ScaLAPACK Evaluation and Performance at the DoD MSRCs [1]

L. S. Blackford and R. C. Whaley
Department of Computer Science
University of Tennessee
Knoxville, Tennessee 37996-1301

Abstract

This report presents performance results for a subset of ScaLAPACK driver routines and PBLAS routines on the Cray T3E, IBM SP, SGI Origin 2000, and SGI Power Challenge Array platforms at the Department of Defense (DoD) CEWES, ARL, and ASC Major Shared Resource Centers (MSRCs). Performance is analyzed using SGI MPI v3.0 versus MPICH version 1.1.0 on the SGI platforms, and MPI versus shmem on the Cray T3E. On the Cray T3E, correctness of the version of ScaLAPACK included in LIBSCI is tested, and performance timings are compared against the freely available version of ScaLAPACK on netlib using the MPIBLACS. On the IBM SP, correctness of the version of ScaLAPACK included in PESSL is tested, and performance timings are compared against the freely available version of ScaLAPACK on netlib using the MPIBLACS. On the SGI platforms, ScaLAPACK performance using distributed memory BLAS (PBLAS) is compared to LAPACK performance using the multi-threaded MP BLAS.

[1]This work was partially supported by the DoD High Performance Computing Modernization Program CEWES Major Shared Resource Center through Programming Environment and Training (PET) under Contract Number DAHC 94-96-C0002, Nichols Research Corporation, subcontract no. NRC CR-96-0011; by the National Science Foundation Grant No. ASC-9005933; by the DoD High Performance Computing Modernization Program ARL Major Shared Resource Center through Programming Environment and Training (PET) under Contract Number DAHC-94-96-C-0010, Raytheon E-Systems, subcontract no. AA23; by the DoD High Performance Computing Modernization Program ASC Major Shared Resource Center through Programming Environment and Training (PET) under Contract Number DAHC-94-96-C-0005, Nichols Research Corporation, subcontract no. NRC CR-96-0011; by the Defense Advanced Research Projects Agency under contract DAAH04-95-1-0077, administered by the Army Research Office; by the Office of Scientific Computing, U.S. Department of Energy, under Contract DE-AC05-84OR21400; and by the National Science Foundation Science and Technology Center Cooperative Agreement No. CCR-8809615.

Contents

1 Introduction 3
2 Overview of Machine Characteristics of DoD MSRC systems 4
 2.1 Discussion 10
3 Cray T3E 10
 3.1 Porting ScaLAPACK and the MPI BLACS to the Cray T3E 10
 3.2 Testing of ScaLAPACK within LIBSCI (CrayLibs) 10
 3.3 Parallel matrix-matrix multiply performance 11
 3.4 Parallel LU factorization/solve performance 16
 3.5 Discussion 19
4 IBM SP 20
 4.1 Testing of ScaLAPACK within PESSL 20
 4.2 Parallel matrix-matrix multiply performance 20
 4.3 Parallel LU factorization/solve performance 24
 4.4 Parallel Cholesky factorization/solve performance 27
 4.5 Discussion 28
5 SGI Origin 2000 28
 5.1 Parallel matrix-matrix multiply performance 28
 5.2 Parallel LU factorization/solve performance 33
 5.3 Discussion 36
6 SGI Power Challenge Array 36
 6.1 Parallel matrix-matrix multiply performance 36
 6.2 Parallel LU factorization/solve performance 38
7 Conclusions and future work 39
Bibliography 39

1 Introduction

ScaLAPACK is a library of high-performance linear algebra routines for distributed-memory, message-passing MIMD computers and networks of workstations supporting PVM and/or MPI. It is a continuation of the LAPACK project, which designed and produced analogous software for workstations, vector supercomputers, and shared-memory parallel computers. Both libraries contain routines for solving systems of linear equations, least squares problems, and eigenvalue problems. The goals of both projects are efficiency, scalability (as the problem size and number of processors grow), reliability (including error bounds), portability, and ease of use. LAPACK will run on any machine where the BLAS are available, and ScaLAPACK will run on any machine where both the BLAS and the BLACS are available.

ScaLAPACK has been incorporated into several commercial packages, including the NAG Parallel Library, IBM Parallel ESSL, and Cray LIBSCI, and is being integrated into the VNI IMSL Numerical Library, as well as software libraries for Fujitsu, Hewlett-Packard/Convex, Hitachi, SGI, and NEC.

This report presents performance timings for version 1.5 of ScaLAPACK [2] on the Cray T3E, IBM SP, SGI Origin 2000, and SGI Power Challenge Array platforms at the Department of Defense (DoD) CEWES, ARL, and ASC Major Shared Resource Centers (MSRCs). The SGI timings were performed using SGI MPI v3.0 and MPICH version 1.1.0, and the optimized SGI BLAS (-lblas). Performance comparisons were also made between ScaLAPACK using distributed memory BLAS (PBLAS) and LAPACK [1], version 2.0, using the optimized SGI MP BLAS (-lblas_mp). For the Cray T3E, performance timings were obtained using Cray MPI and Cray shmem, and the optimized BLAS from LIBSCI (CrayLibs). For the IBM SP, performance timings were obtained using the IBM POE library, specifically MPI, and Parallel ESSL and ESSL.

The timings were conducted between October 1997 and March 1998. During that time, vendor software upgrades for the PCA and O2K were made to correct errors detected during testing and timing of the packages. Timings were performed in batch queue mode (via qsub or LoadLeveler) during regular user mode and dedicated mode. Timing fluctuations were encountered. To obtain up-to-date performance figures, users should use the timing programs provided with LAPACK and ScaLAPACK.

The LAPACK and ScaLAPACK packages are freely available on *netlib* and can be obtained via the World Wide Web or anonymous ftp.

```
http://www.netlib.org/lapack/
http://www.netlib.org/scalapack/
```

Section 2 provides an overview of the machine characteristics of the computer systems utilized at the DoD MSRCs. Sections 3, 4, 5, and 6 present performance data for the Cray T3E, IBM SP, SGI Origin 2000, and SGI Power Challenge Array, respectively, at the DoD MSRCs. Section 7 summarizes our conclusions and suggestions for further study.

2 Overview of Machine Characteristics of DoD MSRC systems

In this section, we indicate the hardware and software that characterized each machine during these timings. The two most important software components for ScaLAPACK are its compute kernel, the serial BLAS, and its communication kernel, the BLACS. The BLACS are in turn usually based on a system-specific message passing library such as MPI or shmem. Therefore, in this section we preview some performance indicators for these kernels.

We have two performance indicators for the compute kernel. The *Peak* performance is the theoretical peak floating point performance for one processor. We can obtain the theoretical peak floating point performance from the clock rate of the chip using the following information:

- The CRAY T3E (based on the Alpha 21164 chip) and the SGI Origin 2000 (based on SGI's R10000 chip) have separate floating point adders and multipliers. This means that if the instruction mix can issue one floating point add and one floating point multiply every cycle (matrix multiply can do this), a peak megaflop rate of twice the clock rate is obtained.
- The IBM SP (based on IBM POWER2 chip) and the SGI Power Challenge Array (based on the SGI R8000 chip) have two floating point units each of which can issue a fused multiply add instruction every clock cycle. This allows these architectures to achieve peak megaflop rates of four times the clock rate, assuming the instruction being executed is expressed as a fused multiply/add (matrix multiply may be expressed in this way).

Tables 1 and 2 provide a snapshot of the CEWES MSRC machines discussed in this report, as they were configured during these timings.

Table 3 describes the ARL MSRC machines discussed in this report, as they were configured during these timings.

Table 4 describes the ASC MSRC machines discussed in this report, as they were configured during these timings.

Table 5 shows the compute kernel indicators, while tables 6, 7, and 8 show the performance of various message passing libraries across the systems.

The measurement labeled F_{MM} is our "achievable peak" for uniprocessor floating point performance, which we have arbitrarily chosen to be a matrix-matrix multiplication of order 500. Since many linear algebra routines derive a large part of their performance from matrix multiply, we can get a rough idea of how well a particular routine is doing by seeing how great a percentage of this "achievable peak" it obtains.

For the communication kernel, we measure two widely-recognized communication benchmarks, the communication *latency* (denoted as t_m) and *bandwidth* (denoted by $1/t_v$). We define the *latency* to be the time it takes to send a 0-byte message. Bandwidth is a measurement of the maximal amount of data that can be transferred between processors per unit of time. For each platform, we report latency and bandwidth for both the BLACS and the message passing library it is based on (e.g., MPI).

