Approximate Inverse Preconditioning and Parallel Computing

Thomas Huckle
Institut für Informatik, Technische Universität München
(TUM), Munich, Germany.
E-mail: huckle@in.tum.de

Abstract

Many practical applications lead to the problem of solving very large sparse systems of linear equations $Ax = b$ on a parallel architecture. This is caused by the need for high accuracy and fine discretization for very large problems; without sparse formulation and the use of parallel computers many applications could not be solved. For linear equations there exist two classes of competing methods: direct and iterative algorithms. Direct solvers often have problems with a loss of sparsity and with their sequential nature. Iterative methods like the conjugate gradient algorithm or GMRES can take full advantage of sparsity and parallelism by using mainly matrix-vector products, but for ill-conditioned matrices they often suffer from slow convergence. To combine the advantages of direct and iterative methods a preconditioner $M$ is included in the iterative method, e.g. in the form $b = Ax = A(My) = (AM)y$. The iterative solver is then applied on the matrix $AM$, where this new system should have a much better condition number than the original problem $A$. $M$ can be seen as an approximate direct solver where we allow only operations that maintain the sparsity and that are well adapted to parallel environments. Therefore, the computation of $M$ and matrix-vector multiplications of the form $Mx$ should be fast to compute in parallel, and the condition number of the new problem $AM$ should be small in order to get fast convergence of the iterative solver applied on the new matrix $AM$.

For preconditioning there exist two major approaches: a direct preconditioner $P$ that gives an approximation of the original matrix $P \approx A$, e.g. ILU or Gauss-Seidel, and an approximate inverse preconditioner $M$, that is an approximation on the inverse $A^{-1} \approx M$. Direct preconditioners are inexpensive to compute and often lead to fast convergence, but they are strongly sequential and in every iteration step we still have to solve a linear system in $P$. Approximate inverse preconditioners are usually much easier to parallelize.

One interesting class of approximate inverse matrices computes the solution of the minimization problem

$$\min_{M \in \Phi} \left\| AM - I \right\|_F,$$

in the Frobenius norm, where $\Phi$ denotes a given sparsity pattern for the allowed nonzero entries of $M$. This minimization problem can be decomposed in $n$ independent minimizations for each column of $M$ in the Euclidean norm

$$\min_{M \in \Phi} \left\| AM_k - e_k \right\|_2.$$

Hence, the computation and the matrix-vector product are very efficient for sparse $M$ and in parallel. We have to consider $n$ Least Squares problems with submatrices $\hat{A} = A(I_k, J_k)$ where the index sets $I_k$ and $J_k$ are given by the pattern $\Phi_k$ and the pattern of $A$. These Least Squares problems are solved by QR-decompositions for $\hat{A}$. 
Unfortunately, we must often allow more entries in $M$ in order to obtain a small norm $\|AM - I\|$ and fast convergence. Note, that the costs for the Least Squares problems are increasing cubic in the size of $A(I_k,J_k)$.

On the other side, to get $\|AM - I\|$ small, $M$ has to reduce this norm relative to the large eigenvalues at first, and the error relative to the small eigenvalues will stay nearly unchanged. This is exactly the property that marks a good smoother in Multigrid methods, because here the large eigenvalues are related to high frequency components that have to be removed by the smoother. Furthermore, ill-posed problems can be regularized by applying the preconditioned conjugate gradient method; here the preconditioner should also improve the condition number relative to the large eigenvalues without amplifying the noise given by the small eigenvalues. Hence, the approximate inverse $M$ is also very useful in this connection.

The main disadvantage of $M$ are the relatively high costs for solving all the Least Squares problems especially if we have to allow more fill-in in order to achieve satisfactorily convergence. Furthermore, we have to determine a promising sparsity pattern $\varphi$, and we need a clever method to distribute the small independent Least Squares problems on different processors. So, various modifications have been developed in the last years:

- Sparsifying the pattern of $A$ to $A_\varepsilon$ by removing small entries,
- using the patterns of $A^k$, $(A^H)^k$, or $(AA^H)^k A$ (or $A_\varepsilon$ instead of $A$),
- using factorized preconditioners, possibly repeatedly,
- enlarging the pattern $\varphi$ dynamically,
- computing a block approximative inverse matrix $M$ blockwise.

To reduce the costs for computing $M$ we want to present different approaches:

- computing the QR-factorizations in sparse mode or Q-free,
- combining Least Squares problems where the underlying submatrices $A(I_k,J_k)$ nearly coincide, and using e.g. updating techniques for the QR-factorizations,
- finding a proper ordering for the block approximate inverse approach,
- distributing the Least Squares problems on different processors with respect to memory and computational time.

By these improvements the computational costs for the Frobenius norm minimization can be reduced in many cases, such that this preconditioner is more likely to be competitive with direct preconditioners or other approximate inverse approaches like AINV or robust ILU.

**Keywords**
Preconditioning, sparse approximate inverse