Communications in Computer and Information Science 1331

More information about this series at http://www.springer.com/series/7899

Vladimir Voevodin · Sergey Sobolev (Eds.)

Supercomputing

6th Russian Supercomputing Days, RuSCDays 2020
Moscow, Russia, September 21–22, 2020
Revised Selected Papers

Springer

Editors
Vladimir Voevodin
Research Computing Center (RCC)
Moscow State University
Moscow, Russia

Sergey Sobolev
Research Computing Center (RCC)
Moscow State University
Moscow, Russia

ISSN 1865-0929 ISSN 1865-0937 (electronic)
Communications in Computer and Information Science
ISBN 978-3-030-64615-8 ISBN 978-3-030-64616-5 (eBook)
https://doi.org/10.1007/978-3-030-64616-5

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

The 6th Russian Supercomputing Days Conference (RuSCDays 2020), was held during September 21–22, 2020. The conference was dedicated to the 65th anniversary of the Research Computing Center, Moscow State University (RCC MSU) and the 100th anniversary of I.S. Berezin, the first RCC MSU director.

The conference was organized by the Supercomputing Consortium of Russian Universities and the Russian Academy of Sciences. The conference organization coordinator was RCC MSU.

Due to COVID-19 pandemic, for the first time the conference was held online. It was a new challenge for the conference organizers. However, online format provides a number of opportunities which are hard to perform offline, e.g. gathering invited speakers from all around the world. At the same time, the Organizing Committee did its best to keep the look and feel of the real-life meeting for attendees, including traditional sets of conference events – sessions, workshops, exhibitions, etc.

The conference was supported by the Russian Foundation for Basic Research and our respected partner (IBM), platinum sponsors (RSC, Merlion, Hewlett Packard Enterprise, Intel, Huawei), gold sponsors (NVIDIA, Dell Technologies in a partnership with CompTek), and silver sponsor (Xilinx). The conference was organized in a partnership with the ISC High Performance conference series.

RuSCDays was born in 2015 as a union of several supercomputing event series in Russia and quickly became one of the most notable Russian supercomputing international meetings. The conference caters to the interests of a wide range of representatives from science, industry, business, education, government, and students – anyone connected to the development or the use of supercomputing technologies. The conference topics cover all aspects of supercomputing technologies: software and hardware design, solving large tasks, application of supercomputing technologies in industry, exaflops-scale computing issues, supercomputing co-design technologies, supercomputing education, and others.

All 106 papers submitted to the conference were reviewed by three referees in the first review round. During single-blind peer reviewing, the papers were evaluated according to their relevance to the conference topics, scientific contribution, presentation, approbation, and related works description. After notification of conditional acceptance, the second review round arranged aimed at the final polishing of papers and also at the evaluation of authors' work after the referees' comments. After the conference, the 55 best papers were carefully selected to be included in this volume.

The proceedings editors would like to thank all the conference committees members, especially the Organizing and Program Committee members as well as the referees and reviewers for their contributions. We also thank Springer for producing these high-quality proceedings of RuSCDays 2020.

October 2020

Vladimir Voevodin
Sergey Sobolev

Organization

Steering Committee

V. A. Sadovnichiy (Chair)	Moscow State University, Russia
V. B. Betelin (Co-chair)	Russian Academy of Sciences, Russia
A. V. Tikhonravov (Co-chair)	Moscow State University, Russia
J. Dongarra (Co-chair)	The University of Tennessee, USA
A. I. Borovkov	Peter the Great Saint-Petersburg Polytechnic University, Russia
Vl. V. Voevodin	Moscow State University, Russia
V. P. Gergel	Lobachevsky State University of Nizhni Novgorod, Russia
G. S. Elizarov	NII Kvant, Russia
V. V. Elagin	Hewlett Packard Enterprise, Russia
A. K. Kim	MCST, Russia
E. V. Kudryashova	Northern (Arctic) Federal University, Russia
N. S. Mester	Intel, Russia
E. I. Moiseev	Moscow State University, Russia
A. A. Moskovskiy	RSC Group, Russia
V. Yu. Opanasenko	T-Platforms, Russia
G. I. Savin	Joint Supercomputer Center, Russian Academy of Sciences, Russia
A. S. Simonov	NICEVT, Russia
V. A. Soyfer	Samara University, Russia
L. B. Sokolinskiy	South Ural State University, Russia
I. A. Sokolov	Russian Academy of Sciences, Russia
R. G. Strongin	Lobachevsky State University of Nizhni Novgorod, Russia
A. N. Tomilin	Institute for System Programming, Russian Academy of Sciences, Russia
A. R. Khokhlov	Moscow State University, Russia
B. N. Chetverushkin	Keldysh Institutes of Applied Mathematics, Russian Academy of Sciences, Russia
E. V. Chuprunov	Lobachevsky State University of Nizhni Novgorod, Russia
A. L. Shestakov	South Ural State University, Russia