Table 1: Characteristics of the Cray T3E (jim) and the IBM SP (osprey) at the CEWES MSRC

	Cray T3E	IBM SP
Processor	64-bit Dec ALPHA processor EV5.6	POWER2 590
Clock speed (MHz)	450	135
Processors per node	1	1
Memory per node (MB)	256	1000
Operating system	UNICOS/mk 2.0.2.19	AIX 4.1.4
BLAS	LIBSCI (CrayLibs 3.0.1.2)	ESSL 2.2.2.4
BLACS	MPI BLACS 1.1α and Cray BLACS	MPI BLACS 1.1α
Communication Software	Cray MPI (mpt.1.2.0.0.6 beta) Cray shmem	POE (2.1.0.17)
C compiler	cc (3.0.1.3)	mpcc (3.1.4.0)
C flags	-O3	-O3 -qarch=pwr2
Fortran compiler	f90 (3.0.1.3)	mpxlf (4.1.0.3)
Fortran flags	-dp -X m -O3,aggress	-O3 -qarch=pwr2
Precision	single (64-bit)	double (64-bit)

The latency values are simple measurements of the time to send a 0-byte message from one processor to another, while the bandwidth figures are obtained by increasing message length until message bandwidth was saturated. We use the same timing mechanism for both the BLACS and the underlying message-passing library.

These numbers are actual timing numbers, not values based on hardware peaks, for instance. Therefore, they should be considered as approximate values or indicators of the observed performance between two nodes, as opposed to precise evaluations of the interconnection network capabilities.

It should be noted that timings for the Cray shmem BLACS are not reported because errors were detected during their testing. The BLACS test suite was downloaded from netlib and run on the Cray shmem BLACS from LIBSCI (CrayLibs 3.0.1.2). The detected errors were reported.

In addition, two bugs in Cray MPI (mpt.1.2.0.0.6 beta) were also detected and reported to the vendor. It was possible to code around these MPI bugs so that the Cray MPI BLACS would run correctly on the Cray T3E and pass all tests of the BLACS Test Suite. Thus, timings for the MPI BLACS are listed in this report. These LIBSCI and Cray MPI errors have been reported to Cray Research and we are awaiting news of their correction.

Table 2: Characteristics of the SGI Origin 2000 (pagh) and the SGI PCA (pca1) at the CEWES MSRC

	SGI O2K	SGI PCA
Processor	R10000 (IP27)	R8000 (IP21)
Clock speed (MHz)	195	90
Processors per node	1	16
Memory per node (MB)	512	512
Operating system	IRIX 6.4	IRIX 6.2
BLAS	SGI BLAS SGI MP BLAS	SGI BLAS SGI MP BLAS
BLACS	MPI BLACS 1.1α	MPI BLACS 1.1α
Communication Software	SGI MPI v3.0	SGI MPI v3.0
C compiler	cc (MIPSpro v7.10)	cc (MIPSpro v7.10)
C flags	-O2 -64 -mips4 -r10000 or -O2 -64 -mips4 -r10000 -mp	-O2 -64 -mips4 -r8000 or -O2 -64 -mips4 -r8000 -mp
Fortran compiler	f77 (MIPSpro v7.10)	f77 (MIPSpro v7.10)
Fortran flags	-O2 -64 -mips4 -r10000 or -O2 -64 -mips4 -r10000 -mp	-O2 -64 -mips4 -r8000 or -O2 -64 -mips4 -r8000 -mp
Precision	double (64-bit)	double (64-bit)

Table 3: Characteristics of the SGI PCA (cosm1 and cosm3) and the SGI Origin 2000 (herman1) at the ARL MSRC

	SGI PCA	SGI O2K
Processor	R8000 (IP21)	R10000 (IP27)
Clock speed (MHz)	75	195
Processors per node	12	1
Memory per node (MB)	170	512
Operating system	IRIX 6.2	IRIX 6.4
BLAS	SGI BLAS SGI MP BLAS	SGI BLAS SGI MP BLAS
BLACS	MPI BLACS 1.1α	MPI BLACS 1.1α
Communication Software	SGI MPI v3.0	SGI MPI v3.0
C compiler	cc (Mongoose v7.1)	cc (Mongoose v7.1)
C flags	-O2 -64 -mips4 -r8000 or -O2 -64 -mips4 -r8000 -mp	-O2 -64 -mips4 -r10000 or -O2 -64 -mips4 -r10000 -mp
Fortran compiler	f77 (Mongoose v7.1)	f77 (Mongoose v7.1)
Fortran flags	-O2 -64 -mips4 -r8000 or -O2 -64 -mips4 -r8000 -mp	-O2 -64 -mips4 -r10000 or -O2 -64 -mips4 -r10000 -mp
Precision	double (64-bit)	double (64-bit)

Table 4: Characteristics of the SGI O2K (hpc03) and the IBM SP (hpc02) at the ASC MSRC

	SGI O2K	IBM SP
Processor	R10000 (IP27)	POWER2 590
Clock speed (MHz)	195	135
Processors per node	1	1
Memory per node (MB)	512	1000
Operating system	IRIX 6.4	AIX 4.1.5
BLAS	SGI BLAS SGI MP BLAS	ESSL 2.2.2.1
BLACS	MPI BLACS 1.1α	MPI BLACS 1.1α
Communication Software	SGI MPI v3.0	POE (2.1.0.22)
C compiler	cc (MIPSpro v7.2)	mpcc (3.1.4.0)
C flags	-O2 -64 -mips4 -r10000 or -O2 -64 -mips4 -r10000 -mp	-O3 -qarch=pwr2
Fortran compiler	f77 (MIPSpro v7.2)	mpxlf (3.2.4.0)
Fortran flags	-O2 -64 -mips4 -r10000 or -O2 -64 -mips4 -r10000 -mp	-O3 -qarch=pwr2
Precision	double (64-bit)	double (64-bit)

Table 5: Level 3 BLAS performance indicator

	Mflop/s	
	F_{MM}	Peak
CEWES MSRC		
SGI PCA	334	380
SGI O2K	318	390
IBM SP	500	540
Cray T3E	549	900
ARL MSRC		
SGI PCA	256	300
SGI O2K	330	390
ASC MSRC		
SGI O2K	335	390
IBM SP	316	540

Table 6: Message passing performance indicators for the Cray T3E-900

	t_m (μs)		$1/t_v$ (MB/s)	
	BLACS	Cray MPI	BLACS	Cray MPI
Cray T3E (CEWES MSRC)	30.3	17.8	115.3	170.9

Table 7: Message passing performance indicators for the IBM SP

	t_m (μs)		$1/t_v$ (MB/s)	
	BLACS	Native	BLACS	Native
IBM SP (MPI) (CEWES MSRC)	57.9	29.0	71.6	96.1
IBM SP (MPI) (ASC MSRC)	66.7	33.1	71.7	96.0

Table 8: Message passing performance indicators for the SGI O2K and SGI PCA

	t_m (μs)				$1/t_v$ (MB/s)			
	BLACS	SGI MPI	BLACS	MPICH	BLACS	SGI MPI	BLACS	MPICH
CEWES MSRC								
SGI O2K	20.6	13.1	41.1	27.1	94.0	130.3	58.7	77.0
SGI PCA	42.1	19.7	121.7	65.8	70.2	78.0	50.1	54.9
ARL MSRC								
SGI O2K	21.9	13.3	43.9	28.7	94.0	131.0	56.4	65.2
SGI PCA								
ASC MSRC								
SGI O2K	22.4	14.7			84.1	135.2		

2.1 Discussion

The most important thing to note from table 5 in this section is the pressing need for ASC to upgrade their version of ESSL. Note that ESSL version 2.2.2.1 achieves approximately 63% (316 Mflop/s versus 500 Mflop/s) of the performance obtained by the newer version (2.2.2.4).

3 Cray T3E

We present performance data for the netlib version of ScaLAPACK and the version of ScaLAPACK in LIBSCI on the Cray T3E-900 (jim) located at the CEWES MSRC. The message-passing libraries used were the Cray shmem library and the Cray MPI library. For all timings, the optimized BLAS in Cray LIBSCI were used.

3.1 Porting ScaLAPACK and the MPI BLACS to the Cray T3E

A few errors were detected in the MPI BLACS and ScaLAPACK in porting them to the Cray T3E. A T3E patch for the MPI BLACS and ScaLAPACK is available on netlib. Details of the "patches" can be found in the respective errata files on netlib.

```
http://www.netlib.org/blacs/errata.blacs
http://www.netlib.org/scalapack/errata.scalapack
```

Also noted in these errata files are Cray-specific modifications that are ONLY required on the Cray T3E due to non-standard features of the T3E compilers and arithmetic.

Two bugs in Cray MPI (mpt.1.2.0.0.6 beta) were also detected and reported to the vendor. It was possible to code around these MPI bugs so that the Cray MPI BLACS would run correctly on the Cray T3E and pass all tests of the BLACS Test Suite. Thus, timings for the MPI BLACS on top of Cray MPI were reported in Table 6. (Previous versions of Cray MPI had been tried, but it was not possible to code around the bugs that were detected. The bugs were reported to the vendor and were fixed in version (mpt 1.2.0.0.6 beta) of the library.)

