NORMA RESEARCH REPORT

T4-D2 (KIAM): (Report M24, 1erMars2022) Parallel LU-SGS method for implicit time integration:

Heterogeneous Implementation of Preconditioners Based on Gauss–Seidel Method for Sparse Block Matrices.

A.R. Magomedov, A.V. Gorobets(*)

March 2022

(*) CAALAB, Keldysh Institute of Applied Mathematics, Moscow

DOI 10.1007/s10598-023-09585-2
Computational Mathematics and Modeling, Vol. 33, No. 4, October, 2022

II. NUMERICAL METHODS

HETEROGENEOUS IMPLEMENTATION OF PRECONDITIONERS BASED ON GAUSS–SEIDEL METHOD FOR SPARSE BLOCK MATRICES

A. R. Magomedov[1] **and A. V. Gorobets**[2]

UDC 519.6

In this paper we present a parallel preconditioner algorithm for the Krylov-type iterative method based on the multicolor Gauss–Seidel method and its transferable heterogeneous software implementation using the OpenCL computational standard. A special feature is its application to block sparse matrices as part of a numerical technique for modeling compressible turbulent flows. Optimization techniques are described that allow obtaining multiple speedups on graphics processors. Results of performance testing on a hybrid computing system are presented.

Keywords: preconditioner, Gauss–Seidel method, sparse matrices, hybrid computing systems.

Introduction

The objective of this work is to create a heterogeneous parallel preconditioner for a solver of a system of linear algebraic equations (SLAE) as part of the implicit time integration algorithm of the NOISEtte [1] simulation code. This code is designed to simulate gas dynamics problems on hybrid supercomputers. The higher-accuracy scheme [3] on unstructured mixed meshes is used for spatial discretization. Time integration is carried out using implicit schemes based on Newton linearization (BDF1, BDF2). Parallel algorithm and heterogeneous implementation for central (CPU) and graphical (GPU) processors are presented in [1]. For multilevel parallelization, the MPI, OpenMP and OpenCL standards are used. The open computing standard OpenCL allows us to use various GPU architectures from different vendors, including NVIDIA, AMD, Intel. The simulation code is also adapted to Russian CPU architecture Elbrus [2].

Currently, the importance of import substitution of software has significantly increased for industrial supercomputer applications. In the scale-resolving simulation of turbulent flows for which the NOISEtte was designed, the size of the time step in the implicit scheme is naturally limited by the dynamics of the simulated flow structures. Therefore, the SLAE with the Jacobi matrix, as a rule, does not require complex solvers: in practice, the Bi-CGSTAB [4] iterative method with a very simple preconditioner based on the block Jacobi method was sufficient. However, in industrial problems, methods based on the Reynolds-averaged Navier–Stokes equations are often used, which allow a much larger time step, which in turn makes the SLAE more stiff and requires the solver to perform more iterations. We need preconditioners that improve convergence more significantly, but there is a problem with them: the matrix changes at each time step, that is why methods with a computationally expensive setup stage (for example, based on resource-intensive LU factorization) are inefficient or inapplicable. Another problem is the extremely limited amount of GPU memory, which also imposes significant restrictions on the choice of method.

Faculty of Computational Mathematics and Cybernetics, Lomonosov Moscow State University, Moscow, Russia.
[1] Email: s02190144@gse.cs.msu.ru.
[2] Email: gorobets@keldysh.ru.

Translated from Prikladnaya Matematika i Informatika, Issue 72, 2023, pp. 38–45.

 1046-283X/22/3304–0438

In this work, the multicolor symmetric Gauss–Seidel method is used, in which the memory and computational costs of construction are insignificant and comparable to those of the Jacobi method.

Parallel Algorithm

A feature of the Jacobian matrix in the implicit scheme for the Navier–Stokes equations is its block structure. The block size of the main SLAE is 5 — according to the number physical or conservative variables. The portrait of the block sparse matrix corresponds to the adjacency of mesh nodes via edges. When a turbulence model is used, one or more additional matrices for turbulent variables appear. The SLAE solver works simultaneously with multiple matrices and vector sets for the main and additional systems. Since the matrices share the same portrait, only the coefficients of the additional matrices are stored. The matrices are stored in the block compressed sparse row (CSR) format. The solver uses a set of kernel subroutines for basic operations of linear algebra, including sparse matrix-vector product (SpMV), linear combination of two vectors $(\mathbf{x} = a\mathbf{y} + b\mathbf{y})$, inner product. To reduce the overhead of data exchange, which is necessary, in particular, for the SpMV, the exchanges are hidden behind the calculations [1].

