Pergamon

Computers Math. Applic. Vol. 31, No. 4/5, pp. 19–26, 1996

0898-1221/96 $15.00 + 0.00

0898-1221(95)00212-X

Massively Parallel Poisson and QR Factorization Solvers

M. LUCKÁ
Institute for Control Theory and Robotics, Slovak Academy of Sciences
Dúbravská cesta 9, 842 37 Bratislava, Slovak Republik
utrrluck@savba.sk

M. VAJTERŠIC
Institute of Informatics, Slovak Academy of Sciences
Dúbravská cesta 9, 840 00 Bratislava, P.O. Box 56, Slovak Republic
kaifmava@savba.sk

E. VIKTORINOVÁ
Institute for Control Theory and Robotics, Slovak Academy of Sciences
Dúbravská cesta 9, 842 37 Bratislava, Slovak Republik
utrrevka@savba.sk

Abstract—The paper brings a massively parallel Poisson solver for rectangle domain and parallel algorithms for computation of QR factorization of a dense matrix A by means of Householder reflections and Givens rotations. The computer model under consideration is a SIMD mesh-connected toroidal $n \times n$ processor array.

The Dirichlet problem is replaced by its finite-difference analog on an $M \times N$ ($M+1, N$ are powers of two) grid. The algorithm is composed of parallel fast sine transform and cyclic odd-even reduction blocks and runs in a fully parallel fashion. Its computational complexity is $O(MN \log L/n^2)$, where $L = \max(M+1, N)$. A parallel proposal of QR factorization by the Householder method zeros all subdiagonal elements in each column and updates all elements of the given submatrix in parallel. For the second method with Givens rotations, the parallel scheme of the Sameh and Kuck was chosen where the disjoint rotations can be computed simultaneously.

The algorithms were coded in MPF and MPL parallel programming languages and results of computational experiments on the MasPar MP-1 system are also presented.

Keywords—Parallel linear algebra, Fast sine transform, Odd-even reduction, QR decomposition, Massively SIMD-type computer arrays.

1. INTRODUCTION

According to observations with running basic linear algebra algorithms on massively parallel SIMD arrays [1,2], only an appropriate tailoring of the algorithm to the machine topology and a careful coding in a machine-close programming language can lead to acceptable performance results. Otherwise, mostly dramatic losses in efficiency may be a consequence. The paper presents massively parallel algorithms for two frequently treated problems of parallel linear algebra.

The first problem concerns solving the discretized Poisson equation with Dirichlet boundary conditions. Parallel direct as well as iterative methods have been already examined for solving this model problem on various architectures. Among the direct methods, e.g., parallel schemes for

This work has been done during a stay of the authors at the Institute for Software Technology and Parallel Systems, University of Vienna (H. P. Zima, director). The opportunity of using the MasPar MP-1 computer for the experiments is highly acknowledged.

the matrix decomposition, marching and cyclic semi-iterative methods, the method of conjugate gradients and the block Stiefel's method have been designed for a linearly connected MIMD processor array. Serious attention has been paid to implementation of multigrid methods for this boundary value problem on both SIMD and MIMD computer types. Rather comprehensive information about the parallel approaches for solving the finite-difference approximations to this problem gives the monograph [3]. Despite that there has been a lot of work done in development of algorithms for solving this problem, our motivation was to solve it by considering the massively parallel concept.

The QR factorization is one of the most important matrix decompositions in numerical linear algebra. It is used when solving least squares problems, SVD decomposition, etc. It factorizes a general matrix into an orthogonal matrix and an upper-triangular matrix. There are two standard ways to compute QR decomposition, the Givens and Householder methods. Both methods work by applying a sequence of orthogonal transformations, either elementary rotations in the Givens method or elementary reflections in the Householder method.

One of the most commonly spread parallel machines with massive parallelism is the MP-1 of the MasPar company [4]. Its massively parallel SIMD array has been used for design and implementation of our algorithms. The paper is organized in five sections. Section 2 brings description of the algorithmical and implementation background. The parallel implementation is described in Section 3. The computational results are presented in Section 4. The concluding Section 5 summarizes the results and presents some outlooks.

2. THEORETICAL AND TECHNICAL PRELIMINARIES

2.1. Poisson Equation

The model problem under discussion concerns the Poisson equation

$$\Delta u(x,y) = f(x,y) \tag{1}$$

on a rectangular region $R = \langle 0,a\rangle \times \langle 0,b\rangle$ with known values $u(x,y) = g(x,y)$ on the boundary of R.

