SIAM J. SCI. STAT. COMPUT.
Vol. 7, No. 1, January 1986

TIMELY COMMUNICATIONS

This is the first paper to be published under the new "timely communications" policy for the SIAM Journal on Scientific and Statistical Computing. Papers that have significant timely content and do not exceed five pages automatically will be considered for a separate section of this journal with an accelerated reviewing process. It will be possible for the note to appear approximately six months after the date of acceptance. The editors are pleased to launch this section with a note on the use of a new computer organization that is likely to be the start of a revolution in scientific and statistical computation.

IMPLEMENTING DENSE LINEAR ALGEBRA ALGORITHMS USING MULTITASKING ON THE CRAY X-MP-4 (OR APPROACHING THE GIGAFLOP)*

JACK J. DONGARRA† AND TOM HEWITT‡

Abstract. This note describes some experiments on simple, dense linear algebra algorithms. These experiments show that the CRAY X-MP is capable of small-grain multitasking arising from standard implementations of *LU* and Cholesky decomposition. The implementation described here provides the "fastest" execution rate for *LU* decomposition, 718 MFLOPS for a matrix of order 1000.

Introduction. Over the past few months we have been experimenting with some simple linear algebra algorithms on the CRAY X-MP-4 multiprocessor. The CRAY X-MP family is a general-purpose multiprocessor system. It inherits the basic vector functions of CRAY-1S, with major architectural improvements for each individual processor. The CRAY X-MP-4 system is a four-processor model housed in a physical chassis identical to the CRAY-1S. The system can be used to perform simultaneous scalar and vector processing of either independent job streams or independent tasks within one job. Hardware in the X-MP enables multiple processors to be applied to a single Fortran program in a timely and coordinated manner.

All processors share a central bipolar memory (of up to 16 million words), organized in 64 interleaved memory banks. Each processor has four memory ports: two for vector fetches, one for vector stores, and one for independent I/O operations. In other words, the total memory bandwidth of the four processors is up to sixteen times that of the CRAY-1S system.

This note describes results obtained from three experiments: *LU* decomposition based on matrix-vector operations, *LU* based on a "best" implementation for the architecture, and an implementation of Cholesky decomposition based on matrix-vector operations.

***LU* decomposition.** The versions of *LU* and Cholesky factorization, used here, are based on matrix-vector modules that allow for a high level of granularity, permitting high performance in a number of different environments, see [2].

The algorithm designed to give the "best" performance on the X-MP architecture is worth noting. It is based on standard Gaussian elimination with partial pivoting. The algorithm is organized such that it zeros out three columns (below the diagonal)

* Received by the editors August 17, 1985.

† Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois 60439. The work of this author was supported in part by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under contract W-31-109-Eng-38.

‡ CRAY Research Inc., Chippewa Falls, Wisconsin 54701.

of the matrix, then applies these transformations to the rest of the matrix. The application of the transformations to the remainder of the matrix is split up among the processors. An assembly language kernel is used to apply the three pivot rows for simultaneous row operations. This reduces memory traffic and allows a single processor of an X-MP to obtain its theoretical maximum sustainable computation rate of 198 MFLOPS. The assembly language kernel was multitasked in the experiment among two, three, and four CPUs. Fortran-callable assembly language synchronization subroutines were used. These require less than half a microsecond for synchronization.

The process of finding the three pivot rows was not multitasked. However, it was written as an assembly language kernel to reduce overhead for small problems. As the factorization proceeds, the size of the relevant vector and submatrix decreases. The final 15×15 block of the reduction was performed by an unrolled [2] version of standard Gaussian elimination. This portion is entirely single-threaded Fortran and runs at two to three times the speed of the Fortran which has not been unrolled. Synchronization was accomplished through a fork-and-join mechanism using the cluster (shared) registers of the X-MP.

Listed in Table 1 are the results of the experiments; speedups range from 1.3 on a problem of size 50×50 to 3.8 for a matrix of order 1000×1000. The performance ranges from 97 MFLOPS to an impressive 718 MFLOPS for four processors on a 1000×1000 matrix. For small problems, startup times dominate the overall performance. It is interesting to note that a system of equations of order 1000 can now be factored and solved in under a second! (The same problem on a VAX 11/780 would take roughly two hours to complete.)

TABLE 1
High-performance LU decomposition.