Symmetric Gauss–Seidel method used as a preconditioner for the Bi-CGSTAB [4] method works as follows. The SLAE $\mathbf{Ax} = \mathbf{b}$, $\mathbf{A} \in \mathbb{R}^{n\times n}$ is solved. Matrix $\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U}$ is represented as the sum of the lower triangular part, the diagonal and the upper triangular part, respectively. The iteration of the method consists of two stages – the solution of the SLAE with a lower triangular matrix (forward substitution) and with an upper triangular matrix (backward substitution):

$$(\mathbf{L} + \mathbf{D})\mathbf{x}^{k+1/2} = \mathbf{b} - \mathbf{U}\mathbf{x}^k, \qquad (\mathbf{D} + \mathbf{U})\mathbf{x}^{k+1} = \mathbf{b} - \mathbf{L}\mathbf{x}^{k+1/2}.$$

The substitutions have data dependencies that prevent GPU parallelization:

$$x_i^{k+1/2} = 1/a_{ii}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{k+1/2} - \sum_{j=i+1}^{n} a_{ij}x_j^k\right).$$

To find the ith position of the vector, all the previous values are needed. To break these dependencies, an approach with reordering of blocks of unknowns is used, which is based on the coloring of the graph, the portrait of the adjacency matrix of which corresponds to the portrait of the block matrix of the system being solved minus the main diagonal. Such coloring is widely used to implement algorithms based on the Gauss–Seidel method on the GPU, see, for example, [5, 6].

To color the vertices of the graph in such a way that there are no adjacent vertices of the same color, we use the very simple greedy algorithm. Blocks of unknowns corresponding to graph vertices are reordered by colors, so in the forward and backward substitutions there are no unknowns of the same color from the new iterative layer. After reordering, the algorithm is fully compatible with the parallel stream processing paradigm used on the GPU: blocks of variables of the same color are grouped; sets of blocks grouped by color are processed in turn; each color is processed in parallel. The coloring algorithm is applied only once at the start of the simulation and does not make a significant contribution to the total computation time. The preconditioner setup stage is characterized by low resource consumption. As in the Jacobi method, only the diagonal blocks of the matrix are inverted, additional memory is allocated to store inverse blocks.

Heterogeneous implementation for CPU and GPU

The multicolor parallel Gauss–Seidel method is implemented trivially for the CPU using OpenMP loop parallelism. After reordering, for each color, a contiguous range of block indices of the unknowns is obtained.

The necessary modification of the forward and backward substitutions consists in adding an outer loop over colors, narrowing the range of the substitution loop to a single color, and adding an OpenMP loop directive to that loop. However, it is important to note that color reordering can significantly degrade the convergence of an iterative solver. Therefore, in this work, instead of the multicolor version, a more efficient decomposition-based version is used on the CPU. A similar algorithm is applied, for example, in [7].

In the CPU version, the unknowns are distributed over subdomains of MPI processes and OpenMP threads by means of mesh graph decomposition with minimization of cut edges. The unknowns are ordered by subdomains, and within the subdomains, the Cuthill–McKee reordering algorithm is used to increase the locality of memory access.

Multiple subdomains are processed simultaneously. Inside the subdomains, the Gauss–Seidel method is used, and at the interfaces between subdomains, in fact, switching to the Jacobi method occurs (i.e., values from the positions of the solution vector belonging to other subdomains are taken for direct substitution from the previous iteration, and for back substitution from the results of direct substitution). This approach gives only a slight degradation of convergence compared to the sequential version, but is poorly suited for the GPU.

In the implementation of the multicolor method for the GPU, instead of a parallel substitution loop, the call to the OpenCL kernel function is used (kernels in OpenCL terminology are subroutines running on the accelerator), processing unknowns of the same color on the GPU in stream processing mode. Accordingly, one iteration of the symmetric method requires two calls to the OpenCL kernel for each color — for forward and backward substitution. The simplest OpenCL implementation is to port directly the CPU version of the multicolor method: the body of the loop over blocks of unknowns of the same color forms the kernel function. Thus, each iteration of the loop becomes a work item, i.e., an elementary task distributed in stream processing. Instead of a loop counter variable, the kernel function has a work item identifier.