3.2 Testing of ScaLAPACK within LIBSCI (CrayLibs)

An optimized version of ScaLAPACK is available in the Cray Scientific Software Library. We tested CrayLibs version 3.0.1.2 and version 3.0.1.3, which includes a subset of routines from ScaLAPACK, version 1.5, from netlib. Previous versions of ScaLAPACK in LIBSCI (CrayLibs) were incompatible with the version of ScaLAPACK on netlib due to a change to the ordering of the array descriptor. As soon as Cray's LIBSCI was updated with ScaLAPACK, version 1.5, this incompatibility problem was alleviated.

Timings for Cray's native BLACS using the shmem library were not reported in Table 6 because errors were detected during their testing. The BLACS Test Suite was downloaded from netlib and run on the Cray shmem BLACS from LIBSCI (CrayLibs 3.0.1.2). The errors detected have been reported to the vendor.

LIBSCI (CrayLibs 3.0.1.2) lacks the following ScaLAPACK routines:

- *psgecon.f, psdbtrf.f, psdbtrs.f, psdttrf.f, psdttrs.f, psgbtrf.f, psgbtrs.f, pspocon.f, psporfs.f, pspbtrf.f, pspbtrs.f, pspttrf.f, pspttrs.f, pstzrzf.f, psgels.f, pssyev.f, psgesvd.f, and psormlq.f*
- *pcgecon.f, pcgerfs.f, pcdbtrf.f, pcdbtrs.f, pcdttrf.f, pcdttrs.f, pcgbtrf.f, pcgbtrs.f, pcpocon.f, pcporfs.f, pcpbtrf.f, pcpbtrs.f, pcpttrf.f, and pcpttrs.f*

In addition, the C interface to the BLACS is not provided in LIBSCI so a set of wrapper routines had to be provided. The wall-clock and cpu timers included in the netlib version of the BLACS are also not provided in the Cray shmem BLACS, so these had to be provided in order to run the ScaLAPACK Test Suite.

The ScaLAPACK Test Suite was run on LIBSCI (CrayLibs 3.0.1.2), and errors were detected in pcgeqlf.f and pssyevx.f. These errors were reported to the vendor.

LIBSCI (CrayLibs 3.0.1.3) includes a few more routines than the previous version, but still lacks the following ScaLAPACK routines:

- *psgecon.f, psdbtrf.f, psdbtrs.f, psdttrf.f, psdttrs.f, psgbtrf.f, psgbtrs.f, pspocon.f, psporfs.f, pspbtrf.f, pspbtrs.f, pspttrf.f, pspttrs.f, pstzrzf.f, psgels.f, pssyev.f, psgesvd.f, and psormlq.f*
- *pcgerfs.f, pcdbtrf.f, pcdbtrs.f, pcdttrf.f, pcdttrs.f, pcgbtrf.f, pcgbtrs.f, pcpocon.f, pcporfs.f, pcpbtrf.f, pcpbtrs.f, pcpttrf.f, and pcpttrs.f*

Running the ScaLAPACK Test Suite on this version of LIBSCI (CrayLibs 3.0.1.3) revealed that the bug in *pcgeqlf.f* had been fixed. Failures in *pssyevx.f* still occur and they are under investigation.

3.3 Parallel matrix-matrix multiply performance

Asymptotically, the performance of the PBLAS will rest on the performance of the corresponding BLAS routine. For smaller problem sizes, lower order costs – primarily communication – will cause performance loss. We therefore see that effects due to BLACS optimality will be seen mostly in the smaller problem sizes. These results have been obtained for the matrix-matrix multiply operation $C \leftarrow C + A\,B$, where A, B, and C are square matrices of order N.

We collected performance data for the Level 3 PBLAS routine PSGEMM from the netlib version of ScaLAPACK and the version of ScaLAPACK in LIBSCI (CrayLibs). Timings were performed during "non-dedicated" time and "dedicated" time using batch queues via "qsub". We were unable to repeat all timings using both methods due to a paucity of dedicated time.

Tables 9 and 11 show performance for non-dedicated runs, while tables 10 and 12 summarize our dedicated results.

Table 9: Speed in Mflop/s for the two versions of PBLAS matrix-matrix multiply routine PSGEMM, NON-DEDICATED time (Cray T3E)

	Process grid	Block size	Values of N 1000	2000	3000	4000	5000
(NETLIB)	2×2	24	–	–	–	–	–
	2×2	48	1873	2067	2118	2126	2142
	2×2	72	–	–	–	–	–
	2×4	24	–	–	–	–	–
	2×4	48	3305	3804	4027	4148	4206
	2×4	72	–	–	–	–	–
	4×4	24	–	–	–	–	–
	4×4	48	5906	7135	7811	8073	8244
	4×4	72	–	–	–	–	–
	4×8	24	–	–	–	–	–
	4×8	48	9332	12376	14348	14898	15646
	4×8	72	8288	12582	13818	15853	15913
(LIBSCI)	2×2	24	1566	1656	1659	1725	1725
	2×2	48	1886	2099	2122	2126	2154
	2×2	72	–	–	–	–	–
	2×4	24	2873	3155	3204	3330	3330
	2×4	48	3349	3911	4081	4186	4261
	2×4	72	–	–	–	–	–
	4×4	24	5286	6004	6277	6547	6547
	4×4	48	6013	7307	7922	8206	8386
	4×4	72	–	–	–	–	–
	4×8	24	9213	11162	11849	12616	12616
	4×8	48	9950	12786	14892	15189	16159
	4×8	72	–	–	–	–	–

Table 10: Speed in Mflop/s for the two versions of PBLAS matrix-matrix multiply routine PSGEMM, DEDICATED time (Cray T3E)

	Process grid	Block size	Values of N				
			1000	2000	3000	4000	5000
(NETLIB)	2×2	24	1639	1759	1802	1808	1818
	2×2	48	1873	2067	2118	2126	2142
	2×4	24	2922	3336	3470	3542	3584
	2×4	48	3308	3807	4028	4148	4206
	4×4	24	5185	6319	6738	6915	7030
	4×4	48	5913	7134	7807	8105	8242
	4×8	24	8478	11172	12536	13069	13437
	4×8	48	9317	12368	14347	14886	15648
(LIBSCI)	2×2	24	1715	1799	1825	1836	1847
	2×2	48	1883	2096	2119	2124	2151
	2×4	24	3119	3473	3542	3623	3667
	2×4	48	3343	3908	4077	4185	4106
	4×4	24	5610	6637	6927	7089	7211
	4×4	48	6022	7310	7920	8207	8384
	4×8	24	9481	11929	13054	13606	13946
	4×8	48	9929	12805	14886	15189	16153

Table 11: Speed in Mflop/s for the two versions of PBLAS matrix-matrix multiply routine PSGEMM, NON-DEDICATED time (Cray T3E)

	Process grid	Block size	Values of N				
			6000	7000	8000	9000	10000
(NETLIB)	2×2	24	–	–	–	–	–
	2×2	48	–	–	–	–	–
	2×2	72	–	–	–	–	–
	2×4	24	–	–	–	–	–
	2×4	48	4176	4250	4272	–	–
	2×4	72	–	–	–	–	–
	4×4	24	–	–	–	–	–
	4×4	48	8340	8303	8403	8463	8504
	4×4	72	–	–	–	–	–
	4×8	24	–	–	–	–	–
	4×8	48	15962	15884	16285	16394	16619
	4×8	72	16337	15617	17097	17226	17261
(LIBSCI)	2×2	24	–	–	–	–	–
	2×2	48	–	–	–	–	–
	2×2	72	–	–	–	–	–
	2×4	24	3345	3351	3354	–	–
	2×4	48	4180	4248	4269	–	–
	2×4	72	–	–	–	–	–
	4×4	24	6618	6698	6741	6750	6833
	4×4	48	8323	8375	8433	8521	8555
	4×4	72	–	–	–	–	–
	4×8	24	12803	13002	13192	13146	13233
	4×8	48	16127	16065	16576	16523	16891
	4×8	72	–	–	–	–	–

Table 12: Speed in Mflop/s for the two versions of PBLAS matrix-matrix multiply routine PSGEMM, DEDICATED time (Cray T3E)

	Process grid	Block size	Values of N				
			6000	7000	8000	9000	10000
(NETLIB)	2×2	24	–	–	–	–	–
	2×2	48	–	–	–	–	–
	2×4	24	3584	3613	3618	–	–
	2×4	48	4173	4223	4253	–	–
	4×4	24	7104	7188	7159	7216	7223
	4×4	48	8340	8304	8407	8474	8503
	4×8	24	13650	13895	13967	14107	14180
	4×8	48	15964	15883	16312	16392	16618
(LIBSCI)	2×2	24	–	–	–	–	–
	2×2	48	–	–	–	–	–
	2×4	24	3644	3677	3669	–	–
	2×4	48	4176	4250	4272	–	–
	4×4	24	7232	7322	7281	7329	7342
	4×4	48	8321	8372	8435	8522	8559
	4×8	24	13970	14192	14325	14471	14537
	4×8	48	16124	16052	16572	16531	16890

3.4 Parallel LU factorization/solve performance

Similarly, we collected performance data for the LU factor/solve driver routine PSGESV from the netlib version of ScaLAPACK and the version of ScaLAPACK in LIBSCI (CrayLibs). PSGESV solves a square linear system of order N by LU factorization with partial row pivoting of a real matrix. For all timings, 64-bit floating-point arithmetic was used. Thus, double precision timings are reported on all computers. The distribution block size is also used as the partitioning unit for the computation and communication phases.

