Algorithm for Sparse Approximate Inverse Preconditioners in the Conjugate Gradient Method*

Ilya B. Labutin

A.A. Trofimuk Institute of Petroleum Geology
and Geophysics SB RAS, 3, acad. Koptyug Ave.,
Novosibirsk 630090, Russia

ilya.labutin@gmail.com

Irina V. Surodina

Institute of Computational Mathematics and
Mathematical Geophysics SB RAS, 6, acad.
Lavrentiev Ave., Novosibirsk 630090, Russia

sur@ommfao1.sscc.ru

Abstract

We propose a method for preconditioner construction and parallel implementations of the Preconditioned Conjugate Gradient algorithm on GPU platforms. The preconditioning matrix is an approximate inverse derived from an algorithm for the iterative improvement of a solution to linear equations. Using a sparse matrix-vector product, our preconditioner is well suited for massively parallel GPU architecture. We present numerical experiments and comparisons with CPU implementations.

Keywords: sparse approximate inverse, preconditioned conjugate gradient, GPU
AMS subject classifications: 68W10

1 Introduction

Our work is motivated by, but not limited to, finite-difference methods applied to 2D and 3D potential field problems arising in mathematical geophysics from well resistivity logging applications. A numerical solution of the potential field problem leads to large sparse linear systems usually solved by iterative methods instead of direct ones.

*Submitted: February 15, 2013; Revised: November 29, 2013; Accepted: December 6, 2013.

The Conjugate Gradient (CG) algorithm is one of the best known iterative methods for solving linear systems with a symmetric positive definite matrix [1]. The performance of the CG can be increased dramatically with the suitable preconditioner. The concept of the preconditioning in iterative methods is to transform the original system into an equivalent system with the same solution, but with better properties. A good preconditioner allows an iterative solution to be computed more quickly for the transformed system than for the original problem [9]. However, the computational overhead of applying the preconditioner must not cancel out the benefit of fewer iterations (see, e.g., [3, 6]).

Modern parallel implementations of the preconditioned conjugate gradient (PCG) on the graphical processing units (GPUs) use sparse approximate inverse (AINV) preconditioners due to their attractive features. First, the columns or rows of the approximate inverse matrix can be generated in parallel. Second, the preconditioner matrix is used in PCG through matrix-vector multiplications, which are easy to parallelize [6].

In this work, we present an algorithm for building a series of AINV preconditioners with (potentially) arbitrary high approximation accuracy. The algorithm presented derives from the Schulz-Hotelling algorithm for the iterative improvement of inverse matrices and solutions to linear equations (see, e.g., [5, 8]). We propose parallel implementations for the preconditioners obtained for the first, third, and seventh order approximations of the inverse matrix.

2 Preconditioned Conjugate Gradient Algorithm

The potential field problem is solved using a finite difference method with the standard five and seven point stencils for 2D and 3D equations, respectively. The approximation leads to a system of linear equations

$$Ax = b,$$

where A is a symmetric positive-definite matrix of order N containing at most five (for 2D) and seven (for 3D) nonzero elements per row. N is the number of potential unknowns, and b is a finite difference approximation of the right-hand side of the equation.

A commonly used and well examined iterative algorithm for solving sparse symmetric positive definite linear systems is the conjugate gradient (CG) method. The convergence rate of the CG method depends not only on the matrix size, but also on the condition number of the matrix

$$\operatorname{cond}(A) = \lambda_{\max}/\lambda_{\min}$$

where $\lambda_{\max}$ and $\lambda_{\min}$ are maximum and minimum eigenvalues of A, respectively [7]. A matrix is called *well conditioned* if it is not too far from the identity matrix, which has $\operatorname{cond}(A) = 1$. A matrix A having smaller condition number leads to faster convergence of the CG method applied to the system $Ax = b$. In typical well resistivity logging applications, the matrix A has a large condition number, i.e. $\operatorname{cond}(A) \gg 1$.