Unfortunately, such a “naive” implementation did not show significant acceleration on the GPU relative to the CPU. To improve the performance of the GPU version, the work item elementary task had to be reduced many times. In the original version, one work item the OpenCL kernel processes one block of 5 unknowns, operating on matrix blocks of 25 coefficients, respectively. In the optimized version, one block is processed by 25 work items at once, one item is responsible for only one coefficient in matrix blocks. The reduction of the sums to 5 positions of the output vector is carried out using the fast local memory of the streaming multiprocessor (compute unit). The work-group size for NVIDIA GPUs is 128, one work group processes 5 blocks, 3 units in the group remain inactive. By reducing elementary tasks in this way, the consumption of register memory per work item is reduced, which allows achieve much higher utilization of GPU streaming multiprocessor functional units.

Performance on CPU and GPU

The performance was tested on a representative case in which a two-bladed helicopter rotor is modeled (Fig. 1). One blade is considered in periodic conditions, the mesh is mixed-element, unstructured, predominantly tetrahedral (80%), consists of 1.3 million nodes. An implicit scheme of the 1st order is used, the Courant number is about 10^3. The stopping criterion for the SLAE solver is the reduction of the residual vector L2 norm with respect to the norm of the right side $\epsilon = 0.01$. The hybrid compute server for testing has two 16-core Intel Xeon Gold 5218 2.30GHz CPUs (2 threads per core, 6 channels DDR4-2666 — 128GB/s); NVIDIA A5000 GPU (24 GB GDDR6, 768 GB/s).

Since reordering by color can degrade convergence, of course, a more efficient decomposition-based version is used on the CPU when demonstrating GPU acceleration relative to the CPU. At each iteration of Bi-CGSTAB, there are two calls to the preconditioner, in which two iterations of the symmetric Gauss–Seidel method are performed. In this test, the average number of Bi-CGSTAB iterations with the CPU version of the preconditioner is 6, for the GPU version it is 10, i.e., the GPU is doing more work due to convergence degradation. The average number of non-zero blocks in the rows of the matrix is 12.1, the number of colors in the coloring is 11, the block size is 5,

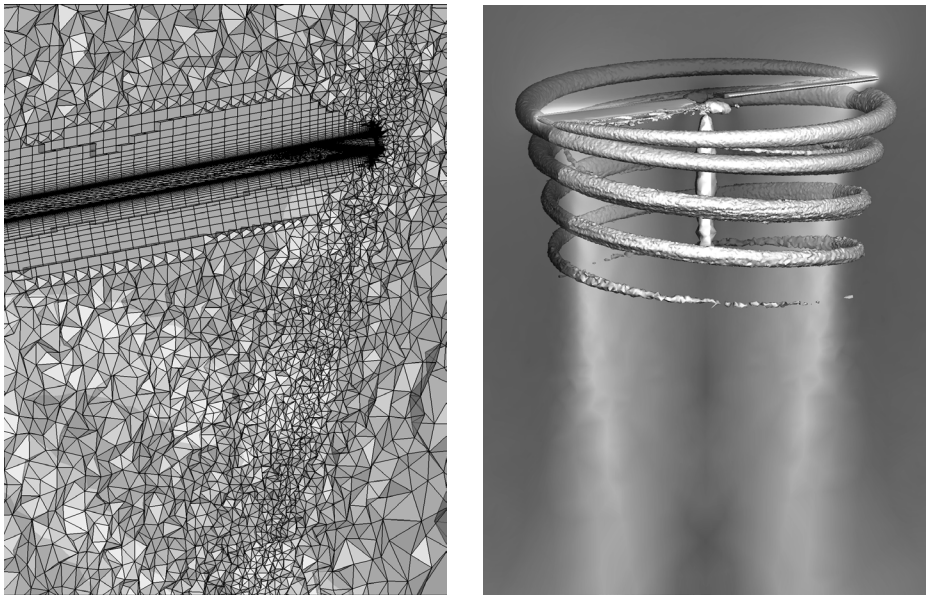

Fig. 1. Test case. A fragment of the unstructured mixed-element mesh (left) and visualization of the tip vortex (right).

Table 1
Performance of Multicolor Gauss–Seidel Method

Version	Devices	MPI	OpenMP	Time, s	Speedup	GFLOPS	GB/s
CPU	1 CPU	1	1	29.5	1	1.3	5.4
CPU	1 CPU	1	32	3.9	7.6	10	42
CPU	2 CPU	2	32	2.2	13.4	17.7	74
GPU-Base	1 GPU	1	32	3.6	8.2	18	46
GPU-Opt	1 GPU	1	32	0.52	57	125	519
CPU,	1 CPU,	2	28,	0.48	61.5	135	563
GPU-Opt	1 GPU		4				
GPU-Opt	2 GPU	2	32	0.3	98	216	900

and the total number of unknowns is 6.8 million. The results are presented in Table 1 for three versions: the version for central processors (denoted by CPU), the basic version for the GPU (GPU-Base), the optimized version for the GPU (GPU-Opt). The table shows the resources involved; number of MPI processes; number of OpenMP threads per process (if different, separated by commas for all processes); the time spent on the preconditioner when solving the SLAE, averaged over 10 time steps; acceleration relative to sequential mode on the CPU; performance in floating point operations per second (FLOPS); lower estimate for memory bandwidth utilization.

