ZASTOSOWANIA MATEMATYKI
APPLICATIONES MATHEMATICAE
XVIII, 1 (1983), p. 97-105

J. S. KOWALIK (Pullman, Wash.), S. P. KUMAR (Coral Gables, Fla.)
and E. R. KAMGNIA (Urbana-Champaign, Ill.)

AN IMPLEMENTATION OF THE FAST GIVENS TRANSFORMATIONS ON A MIMD COMPUTER

1. Introduction. By introducing simultaneous execution it is possible to design computers that can operate faster than conventional machines. Several super fast computers utilizing this principle have already been built. Among them CRAY-1, CDC STAR, ILLIAC IV, and STARAN-IV are perhaps best known. Recently a new parallel computer called *HEP* (*heterogeneous element processor*) has been made available. This is a MIMD machine (multiple-instruction multiple-data) or, more specifically, a resource sharing computer as defined by Flynn [1]. HEP is able to execute different instructions upon multiple data streams simultaneously, unlike vector or array processors which can handle many data streams at the same time but can only execute the same instruction on all the data streams. Fig. 1 illustrates this capability of a MIMD computer.

The HEP machine consists of process execution modules (PEM's), program memory, and data memory modules. Each PEM has 2,048 internal general-purpose registers. All data and instruction words are 64 bits long. In the future configuration there will be up to 16 PEM's and data memory modules. All data memory modules will be accessible to all PEM's via a high-speed data switch network. Currently, the HEP machine consists of one PEM (1), program memory, and a set of general purpose registers. One PEM executes a maximum of 10 million instructions per second (MIPS). It can concurrently process from 1 to 8 instruction streams at the rate of 1.25 MIPS per each stream. It is significant to point out that the HEP machine does not degrade its performance whether it operates on vector or scalar operands. This is in contrast to array machines whose performance is sensitive to the length of processed vectors.

From the logical point of view (or a programmer view) the current HEP configuration can be regarded as composed of $p \leqslant 8$ interconnected processors which cooperate and simultaneously process up to 8 streams

(1) The HEP computer as described in the paper has been now extended to a four PEM configuration. [Note added in proof]

of instructions. A more detailed description of HEP and its target configuration can be found in [8].

In this paper we investigate the fast Givens transformations which can be used to solve square and rectangular systems of algebraic linear equations. First we assume that there is available an unlimited number

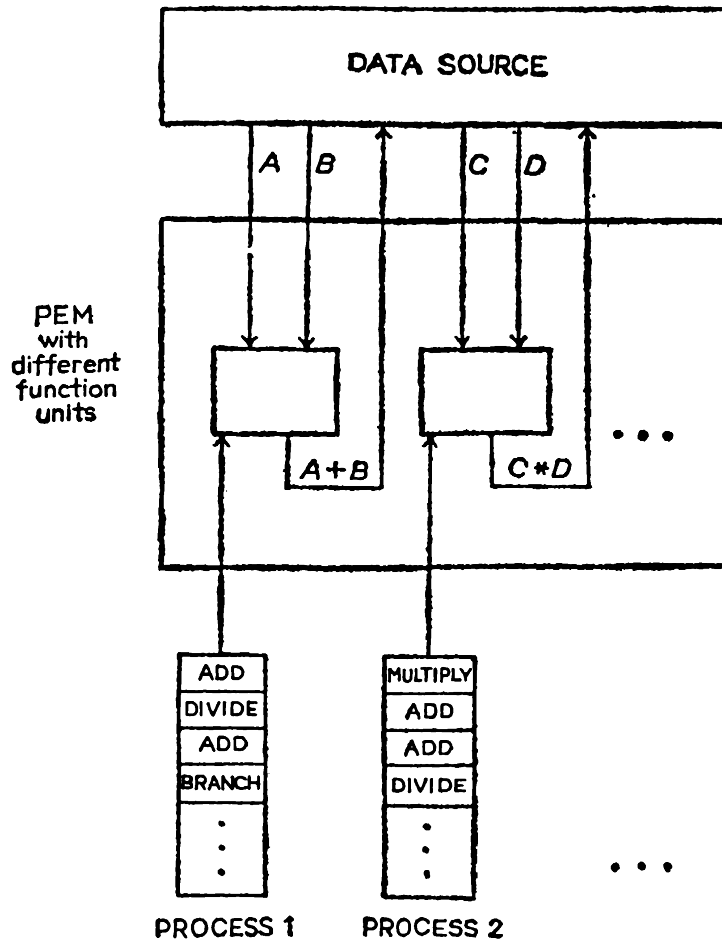

Fig. 1. A MIMD computer

of processors and show that a system of n linear equations $Ax = b$ can be solved in $T_p = 11n-15$ steps using $p = O(n^2)$ processors, where one step is equivalent to one arithmetic operation. This part of our analysis is akin to earlier results by Sameh and Kuck ([6], [7]). Then we postulate only $O(n)$ processors and present a practical parallel algorithm which was programmed and executed on the HEP parallel computer.