The discretization of the above equation according to the familiar five-point stencil on an $M \times N$ grid leads to the block-tridiagonal linear algebraic system

$$G\bar{u} = \bar{v} \tag{2}$$

of the size $MN \times MN$, where not more than five nonzero entries are placed in a row or column of G. (Throughout the paper, we assume that the parameters M and N are of the form $M = 2^q - 1$ and $N = 2^p$ (for some positive integers p, $q \geq 2$). Further, L will denote $\max(M+1, N)$ and $\log L$ will be used for $\log_2 L$.) The complexity for serial solution of the above linear system ranges from $O(M^2N)$ through $O(MN \log L)$ to asymptotically optimal value $O(MN)$. The former estimation is relevant for elimination-based methods, the value $O(MN \log L)$ is valid for Fourier-based approaches, and the latter one corresponds to the multigrid approach.

For our solver, the Fourier-based method has been chosen which relates to the orthogonal decomposition of the matrix G [2].

The computation of the solution vector $\bar{u}$ follows in four phases:

(P0) creation of the right-hand side vector $\bar{v}$ of the size MN;
(P1) transformation of $\bar{v}$ by a block matrix whose blocks correspond to one-dimensional *sine* transform of the length M each;
(P2) solution of M positively definite tridiagonal Toeplitz systems with right-hand sides of the length N which result from the phase (P1);
(P3) final *sine* transformations of M vectors which are composed from the solutions of (P2).

The phase (P1) computes the vectors

$$\tilde{v}_j = S\bar{v}_j, \qquad j = 1, 2, \ldots, N \tag{3}$$

where the $M \times M$ matrix S is defined by $S_{ij} = \sqrt{\frac{2}{M+1}} \sin\left(\frac{ij\pi}{M+1}\right)$ and $\bar{v}_j$ is the j^{th} M-block of the vector $\bar{v}$ from (2). These computations follow via the discrete *sine* transform which is defined for general M-vectors $\bar{x}, \bar{y}$ (we remind $M = 2^q - 1$) by

$$y_i = \sum_{j=1}^{M} x_j \sin\left(\frac{ij\pi}{M+1}\right), \qquad i = 1, 2, \ldots, M. \tag{4}$$

We will adapt the efficient approach from [2] consisting of a precalculation, FFT evaluation and a post-calculation stage.

The work in phase (P2) consists of solving the systems

$$T_{\lambda_i}\bar{x}_i = \bar{y}_i, \qquad i = 1, 2, \ldots, M \tag{5}$$

where $T_{\lambda_i} = (-\rho^2, \lambda_i, -\rho^2)$ are tridiagonal Toeplitz matrices of the order N,

$$\lambda_i = 2\left(1 + \frac{1}{\rho^2}\right) - 2\cos\left(\frac{i\pi}{M+1}\right) \tag{6}$$

with $\rho = (b/(M+1))/(a/(N+1))$ and the N-vector $\bar{y}_i$ contains the i^{th} components of the transformed vectors from the phase (P1). These systems are solved by a proper adaptation of the cyclic odd-even method for this case, where not just one but M systems from (5) can be solved concurrently.

2.2. QR Factorization

QR factorization decomposes a general matrix A of size $M \times N$ into an orthogonal matrix Q of the size $M \times M$ and an upper-triangular matrix R of the size $M \times N$ by

$$Q^{\top}A = R. \tag{7}$$

Both Householder and Givens methods work by applying a sequence of orthogonal transformations to A, zeroing out elements in an order given by a rule

$$A^{(k)} = \left(P^{(k)}\right)^{\top} A^{(k-1)}. \tag{8}$$

In Householder's method, one whole subdiagonal column of elements at the same time is annihilated. This is done with Householder reflection P which is an orthogonal transformation $P = I - 2\bar{v}\bar{v}^{\top}$. For a given vector $\bar{x}$, it is possible to choose $\bar{v}$ such that $P\bar{x}$ is parallel to $\bar{e}_1$ (the first unit vector). Let $\bar{a}_1$ be the first column of A and define

$$\bar{v}_1 = \mu_1\bar{u}_1, \qquad \bar{u}_1^{\top} = (a_{11} - s_1, a_{21}, \ldots, a_{N1}),$$

where (9)

$$s_1 = \pm\left(\bar{a}_1^{\top}\bar{a}_1\right)^{1/2}, \qquad \mu_1 = \left(2s_1^2 - 2a_{11}s_1\right)^{-1/2}.$$

In the Givens method, one element at a time is set to zero by plane Givens rotation. A plane rotation matrix P_{ij} is equal to the identity matrix except for $p_{ii} = p_{jj} = c_{ij} = \cos\Theta$, $p_{ij} = -p_{ji} = s_{ij} = \sin\Theta$ and $c_{ij}^2 + s_{ij}^2 = 1$.