Timings were performed during "non-dedicated" time and "dedicated" time using batch queues via "qsub".

Table 13: Speed in Mflop/s for the two versions of the LU factor/solve routine PSGESV for square matrices of order N, NON-DEDICATED time (Cray T3E)

	Process Grid	Block Size	Values of N								
			1000	2000	3000	4000	5000	7500	10000	12500	15000
(NETLIB)	1×4	24	678	1050	1251	1348	1426	1539	1598	–	–
	1×4	32	664	1042	1259	1377	1467	1608	1678	–	–
	1×4	48	598	995	1232	1376	1483	1660	1761	–	–
	1×4	72	509	886	1133	1294	1414	1623	1751	–	–
	2×4	24	677	1409	1917	2239	2459	2802	2997	3126	–
	2×4	32	638	1402	1907	2242	2513	2906	3131	3287	–
	2×4	48	628	1336	1881	2226	2527	2991	3273	3462	–
	2×4	72	561	1217	1744	2097	2441	2920	3237	3479	–
	2×8	24	808	1984	2944	3645	4112	5037	5483	5818	6046
	2×8	32	767	1927	2849	3533	4098	5107	5621	6016	6273
	2×8	48	726	1772	2699	3355	3953	5048	5680	6160	6493
	2×8	72	632	1545	2387	3012	3671	4712	5417	5995	6339
	4×8	24	780	2318	3793	5233	6206	8287	9544	10459	11037
	4×8	32	754	2182	3703	4852	6166	8295	9729	10726	11425
	4×8	48	737	2096	3460	4929	5940	8264	9765	10965	11736
	4×8	72	682	1917	3201	4491	5594	7749	9406	10557	11443
(LIBSCI)	1×4	24	742	1089	1279	1368	1449	1558	–	–	–
	1×4	32	709	1075	1284	1394	1485	1623	–	–	–
	1×4	48	637	1019	1253	1393	1504	1680	–	–	–
	1×4	72	526	903	1145	1307	1426	1637	–	–	–
	2×4	24	881	1623	2089	2370	2562	2870	3042	3164	–
	2×4	32	820	1609	2072	2359	2615	2972	3169	3321	–
	2×4	48	785	1512	2031	2341	2627	3060	3320	3500	–
	2×4	72	676	1360	1865	2196	2523	2979	3279	3512	–
	2×8	24	1107	2362	3286	3936	4395	5225	5625	5937	6137
	2×8	32	1022	2282	3184	3820	4364	5290	5760	6146	6376
	2×8	48	936	2043	2961	3598	4191	5223	5820	6276	6585
	2×8	72	786	1732	2563	3207	3826	4857	5523	6087	6415
	4×8	24	1170	2987	4612	5966	6991	8849	9953	10760	11273
	4×8	32	1128	2891	4505	5670	6912	8855	10130	11031	11642
	4×8	48	1051	2648	4173	5523	6627	8762	10147	11249	11971
	4×8	72	920	2304	3680	4968	6035	8170	9675	10787	11632

Table 14: Speed in Mflop/s for the two versions of the LU factor/solve routine PSGESV for square matrices of order N, DEDICATED time (Cray T3E)

	Process Grid	Block Size	Values of N								
			1000	2000	3000	4000	5000	7500	10000	12500	15000
(NETLIB)	1×4	24	–	–	–	–	–	–	–	–	–
	2×4	24	–	–	–	–	–	–	–	–	–
	2×8	24	–	–	–	–	–	–	–	–	–
	4×8	24	784	2275	3657	4997	6255	8333	9661	10500	11115
(LIBSCI)	1×4	24	743	1088	1278	1369	1449	1559	–	–	–
	1×4	32	709	1074	1281	1393	1489	1626	–	–	–
	1×4	48	637	1019	1252	1395	1504	1682	–	–	–
	2×4	24	883	1628	2091	2375	2566	2874	3045	3167	–
	2×4	32	819	1610	2076	2360	2615	2974	3169	3325	–
	2×4	48	785	1512	2037	2343	2632	3066	3323	3505	–
	2×8	24	1107	2359	3290	3937	4397	5228	5627	5939	6138
	2×8	32	1023	2282	3187	3817	4365	5288	5767	6134	6367
	2×8	48	935	2045	2962	3602	4193	5223	5824	6280	6588
	4×8	24	1162	2992	4617	5972	6999	8853	9954	10764	11278
	4×8	32	1134	2894	4509	5673	6922	8866	10127	11038	11629
	4×8	48	1054	2647	4179	5526	6634	8769	10147	11250	11974

3.5 Discussion

For these timings, we note that the performance of PSGEMM for large problem sizes is very close to our "achievable peak". Asymptotically, this will be true of any system (as the $O(N^3)$ computation dominates the $O(N^2)$ communication costs). However, due to the speed of its communication, the T3E was the only system to reach this peak with the selected problem sizes.

For both matrix multiply and LU, dedicated and non-dedicated runs showed little variation. This seems to indicate that the T3E's queuing system does a good job of isolating the different jobs.

From these timings, it appears LIBSCI's use of shmem (as opposed to the netlib's use if MPI) pays off. What we see is that LIBSCI routines get better performance than their netlib equivalents, but that the difference narrows as we increase the problem size, or in the case of PSGEMM, increase block size. We draw the conclusion that this performance win is mainly in communication since both of these changes tend to minimize the communication costs.

In a related note, it is easily seen that the performance of PSGEMM increases as we increase the block size; this is not true for LU. This is because large block sizes increase load imbalance for LU; PSGEMM, where the operation may be almost arbitrarily reordered, does not become load-imbalanced as the block size is increased. With large blocking factors, there is more work done per BLAS invocation, thus allowing a greater portion of the asymptotic peak to be reached. Despite this, we still restrain our PSGEMM timings to blocking factors that are roughly the same as for our LU timings, since few applications use PSGEMM in isolation.

4 IBM SP

We present performance data on the IBM SP (osprey) for the netlib version of ScaLAPACK and the version of ScaLAPACK in PESSL on the IBM SP (osprey) located at the CEWES MSRC and the IBM SP (hpc02) located at the ASC MSRC. The message-passing library used was the IBM POE library, specifically MPI, and the optimized BLAS library used was the ESSL BLAS.

4.1 Testing of ScaLAPACK within PESSL

An optimized version of ScaLAPACK is available in the IBM Parallel Scientific Software Library (PESSL). We tested PESSL version 2.2.2.4 on the IBM SP (osprey) at the CEWES MSRC. At the time of this report, PESSL was not available on the IBM SP (hpc02) at the ASC MSRC.

Parallel ESSL (version 2.2.2.4) lacks the following ScaLAPACK routines:

- *pslamch.f, pslange.f, pslacpy.f, pslaset.f, pslapiv.f, psgecon.f, psgerfs.f, psdbtrf.f, psdbtrs.f, psgbtrf.f, psgbtrs.f, pslansy.f, pspocon.f, psporfs.f, psgeqrf.f, psgeqlf.f, psgerqf.f, psgeqpf.f, pstzrzf.f, psgeqlf.f, psgels.f*

and their dependent auxiliary subroutines. The following PBLAS routines were also missing or replaced with slightly different functionality:

- *ptopset.c, ptopget.c, and pbdtran.f.*

In addition, the C interface to the BLACS is not provided in PESSL. The wall-clock and cpu timers included in the netlib version of the BLACS are also not provided in the IBM BLACS, so these had to be provided in order to run the ScaLAPACK Test Suite.

The ScaLAPACK Test Suite was run on PESSL (version 2.2.2.4).

4.2 Parallel matrix-matrix multiply performance

Asymptotically, the performance of the PBLAS will rest on the performance of the corresponding BLAS routine. For smaller problem sizes, lower order costs, primarily communication, will cause performance loss. We therefore see that effects due to BLACS optimality will be seen mostly in the smaller problem sizes. These results have been obtained for the matrix-matrix multiply operation $C \leftarrow C + A\,B$, where A, B, and C are square matrices of order N.

