Conclusions

Temperature stratification can be a significant source of error during hot-wire measurements in low-speed, open-return wind tunnels operating in an enclosed room. Errors of 4–5% in mean velocity measurements were observed at low (≈ 5 m/s) flow speeds in the tunnel of Fig. 1. The stratification could be eliminated by thoroughly mixing the air just upstream of the wind-tunnel inlet. Since the facility is equipped with adequate turbulence management (honeycombs and screens), the mixing was accomplished without reduction of flow quality.

Reference

[1]Loehrke, R. I and Nagib, H. M., "Experiments on Management of Freestream Turbulence," AGARD-R-598, 1972.

Parallelism in the Solution of Large Full Linear Systems Using a Matrix Partition Approach

Robert Morris Hintz*
General Microelectronics Corporation, San Diego, California

SOLUTION-TIME and data-storage requirements for the solution of full linear systems have become increasingly unwieldy with problem growth. Solution time for a 20,000-order complex system is about 1 day at 250 megaflops (millions of floating-point operations per second), and the data-storage requirements exceed 800 million words. Real systems require about a quarter of the solution time of complex and half the storage. Solution time is approximately proportional to the cube of problem size and storage requirements to the square of problem size. Thus, larger problems become even more intractable at an alarming rate.

Two parallel solution algorithms are proposed that can greatly decrease the solution time of full systems by employing multiple processes in the problem solution. Furthermore, the algorithms lend themselves to extremely efficient operation on disk resident problems because of benign demands on data transfer to and from disk. The basis for the algorithms is LU decomposition with partial row pivoting.[1] It is recommended that pivoting be used only sparingly because it can dramatically retard the performance of the algorithms. The parallel algorithms presented here do not include pivoting strategy.

The linear system to be solved is given by

$$Ax = b \tag{1}$$

where A is a full coefficient matrix, x and b are vectors, and the solution x is required, given A and b. Using LU decomposition, a nonsingular A matrix may be factored into the product of a lower and upper triangle matrix, as given in Eq. (2)[1]:

$$A = LU \tag{2}$$

where solution x then can be determined from the two-stage forward reduction (or forward elimination[1]) and back substitution.[1] The great bulk of the numerical effort is in the factoring algorithm, and multiple solutions may be easily obtained once A is factored.

The focus here is the decomposition of the A matrix using the matrix partition approach discussed in Ref. 2. The decomposition is implemented by partitioning the A matrix into square submatrices. Boundary partitions at the lower and right edges of A are, in general, not square. The partition-factoring algorithm[2] is restated as Eqs. (3a) and (3b). The diagonal partition submatrices L_{ii} and U_{jj} of

$$L_{ii}U_{ij} = A_{ij} - \sum_{k=1}^{i-1} L_{ik}U_{kj} \qquad i \le j \tag{3a}$$

$$L_{ij}U_{jj} = A_{ij} - \sum_{k=1}^{j-1} L_{ik}U_{kj} \qquad i > j \tag{3b}$$

Equations (3) are triangular, and all other submatrices are either square or rectangular. The solution of Eqs. (3) consists of a series of matrix multiplication and subtraction operations (to form $A - LU$ terms), followed the solution of a triangular system. More specifically, when $i = j$, the triangular system of Eq. (3a) is a factoring solution; when $i \neq j$, the triangular system of Eq. (3a) is a forward reduction, and the triangular system of Eq. (3b) is the transpose of a forward reduction.

The maximum partition size of the submatrices of Eqs. (3) is limited only by the amount of main data memory available to a computer at execution time. The technique thus lends itself to mapping large disk resident problems into an assemblage of in-core problems. Systems such as the order 20,000 problem previously discussed can be solved in a few million words of memory with negligible loss to input/output (I/O) when using concurrent computation and I/O.

In fact, a principal advantage of this technique is the very large computational effort generated per I/O request for a disk resident problem. Algorithm I/O can be structured to yield a full partition matrix multiply for each partition matrix read.[2] In other words, for complex arithmetic, $8m^3$ floating-point operations can be generated per $2m^2$ words read for a partition size of m. This translates to $4m$ floating-point operations per word of I/O, which gives a tremendous advantage to an I/O device supplying input to a computer concurrently with computation. These I/O efficiencies, coupled with the inherent parallelism of the partition algorithms, makes them extremely attractive.

The parallel factoring algorithm is developed here for both order n processes and order n^2 processes. Throughput of an order n^3 process operated on by order n processes is order n^2; an order n^3 process operated on by order n^2 processes is order n. The parallel algorithms are developed with each parallel set of processes collected in a group denoted as a step. All operations within a step may be executed in parallel with minor exceptions as noted.

First, the algorithm is parallelized with order n processes. The solution steps 1–n are given as follows.

Step 1: $i = 1$ in Eq. (3a), $j = 1$ in Eq. (3b) ($2n - 1$ processes):

$$L_{11}U_{11} = A_{11}$$

then, for the first row and column of partitions,

$$L_{11}U_{1j} = A_{1j}, \qquad j = 2,3,\ldots,n$$

$$L_{i1}U_{11} = A_{i1}, \qquad i = 2,3,\ldots,n$$

Step 2: $i = 2$ in Eq. (3a), $j = 2$ in Eq. (3b)† ($2n - 3$ processes):

$$L_{22}U_{2j} = A_{2j} - L_{21}U_{1j}, \qquad j = 2,3,\ldots,n$$

$$L_{i2}U_{22} = A_{i2} - L_{i1}U_{12}, \qquad i = 3,4,\ldots,n$$

Received March 1, 1988; revision received Sept. 6, 1988.

*Senior Applications Engineer.