Each transformation affects only rows i and j, and Θ is chosen so that the rotation annihilates one subdiagonal element of A [5].

2.3. MasPar Features

MasPar MP-1 is a massively parallel SIMD computer system with a very large number of simple processors (at least 1024) all executing the same program at once. Its Data Parallel Unit (DPU) contains a two-dimensional matrix of Processor Elements (PE) that does all the parallel processing and is controlled by Array Control Unit (ACU). Each PE in PE-array has its own memory and high performance registers. It receives an instruction from ACU and then executes it, but on different data. There are two possibilities of communications between PEs: on a straight line or by means of the global router. The former are called X-net communications and enable one to send (or receive) the data element any distance in one of the basic eight directions. They are significantly faster than global router communications, but the latter are more general and have no built-in direction of the communications. This data parallel programming model simplifies the programming of local memory parallel architectures by associating a processor with every data element in a computation (at least conceptually).

MasPar Fortran language is an implementation of Fortran 90 for the MP-1, that provides data parallel control through its array extensions and intrinsic functions [6]. The MPL (MasPar Parallel Language) is an adaptation of C-language for this computer [7].

3. THE PARALLEL IMPLEMENTATION

3.1. Poisson Solver

For the execution of phases (P0)–(P3), an $n \times n$ (n being a power of two, $n \leq M+1$, N) array MP-1 of MasPar will be considered. The assignment of grid points to processors will be in the virtual manner, i.e., one physical processor takes a responsibility for $p = MN/n^2$ grid points.

The phase (P0) can be computed entirely in parallel by applying finite-difference approximations to the grid points close to the boundary with a point-wise contribution of function f to each element of $\bar{v}$.

The phases (P1) and (P3) are executable either via the classical matrix by matrix multiplication or by the *sine* variant of the parallel FFT algorithm. Matrix multiplication methods already exist for MP-1 and a report is given in [1]. There exist two approaches for how to compute the *sine* transform via FFT on a parallel processor array. The computation follows concurrently on columns, the number of which is equal to the horizontal dimension of the array. The first way is represented by a fill-in of the input vectors (length $M+1$) with $M+1$ zeroes to create enlarged vectors (length $2(M+1)$). Thus, an odd function is obtained with added elements equal to 0. These enlarged vectors are then transformed by means of FFT. Another algorithmical opportunity [2] offers a fact, that all input data are real. In this case, a suitable choice and reordering of the input and output data enable to apply the complex FFT of the length $M+1 = 2^q$. For the calculation of one-dimensional FFT, a modification of the algorithm from [8] has been used. The parallel implementation strategy of this phase follows straightforwardly from the description given in the previous section.

To realize the phase (P2), a solver for concurrent computation of solutions of M tridiagonal Toeplitz systems has been designed. The solver reflects the algorithmic pattern of the cyclic odd-even reduction [3], and we have achieved a well-balanced parallel execution of the reduction and expansion algorithm's levels. Both phases proceed in parallel for all rows of the input matrix which is stored by blocks, the size of which equals to that of the processor array. In every step, the blocks are shifted to the right and to the left across the array in the horizontal direction by means of the X-net sendings. Since the distance of interacting data is always a power of two value, this local communication mechanism was applied profitably. A disadvantage of the reduction and expansion phases is a not optimal work balancing because the active set of processors increases (decreases) by a factor of 2 in every step of the expansion (reduction).

Complexity of the algorithm depends linearly on the value $p = MN/n^2$ which characterizes the mapping of the grid onto the processor array. The cyclic odd-even reduction block, as well as the FFT modification for the *sine* transform, need $O(\log L)$ parallel operational steps. The computation of both methods can be scaled perfectly into p portions, each of them being performed on the $n \times n$ array. Thus, the algorithm's complexity is $O(p \log L)$. If instead of FFT the matrix multiplication routines would be applied, the complexity for the phases (P1) and (P3) would increase to $O(pM)$.