The effectiveness of the CG method can be increased by applying a preconditioning technique that improves the properties of the matrix of a system. Suppose that M is a symmetric, positive definite matrix that approximates A, but is easier to invert. We can solve $Ax = b$ indirectly by solving

$$M^{-1}Ax = M^{-1}b. \tag{1}$$

If $\operatorname{cond}(M^{-1}A) \ll \operatorname{cond}(A)$ or if the eigenvalues of $M^{-1}A$ are better clustered than those of A, we can iteratively solve equation (1) more quickly than the original problem [9].

The iteration scheme of the preconditioned CG (PCG) algorithm is

$$\begin{array}{ll} \text{Initialization:} & x_0, r_0 = b - Ax_0, Mz_0 = r_0, p_0 = z_0\,; \\ 1. & q_i = Ap_i, \alpha_i = \dfrac{z_i^T r_i}{p_i^T q_i}\,; \\ 2. & x_{i+1} = x_i + \alpha_i p_i, r_{i+1} = r_i - \alpha_i q_i\,; \\ 3. & Mz_{i+1} = r_{i+1}\,; \\ 4. & \beta_i = \dfrac{z_{i+1}^T r_{i+1}}{z_i^T r_i}, p_{i+1} = r_{i+1} + \beta_i p_i. \end{array} \qquad (2)$$

Each iteration requires one matrix-vector product, two inner products, three vector updates, and the solution of the linear system at Step 3. For the NVIDIA GPU implementation, all the required operations can be found in the standard CUBLAS library. Matrix-vector product can be implemented efficiently on a GPU for sparse matrices (see, e.g., [6, 2]).

3 Approximate Inverse Preconditioner

To eliminate linear system solving at Step 3, M^{-1} can be computed as an approximation to the inverse of A. In this case, Step 3 is replaced with a matrix-vector product. We derive our algorithm for constructing AINV preconditioners from the procedure developed by Schulz [8] and Hotelling [5] for the iterative improvement of inverse matrices and solutions to linear equations.

Let us assume that D_0 is an initial approximation of A^{-1} and satisfies the condition

$$\|R_0\| \le k < 1, \quad R_0 = I - AD_0, \qquad (3)$$

where the matrix norm is any norm which is easy to compute. Next, consider the formal manipulation

$$\begin{aligned} A^{-1} &= (D_0 D_0^{-1})A^{-1} = D_0(D_0^{-1}A^{-1}) \\ &= D_0(AD_0)^{-1} = D_0(I - R_0)^{-1} \\ &= D_0(I + R_0 + R_0^2 + R_0^3 + \dots). \end{aligned}$$

We can define the sequence of matrices D_n by $D_{n+1} = D_n(I + R_n)$, where $R_n = I - AD_n$. What is more that

$$\begin{aligned} R_n &= I - AD_{n-1}(I + R_{n-1}) = I - (I - R_{n-1})(I + R_{n-1}) \\ &= R_{n-1}^2 = R_{n-2}^4 = \dots = R_0^{2^n}. \end{aligned}$$

Hence, we have

$$D_n = A^{-1}(I - R_0^{2^n}). \qquad (4)$$

With (3) and (4) we can build an iterative process which converges to A^{-1} [4], and the error is

$$\|D_n - A^{-1}\| \le \|D_0\| \frac{k^{2^n}}{1-k}. \qquad (5)$$

In addition, the algorithm preserves the symmetry of the approximation to A^{-1} if the initial approximation is a symmetric matrix. Assuming $A^T = A$ and $D_{n-1}^T = D_{n-1}$,

$$\begin{aligned} D_n^T &= 2D_{n-1}^T - (D_{n-1}AD_{n-1})^T \\ &= 2D_{n-1}^T - D_{n-1}^T A^T D_{n-1}^T \\ &= 2D_{n-1} - D_{n-1}AD_{n-1} \\ &= D_n. \end{aligned}$$

With a reasonable initial approximation D_0, we can use D_n as the preconditioning matrix M^{-1} at Step 3 of the PCG algorithm (2).