Program Committee

Vl. V. Voevodin (Chair)	Moscow State University, Russia
R. M. Shagaliev (Co-chair)	Russian Federal Nuclear Center, Russia

M. V. Yakobovskiy (Co-chair)	Keldysh Institutes of Applied Mathematics, Russian Academy of Sciences, Russia
T. Sterling (Co-chair)	Indiana University Bloominton, USA
S. I. Sobolev (Scientific Secretary)	Moscow State University, Russia
A. I. Avetisyan	Institute for System Programming, Russian Academy of Sciences, Russia
D. Bader	Georgia Institute of Technology, USA
P. Balaji	Argonne National Laboratory, USA
M. R. Biktimirov	Russian Academy of Sciences, Russia
A. V. Bukhanovskiy	ITMO University, Russia
J. Carretero	University Carlos III of Madrid, Spain
Yu. V. Vasilevskiy	Keldysh Institutes of Applied Mathematics, Russian Academy of Sciences, Russia
V. E. Velikhov	National Research Center "Kurchatov Institute", Russia
V. Yu. Volkonskiy	MCST, Russia
V. M. Volokhov	Institute of Problems of Chemical Physics, Russian Academy of Sciences, Russia
R. K. Gazizov	Ufa State Aviation Technical University, Russia
B. M. Glinskiy	Institute of Computational Mathematics and Mathematical Geophysics, Siberian Branch of Russian Academy of Sciences, Russia
V. M. Goloviznin	Moscow State University, Russia
V. A. Ilyin	National Research Center "Kurchatov Institute", Russia
V. P. Ilyin	Institute of Computational Mathematics and Mathematical Geophysics, Siberian Branch of Russian Academy of Sciences, Russia
S. I. Kabanikhin	Institute of Computational Mathematics and Mathematical Geophysics, Siberian Branch of Russian Academy of Sciences, Russia
I. A. Kalyaev	NII MVS, South Federal University, Russia
H. Kobayashi	Tohoku University, Japan
V. V. Korenkov	Joint Institute for Nuclear Research, Russia
V. A. Kryukov	Keldysh Institutes of Applied Mathematics, Russian Academy of Sciences, Russia
J. Kunkel	University of Hamburg, Germany
S. D. Kuznetsov	Institute for System Programming, Russian Academy of Sciences, Russia
J. Labarta	Barcelona Supercomputing Center, Spain
A. Lastovetsky	University College Dublin, Ireland
M. P. Lobachev	Krylov State Research Centre, Russia
Y. Lu	National University of Defense Technology, China
T. Ludwig	German Climate Computing Center, Germany
V. N. Lykosov	Institute of Numerical Mathematics, Russian Academy of Sciences, Russia

I. B. Meerov	Lobachevsky State University of Nizhni Novgorod, Russia
M. Michalewicz	University of Warsaw, Poland
L. Mirtaheri	Kharazmi University, Iran
S. G. Mosin	Kazan Federal University, Russia
A. V. Nemukhin	Moscow State University, Russia
G. V. Osipov	Lobachevsky State University of Nizhni Novgorod, Russia
A. V. Semyanov	Lobachevsky State University of Nizhni Novgorod, Russia
Ya. D. Sergeev	Lobachevsky State University of Nizhni Novgorod, Russia
H. Sithole	Centre for High Performance Computing, South Africa
A. V. Smirnov	Moscow State University, Russia
R. G. Strongin	Lobachevsky State University of Nizhni Novgorod, Russia
H. Takizawa	Tohoku University, Japan
M. Taufer	University of Delaware, USA
V. E. Turlapov	Lobachevsky State University of Nizhni Novgorod, Russia
E. E. Tyrtyshnikov	Institute of Numerical Mathematics, Russian Academy of Sciences, Russia
V. A. Fursov	Samara University, Russia
L. E. Khaymina	Northern (Arctic) Federal University, Russia
T. Hoefler	Eidgenössische Technische Hochschule Zürich, Switzerland
B. M. Shabanov	Joint Supercomputer Center, Russian Academy of Sciences, Russia
L. N. Shchur	Higher School of Economics, Russia
R. Wyrzykowski	Czestochowa University of Technology, Poland
M. Yokokawa	Kobe University, Japan