CONCLUSIONS

As a result of the work, a heterogeneous implementation of the preconditioner was created, which allows using multiple CPUs and GPUs. From Table 1 we can draw the following conclusions. The use of coloring for the symmetric Gauss–Seidel method makes it possible to obtain a significant acceleration on GPU versus CPU even

though there is some convergence degradation due to color reordering. For these device models, the acceleration on the GPU relative to the 16-core CPU was 7.5 times, which gives the practical equivalent of about 120 cores from one GPU.

It should be noted that the NVIDIA A5000 GPU model is relatively inexpensive, about 3 times cheaper than the currently widely used NVIDIA V100 computing GPUs. A heterogeneous mode with simultaneous use of the CPU and GPU gives a certain gain, but with such a performance ratio it does not make much sense: the potential acceleration from using the CPU in addition to the GPU is only 13%. In fact, less than 8% is obtained due to the losses added for data exchange and inevitable load imbalance, since the CPU and GPU subdomains are balanced by the performance ratio of the entire simulation algorithm, and not of the preconditioner alone.

The main conclusion is that when working with such small-block sparse matrices, it is very important to optimize the OpenCL kernel functions for the GPU streaming multiprocessor architecture, which consists in minimizing the tasks of work items and tuning the work-group size. With appropriate optimization, the GPU handles very effectively such a computational algorithm, the processing speed is close to 70% of the memory bandwidth, in other words, the resulting performance is about 70% of the theoretically achievable on the given device and algorithm.

Acknowledgements

This work was funded by the RSF project 19-11-00299.

The research is carried out using the equipment of the shared research facilities of HPC computing resources at Lomonosov Moscow State University, the equipment of Shared Resource Center of KIAMRAS (http://ckp.kiam.ru), the infrastructure of the Shared Research Facilities "High Performance Computing and Big Data" (CKP "Informatics") of FRC CSC RAS (Moscow). The authors thankfully acknowledges these institutions.

REFERENCES

1. A. Gorobets and P. Bakhvalov, "Heterogeneous CPU + GPU parallelization for high-accuracy scale-resolving simulations of compressible turbulent flows on hybrid supercomputers," *Computer Physics Communications*, **271**, 108231 (2022); https://doi.org/10.1016/j.cpc.2021.108231.
2. A. V. Gorobets, M. I. Neiman-Zade, S. K. Okunev, A. A. Kalyakin, and S. A. Soukov, "Performance of Elbrus-8C Processor in Supercomputer CFD Simulations," *Mathematical Models and Computer Simulations*, **11**, 914–923 (2019); https://doi.org/10.1134/S2070048219060073.
3. P. Bakhvalov and T. Kozubskaya, "EBR-WENO scheme for solving gas dynamics problems with discontinuities on unstructured meshes," *Computers & Fluids*, **157**, 312–324 (2017); https://doi.org/10.1016/j.compfluid.2017.09.004.
4. H. A. Van der Vorst, "Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems," *SIAM Journal on Scientific Computing*, **13**, 631–644 (1992); https://doi.org/10.1137/0913035.
5. E. Phillips and M. Fatica, "A CUDA implementation of the high performance conjugate gradient benchmark," *PMBS 2014. Lecture Notes in Computer Science*, **8966**, Springer, Cham, 68–84 (2015); https://doi.org/10.1007/978-3-319-17248-4_4.
6. I. Menshov and P. Pavlukhin, "Highly scalable implementation of an implicit matrix-free solver for gas dynamics on GPU-accelerated clusters," *J. Supercomputing*, **73**, 631–638 (2017); https://doi.org/10.1007/s11227-016-1800-1.
7. M. N. Petrov, V. A. Titarev, S. V. Utyuzhnikov, and A. V. Chikitkin, "A multithreaded OpenMP implementation of the LU-SGS method using the multilevel decomposition of the unstructured computational mesh," *Computational Mathematics and Mathematical Physics*, **57**, No. 11, 1856–1865 (2017); https://doi.org/10.1134/S0965542517110124.