4 Numerical Experiments

For the numerical experiments, we use well-known Jacobi preconditioner $D_0 = J = \operatorname{diag}\{a_{11}^{-1}, a_{22}^{-1}, \dots, a_{nn}^{-1}\}$ as an initial approximation to A^{-1}. In this case, the matrix D_1 has the form

$$D_1 = J + J(I - AJ).$$

According to (5), the matrix D_1 has better properties than $D_0 = J$. Also, we can see that D_1 has the same structure as the matrix A, so that the same matrix-vector product implementation can be used for iterations (Step 1) and for preconditioning (Step 3) in PCG algorithm (2). Next, the matrices D_2 and D_3 can be expressed in factorized form via D_1:

$$\begin{aligned} D_2 &= D_1 + D_1(I - AD_1) = 2D_1 - D_1AD_1, \\ D_3 &= D_2 + D_2(I - AD_2), \\ &= (2D_1 - D_1AD_1)(2I - A(2D_1 - D_1AD_1)) \\ &= 2(2D_1 - D_1AD_1) - (2D_1 - D_1AD_1)A(2D_1 - D_1AD_1). \end{aligned} \tag{6}$$

Such factorized form is eminently suitable for computations, because we know the structure of the matrices A and D_1. It is hard to use the explicit form of the matrix D_i with the indexes 2 and higher, because the number of fill-ins increases accordingly. For example, the matrix D_1 contains 5 diagonals for a 2D finite difference grid. D_2 contains 25 diagonals, and D_3 contains113 diagonals. On the other hand, D_1, D_2 and D_3, can be expressed as

$$\begin{aligned} D_1 &= D_0(I + R_0), \\ D_2 &= D_1(I + R_0^2) = D_0(I + R_0)(I + R_0^2) \\ &= D_0(I + R_0 + R_0^2 + R_0^3), \\ D_3 &= D_2(I + R_2) = D_2(I + R_0^4) \\ &= D_0(I + R_0 + R_0^2 + R_0^3)(I + R_0^4) \\ &= D_0(I + R_0 + R_0^2 + R_0^3 + R_0^4 + R_0^5 + R_0^6 + R_0^7). \end{aligned}$$

We can see that D_1, D_2 and D_3 are the first, third, and seventh order approximations of A^{-1}, respectively.

We first investigate the behaviour of our AINV preconditioners on a matrix arising from a 2D potential field problem discretized by a finite difference method on a nonuniform grid. The resulting matrix has $n = 17,139$. Table 1 shows the condition number of the initial matrix and preconditioned matrices. We notice that the $D_3(J)$ preconditioner is significantly better than the Jacobi preconditioner.

Next, we compare the generated preconditioners with the sequential Jacobi preconditioned CG implementation on the same finite difference grids produced for 2D and 3D potential field problems. For the numerical experiments, we used processors

CPU: Intel Xeon X5670 2.93 GHz (using Intel Fortran 11.0),

GPU: NVIDIA Tesla M2090, 512 Core, 4 GB RAM (using CUDA).

Matrix	**Condition number**
A	$4.5377 * 10^7$
AJ	$3.0542 * 10^5$
$AD_1(J)$	$7.6542 * 10^4$
$AD_2(J)$	$3.8271 * 10^4$
$AD_3(J)$	$1.9135 * 10^4$

Table 1: Condition number of AM^{-1}

Method	n**=17,139**			n**=37,846**			n**=76,136**		
	iter. count	**time, sec**	**speed up**	**iter. count**	**time, sec**	**speed up**	**iter. count**	**time, sec**	**speed up**
$CG(J)$	2835	1.21	-	4523	3.75	-	7204	14.34	-
$D_0 = J$	2684	0.26	5	4073	0.41	9	5731	0.73	20
$D_1(J)$	1325	0.16	8	2038	0.27	14	2866	0.52	26
$D_2(D_1)$	942	0.12	10	1451	0.22	17	2055	0.46	31
$D_3(D_1)$	668	0.11	11	1038	0.19	20	1458	0.40	36

Table 2: Performances of the PCG algorithm and speed-up.
2D potential field problem.