We collected performance data for the Level 3 PBLAS routine PDGEMM from the netlib version of ScaLAPACK and the version of ScaLAPACK in PESSL (version 2.2.2.4). Timings were performed during "non-dedicated" time and "dedicated" time using batch queues via "qsub". Dedicated time on this machine was not truly dedicated, as other people could still log in to the machine. With this caveat, we can state that dedicated and non-dedicated runs are within clock resolution of each other. Both would occasionally show large, non-repeatable performance drops, probably due to system interference.

We present in tables 15 and 16 performance timings for the netlib version of PDGEMM versus the PESSL version of PDGEMM. These timings were obtained during "non-dedicated" time over two days. Two sets of timings are included to illustrate the variation in timings that were encountered.

Comparing the data in Tables 5, 15, and 16, we can see that the PBLAS routine PDGEMM achieves 74–89% of the per processor DGEMM performance on the IBM SP. PESSL PDGEMM performance timings are very similar.

Table 15: Speed in Mflop/s for the two versions of the matrix-matrix multiply routine PDGEMM, NON-DEDICATED time (IBM SP)

	Process grid	Block size	Values of N				
			1000	2000	3000	4000	5000
CEWES MSRC							
(NETLIB)	2×2	50	1703	1818	1848	1873	1865
	2×2	50	1726	1829	1858	1520	1881
	2×4	50	2940	3395	3496	3605	3622
	2×4	50	2998	3271	3559	3628	3647
	4×4	50	5046	6175	6714	6904	6992
	4×4	50	4884	6106	6795	6912	7003
	4×8	50	6813	10659	11278	12583	12561
	4×8	50	5865	10410	11202	12401	12401
(PESSL)	2×2	50	1699	1828	1842	1862	1870
	2×2	50	1725	1846	1861	1881	1888
	2×4	50	2790	3269	3448	3535	3585
	2×4	50	2847	3330	3479	3570	3613
	4×4	50	4638	5957	6421	6763	6841
	4×4	50	4703	5901	6501	6770	6899
	4×8	50	6325	9733	10803	11940	12312
	4×8	50	5742	9287	10728	11995	12472
ASC MSRC							
(NETLIB)	2×2	50	1102	1152	1125	1057	–
	2×2	64	943	1040	1044	1068	–
	2×4	50	1976	2058	2221	1999	2096
	2×4	64	1657	1841	2034	2061	2013
	4×4	50	5221	3603	4173	4244	3988
	4×4	64	3302	3868	4035	3961	3885
	4×8	50	4675	6583	7622	7798	7223
	4×8	64	6442	6756	6961	7045	7487

Table 16: Speed in Mflop/s for the two versions of the matrix-matrix multiply routine PDGEMM, NON-DEDICATED time (IBM SP)

	Process grid	Block size	Values of N				
			6000	7000	8000	9000	10000
CEWES MSRC							
(NETLIB)	2×2	50	–	–	–	–	–
	2×2	50	–	–	–	–	–
	2×4	50	3670	3683	3215	–	–
	2×4	50	3693	3723	3686	–	–
	4×4	50	7130	7245	7296	7299	7326
	4×4	50	7165	7282	7331	7342	7373
	4×8	50	13422	13442	13905	13731	14053
	4×8	50	13350	13445	13927	13738	14049
(PESSL)	2×2	50	–	–	–	–	–
	2×2	50	–	–	–	–	–
	2×4	50	2564	3675	3603	–	–
	2×4	50	3659	3701	3638	–	–
	4×4	50	6996	7126	7188	7217	7266
	4×4	50	7064	7152	7220	6248	6819
	4×8	50	12958	13195	13508	13569	13766
	4×8	50	12965	13249	13545	13598	13832
ASC MSRC							
(NETLIB)	2×2	50	–	–	–	–	–
	2×2	64	–	–	–	–	–
	2×4	50	–	–	–	–	–
	2×4	64	–	–	–	–	–
	4×4	50	4311	4183	3966	–	–
	4×4	64	4034	4135	–	–	–
	4×8	50	7911	7859	7771	8057	7901
	4×8	64	7779	7884	7717	7768	7665

4.3 Parallel LU factorization/solve performance

Tables 17 and 18 illustrate the speed of the ScaLAPACK driver routine PDGESV for solving a square linear system of order N by LU factorization with partial row pivoting of a real matrix. For all timings, 64-bit floating-point arithmetic was used. Thus, double precision timings are reported. The data distribution block size is also used as the partitioning unit for the computation and communication phases.

We collected performance data for the LU factor/solve routine PDGESV from the netlib version of ScaLAPACK and from PESSL.

We present in tables 17 and 18 performance timings for the netlib version of PDGESV versus the PESSL version of PDGESV. These timings were obtained during "non-dedicated" time over two days via the "qsub" queuing system at the CEWES MSRC and the LoadLeveler queuing system at the ASC MSRC. Two sets of timings (for CEWES MSRC) are included to illustrate the variation in timings that were encountered.

One obvious inconsistency is the poor PDGESV performance for small problem sizes when we used two dimensional grids (eg. the 2×4, 2×8 and 4×8 grid sizes). This is easily explained: two dimensional grids are required for scalability. However, they perform poorly for small problem sizes due to increased latency-bound communication along the columns of the process grid. This is a particular problem on the SP, which has a very fast compute kernel and a very high communication latency. To demonstrate that this was the problem, table 17 shows the timings for small problem sizes on the appropriate one dimensional grid. Notice that, as predicted, they have superior performance for small problem sizes. These timings indicate that a 1×8 grid is probably superior to a 2×4 grid for reasonable problem sizes; for this modest number of processors, very large problems are required for the superior scalability of the two dimensional grids to offset their weakness of increased alpha-bound communication.

Table 17: Speed in Mflop/s for the two versions of the LU factor/solve routine PDGESV for square matrices of order N, NON-DEDICATED time (IBM SP)

	Process Grid	Block Size	Values of N 1000	2000	3000	4000	5000	7500	10000	12500	15000
CEWES MSRC											
(NETLIB)	1×4	40	703	979	1152	1251	1373	1516	–	–	–
	1×4	50	633	884	1067	1172	1300	1453	–	–	–
	1×4	50	676	954	1137	1243	1366	1511	–	–	–
	2×4	40	533	1106	1510	1821	2051	2475	2742	2858	–
	2×4	50	543	1055	1455	1728	1962	2385	2671	2476	–
	2×4	50	555	1106	1482	1807	2025	2471	2747	2919	–
	1×8	40	944	1498	1853	2055	2343	2688	2413	3029	–
	1×8	50	868	1425	1776	2011	2280	2643	2889	3029	–
	2×8	40	662	1629	2350	2957	3475	4363	4970	5263	5613
	2×8	50	632	1581	2228	2748	3209	4111	4744	5164	5506
	2×8	50	597	1526	2288	2846	3308	4226	4836	5253	4954
	1×16	40	1080	2084	2723	3126	3653	4434	4972	5300	5669
	1×16	50	1006	1881	2501	2904	3446	4213	4785	5157	5503
	4×8	40	485	1648	2753	3701	4505	6140	7451	8451	9285
	4×8	50	566	1633	2643	3500	4261	5925	7115	8188	8980
	4×8	50	574	1662	2634	3521	4249	5948	7200	8261	9029
	1×32	40	1169	2423	3389	4186	5110	6551	7663	8439	7228
	1×32	50	1056	2105	3062	3682	4628	5938	7002	7878	8598
(PESSL)	1×4	40	959	1321	1469	1515	1605	1688	–	–	–
	1×4	50	1027	1365	1508	1549	1639	1714	–	–	–
	1×4	50	154	1253	1490	1546	1231	1711	–	–	–
	2×4	40	891	1897	2288	2598	2759	3047	3212	3249	–
	2×4	50	1079	1911	2333	2625	2809	3088	3253	3336	–
	2×4	50	1082	1897	2342	2609	2811	3083	3244	3331	–
	1×8	40	1139	2029	2448	2648	2864	3142	3247	3322	–
	1×8	50	1160	2005	2443	2663	2892	3161	3287	3367	–
	2×8	40	759	2683	3770	4399	4871	5624	6022	5228	5893
	2×8	50	1098	2600	3671	4473	4929	5704	6115	6358	6580
	2×8	50	1086	2611	3763	4437	4922	5681	6109	6349	6575
	1×16	40	793	2219	3066	3634	4216	5027	5517	5856	6125
	1×16	50	1037	2432	3413	3716	4516	5322	5774	6077	6339
	4×8	40	645	2670	4580	6054	7177	9106	10367	11119	11678
	4×8	50	1103	2579	4404	6065	7407	9319	10514	11311	11936
	4×8	50	1095	2505	4415	6159	7366	9297	6327	7907	11905
	1×32	40	628	1971	3331	4334	5338	7198	8449	9318	10111
	1×32	50	936	2020	3682	4815	6019	7879	9146	10024	10686