3.2. QR Algorithm

A parallel proposal of QR factorization by Householder method zeros all subdiagonal elements in each column and updates all elements of the given submatrix in parallel. This method starts from the left, working with the whole matrix A, and continues to the right, working on smaller and smaller submatrices. When the data elements of the matrix A are stored into the PE array, then the following algorithm modifies A to upper-triangular matrix (and simultaneously the orthogonal matrix Q is generated which is the unit matrix initially):

(1) for given column compute the Householder vector $\bar{v}$, respectively, $\bar{u}$ by means of formulas (9);
(2) $\bar{v}$ is copied by the spread function along the first dimension (from the bottom to the top or vice versa) to all processors where the submatrix is updated, and so the matrix V is generated;
(3) $A * V$ dot-products are performed in a single step (it is a dot-multiplication) and then accumulated and saved in a row vector $\bar{w}^\top$;
(4) $\bar{w}^\top$ is copied by the spread function and dot-multiplied with the matrix V and then added to the matrix A;
(5) vector $\bar{v}$ is copied (by the spread function) along the second dimension (from the left to the right or vice versa) and dot-multiplied with $2 * V$;
(6) after adding "one" to diagonal processor elements, the matrix P is generated;
(7) the matrices Q and P are multiplied (MATMUL function) and the result is stored in the matrix Q.

This is repeated for all columns of A, and then the matrix A is transformed into the matrix R. The arithmetical complexity of this algorithm is $O(N + N \log N)$.

Givens transformation is a plane rotation that combines two rows of A in order to annihilate one element. In order to parallelize this reduction, the basic idea is to annihilate more than one element at the time. In this process, various rows are combined in such a way that previously introduced zeros are not destroyed. In the case of Givens method, there was implemented an algorithm which utilizes parallel ordering proposed by Sameh and Kuck. According to this ordering, the independent rotations are computed in parallel so that as many processors are activated as allowed for preserving all zero elements. A modification of elements of competent rows is performed also in parallel. The algorithm computes the values c, s for all independent rotations in prescribed order and stores them into vectors $\bar{c}$ and $\bar{s}$. Vectors are shifted one position down and matrices $C1$ and $S1$ are created by means of the spread function that copies vectors $\bar{c}$ and $\bar{s}$ along the first dimension.

This algorithm takes $2N-3$ steps for a square matrix and an $M+N-2$ steps for $M \times N$ matrix, each step being the time necessary to achieve a set of the independent Givens rotations [9].

4. NUMERICAL EXPERIMENTS

The algorithms were implemented on two-dimensional processor array MP-1 of MasPar of the size 32×32 [4]. The computations for the Poisson equation were performed in the MPF as well as MPL parallel programming languages (for both single and double precision arithmetics). On

the unit square domain, the discretization in both directions were chosen for $N = 32, 64, 128, 256$ (MPF codes) and for $N = 32, 64, 128, 256, 512$ (MPL codes).

The algorithm is coded in MPF in four variants. The first one, F-POISMAT-F, contains both main program blocks, i.e., the transformation and the tridiagonal solver, programmed in MPF. The transformations are computed by a classical matrix multiplication procedure written in MPF language. The computer library contains a MATMUL routine for multiplication of real matrices. This was implemented on the place of forward and backward transformations in the code F-POISMAT-L. It is to note that F-POISMAT-L is not a pure Fortran code because the MATMUL is just callable from MPF, but it is not programmed in it. It is assumed that it is developed in a low-level machine-close language. A third Fortran-based Poisson solver version is F-POISSING-F. The first and third computational phases are calculated by our MPF subroutine based on the fast *sine* transform algorithm given in [2]. The fourth code F-POISSING-L is generated by inserting the *sine* transform routine coded in MPL into the F-POISMAT-L instead of the MATMUL in the transformation phases (P1) and (P3) of the algorithm.

The timings (in msec) for these four versions are given for the single and the double precision arithmetics in Table 1 and Table 2, respectively. It is straightforward to observe that the slowest version is F-POISMAT-F. As seen from the Table 1, a faster performance has been achieved for the F-POISMAT-L against the F-POISSING-F code. The complexity for the FFT used in F-POISSING-F is asymptotically lower than that one for the MATMUL used in F-POISMAT-L. This is due to the fact that the code F-POISMAT-L contains transformation parts written in a lower level language than MPF in which the F-POISSING-F routine is written entirely. The complexity preference in favor of the FFT becomes apparent when comparing the F-POISMAT-L with F-POISSING-L, where in both codes the transformations are performed by means of routines written in MPL. The MPL-based codes of our algorithm are presented in three versions. In L-POISMAT-L, the transformation blocks are replaced by the MATMUL routine from the computer library (as it was the case in F-POISMAT-L) and the tridiagonal block was coded in MPL. (Thus, the difference between F-POISMAT-L and L-POISMAT-L lies in different programming realizations of the second computational phase of the algorithm.)

Table 1. MPF code timings—single precision.

N	F-POISMAT-F	F-POISSING-F	F-POISMAT-L	F-POISSING-L
32	218	179	121	70
64	2003	407	261	165
128	15011	1277	833	508
256	133578	4902	3598	1890

Table 2. MPF code timings—double precision.