2. Fast Givens transformation. To describe the use of fast Givens transformations, consider first a $(2\times k)$-matrix B which is scaled by a diagonal matrix $D = \mathrm{diag}(d_1, d_2)$, $d_1, d_2 > 0$, and A such that $A = D^{1/2}B$.

If we wish to eliminate the element A_{21} of A, we can form the matrix G and use the standard Givens transformation

$$GA = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} A,$$

where $c = A_{11}/r$, $s = A_{21}/r$, and $r = \pm(A_{11}^2+A_{21}^2)^{1/2}$.

Alternatively, we may work with D and B separately and use an orthogonal transformation on $D^{1/2}B$ implicitly. This approach, due to Gentleman [2], is commonly known as the *fast Givens transformation.* One step of this transformation consists of the following calculations:

$$a = \frac{B_{21}}{B_{11}}, \quad \beta = \frac{d_2}{d_1}a, \quad \text{and} \quad t = a\beta.$$

Now the matrices D and B are replaced by

$$\tilde{D} = \operatorname{diag}(\tilde{d}_1, \tilde{d}_2) = \frac{1}{1+t}D$$

and

(1)
$$\tilde{B} = \begin{bmatrix} 1 & \beta \\ -a & 1 \end{bmatrix} B = HB.$$

The scaled plane rotation matrix H is the fast Givens transformation matrix. If G is the standard Givens rotation matrix applied to A, then the relation between G and H is

(2)
$$G = \tilde{D}^{1/2}HD^{-1/2}.$$

The matrix A is replaced by $GA = \tilde{D}^{1/2}HD^{-1/2}D^{1/2}B = \tilde{D}^{1/2}\tilde{B}$. It is straightforward to verify that G in (2) is an orthogonal matrix.

From (1) we get

(3)
$$\begin{aligned} \tilde{B}_{11} &= B_{11}(1+t), \\ \tilde{B}_{21} &= 0, \\ \tilde{B}_{1i} = B_{1i}+\beta B_{2i}, \quad \tilde{B}_{2i} &= B_{2i}-aB_{1i} \quad (2 \leqslant i \leqslant k). \end{aligned}$$

As can be seen, the transformation requires approximately $2k$ multiplications and additions, and square roots are not needed.

Rewriting the formulas for $\tilde{d}_1$ and $\tilde{d}_2$ we obtain

$$\tilde{d}_1 = \frac{d_1^2 B_{11}^2}{d_1 B_{11}^2 + d_2 B_{21}^2} \quad \text{and} \quad \tilde{d}_2 = \frac{d_2 d_1 B_{11}^2}{d_1 B_{11}^2 + d_2 B_{21}^2}.$$

To preserve numerical stability of the factorization, an operation equivalent to row interchange is required. This results in two alternative

forms of H. One is as in (1) and the other is

$$H = \begin{bmatrix} 1/\beta & 1 \\ -1 & 1/\alpha \end{bmatrix}. \tag{4}$$

The transformation matrix H defined by (1) is applied if $|c| > |s|$ or, equivalently,

$$d_1 B_{11}^2 > d_2 B_{21}^2. \tag{5}$$

In this case $\tilde{B}_{11}$ is computed from (3):

$$\tilde{B}_{11} = B_{11}\left(1 + \frac{d_2 B_{21}^2}{d_1 B_{11}^2}\right).$$

If (5) does not hold, then the transformation matrix is defined by (4) and

$$\tilde{B}_{11} = B_{21}\left(1 + \frac{d_1 B_{11}^2}{d_2 B_{21}^2}\right).$$

The formulas to calculate $\tilde{d}_1$, $\tilde{d}_2$, $\tilde{B}_{1i}$, and $\tilde{B}_{2i}$ for $2 \leqslant i \leqslant k$ are appropriately modified.

Furthermore, periodic rescaling of D and H has been implemented to avoid underflow or overflow in D. The reader interested in details of the numerical implementation of the fast Givens method is referred to [4] and [5].