Table 18: Speed in Mflop/s for the two versions of the LU factor/solve routine PDGESV for square matrices of order N, NON-DEDICATED time (IBM SP)

	Process Grid	Block Size	Values of N								
			1000	2000	3000	4000	5000	7500	10000	12500	15000
ASC MSRC											
(NETLIB)	1×4	50	–	–	–	–	–	–	–	–	–
	1×4	64	399	611	730	782	834	903	–	–	–
	2×4	50	–	–	–	–	–	–	–	–	–
	2×4	64	413	834	1081	1225	1358	1456	1525	1682	–
	1×8	50	–	–	–	–	–	–	–	–	–
	1×8	64	636	932	1162	1274	1431	1594	1710	–	–
	2×8	50	–	–	–	–	–	–	–	–	–
	2×8	64	352	1206	1646	2016	2299	2645	3053	3269	3425
	1×16	50	–	–	–	–	–	–	–	–	–
	1×16	64	697	1308	1673	1832	2217	2644	2935	3137	3287
	4×8	50	–	–	–	–	–	–	–	–	–
	4×8	64	281	1344	2169	2856	3375	4365	4956	5565	5949
	1×32	50	–	–	–	–	–	–	–	–	–
	1×32	64	758	1542	2098	2617	3099	3928	4570	5050	5426

4.4 Parallel Cholesky factorization/solve performance

Since LU is often heavily optimized for benchmarking purposes, a performance comparison of the Cholesky factorization was also conducted. Table 19 illustrates the speed of the ScaLAPACK driver routine PDPOSV for solving a symmetric positive definite linear system of order N by Cholesky factorization. For all timings, 64-bit floating-point arithmetic was used. Thus, double precision timings are reported. The data distribution block size is also used as the partitioning unit for the computation and communication phases.

We collected performance data for the Cholesky factor/solve routine PDPOSV from the netlib version of ScaLAPACK and from PESSL. The PESSL Cholesky factorization also consistently outperformed the netlib implementation, but not to the extent of LU.

Table 19: Speed in Mflop/s for the two versions of the Cholesky factor/solve routine PDPOSV for matrices of order N, NON-DEDICATED time (IBM SP)

	Process Grid	Block Size	Values of N								
			1000	2000	3000	4000	5000	7500	10000	12500	15000
CEWES MSRC											
(NETLIB)	2×2	40	856	1197	1262	1440	1490	1570	–	–	–
	2×2	50	838	1190	1232	1446	1502	1598	–	–	–
	2×2	64	791	1135	1305	1436	1507	1598	–	–	–
	2×4	40	1071	1746	2108	2353	2558	2721	2331	3016	–
	2×4	50	1043	1728	2110	2396	2567	2860	3036	3123	–
	2×4	64	954	1520	2063	2307	2516	1655	3019	3141	–
	4×4	40	1455	2870	3575	4267	4591	5302	5642	5849	5070
	4×4	50	1479	2792	3663	4189	4573	5330	5619	5838	6161
	4×4	64	1387	2547	3380	4038	4530	5248	5676	4377	6185
	4×8	40	1088	3849	5195	6171	7288	8817	9613	10468	10916
	4×8	50	1758	3726	5046	6178	7216	8740	9851	10623	10410
	4×8	64	1587	3297	4852	5895	6900	8581	9708	11110	10919
(PESSL)	2×2	40	811	1143	1268	1344	1381	1461	–	–	–
	2×2	50	925	1223	1338	1430	1474	1544	–	–	–
	2×2	64	970	1287	1422	1485	1530	1605	–	–	–
	2×4	40	1100	1943	2271	2486	2583	2801	2912	2914	–
	2×4	50	1298	2057	2396	2600	2753	2952	3063	3093	–
	2×4	64	1012	2106	2463	2681	2833	3050	3126	3246	–
	4×4	40	1229	2717	3442	3909	4254	4894	5138	5391	5553
	4×4	50	1618	2934	3630	4203	4535	5186	5528	5758	5832
	4×4	64	1617	3042	3802	4471	4785	5365	5753	5959	6142
	4×8	40	859	3841	5388	6468	7176	8601	9427	9990	10429
	4×8	50	1680	3986	5312	6649	7690	9100	10025	10616	11010
	4×8	64	1501	3840	5512	6826	7790	9328	10339	10920	11410

4.5 Discussion

One surprising result is how well the one dimensional process grids perform. Due to the high latency for communication, and the speed of the compute node, one dimensional grids are competitive for these problem sizes even up to 32 nodes.

Comparing PESSL and netlib PDGEMM shows that they are within clock resolution of each other. For PDGESV, the difference is remarkable. PESSL significantly outperforms its netlib equivalent for all cases. Obviously, this routine has been heavily optimized by IBM. The only difference apparent to the user is that PESSL does not apply the pivots to the L portion of the LU factorization. This means that if a user wishes to utilize the factorization itself (as opposed to using it only in the solve), the pivot vector must be applied manually. The long and short of this is that users would be well-advised to use the PESSL LU, unless they have a specific need for the actual factorizations.

5 SGI Origin 2000

We present performance data on the SGI Origin 2000 for the netlib version of ScaLAPACK using the distributed-memory BLAS (PBLAS), and the netlib version of LAPACK using the SGI multi-threaded BLAS (-lblas_mp). The message-passing library used was the SGI MPI v3.0 library. The optimized SGI BLAS (in -lblas) were used for the ScaLAPACK timings and the SGI MP BLAS (in -lblas_mp) were used for the LAPACK timings.

5.1 Parallel matrix-matrix multiply performance

We perform comparison timings of the distributed-memory PBLAS matrix-matrix multiply routine PDGEMM using the SGI BLAS (-lblas) versus the multi-threaded DGEMM in SGI BLAS MP (-lblas_mp).

Asymptotically, the performance of the PBLAS will rest on the performance of the corresponding BLAS routine. For smaller problem sizes, lower order costs, primarily communication, will cause performance loss. We therefore see that effects due to BLACS optimality will be seen mostly in the smaller problem sizes.

Timings were performed during "dedicated" time when we were alone on the machine, and if available, in "non-dedicated" time using batch queues via "qsub". Variances in timings were encountered in both "dedicated" and "non-dedicated" time.

Tables 20 and 21 shows the performance results obtained by the general matrix-matrix multiply PBLAS routine PDGEMM on the SGI Origin 2000. These results have been obtained for the matrix-matrix multiply operation $C \leftarrow C + AB$, where A, B, and C are square matrices of order N.

You can control the number of threads to which the MP BLAS are spawned by setting the environment variable MP_SET_NUMTHREADS. Otherwise, libblas_mp uses all processors on the machine.

Comparing the data in Tables 5, 20, and 21, we can see that the PBLAS routine PDGEMM achieves 80–90% of the per processor DGEMM performance on the SGI O2K.

We then repeated these same timings during "non-dedicated" time via batch queues and "qsub" at ARL. These results are contained in tables 22 and 23.

Table 20: Speed in Mflop/s for matrix-matrix multiply, DEDICATED time (SGI O2K)

	Process grid	Block size	Values of N 1000	2000	3000	4000	5000
CEWES MSRC							
Message-Passing	2×2	64	1018	1044	1140	1142	1118
	2×4	64	1924	1963	2091	2187	2127
	4×4	64	3209	3941	3999	3989	3929
	4×8	64	6306	7249	7752	7798	7585
Threaded	4	64	1174	1172	1202	1185	1229
	8	64	2097	2298	2407	2343	–
	16	64	2775	4143	4493	4421	–
	32	64	5671	5666	6935	7840	7726
ARL MSRC							
Message-Passing	2×2	64	1122	1091	1068	1148	1118
	2×4	64	1989	2017	2127	2229	2171
	4×4	64	3583	3954	4029	4042	3907
	4×8	64	3085	7302	7747	7847	7498
Threaded	4	64	1201	1165	1200	1206	–
	8	64	2183	2119	2346	2325	–
	16	64	3074	4224	4545	4510	–
	32	64	1266	4188	7033	8022	7618

We show timing numbers for message-passing (i.e. ScaLAPACK) and threaded (i.e. blas_mp) matrix multiplication. Here we see that ScaLAPACK is slightly slower than the threaded implementation for large problems and/or small numbers of processors. This is to be expected. As previously mentioned, two factors govern parallel matrix multiplication speed: communication and computation. Communication effects will be seen primarily in the case where the work per processor is low (i.e., a small problem size, or a fixed problem size with many processors), whereas computation speed will affect all problem sizes and dictate the asymptotic performance.