	MFLOPS #processors				Speedup over 1 processor #processors		
Order	1	2	3	4	2	3	4
50	97	124	135	145	1.28	1.39	1.49
100	145	230	281	325	1.64	1.94	2.24
200	172	330	426	526	1.81	2.47	3.05
400	183	353	507	652	1.93	2.77	3.56
600	186	364	535	689	1.96	2.87	3.70
1000	188	372	550	718	1.98	2.92	3.81

For comparison we give in Table 2 the results for *LU* decomposition based on matrix-vector operations. The parallelism here is gained by simply splitting the matrix-vector operation across the processors (see [1] for details on the algorithm). The

TABLE 2
LU based on matrix-vector operations.

Order	MFLOPS (4 processors)	Ratio of algorithms from Table 1/Table 2
50	57	2.54
100	167	1.95
200	343	1.53
400	537	1.21
600	608	1.13
1000	675	1.06

matrix-vector modules have been coded in assembly language. The same number of operations is performed here as in the version that gives "best" performance.

As we can see, for lower order matrix problems, the high-performance algorithm is far more efficient. For large problems, however, there is not too much difference; for the matrix of order 1000 there is only a difference of 6% in the running time. This fact shows one of the strengths in using the matrix-vector design for algorithms of this nature. The matrix-vector modules can be easily changed as we go to a different architecture, but the basic algorithm is unaltered.

Cholesky decomposition. A version of Cholesky decomposition for a symmetric positive definite matrix was implemented on the CRAY X-MP-4 based on matrix-vector routines (see [2] for algorithm details). Table 3 gives the performance of that routine when run on 1 and 4 processors. In addition, the algorithm was reorganized to compute information necessary to perform four steps of the decomposition during the same step. This results in four independent "full size" matrix-vector multiplications. This information is listed in the last column of Table 3.

TABLE 3
Cholesky decomposition based on matrix-vector operations.

Order	MFLOPS 1 processor	MFLOPS 4 processors (MV split)	Speedup	MFLOPS 4 processors (4 MV ops)
50	59	52	.88	97
100	117	154	1.32	264
200	163	345	2.12	460
400	184	544	2.96	631
600	189	623	3.30	683
1000	193	689	3.58	733

Synchronization was accomplished via the X-MP shared registers and semaphores. Multiprocessing overhead in this case is largely the result of code changes and of the breaking of a large piece of work into smaller pieces—each of which has essentially the same startup time as the original large piece.

The apparent overhead for multitasking small problems is largely caused by the following items:

1. Single-threaded portions of work. Finding pivot rows, scaling rows of the matrix, the scalar square root in Cholesky decomposition—these are all of great importance in the current study.
2. Less parallel work per processor in the computational kernel. Even in scalar operations the X-MP uses a considerable amount of parallelism. In vector mode, 100 floating-point operations can be performed in the time it takes to call a subroutine and scores of flops in the time required to initialize a DO loop.
3. Additional memory-bank conflicts. The single CPU times were run with no activity in the other three CPUs. When four processors are active, additional conflicts will occur, though in this case the effect is small as the matrix-vector operations are conflict-insensitive on the X-MP.
4. Synchronization overhead. This is dominated by the time a CPU is waiting for all of its vector memory references to be completed. Thus, it is generally wise to complete all vector memory references before synchronization, as it eliminates the possibility of an inter-CPU memory race condition.

Conclusions. The CRAY X-MP is capable of small granularity multitasking. For large problems that are both vectorized and parallelized, performance of the X-MP-4 should be in the range of 400 to 700 MFLOPS. For smaller problems the startup time of the parallel processes can dominate the execution time. This feature becomes more important as more processors are applied to the problem.

Note. Since the time this work was conducted, CRAY Inc. Research has introduced micro-tasking, which provides a mechanism for the user to conveniently, and with low overhead, exploit small-granularity parallelism from Fortran programs with compiler directives.

REFERENCES

[1] STEVE S. CHEN, JACK J. DONGARRA AND CHRISTOPHER C. HSIUNG, *Multiprocessing for linear algebra algorithms on the* CRAY X-MP-2: *Experiences with small granularity*, J. Parallel and Distributed Computing, 1 (1984), pp. 22-31.

[2] JACK J. DONGARRA AND STANLEY C. EISENSTAT, *Squeezing the most out of an algorithm in* CRAY *Fortran*, ACM Trans. Math. Software, 10 (1984), pp. 221-230.