, the memory bandwidth is high enough that the Level 2 BLAS are as efficient as the Level 3 BLAS, and choosing $NB = 1$ (i.e., using the unblocked algorithm) gives much better performance. This is particularly true for the block formulations of the QR factorization and reduction routines, since the block algorithm requires more operations than the unblocked algorithm, and the extra work is justified only if the Level 3 BLAS are faster than the Level 2 BLAS.

- Choice of block algorithm: Studies with different block algorithms for operations such as the LU factorization (DGETRF) often showed more dramatic differences than the choice of block size within the same algorithm. Details are given in the following section.

- Choice of unblocked algorithm: In LAPACK, the unblocked algorithm is always chosen to be the Level 2 BLAS equivalent of the higher level blocked algorithm, but a hybrid

degrade for large problems; this property is frequently called *scalability*. For the subrout in LAPACK, running time depends almost entirely on a problem's dimension alone, not just for algorithms with fixed operation counts like Gaussian elimination, but also for routines that iterate (to find eigenvalues). Hence we can do performance tuning for the average case with some confidence that our optimizations will hold independent of the actual data.

Portability in its most inclusive sense means that the code is written in a standard lan guage (say Fortran), and that the source code can be compiled on an arbitrary machine with an arbitrary Fortran compiler to produce a program that will run correctly and efficiently. We call this the "mail order software" model of portability, since it reflects the model used by software servers like *netlib* [10]. This notion of portability is quite demanding. It requires that all relevant properties of the computer's arithmetic and architecture be discovered at runtime within the confines of a Fortran code. For example, if the overflow threshold is im portant to know for scaling purposes, it must be discovered at runtime *without overflowing* since overflow is generally fatal. Such demands have resulted in quite large and sophis ticated programs [12] which must be modified continually to deal with new architectures and software releases. The mail order software notion of portability also means that codes generally must be written for the worst possible machine expected to be used, thereby often degrading performance on all the others.

2 LAPACK Overview

Teams at the University of Tennessee, The University of California at Berkeley, the Courant Institute of Mathematical Sciences, the Numerical Algorithms Group, Ltd., Cray Research Inc., Rice University, Argonne National Laboratory, and Oak Ridge National Laboratory have developed a transportable linear algebra library called LAPACK (short for Linear Algebra Package) [1]. The library is intended to provide a coordinated set of subroutines to solve the most common linear algebra problems and to run efficiently on a wide range of high-performance computers.

LAPACK provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems and singular value problems. The associated matrix factorizations (LU, Cholesky, QR, SVD, Schur, generalized Schur) are provided, as are related computations such as reordering the Schur factorizations and estimating condition numbers. Matrices may be dense or banded, but there is no provisions for general sparse matrices. In all areas, similar functionality is provided fo real and complex matrices, in both single and double precision. LAPACK is in the public domain and available from *netlib*.

The library is written in standard Fortran 77. The high performance is attained by doing most of the computation in the BLAS [8, 9], a standardized set of matrix-vector and matrix-matrix subroutines. Although Fortran implementations of the BLAS are provided with LAPACK, and many optimizing compilers can recognize some of the parallel con structs in these codes, consistent high performance can generally be attained only by using implementations optimized for a specific machine. In particular, most of the parallelism in LAPACK is embedded in the BLAS and is invisible to the user.

Besides depending upon locally implemented BLAS, good performance also requires knowledge of certain machine-dependent *blocksizes*, which are the sizes of the submatrices

LAPACK Working Note: 44
Performance of LAPACK: A Portable Library of Numeri
Linear Algebra Routines
(University of Tennessee Tech Report CS-92-156, May 1

Edward Anderson
Mathematical Software Group
Cray Research Inc.
Eagan, MN 55121

Jack Dongarra
Department of Computer Science
University of Tennessee
Knoxville, Tennessee 37996-1301
and
Oak Ridge National Laboratory
Mathematical Sciences Section
Oak Ridge, Tennessee 37831

November 13, 1992

1 Introduction

The goal of the LAPACK project was to modernize the widely used LINPACK [7] and EISPACK [14, 13] numerical linear algebra libraries to make them run efficiently on shared memory vector and parallel processors. On these machines, LINPACK and EISPACK are inefficient because their memory access patterns disregard the multilayered memory hierarchies of the machines and spend too much time moving data instead of doing useful floating point operations. LAPACK tries to cure this by reorganizing the algorithms to use a standardized set of block matrix operations known as the BLAS (Basic Linear Algebra Subprograms). These block operations can be optimized for each architecture to account for the memory hierarchy, and so provide a transportable way to achieve high efficiency on diverse modern machines.

We say "transportable" instead of "portable" because for fastest possible performance LAPACK requires that highly optimized block matrix operations be already implemented on each machine. Many computer vendors and researchers have developed optimized versions of the BLAS for specific environments, and we report some of their results in the context of LAPACK in this paper. Among other things, *efficiency* means that the performance (measured in millions of floating point operations per second, or megaflops) should not

*This work was supported in part by NSF Grant No. ASC-8715728, Applied Mathematical Sciences su program of the Office of Energy Research, U. S. Department of Energy, under Contract DE-AC05 and Cray Research Inc.