Let us briefly summarize the advantages/drawbacks of each technique. The communication inherent in the threaded BLAS will likely be controlled by the hardware. This allows for more efficient communication, as the latencies inherit in software communication (eg., MPI interface) are not added to each communication. This implies threaded matrix multiply will have a slight advantage over message passing for small problem sizes, as its communication will be faster.

The main difference in the algorithms, however, is the data decomposition. Without access to the source code for the threaded BLAS, we can at best guess what matrix decomposition is being employed there. Our understanding is that all matrices start out on one processor. Then, the most probable case is that the threaded BLAS simply partition the columns of the result matrix C among the processors, and then farm out the corresponding sections of B and the entire matrix A to all processors.

Table 21: Speed in Mflop/s for the PBLAS matrix-matrix multiply routine PDGEMM, DEDICATED time (SGI O2K)

	Process grid	Block size	Values of N				
			6000	7000	8000	9000	10000
CEWES MSRC							
Message-passing	2×2	64	1161	1121	1160	–	–
	2×4	64	2254	2186	2260	2211	2300
	4×4	64	4018	4012	4068	4276	4494
	4×8	64	7783	7644	7834	8290	8482
ARL MSRC							
Message-passing	2×2	64	1171	1133			
	2×4	64	2263	2188	2295	2237	2295
	4×4	64	4119	4129	4134	4317	4571
	4×8	64	7841	7778	7860	8405	8612

ScaLAPACK, on the other hand, starts with all three matrices distributed and then uses an outer-product based algorithm, where column panels of A and row panels of B are sent among the processes, which then do a series of rank-K updates to produce C.

This *may* give the threaded BLAS a slight advantage in computation speed, since an outer-product multiply (ScaLAPACK) requires more memory writes than an inner-product multiply (probably what blas_mp uses). This would explain why the threaded BLAS are slightly faster for large problem sizes.

The outer product multiply has two advantages, due to the way it performs the communication. First, it will have better load balance because the messages being sent are smaller (less time waiting until computation may begin). More importantly, its communication is pipelined, which significantly reduces communication costs. This effect should increase with the number of nodes. Therefore, we see ScaLAPACK being faster than the blas_mp for the cases where the number of nodes is large and the problem size is not large enough for the computation term to dominate.

The above analysis holds true for the block sizes that we use in LU. As we increase the blocking factor, the distribution used in ScaLAPACK becomes more like that proposed for the threaded BLAS. To confirm this idea, we ran a few cases with larger blocking factors and, as shown above, performance was indeed improved. As before, however, these large blocking factors are usually not used in applications (such as LU), so we do not concentrate on them here.

Table 22: Speed in Mflop/s for matrix-matrix multiply on SGI O2K, NON-DEDICATED time (SGI O2K)

	Process grid	Block size	Values of N				
			1000	2000	3000	4000	5000
ARL MSRC							
Message-Passing	2×2	64	1096	1062	1101	1157	1137
	2×2	128	1175	1162	1158	812	1183
	2×2	256	1186	941	1160	1132	1195
	2×4	64	1967	1985	2127	2228	2185
	4×4	64	3605	3879	4045	4061	3959
	4×4	128	3845	4194	4193	4280	4244
	4×8	64	6138	7289	7865	7897	7826
Threaded	4	64	–	–	–	–	–
	8	64	–	–	–	–	–
	16	64	–	–	–	–	–
	32	64	3290	4882	7343	7645	7813
ASC MSRC							
Message-Passing	2×2	64	1059	980	1035	1022	1062
	2×4	64	1857	1883	1719	2022	1519
	4×4	64	830	1390	2018	2213	2241
	4×8	64	–	–	–	–	–

Table 23: Speed in Mflop/s for matrix-matrix multiply, NON-DEDICATED time (SGI O2K)

	Process grid	Block size	Values of N				
			6000	7000	8000	9000	10000
ARL MSRC							
Message-passing	2×2	64	1120	736	–	–	–
	2×2	128	1198	1209	–	–	–
	2×2	256	1201	1187	–	–	–
	2×4	64	2162	1425	1011	2300	2323
	4×4	64	4125	4110	3927	4485	4562
	4×4	128	4311	3358	2928	4798	4780
	4×8	64	7980	7915	7635	8606	8674
Threaded	4	64	1181	1235	1116	867	–
	8	64	2362	2251	2220	2183	–
	16	64	3237	3227	4060	4326	–
	32	64	8409	8909	8498	8017	8465
ASC MSRC							
Message-Passing	2×2	64	–	–	–	–	–
	2×4	64	1799	1590	1545	–	–
	4×4	64	2385	1359	1414	2707	1879
	4×8	64	–	–	–	–	–

5.2 Parallel LU factorization/solve performance

Table 24 illustrates the speed of the ScaLAPACK driver routine PDGESV using distributed-memory BLAS (PBLAS) versus the LAPACK routine DGESV using the multi-threaded BLAS. PDGESV or DGESV solves a square linear system of order N by LU factorization with partial row pivoting of a real matrix. For all timings, 64-bit floating-point arithmetic was used. Thus, double precision timings are reported. The distribution block size is also used as the partitioning unit for the computation and communication phases. These timings were performed during dedicated time, and yet variances were still encountered.

One obvious inconsistency is the poor PDGESV performance for small problem sizes when we used two dimensional grids (eg. the 2×8 and 4×8 grid sizes). This is easily explained: two dimensional grids are required for scalability. However, they perform poorly for small problem sizes due to increased latency-bound communication along the columns of the process grid. To demonstrate that this was the problem, table 26 shows the timings for small problem sizes on the appropriate 1D grid. Notice that even though these timings were done during non-dedicated time and executed interactively, they have superior performance for small problem sizes.

Table 24: Speed in Mflop/s of LU factor/solve for square matrices of order N, DEDICATED time (SGI O2K)

	Process Grid	Block Size	Values of N								
			1000	2000	3000	4000	5000	7500	10000	12500	15000
CEWES MSRC											
Message-Passing	1×4	64	532	705	773	821	859	917	970	966	–
	1×8	64	709	1113	1280	1379	1451	1581	1743	1752	1818
	2×8	64	541	891	2074	2313	2575	2938	3229	3339	3373
	4×8	64	623	1485	2888	2541	4190	5011	5571	5902	6096
Threaded	4	64	759	810	895	923	937	–	–	–	–
	8	64	1024	1145	1349	1456	–	–	–	–	–
	16	64	1234	1500	1805	1956	–	–	–	–	–
	32	64	1132	1453	1948	1990	2273	–	–	–	–
ARL MSRC											
Message-Passing	1×4	64	567	747	804	865	888	936	1016	1004	–
	1×8	64	681	1169	1333	1409	1467	1637	1824	1811	1889
	2×8	64	368	904	2113	2379	2613	3013	3317	3436	3584
	4×8	64	561	1559	3140	2592	4323	5071	5691	6051	6345
Threaded	4	64	758	814	897	932	–	–	–	–	–
	8	64	1058	1176	1377	1464	–	–	–	–	–
	16	64	1249	1518	1840	1988	–	–	–	–	–
	32	64	1176	1488	2008	2077	2283	–	–	–	–

The main thing to note in these timings is that ScaLAPACK maintains scalability as

Table 25: Speed in Mflop/s of LU factor/solve for square matrices of order N, NON-DEDICATED time (SGI O2K)

	Process Grid	Block Size	Values of N								
			1000	2000	3000	4000	5000	7500	10000	12500	15000
ARL MSRC											
Message-Passing	1×4	64	585	741	816	699	774	922	1012	1036	–
	2×4	64	472	708	1315	1442	1561	1514	1567	1738	1839
	1×8	64	–	–	–	–	–	–	–	–	–
	2×8	64	554	907	2127	2382	2669	2744	2890	3292	3487
	4×8	64	818	1554	3101	2601	4327	5240	5786	6152	5751
Threaded	4	64	–	–	–	–	–	–	–	–	–
	8	64	–	–	–	–	–	1480	1719	1680	1789
	16	64	–	–	–	–	–	–	–	–	–
	32	64	1091	1446	1965	2096	2426	–	3316	–	–
ASC MSRC											
Message-Passing	1×4	64	532	730	793	853	873	963	–	–	–
	2×4	64	579	732	1356	1458	1555	1701	1826	1883	–
	1×8	64	751	1174	1352	1475	1568	1734	1878	1912	–
	2×8	64	705	948	2232	2506	2750	3139	3427	3601	3701
	1×16	64	877	1595	1974	2225	2434	2825	3154	3264	3503
	4×8	64	820	1872	3129	2667	4369	5257	5844	6213	6469

the problem size and number of processors is increased, while the threaded code does not. This is because ScaLAPACK knows precisely what operation is being performed, and is thus better able to schedule communication (i.e., make use of pipelining, avoid unnecessary communication, etc).

