Application Brief
Intel Scientific Computers

LINPACK and EISPACK on the Intel iPSC™

Roger Golliver, Bill Hughey, & Cleve Moler
Intel Scientific Computers

LINPACK [1] is a library of FORTRAN subroutines for analyzing and solving systems of linear algebraic equations and least squares problems involving dense, stored matrices. EISPACK [2], [3] is library of FORTRAN subroutines for solving various matrix eigenvalue problems.

One way in which matrix computations can be done on the iPSC™ is to use the standard, sequential LINPACK and EISPACK subroutines simultaneously on each of the individual processors. The parallelism is obtained by solving many different matrix problems at the same time, with no communication between the processors. The memory available on the standard model of the iPSC provides storage for matrices of order up to about 180 on each processor.

Larger matrix computations can be carried out on the iPSC by distributing the data and the arithmetic across many processors [4]. The algorithms in LINPACK are organized so that almost all the innermost loops access the columns, rather than the rows, of the matrices. A frequent operation involves adding scalar multiples of one column in a matrix to each of the other columns. This column orientation is carried over to the parallel generalizations of the algorithms. A matrix of order n is distributed over p processors by storing n/p columns in each processor. If n is not exactly divisible by p, then some processors get one more column than others. A double precision matrix of order 1000 can be stored on 32 processors, for example, with 32 columns in the first 8 processors and 31 columns in the remaining 24 processors. A matrix of order 2000 can be stored on 128 processors. The iPSC-VX has over twice as much memory per node, and hence can handle a matrix of order 1000 on 16 processors.

A typical LINPACK or EISPACK factorization algorithm has an outer loop on an index, say k, which goes across the columns of the matrix. In the k-th step of the parallel generalization, the processor which holds the k-th column serves as the root processor for that step. The root processor does a short computation involving only the k-th column. The result of that computation is broadcast to all the other processors, using a "spanning tree" communication pattern that makes effective use of the hypercube interconnection between the processors. Each processor then uses this new data to modify its own columns. As the outer loop proceeds, different processors serve as root at different steps.

For large problems, most of the time in these parallel algorithms is spent doing arithmetic; only a small portion of the time is spent waiting for the root processor or participating in the communication. The idle time and communication costs are asymptomatically negligible - as the order of the matrix is increased, the portion of processor time attributable to overhead goes to zero.

This column oriented approach to parallel matrix computation has several significant advantages:

- The parallel algorithms retain the numerical stability properties of the underlying sequential algorithms.
- Available software can be readily modified to implement the parallel algorithms.
- Larger problems can be solved by increasing the number of processors, or the memory on the individual processors.
- Operations on entire columns provide long vectors for efficient use of the VX processor.
- Communication and load balancing overheads are asymptomatically negligible as problem size increases.

