Efficient Algebraic Multigrid Preconditioners on Clusters of GPUs

Ambra Abdullahi Hassan[1], Valeria Cardellini[1], Pasqua D'Ambra[2], Daniela di Serafino[3], and Salvatore Filippone[4]

[1] Università degli Studi di Roma "Tor Vergata", Roma, Italy
{ambra.abdullahi,cardellini}@uniroma2.it
[2] Istituto per le Applicazioni del Calcolo "Mauro Picone", CNR, Napoli, Italy
pasqua.dambra@cnr.it
[3] Università degli Studi della Campania "Luigi Vanvitelli", Caserta, Italy
daniela.diserafino@unicampania.it
[4] Cranfield University, Cranfield, UK
salvatore.filippone@cranfield.ac.uk

Abstract. Many scientific applications require the solution of large and sparse linear systems of equations using iterative methods, this operation accounting for a large percentage of the computing time. In these cases, the choice of the preconditioner is crucial for the convergence of the iterative method. Given the broad range of applications, great effort has been put in the development of efficient preconditioners ensuring algorithmic scalability: in this sense, multigrid methods have been proved to be particularly promising. Additionally, the advent of GPUs, now found in many of the fastest supercomputers, poses the problem of implementing efficiently these algorithms on highly parallel architectures; this is made more difficult by the fact that the solution of sparse triangular systems, a common kernel in many types of preconditioners, is extremely inefficient on GPUs.

In this paper, we use the PSBLAS and MLD2P4 libraries to explore various issues that affect the efficiency of multilevel preconditioners on GPUs, both in terms of execution speed and exploitation of computational cores, as well as the algorithmic efficiency in guaranteeing convergence to solution in a number of iterations independent of the parallelism degree. We investigate these issues in the context of linear systems arising from groundwater modeling application of the filtration of 3D incompressible single-phase flows through porous media.

1 Introduction

2 Multilevel preconditioner

Multilevel preconditioner e nuclei computazionali dellalgoritmo di applicazione.

3 Sparse linear algebra on GPUs

General Purpose Graphics Processing Units (GPGPUs) [4] are today an established and attractive choice in the world of scientific computing, found in many among the fastest supercomputers on the Top 500 list, and offered in standard Cloud infrastructure services, e.g. in Amazon EC2. It is therefore not surprising that they are at the focus of many research efforts revolving around the efficient implementation of scientific software, including solvers for sparse linear system of equations.

Many applications use iterative methods to solve sparse linear systems; the most popular ones are those based on Krylov subspace projection methods [6].

From a software point of view, all Krylov methods employ the matrix A to perform matrix-vector products $y \leftarrow Ax$, whose efficient implementation provides significant challenges on all modern computer architectures. The SpMV kernel is well-known to be a memory bounded application; and its bandwidth usage is strongly dependent on both the input matrix and on the underlying computing platform(s). For a detailed overview of the implementation issues and a survey of available techniques for this kernel on GPUs, see [3].

For most real-world problems,it is necessary to combine the Krylov solvers with *preconditioners*. A preconditioner is a transformation applied to the coefficient matrix such that the resulting linear system has better convergence properties. Examples of popular preconditioners include Jacobi and Gauss-Seidel iterations, Block Jacobi and Additive Schwarz coupled with incomplete factorizations, and Algebraic Multigrid.

To implement a linear system solver on a parallel computing architecture we typically use a *domain decomposition* approach, in which subsets of the computational domain are assigned to different computing nodes. Domain decomposition requires to achieve a number of trade-offs among multiple factors:

- Many preconditioners, e.g. Gauss-Seidel or incomplete factorizations, employ kernels for the solution to sparse triangular systems; in using sparse triangular solvers, we normally want the sparsity pattern to be of the same order as that of the original coefficient matrix;
- However, the amount of parallelism available in a sparse triangular solution is dependent on the number of nonzeros in the sparse factors, and is usually very small;
- To overcome this issue, most preocnditioners use a Block Jacobi or hybrid Gauss-Seidel strategy, in which the sparse triangular solve is used only within the local subdomain, while employing a global Jacobi-like correction;
- Unfortunately, the convergence properties of these local schemes deteriorate with increased parallelism.

The last phenomenon is often described as a loss of *algorithmic scalability*, i.e. the ability to obtain a convergence history that is independent of the degree of parallelism. In summary, simple preconditioning schemes such as Point Jacobi have good algorithmic scalability, because their convergence properties do

not change much with an increasing number of processes, but the more sophisticated Gauss-Seidel and incomplete factorizations are a lot more efficient in serial computations.

Multilevel corrections, or Algebraic Multigrid preconditioners, offer a way to recover algorithmic scalability: it is possible to apply efficient local approximate solvers, while a good convergence history is recovered by relying on the multilevel corrections.

Finding the preconditioning combination that gives the best overall performance is thus a very complex enterprise. In this context, the ability to combine multiple options in a flexible framework such as the one we described in [2] is an essential ingredient.

3.1 Preconditioners on GPUs

Additional considerations are necessary when implementing preconditioning operations on GPUs. First of all, implementing the solution of sparse triangualr systems on GPUs is extremely difficult. As an example, the conjugate gradient preconditioned with incomplete LU available in CUDA CuSPARSE since version 4.0 [5] achieves at best a speedup factor of about 2 over a standard CPU implementation. The main reason is that the GPU requires a massive amount of data parallelism to be exploited, and in the sparse triangular factors the amount of available parallelism is limited by the sparsity. The two main options that are available involve the use of matrix-vector products as their main kernel:

- Point-Jacobi iterations;
- (Local) preconditioning through sparse approximation of the inverses.

For a discussion of sparse approximate inverses on GPUs see [1].

3.2 Implementation in PSBLAS/MLD2P4

The implementation of the

4 Results

Distribuzione dati fatta tenendo conto della geometria (3D) del problema e giustificandola rispetto al vantaggio che fornisce in termini di rapporto calcolo/comunicazione

Acknowledgements

References

1. D. Bertaccini and S. Filippone. Sparse approximate inverse preconditioners on high performance gpu platforms. *Computers & Mathematics with Applications*, 71(3):693 – 711, 2016.

2. P. D'Ambra, D. di Serafino, and S. Filippone. MLD2P4: a package of parallel algebraic multilevel domain decomposition preconditioners in Fortran 95. *ACM Trans. Math. Softw.*, 37(3):7–23, 2010.
3. S. Filippone, V. Cardellini, D. Barbieri, and A. Fanfarillo. Sparse matrix-vector multiplication on gpgpus. *ACM Trans. Math. Softw.*, 43(4):30:1–30:49, Jan. 2017.
4. D. Luebke, M. Harris, N. Govindaraju, A. Lefohn, M. Houston, J. Owens, M. Segal, M. Papakipos, and I. Buck. GPGPU: general-purpose computation on graphics hardware. In *Proc. of 2006 ACM/IEEE Conf. on Supercomputing*, SC '06, 2006.
5. M. Naumov. Incomplete-LU and Cholesky preconditioned iterative methods using CUSPARSE and CUBLAS. Technical report, NVIDIA Corporation, June 2011.
6. Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2nd edition, 2003.