To solve the system of linear equations $A\boldsymbol{x} = \boldsymbol{b}$, where A is a non-singular matrix $n \times n$, we proceed as follows:

(i) factorize the augmented matrix $[A, \boldsymbol{b}]$ using fast Givens transformations, i.e., compute $\hat{D}$, R, and $\hat{\boldsymbol{b}}$ such that $Q[A, \boldsymbol{b}] = \hat{D}^{1/2}[R, \hat{\boldsymbol{b}}]$, where R is upper-triangular, Q is the product of all orthogonal transformations required to triangularize A, and $\hat{D}$ is diagonal;

(ii) solve $R\boldsymbol{x} = \hat{\boldsymbol{b}}$.

Note that square roots are never computed; neither is explicitly computed the matrix Q.

3. An algorithm using $O(n^2)$ processors. It can be easily shown that having $2(k+1)$ processors which perform arithmetic calculations simultaneously we can eliminate one element of B (such as B_{21}) in 4 steps.

Let A be a non-singular square matrix of size n. The sequential Givens transformations annihilate the subdiagonal non-zero elements of A one at a time, while preserving all previously introduced zeros. This annihilation process can be performed columnwise and for each column l of A as many as $n-l$ plane rotations are required. In a parallel algorithm more than one Givens rotation can be performed at the same time.

One possible scheme of annihilation is shown in Fig. 2. Numbers in Fig. 2 show the sequence in which the elements of A are annihilated.

For instance, the fifth transformation consists of 3 simultaneous rotations and eliminates 3 subdiagonal elements of A. The total number of parallel transformations is $2n-3$, e.g., for $n=6$ we have 9 transformations.

Thus the orthogonal factor is $Q = Q_{2n-3}Q_{2n-4}\cdots Q_2Q_1$, where Q_k, $k=1,2,\ldots,2n-3$, is a parallel transformation representing one or

*					
1	*				
2	3	*			
3	4	(5)	*		
4	(5)	6	7	*	
(5)	6	7	8	9	*

Fig. 2. Annihilation pattern for a square matrix, $n = 6$. Elements in the circles identify the transformation requiring the maximum number of processors

*					
1	*				
1	2	*			
1	3	4	*		
2	4	5	6	*	
3	5	6	7	8	*

Fig. 3. A slightly better elimination pattern

more simultaneous rotations. Since the entire annihilation scheme requires $2n-3$ parallel transformations, the total time to calculate $\hat{D}$, R, and $\hat{b}$ by this parallel Givens algorithm is $T_p = 4(2n-3) = 8n-12$ steps. Additional time may be required to scale periodically the matrices D and H.

The elimination pattern shown in Fig. 2 is not optimal. The reader can verify that we can use one less transformation to triangularize a square matrix of size $n = 6$ (Fig. 3).

The sequential algorithm requires $T_1 = 4\cdot 3^{-1}n^3+O(n^2)$ steps. The speedup S_p achieved by the parallel algorithm is $S_p = T_1/T_p = 6^{-1}n^2+ +O(n)$. To calculate the maximum number of processors needed to factorize A, we assume for simplicity that n is even. It follows from the annihilation pattern (see Fig. 2) that

$$p = 2\sum_{t=0}^{n/2-1}(n+1-t) = \frac{3}{4}n^2+\frac{3}{2}n.$$

To factorize $[A, b]$ in the time T_p, the number of required processors is $3\cdot 4^{-1}n^2+5\cdot 2^{-1}n$. The efficiency for sufficiently large values of n is $E_p = S_p/p = .22$. This low efficiency is due to the limited use of processors in the initial and the final stages of the Givens transformation process.

We now return to the problem of solving the linear set of equations $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, where A is $n \times n$. After $[\boldsymbol{A}, \boldsymbol{b}]$ is factorized, we have to solve the triangular system $\boldsymbol{R}\boldsymbol{x} = \hat{\boldsymbol{b}}$, and this requires $3(n-1)$ steps using $n-1$ processors, if we apply, for instance, the column sweep algorithm [3]. Thus, the total time for solving $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ by this parallel fast Givens transformation followed by the column sweep method is approximately $T_p = (8n-12)+(3n-3) = 11n-15$.

4. An algorithm using only $O(n)$ processors. We focus our attention on the case where the number of available processors is $p = \lceil (n-1)/2 \rceil$ and assume that Givens rotations are performed sequentially. In fact, if Givens rotations are sequential, it is not possible to use more processors than $\lceil (n-1)/2 \rceil$ since each rotation requires two rows of A. Let T_j^i be a computationally indivisible task defined by $T_j^i = GIVENS(i, j)$, where $GIVENS(i, j)$ is a subroutine call that rotates rows i and j and eliminates the (j, i)-th element of A. With each task T_j^i there are associated two, possibly overlapping, ordered sets of memory cells, the domain D_T and the range R_T. When T_j^i is initiated, it reads the values stored in its domain and writes values into its range cells. We say that two tasks T and $\hat{T}$ are *non-interfering* if either (i) or (ii) holds:

(i) T is a temporal successor or predecessor of $\hat{T}$,

(ii) $R_T \cap R_{\hat{T}} = R_T \cap D_{\hat{T}} = D_T \cap R_{\hat{T}} = \emptyset$ (empty).