Method	n**=1,458,600**			n**=4,111,848**		
	iter. count	**time, sec**	**ratio**	**iter. count**	**time, sec**	**ratio**
$CG(J)$	4769	190	-	7445	673	-
$D_0 = J$	4981	6.98	27	7754	28.43	24
$D_1(J)$	2495	4.35	44	3666	17.50	38
$D_2(D_1)$	1885	5.56	34	2692	21.85	30

Table 3: Performances of the PCG algorithm and speed-up.
3D potential field problem.

In Table 2, we compare GPU implementations of D_0, D_1, D_2, and D_3 preconditioners with CPU implementation of CG with Jacobi preconditioner for the 2D potential field problem. We use three finite difference grids of different sizes typical for well resistivity logging applications. As expected, the lower the condition number of the matrix, the smaller the number of iterations. The speed-up increases accordingly. For the largest problem, the GPU implementation of D_3 is 36 times faster than the CG algorithm on the CPU. However, the speed-up of D_2 is relatively the same as the speed-up of D_3, despite of significant difference in the number of iterations. This fact is expected, because an application of the D_2 preconditioner requires only three matrix-vector products (see (6)), while D_3 requires seven matrix-vector products.

In Table 3, we compare GPU implementations of D_0, D_1, and D_2 preconditioners with a CPU implementation of CG with the Jacobi preconditioner for the 3D potential field problem. For the largest problem, the GPU implementation of D_1 is 38 times faster than the CG algorithm on the CPU. For the large matrices, the number of matrix-vector products in the preconditioning application becomes important. D_1 is faster than D_2 for the 3D grid because it requires only one matrix-vector product. However for the 2D grid, D_2 is faster than D_1, see Table 2.

In summary, the higher the index of the preconditioner obtained in the iteration (4), the higher the number of fill-ins in the resulting matrix. The choice of the preconditioner in this case is a trade-off between the quality of the resulting matrix in terms of (5) and performance of the preconditioning procedure, which depends on sparsity pattern and number of matrix-vector products in factorized form of the preconditioner.

5 Conclusion

We have presented an algorithm for generating preconditioners and parallel GPU implementations of the preconditioned conjugate gradient algorithm for linear systems with symmetric positive definite matrices. Our preconditioners, derived from the Schulz-Hotelling algorithm for the iterative improvement of inverse matrices and solutions to linear equations, is an approximate inverse and therefore can be used in the PCG algorithm through a sparse matrix-vector product. In the future, we plan to investigate other preconditioners generated by the algorithm presented here, but with different initial approximations of A^{-1}.

References

[1] R. Barrett, M. W. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, Society for Industrial and Applied Mathematics, Philadelphia, 1993, URL: `http://www.netlib.org/templates/templates.pdf`

[2] N. Bell and M. Garland, *Efficient Sparse Matrix-Vector Multiplication on CUDA*, Technical Report, NVIDIA, 2008.

[3] J. Dongarra, I. Duff, D. Sorensen, H. Van der Vorst, *Numerical Linear Algebra for High-Performance Computers*, Society for Industrial and Applied Mathematics, Philadelphia, 1998.

[4] D. K. Faddeev and V. N. Faddeeva, *Computational Methods of Linear Algebra*, Fizmatgiz, Moscow, 1963. (in Russian)

[5] H. Hotelling, Analysis of a complex of statistical variables into principal components, *Journal of Educational Psychology*, 1933, pp. 417–441.

[6] R. Li and Y. Saad, *GPU-Accelerated Preconditioned Iterative Linear Solvers*, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, 2010,

[7] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd edition, Society for Industrial and Applied Mathematics, Philadelphia, 2003.

[8] G. Schulz, Iterative Berechnung der reziproken Matrix, *Zeitschrift für Angewandte Mathematik und Mechanik (ZAMM)*, 1933, pp. 57–59.

[9] J. R. Shewchuk, *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*, School of Computer Science, Carnegie Mellon University, Pittsburgh, 1994.