Industrial Committee

A. A. Aksenov (Co-chair)	Tesis, Russia
V. E. Velikhov (Co-chair)	National Research Center “Kurchatov Institute”, Russia
A. V. Murashov (Co-chair)	T-Platforms, Russia
Yu. Ya. Boldyrev	Peter the Great Saint-Petersburg Polytechnic University, Russia
M. A. Bolshukhin	Afrikantov Experimental Design Bureau for Mechanical Engineering, Russia
R. K. Gazizov	Ufa State Aviation Technical University, Russia
M. P. Lobachev	Krylov State Research Centre, Russia
V. Ya. Modorskiy	Perm National Research Polytechnic University, Russia
A. P. Skibin	Gidropress, Russia
S. Stoyanov	T-Services, Russia

A. B. Shmelev	RSC Group, Russia
S. V. Strizhak	Hewlett-Packard, Russia

Educational Committee

V. P. Gergel (Co-chair)	Lobachevsky State University of Nizhni Novgorod, Russia
Vl. V. Voevodin (Co-chair)	Moscow State University, Russia
L. B. Sokolinskiy (Co-chair)	South Ural State University, Russia
Yu. Ya. Boldyrev	Peter the Great Saint-Petersburg Polytechnic University, Russia
A. V. Bukhanovskiy	ITMO University, Russia
R. K. Gazizov	Ufa State Aviation Technical University, Russia
S. A. Ivanov	Hewlett-Packard, Russia
I. B. Meerov	Lobachevsky State University of Nizhni Novgorod, Russia
V. Ya. Modorskiy	Perm National Research Polytechnic University, Russia
S. G. Mosin	Kazan Federal University, Russia
I. O. Odintsov	RSC Group, Russia
N. N. Popova	Moscow State University, Russia
O. A. Yufryakova	Northern (Arctic) Federal University, Russia

Organizing Committee

Vl. V. Voevodin (Chair)	Moscow State University, Russia
V. P. Gergel (Co-chair)	Lobachevsky State University of Nizhni Novgorod, Russia
B. M. Shabanov (Co-chair)	Joint Supercomputer Center, Russian Academy of Sciences, Russia
S. I. Sobolev (Scientific Secretary)	Moscow State University, Russia
A. A. Aksenov	Tesis, Russia
A. P. Antonova	Moscow State University, Russia
A. S. Antonov	Moscow State University, Russia
K. A. Barkalov	Lobachevsky State University of Nizhni Novgorod, Russia
M. R. Biktimirov	Russian Academy of Sciences, Russia
Vad. V. Voevodin	Moscow State University, Russia
T. A. Gamayunova	Moscow State University, Russia
O. A. Gorbachev	RSC Group, Russia
V. A. Grishagin	Lobachevsky State University of Nizhni Novgorod, Russia
S. A. Zhumatiy	Moscow State University, Russia
V. V. Korenkov	Joint Institute for Nuclear Research, Russia
I. B. Meerov	Lobachevsky State University of Nizhni Novgorod, Russia

D. A. Nikitenko	Moscow State University, Russia
I. M. Nikolskiy	Moscow State University, Russia
N. N. Popova	Moscow State University, Russia
N. M. Rudenko	Moscow State University, Russia
A. S. Semenov	NICEVT, Russia
I. Yu. Sidorov	Moscow State University, Russia
L. B. Sokolinskiy	South Ural State University, Russia
K. S. Stefanov	Moscow State University, Russia
V. M. Stepanenko	Moscow State University, Russia
N. T. Tarumova	Moscow State University, Russia
A. V. Tikhonravov	Moscow State University, Russia
A. Yu. Chernyavskiy	Moscow State University, Russia
P. A. Shvets	Moscow State University, Russia
M. V. Yakobovskiy	Keldysh Institutes of Applied Mathematics, Russian Academy of Sciences, Russia

Contents

Parallel Algorithms

Supercomputer Simulation

Resource-Efficient Parallel CG Algorithms for Linear Systems Solving on Heterogeneous Platforms

Nikita S. Nedozhogin(✉), Sergey P. Kopysov, and Alexandr K. Novikov

Udmurt State University, Izhevsk, Russia
nedozhogin07@gmail.com, s.kopysov@gmail.com,
alexander.k.novikov@gmail.com