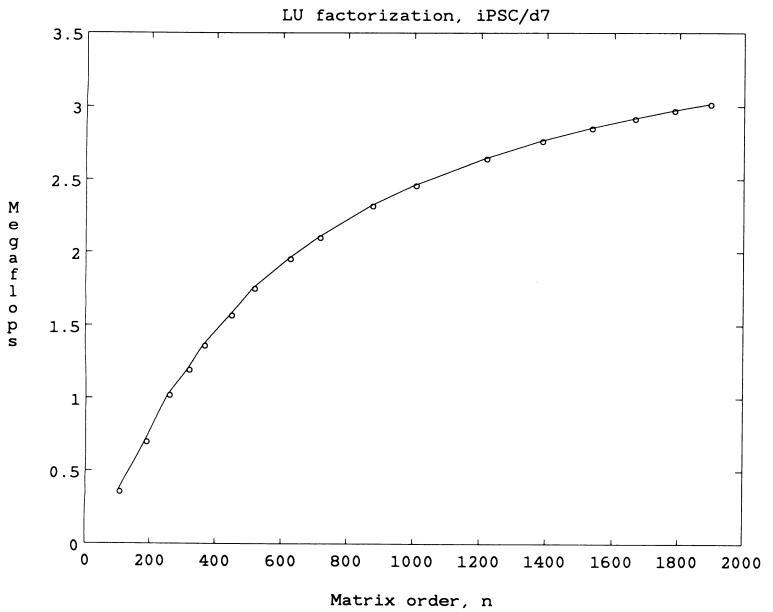
iPSC PERFORMANCE

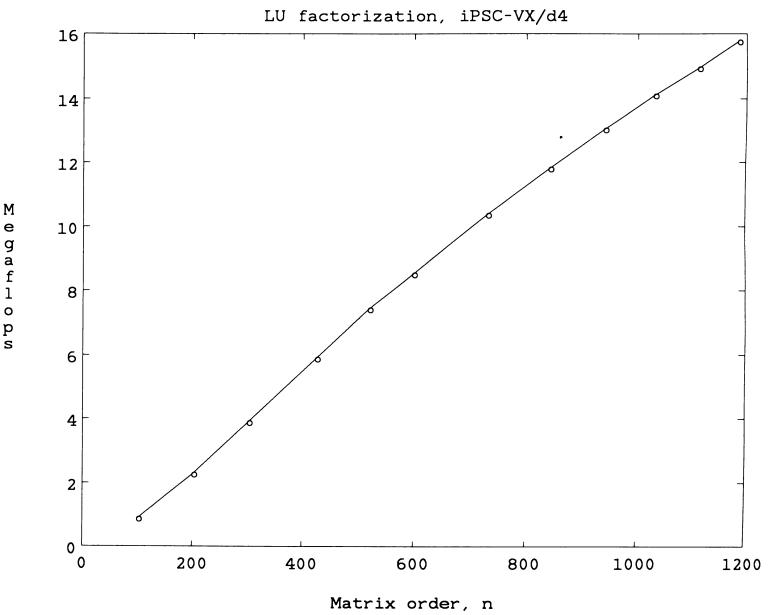
The most important subroutine in LINPACK is DGEFA, which uses Gaussian elimination with partial pivoting to compute the LU factorization of a double precision, general matrix. The parallel generalization of this routine uses exactly the same algorithm on a matrix distributed by columns across the processors. It has the same roundoff error and accuracy properties. Its performance, measured in megaflops, on two different models of the iPSC is shown in the two graphs.

The first graph is for an iPSC/d7, which has 128 processors using 80287 math coprocessors. For the inner loop of DGEFA, which is the vector operation DAXPY, this coprocessor can approach 0.03 megaflops. For very large problems, 128 processors approach 128 times 0.03, or about 3.8 megaflops. The available memory limits the problem size to about $n = 2000$. The performance obtained is a little over 3 megaflops, which is over 75% of peak performance.

The second graph is for an iPSC-VX/d4, which has 16 processors and a vector floating point unit associated with each processor. A single VX processor would approach 2.66 megaflops on very long DAXPY's. As the problem size is increased to n = 1200, which is the maximum allowed by current memory size, the processing rate approaches 16 megaflops for 16 nodes, or about 37% of peak performance. This percentage will be increased by adding more memory per node and by improving the performance of the hypercube message passing hardware.

Jack Dongarra of Argonne National Laboratory has continued the collection of the performance data originated in the LINPACK Users' Guide and periodically issues a technical report which has come to be known as "The LINPACK Benchmark." Dongarra's data comes primarily from running unaltered versions of DGEFA and its companion DGESL. Our performance results do not quite belong in Dongarra's collection because we have altered the source code to incorporate message passing.





References:

[1] J. J. Dongarra, J. R. Bunch, C. B. Moler and G. W. Stewart, *LINPACK Users' Guide*, SIAM Publications, 368 pp., 1979.

[2] B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema and C. B. Moler, *Matrix Eigensystem Routines -- EISPACK User's Guide*, Springer-Verlag, Lecture Notes in Computer Science No. 6, Second Edition, 552 pp., 1976.

[3] B. S. Garbow, J. M. Boyle, J. J. Dongarra and C. B. Moler, *Matrix Eigensystem Routines -- EISPACK Guide Extension*, Springer Verlag, Lecture Notes in Computer Science No. 51, 343 pp., 1977.

[4] Cleve Moler, "*Matrix Computation on Distributed Memory Multiprocessors*," Hypercube Multiprocessors 1986, (Michael Heath, editor), 181-195, SIAM, 1986.