A set of tasks is said to *contain mutually non-interfering tasks* if T_j^i and T_l^k are non-interfering for all indices i, j, l, and k which belong to the set. The transformation matrix Q_k, $1 \leqslant k \leqslant 2n-3$, consists of tasks

$$\{T_j^i \mid 1 \leqslant i \leqslant n-1,\ i < j \leqslant n,\ i+j = k+2,\ k = 1, 2, \ldots, 2n-3\}$$

which are mutually non-interfering and can be executed in parallel for each k.

Any algorithm triangularizing the matrix A must include the condition that no new non-zeros are reintroduced in the process of rotation. This means that all tasks have to obey the following precedence constraint: T_j^l is completed before T_j^i can begin for all $l < i$. The annihilation scheme we implement follows the pattern of Fig. 2 which means that another precedence constraint is enforced: T_j^i is completed before T_l^i can begin for all $l > j$. The pair consisting of the set of tasks T_j^i and the partial order representing temporal precedence constraints is called a *task system* which can be conveniently visualized as a directed acyclic graph with no redundant (transitive) arcs. The task system of the Givens transformation problem is shown in Fig. 4.

The longest (critical) path in this graph is

$$S = \{T_2^1, T_3^1, \ldots, T_n^1, T_n^2, \ldots, T_n^{n-1}\},$$

and since the time length of T_j^i is $L(T_j^i) = 4(n-i+1)+7$ steps, we get $L(S) = 6n^2+8n-25$ steps, where one step corresponds to one arithmetic operation. The value of $L(S)$ determines the best possible execution time of the considered task system. This execution time can be achieved with $p = \lceil (n-1)/2 \rceil$ processors. There are several possible schedules and the

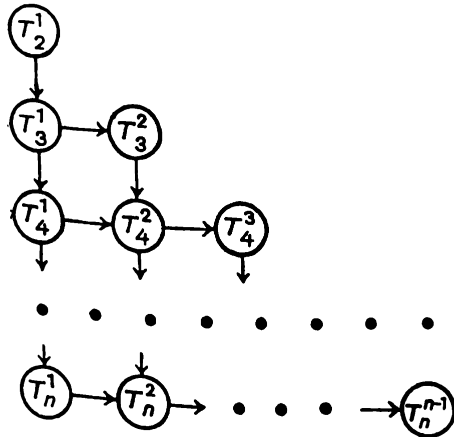

Fig. 4. Precedence graph for tasks T_j^i

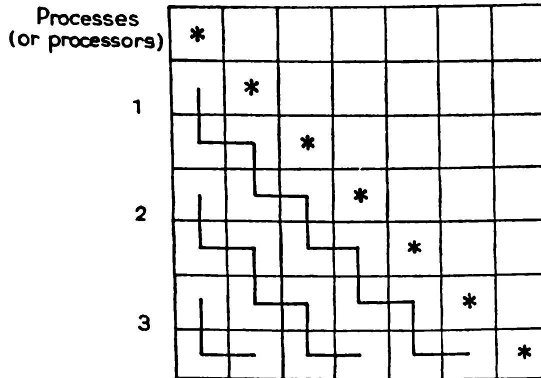

Fig. 5. Scheduling scheme for $p = \lceil (7-1)/2 \rceil = 3$

one we have selected is shown in Fig. 5. In this schedule the processors are assigned to the tasks as follows:

processor 1: tasks $T_2^1, T_3^1, T_3^2, \ldots, T_n^{n-1}$,

processor 2: tasks $T_4^1, T_5^1, T_5^2, \ldots, T_n^{n-3}$

and, in general, the processor P_j, $1 \leqslant j \leqslant \lceil (n-1)/2 \rceil$, executes the tasks $T_{2j}^1, T_{2j+1}^1, T_{2j+1}^2, \ldots, T_n^{n-2j+1}$. To enforce the partial order temporal constraints the processors have to be synchronized. For example, the processor P_1 should not start T_4^2 before the processor P_2 completed T_4^1, and the processor P_2 should not start T_4^1 before the processor P_1 completed T_3^1. It is also necessary to create new processes at different times. For example, for the scheduling scheme presented in Fig. 5 the process 1 starts first and executes the tasks T_2^1 and T_3^1. Now the second process is created since T_4^1 and T_3^2 can be executed in parallel.