N	F-POISMAT-F	F-POISSING-F	F-POISMAT-L	F-POISSING-L
32	343	236	128	108
64	3082	576	320	273
128	23239	1905	1253	829
256	183813	7376	5872	2983

The fast *sine* transform algorithm from [2] programmed in MPL has been used in the code L-POISSING-L for both the transformation phases of the Poisson solver. When these transforms are computed by doubling the transform length with inserting additional zeroes into transformed vectors, the code L-POISSIND-L has been developed entirely in MPL. As it was the situation with MPF, the timings for these MPL-based codes are presented (in msec) for the single precision in Table 3 and for the double precision in Table 4. Since the time for the generation of the right-hand side vector of the system and for the tridiagonal solver block are the same in all three MPL codes, the difference among them results from different execution of both the transformation phases. Because of the higher complexity order for the matrix multiplication routine, the algorithm

L-POISMAT-L delivers worse results than the two remaining approaches which are based on the FFT method. From these two codes, L-POISSING-L is faster than L-POISSIND-L which needs to compute the transformed vectors of the double lengths. The parallel variants of Householder and Givens methods for the QR decomposition, as described in the previous section, were coded in MPF. The results (in msec) for different matrix sizes are given in Table 5.

Table 3. MPL code timings—single precision.

N	L-POISMAT-L	L-POISSIND-L	L-POISSING-L
32	101	70	43
64	198	130	94
128	590	359	258
256	2649	1238	902
512	14406	3838	3371

Table 4. MPL code timings—double precision.

N	L-POISMAT-L	L-POISSIND-L	L-POISSING-L
32	112	106	50
64	263	213	109
128	921	734	332
256	4235	2279	1226

Table 5. MPF code timings—single precision.

	$N = 32$	$N = 64$	$N = 128$	$N = 256$	$N = 350$
Householder algorithm	414	2200	13468	89035	217308
Givens algorithm	219	751	3492	20144	47579

5. CONCLUSIONS AND OUTLOOKS

It follows from comparisons of seven developed parallel codes for solving the discretized Poisson problem that the MPL codes are faster than those written in MPF, the codes which use the FFT variants are generally faster than those which are based on the matrix multiplication, and it was observed that the fastest code is MPL-based L-POISSING-L with parallel *sine* transform and cyclic odd-even reduction blocks. For QR methods, the experiments show that QR-decomposition with Givens rotations is 4.5 times faster than that one with Householder transformations. The outlooks for a future work on the Poisson problem could be in an implementation of the Cannon's and Winograd's algorithms in place of the library routine MATMUL (MPL-based codes). Further, an analysis of other Toeplitz solvers for solving the phase (P2) (a set of tridiagonal systems) from the point of view of highly concurrent execution for SIMD two-dimensional arrays with toroidal topology will be performed. For QR factorization, it is intended to write the algorithms in the MPL code and as follows from the comparison of MPF and MPL performance, it is expected that these times will be substantially smaller than those given in the Table 5.

REFERENCES

1. P. Bjørstad, F. Manne, T. Sørevik and M. Vajteršic, Efficient matrix multiplication on SIMD computers, *SIAM J. Matrix Anal. Appl.* **13**, 386–401 (1992).
2. G. Golub and J.M. Ortega, *Scientific Computing: An Introduction with Parallel Computing*, Academic Press; Harcourt Brace Jovanovich, Publishers, San Diego, (1993).
3. M. Vajteršic, *Algorithms for Elliptic Problems: Efficient Sequential and Parallel Solvers*, Kluwer Academic Publishers, Dordrecht, (1993).
4. T. Blank, The MasPar MP-1 architecture, *Proc. Compcon Spring 90*, 20–24 (1990).
5. G. Svensson, *Some Linear Algebra on The CM-2*, Department of Mathematics, Linkoping University, Linkoping, Sweden, (1992).
6. *MasPar Fortran Programming Manuals*, Software Version 2.0, Revision A6, MasPar Computer Corporation, Sunnyvale, CA, (1992).

7. *MasPar Programming Language Manuals*, Software Version 3.0, Revision A2, MasPar Computer Corporation, Sunnyvale, CA, (1992).
8. M. Lucká, An effective algorithm for computation of two-dimensional Fourier transform for $N \times M$ matrices, In *Proc. Parallel Computation*, Lecture Notes in Computer Science **734**, pp. 64–71, Springer-Verlag, (1993).
9. M. Cosnard and Y. Robert, Complexity of parallel QR factorization, *Journal of the Association for Computing Machinery* **33** (4), 712–723 (1986).