Solving Quasi Block Diagonal Linear Systems

Viviana Arrigoni
arrigoni@di.uniroma1.it
Department of Computer Science
Sapienza University of Rome, Italy

Annalisa Massini
massini@di.uniroma1.it
Department of Computer Science
Sapienza University of Rome, Italy

ABSTRACT

A hybrid MPI/OpenMP solver is implemented for a class of sparse linear systems that we call *quasi block diagonal*. The system uses specific formats for sparse matrices and implements preconditioned Jacobi in order to reduce memory storage requirements. We compare our solver with solvers available in HPC libraries.

KEYWORDS

Computational Linear Algebra, Sparse Matrices, MPI, OpenMP

1 INTRODUCTION

Sparse matrices arise from problems in several fields, as they result from the discretization of partial differential equations (PDEs) when modelling phenomena spanning over the widest scientific range. In engineering, Finite Element Model (FEM), see e.g., [3], is widely used to model structural components of engines through large and sparse matrices having denser blocks distributed along their diagonal. We refer to them as *quasi block diagonal matrices*. They represent the coefficient matrices of the PDEs that model the system under study, and that are discretized and solved numerically integrating multiple large, sparse linear systems. For this reason, in the last decades many efforts have been devoted to devise fast linear solvers that would adapt to the most recent parallel architectures, and that would take advantage of specific sparsity patterns. Here, we present a hybrid solver that exploits the preconditioned Jacobi method to increase sparsity, thus allowing very effective memory storage, also thanks to specific sparse formats. In our implementation, we use both MPI and OpenMP. We test our solver on a cluster.

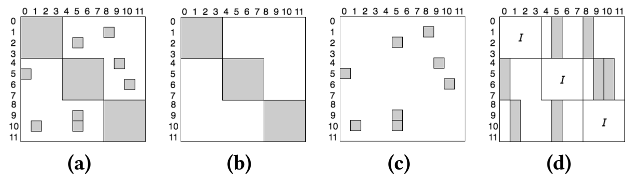

Figure 1: Sparsity patterns of matrices: (a) A, **(b)** D, **(c)** R, **(d)** S.

2 ALGORITHM AND IMPLEMENTATION

Let $Ax = b$ be the linear system to solve, where A is quasi block diagonal, meaning that it can be represented as the sum of its block diagonal part, D, and its off block diagonal part, R (see Figures 1a-1c), as described in [1], taking inspiration from the factorization phase of SPIKE algorithm, see e.g., [2]. The linear system is preconditioned by D^{-1}, becoming $D^{-1}Ax = D^{-1}b$, that is $(I + D^{-1}R)x = D^{-1}b$. We denote $S = I + D^{-1}R$, $G = D^{-1}R$ and $f = D^{-1}b$. The sparsity patterns of D and G do not intersect, and G is formed by small columns (see Figure 1d). The preconditioned linear system $Sx = f$ has provable convergence properties and it is solved using the Jacobi algorithm, that can be suitably implemented in hybrid multiprocessors/threading architectures.

We assume we have as many processors as the number of diagonal blocks of A. To each processor p we assign a diagonal block D_p in full format, and the elements of R having the same row indexes as D_P, R_p, in coordinate format. Every processor p computes D_p^{-1} (using the Lapack routine *gesv), $S_p = D_p^{-1}b_p$ (using the BLAS routine *gemv) and $S_p = D_p^{-1}R_p$ (using OpenMP). These operations can be computed in perfect parallelism since the inverse of D is $D^{-1} = diag(D_0^{-1}, \ldots, D_{k-1}^{-1})$, and $G = \left[D_0^{-1}R_0; \ldots; D_{k-1}^{-1}R_{k-1}\right]$. Only the G component of matrix S is stored using a simplified Ellpack format, where the column indexes matrix is reduced to be a 1D array, exploiting the sparsity pattern of S. Afterwards every processor informs the others about what components of the approximated solution vector $\tilde{x}$ it needs at each iteration of the Jacobi algorithm. Then, each processor p applies the Jacobi algorithm to compute its portion of the approximated solution $\tilde{x}_p$, and exchanges the required entries of $\tilde{x}_p$ with other processors.

3 PERFORMANCE EVALUATION

We tested our solver on Galileo cluster, Cineca (Bologna, Italy). The speed-up with respect to the sequential implementation of preconditioned Jacobi is depicted in Figure 2, and highlights the strong scalability and the negligible communication cost of our solver. Figure 3 shows the outperforming results in terms of execution times of our hybrid solver in comparison with ScaLapack and Intel MKL pardiso solvers for clusters, considering blocks of size 1000 and matrices R_p having 100 nonzero entries each.

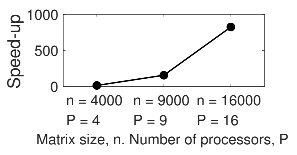

Figure 2: Speed-up wrt sequential implementation.

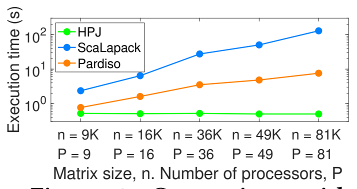

Figure 3: Comparison with ScaLapack and Intel MKL pardiso solvers.

The proposed hybrid MPI/OpenMP solver minimizes memory occupation and communication costs, thus outperforming state-of-the-art solvers integrated in HPC libraries.

REFERENCES

[1] Manguoglu M. Bolukbasi, E. S. 2016. A multithreaded recursive and nonrecursive parallel sparse direct solver. In *Advances in Computational Fluid-Structure Interaction and Flow Simulation*. Springer, 283–292.
[2] Sameh A. Polizzi, E. 2006. A parallel hybrid banded system solver: the SPIKE algorithm. *Parallel computing* 32, 2 (2006), 177–194.
[3] Y. Saad. 2003. *Iterative Methods for Sparse Linear Systems: Second Edition*. Society for Industrial and Applied Mathematics.