Step 3: $i = 3$ in Eq. (3a), $j = 3$ in Eq. (3b)† ($2n - 5$ processes):

$$L_{33}U_{3j} = A_{3j} - L_{31}U_{1j} - L_{32}U_{2j}, \qquad j = 3,4,\ldots,n$$

$$L_{i3}U_{33} = A_{i3} - L_{i1}U_{13} - L_{i2}U_{23}, \qquad i = 4,5,\ldots,n$$

Step 4: $i = 4$ in Eq. (3a), $j = 4$ in Eq. (3b)† ($2n - 7$ processes):

$$L_{44}U_{4j} = A_{4j} - L_{41}U_{1j} - L_{42}U_{2j} - L_{43}U_{3j}, \quad j = 4,5,\ldots,n$$

$$L_{i4}U_{44} = A_{i4} - L_{i1}U_{14} - L_{i2}U_{24} - L_{i3}U_{34}, \qquad i = 5,6,\ldots,n$$

Step n: $i = n$ in Eq. (3a) (single process):

$$L_{nn}U_{nn} = A_{nn} - L_{n1}U_{1n} - L_{n2}U_{2n} - L_{n3}U_{3n} - \cdots - L_{n,n-1}U_{n-1,n}$$

Note that, upon completion of the matrix multiply and subtracts in steps 2–4, the process for the first value of j (i.e., $j = 2$ for step 2, $j = 3$ for step 3, etc.) must complete before the remaining processes can complete.

Additional parallelism may be invoked in the preceding algorithm by reorganizing it to implement order n^2 processes in a recursive fashion. The startup procedure is repeated as above for step 1. At step 2, $(n-1)^2$ processes are employed to compute the partial sum $A_{ij}^{(2)}$ for all $i,j \geq 2$. At the end of the step, the second row and column of processes then is completed with factoring and forward reduction operations as shown, and the next step is begun with $(n-2)^2$ processes. The recursive algorithm proceeds, completing a row and column of partitions at each step and releasing those processes, until finishing with a single process at step n.

Solution steps 1–n are now given for a recursive factoring algorithm with order n^2 processes.

Step 1: $2n - 1$ processes:

$$L_{11}U_{11} = A_{11}$$

then, for the first row and column of partitions,

$$L_{11}U_{1j} = A_{1j}, \qquad j = 2,3,\ldots,n$$

$$L_{i1}U_{11} = A_{i1}, \qquad i = 2,3,\ldots,n$$

Step 2: $(n-1)^2$ processes:

$$A_{ij}^{(2)} = A_{ij} - L_{i1}U_{1j} \quad \text{for all} \quad i,j \geq 2$$

then,

$$L_{22}U_{22} = A_{22}^{(2)}$$

then, for the second row and column of partitions,

$$L_{22}U_{2j} = A_{2j}^{(2)}, \qquad j = 3,4,\ldots,n$$

$$L_{i2}U_{22} = A_{i2}^{(2)}, \qquad i = 3,4,\ldots,n$$

Step 3: $(n-2)^2$ processes:

$$A_{ij}^{(3)} = A_{ij}^{(2)} - L_{i2}U_{2j} \quad \text{for all} \quad i,j \geq 3$$

then,

$$L_{33}U_{33} = A_{33}^{(3)}$$

then, for the third row and column of partitions,

$$L_{33}U_{3j} = A_{3j}^{(3)}, \qquad j = 4,5,\ldots,n$$

$$L_{i3}U_{33} = A_{i3}^{(3)}, \qquad i = 4,5,\ldots,n$$

Step 4: $(n-3)^2$ processes:

$$A_{ij}^{(4)} = A_{ij}^{(3)} - L_{i3}U_{3j} \quad \text{for all} \quad i,j \geq 4$$

then,

$$L_{44}U_{44} = A_{44}^{(4)}$$

then, for the fourth row and column of partitions,

$$L_{44}U_{4j} = A_{4j}^{(4)}, \quad j = 5,6,\ldots,n$$

$$L_{i4}U_{44} = A_{i4}^{(4)}, \quad i = 5,6,\ldots,n$$

$$\vdots$$

Step k: $(n-k+1)^2$ processes:

$$A_{ij}^{(k)} = A_{ij}^{(k-1)} - L_{i,k-1}U_{k-1,j}, \qquad i,j \geq k$$

$$L_{kk}U_{kk} = A_{kk}^{(k)}$$

$$L_{kk}U_{kj} = A_{kj}^{(k)}, \quad j = k+1,k+2,\ldots,n, \quad k = 5,6,\ldots,n-1$$

$$L_{ik}U_{kk} = A_{ik}^{(k)}, \quad i = k+1,k+2,\ldots,n$$

$$\vdots$$

Step n: single process:

$$A_{nn}^{(n)} = A_{nn}^{(n-1)} - L_{n,n-1}U_{n-1,n}$$

$$L_{nn}U_{nn} = A_{nn}^{(n)}$$

It is interesting to examine the degenerate form of the n^2 process factoring algorithm for a partition size of 1. The matrix partition size can be reduced in the limit to 1, at which time the partition matrix factoring algorithm becomes an algorithm operating on the elements of the matrix itself. The order n^2 algorithm, when used as an elemental algorithm, thus reduces the throughput for factoring a matrix to the same order as an inner product. The factoring process is thus a massively parallel algorithm when formulated as stated.

References

[1]Golub, G. H. and Van Loan, C. F., *Matrix Computations,* Johns Hopkins Univ. Press, Baltimore, MD, 1983.

[2]Hintz, R. M., "FPS-164/MAX General Purpose Math Package for Larger Than Core Applications," *Proceedings of 1987 ARRAY Conference,* Montreal, Canada, April 1987, pp. 61–69.