Table 26: Speed in Mflop/s of ScaLAPACK PDGESV for square matrices of order N, 1D process grids, NON-DEDICATED time (SGI O2K)

	Process Grid	Block Size	Values of N								
			1000	2000	3000	4000	5000	7500	10000	12500	15000
CEWES MSRC											
Message passing	1×8	64	560	768	1316	1564	1784	2174	2444	2644	–
	1×16	64	566	876	1676	2098	2455	3192	3745	4192	4504
	1×32	64	555	919	1858	2445	2970	4020	5041	4173	6539

5.3 Discussion

The SGI Cray Scientific Library (SCSL) has recently become available. SCSL is tuned for the R10000 Origin systems and will be the replacement for SGI's CompLib and Cray's LIBSCI. As future work, we would like to conduct performance evaluations of this library as it would contain a machine-specific optimized version of ScaLAPACK for the SGI Origin 2000.

6 SGI Power Challenge Array

We present performance data on the SGI Power Challenge Array for the netlib version of ScaLAPACK using the distributed-memory BLAS (PBLAS), and the netlib version of LAPACK using the SGI multi-threaded BLAS (-lblas_mp). The message-passing library used was the SGI MPI v3.0 library. The optimized SGI BLAS in (-lblas) were used for the ScaLAPACK timings and the SGI MP BLAS in (-lblas_mp) were used for the LAPACK timings.

6.1 Parallel matrix-matrix multiply performance

We perform comparison timings of the distributed-memory PBLAS matrix matrix multiply routine PDGEMM using the SGI BLAS (-lblas) versus the multi-threaded DGEMM in SGI BLAS MP (-lblas_mp).

Asymptotically, the performance of the PBLAS will rest on the performance of the corresponding BLAS routine. For smaller problem sizes, lower order costs, primarily communication, will cause performance loss. We therefore see that effects due to BLACS optimality will be seen mostly in the smaller problem sizes.

Timings were performed during "dedicated" time. Variances in timings were encountered.

Tables 27 and 28 show the performance results obtained by the general matrix-matrix multiply PBLAS routine PDGEMM and the multi-threaded SGI MP BLAS routine DGEMM on the SGI Power Challenge Array. These results have been obtained for the matrix-matrix multiply operation $C \leftarrow C + A\,B$, where A, B, and C are square matrices of order N.

You can control the number of threads to which the MP BLAS are spawned by setting the environment variable MP_SET_NUMTHREADS. Otherwise, libblas_mp uses all processors on the machine.

Comparing the data in Tables 5, 27, and 28, we can see that the PBLAS routine PDGEMM achieves 67–81% of the per processor DGEMM performance on the SGI PCA.

The overall analysis of threaded versus message passing should be the same for the Power Challenge Array as it was for Origin 2000. However, the number of processors available to us is less, so the lack of scalability is not as evident for these problem sizes.

Table 27: Speed in Mflop/s for matrix-matrix multiply, DEDICATED time (SGI PCA)

	Process grid	Block size	Values of N				
			1000	2000	3000	4000	5000
CEWES MSRC							
Message-passing	2×2	64	1087	1005	1016	1037	1027
	2×4	64	2033	2025	1926	1852	1888
	4×4	64	2700	3708	3208	3350	3367
Threaded	4	64	1239	1236	1236	1236	1274
	8	64	2389	2469	2422	2441	–
	16	64	4194	4559	4326	4924	–
ARL MSRC							
Message-passing	2×2	64	931	883	893	914	–
	2×4	64	1752	1711	1719	1750	1731
	4×4	64	–	–	–	–	–
Threaded	4	64	1001	1027	1017	1015	1042
	8	64	269	254	267	266	–
	16	64	–	–	–	–	–

Table 28: Speed in Mflop/s for matrix-matrix multiply, DEDICATED time (SGI PCA)

	Process grid	Block size	Values of N				
			6000	7000	8000	9000	10000
CEWES MSRC							
Message-passing	2×2	64	1030	1014	949	–	–
	2×4	64	1951	1910	1806	1919	1896
	4×4	64	3511	3533	3315	3602	3543
ARL MSRC							
Message-passing	2×2	64	–	–	–	–	–
	2×4	64	1752	–	–	–	–
	4×4	64	–	–	–	–	–

6.2 Parallel LU factorization/solve performance

Table 29 illustrates the speed of the ScaLAPACK driver routine PDGESV using distributed-memory BLAS (PBLAS) versus the LAPACK routine DGESV using the multi-threaded BLAS. PDGESV/DGESV solves a square linear system of order N by LU factorization with partial row pivoting of a real matrix. For all timings, 64-bit floating-point arithmetic was used. Thus, double precision timings are reported. The distribution block size is also used as the partitioning unit for the computation and communication phases.

Timings were performed during "dedicated" time. Variances in timings were encountered.

The overall analysis of threaded versus message passing should be the same for the power challenge array as it was for origin 2000. However, the number of processors available to us is less, so the lack of scalability is not as evident for these problem sizes.

Table 29: Speed in Mflop/s of LU factor/solve for square matrices of order N, DEDICATED time (SGI PCA)

	Process Grid	Block Size	Values of N								
			1000	2000	3000	4000	5000	7500	10000	12500	15000
CEWES MSRC											
Message-Passing	1×4	64	535	643	684	721	745	775	–	–	–
	1×8	64	660	671	900	1166	1248	1281	1415	1360	1379
	2×8	64	335	1247	1537	1705	1943	2319	2504	2567	2608
Threaded	4	64	696	707	831	870	890	–	–	–	–
	8	64	951	978	1241	1342	–	–	–	–	–
	16	64	800	1116	1585	1745	–	–	–	–	–
ARL MSRC											
Message-Passing	1×4	64	528	622	640	665	681	710	–	–	–
	1×8	64	701	1105	1108	1145	1187	1271	1290	–	–
	2×8	64	–	–	–	–	–	–	–	–	–
Threaded	4	64	592	604	710	746	764	–	–	–	–
	8	64	225	224	234	237	–	–	–	–	–
	16	64	–	–	–	–	–	–	–	–	–

7 Conclusions and future work

Of all the machines, the Cray T3E appeared to have the most repeatable timings, both for dedicated and non-dedicated runs. The IBM SP also did not seem strongly affected by whether the machine was dedicated or not; however, timings were never more than roughly repeatable on this platform. Also, the IBM SP would occasionally show large dips in performance.

The SGI Origin 2000 timings were probably the least repeatable. The timings reported in this paper for a particular grid size were always obtained in one run, but often that run was selected as the best out of several runs (by best, we mean the run with the smoothest (i.e., steadily increasing) performance curve). Even so, these runs are far from smooth.

On the Cray T3E, the routines present in LIBSCI were always slightly faster than the equivalent from netlib ScaLAPACK. Because this performance win decreased with problem size, it is probably due to a lower order term such as communication. In particular, LIBSCI's use of shmem probably allows for faster communication than the publicly available MPI-based implementation.

On the IBM SP, there was no noticeable difference between the matrix multiply supplied by PESSL and that supplied by the netlib version of ScaLAPACK. PESSL had a much faster version of LU. The only difference between the two routines as far as the user is concerned is the form of the factorization, which is more complex in the PESSL implementation. Since LU is often heavily optimized for benchmarking purposes, a performance comparison of the Cholesky factorization was also conducted. The PESSL Cholesky factorization also consistently outperformed the netlib implementation, but not to the extent of LU.

On the SGI Origin 2000 and the SGI Power Challenge Array, threaded codes showed a slight advantage over ScaLAPACK for the matrix multiply. For LU, threaded codes did well for small problem sizes and/or small numbers of nodes, but were not as scalable as their ScaLAPACK equivalents. Due to time constraints, we have not presented threaded results for many of the larger problem sizes for the LU factorization. Future work should extend the threaded timings to these larger problem sizes to ensure that the general tendencies we have seen so far continue throughout the performance curve.

Bibliography

[1] 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*, Society for Industrial and Applied Mathematics, Philadelphia, PA, second ed., 1995.

[2] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.

[3] J. Dongarra and R. C. Whaley, *A user's guide to the BLACS v1.1*, Computer Science Dept. Technical Report CS-95-281, University of Tennessee, Knoxville, TN, 1995. (Also LAPACK Working Note #94).

[4] J. J. Dongarra, J. Du Croz, I. S. Duff, and S. Hammarling, *A set of Level 3 Basic Linear Algebra Subprograms*, ACM Trans. Math. Soft., 16 (1990), pp. 1–17.

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

[6] C. L. Lawson, R. J. Hanson, D. Kincaid, and F. T. Krogh, *Basic linear algebra subprograms for Fortran usage*, ACM Trans. Math. Soft., 5 (1979), pp. 308–323.