The main synchronization mechanism in the HEP Fortran language is that of the well-known producer-consumer synchronization using busy waiting. This mechanism is realized via the so-called asynchronous variables, the names of which begin with a special symbol. Each such variable can be in a state *FULL* or *EMPTY*. The reading may only take place when the state of a variable is *FULL* and writing (assignment) may only take place when the state is *EMPTY*. Writing an asynchronous variable always sets the state to *FULL* and reading sets it to *EMPTY*. The producer-consumer synchronization consists essentially of the two

conditional actions: (i) wait until *EMPTY* and then write, and (ii) wait until *FULL* and then read.

The second aspect of synchronization which is creating a new process is accomplished in HEP Fortran by an instruction called *CREATE*. *CREATE* initiates a parallel instruction stream which will terminate when a *RETURN* statement is encountered. Using these programming facilities we can fully control the execution of a MIMD parallel program.

TABLE 1. Actual (A) and predicted (P) speedup and efficiency. Execution time is measured in seconds

n	p	T_1	T_p	S_p	E_p	
5	2	.0036	.0025	1.44	.72	A
				1.40	.70	P
7	3	.0087	.0045	1.93	.64	A
				1.83	.61	P
9	4	.0168	.0072	2.33	.58	A
				2.27	.57	P
10	5	.0222	.0087	2.55	.51	A
				2.50	.50	P
11	5	.0286	.0105	2.72	.54	A
				2.72	.54	P
13	6	.0448	.0146	3.07	.51	A
				3.16	.52	P
15	7	.0660	.0194	3.34	.47	A
				3.61	.51	P
17	8	.0927	.0256	3.62	.45	A
				4.01	.50	P

The discussed Givens method was programmed and run on the HEP computer. Since for this machine $1 \leqslant p \leqslant 8$ and in our algorithm we use $p = \lceil (n-1)/2 \rceil$ processors, we have triangularized several matrices with $n \leqslant 17$. Table 1 summarizes the predicted and actual values of the speedup S_p and efficiency E_p. The predicted values are calculated from the relations

$$S_p = \frac{T_1}{T_p} = \frac{4 \cdot 3^{-1} n^3 + O(n^2)}{6n^2 + O(n)} \simeq \frac{2}{9} n \quad \text{and} \quad E_p = \frac{S_p}{p} = \frac{4}{9} \frac{n}{n-1}.$$

The actual values were obtained by timing the HEP runs.

The small differences between the predicted and actual values of S_p and E_p can be attributed to machine time required for *CREATE* statements, extra synchronization variables used in the program, DO-loop control, data dependent scaling operations in the Givens subroutine, and other program overhead items which are not accounted for in our formulas.

In the present HEP configuration it is not meaningful to consider a loss of time due to the processor-to-processor or processor-to-memory communication.

Acknowledgement. We express our thanks to Dr. R. Lord of the Washington State University for his helpful comments and to the reviewer whose remarks have helped to improve the quality of the paper.

References

[1] M. J. Flynn, *Some computer organizations and their effectiveness*, IEEE Trans. Computers C21 (1972), p. 948-960.

[2] W. M. Gentleman, *Least squares computation by Givens transformations without square roots*, J. Inst. Math. Appl. 12 (1973), p. 329-336.

[3] D. J. Kuck, *A survey of parallel machine organization and programming*, Comput. Surveys 9 (1977), p. 29-59.

[4] C. L. Lawson and R. J. Hanson, *Solving least squares problems*, New York 1974.

[5] – D. R. Kincaid and F. T. Krogh, *Basic linear algebra subprograms for FORTRAN usage*, Report SAND 77-0898, Sandia Laboratories, Albuquerque 1977.

[6] A. H. Sameh and D. J. Kuck, *Linear system solvers for parallel computers*, Internal report, Dept. of Computer Sci., Univ. of Illinois, Urbana-Champaign 1974.

[7] – *On stable linear system solvers*, J. Assoc. Comput. Mach. 25 (1978), p. 81-91.

[8] B. J. Smith, *A pipelined shared resource MIMD computer*, p. 6-8 in: Proc. 1978 Internat. Conference on Parallel Processing, Long Beach, Cal., 1978.

WASHINGTON STATE UNIVERSITY
PULLMAN, WASH. 99164

UNIVERSITY OF ILLINOIS
URBANA-CHAMPAIGN, ILL. 61801

UNIVERSITY OF MIAMI
CORAL GABLES, FLA. 33124

Received on 15. 4. 1980;
revised version on 3. 11. 1980