Abstract. The article discusses the parallel implementation of solving systems of linear algebraic equations on the heterogeneous platform containing a central processing unit (CPU) and graphic accelerators (GPU). The performance of parallel algorithms for the classical conjugate gradient method schemes when using the CPU and GPU together is significantly limited by the synchronization points. The article investigates the pipeline version of the conjugate gradient method with one synchronization point, the possibility of asynchronous calculations, load balancing between the CPU and GPU when solving the large linear systems. Numerical experiments were carried out on test matrices and computational nodes of different performance of a heterogeneous platform, which allowed us to estimate the contribution of communication costs. The algorithms are implemented with the combined use of technologies: MPI, OpenMP and CUDA. The proposed algorithms, in addition to reducing the execution time, allow solving large linear systems, for which there are not enough memory resources of one GPU or a computing node. At the same time, block algorithm with the pipelining decreases the total execution time by reducing synchronization points and aggregating some messages in one.

Keywords: Parallel computing on heterogeneous platform · Hybrid parallel technologies · Conjugate gradient method · Reduction of communications

1 Introduction

Computing accelerators are contained in the computing nodes of supercomputers and are used quite successfully in solving many computing problems despite the fact that the central processor (CPU) is idle after running the core functions on the accelerator. There are several more important conditions for which the

Supported by Russian Foundation for Basic Research (RFBR) according to the research project 17-01-00402. The work was carried out with the financial support of Udmurt State University in the contest of the grants "Scientific Potential", project No. 2020-04-03.

V. Voevodin and S. Sobolev (Eds.): RuSCDays 2020, CCIS 1331, pp. 122–133, 2020.
https://doi.org/10.1007/978-3-030-64616-5_11

joint use of the CPU and accelerators (for example, GPU) in parallel computing within the same problem seems to be promising.

Each architecture of the CPU and GPU has unique features and, accordingly, is focused on solving certain tasks for which typical, for example, high performance or low latency. Heterogeneous nodes containing and sharing CPU plus GPUs can provide an effective solution to a wide range of problems or one problem for which the parallel properties of the algorithms are changed and determined to be executed on one or another computing device. We also note the high energy efficiency of the heterogeneous computing systems.

One of the computationally complex operations in numerical methods is the solution of linear systems (SLAE). Currently, many parallel algorithms have been proposed that provide high performance and scalability when solving large sparse systems of equations on modern multiprocessor with a hierarchical architecture.

The construction of hybrid solvers with a combination of direct and iterative methods for solving SLAE allows the use of several levels of parallelism [1–5]. So in [6] a hybrid method for solving systems of equations of Schur complement by preconditioned iterative methods from Krylov subspaces was built and implemented when used together the cores of central (CPU) and graphic processing units (GPU). The classical preconditioned conjugate gradient method [7] was applied for the block ordered matrix and the separation of calculations in matrix operations between the CPU and one or more GPUs, when the system of equations in Schur complement was solved in parallel.

Currently, there are several approaches for computing load scheduling of the parallel conjugate gradient method (CG) on heterogeneous Multi-CPUs/Multi-GPUs platforms. Firstly, the separate steps of the CG algorithm are executed on a CPU or GPU, such as a preconditioner [8].

Second approach is based on the data mapping (matrix block and vectors) or separate tasks are carried out of all CG steps on the CPU or GPU [6].

In [9] task scheduling on Multi-CPUs/Multi-GPUs platforms for the classical CG from the `PARALUTION` library is executed in the `StarPU`. It's unified runtime system for heterogeneous multicore architectures which is developed in the INRIA laboratory (France).

In this paper, we consider an approach that reduces the cost of data exchanging between the CPU and GPU by reducing the number of synchronization points and pipelining of computing when SLAE is solved on heterogeneous platforms.

The Krylov subspace methods are some of the most effective options for solving large-scale linear algebra problems. However, the classical Krylov subspace algorithms do not scale well on modern architectures due to the bottleneck related to synchronization of computations. Pipeline methods of the Krylov subspace [10] with hidden communications provide high parallel scalability due to the global communications overlapping with computing, performing matrix-vector and dot products. The first work on reducing communications was related to a variant of the conjugate gradient method, having one communication at each iteration [11], using the three-term recurrence relations CG [12].

The next stage of development was the emergence of s-step methods from Krylov subspaces [13], in which the iterative process in the s-block uses various bases of Krylov subspaces. As a result, it was possible to reduce the number of synchronization points to one per s iterations. However, for a large number of processors (cores), communications can still take significantly longer than computing a single matrix-vector product. In [14] it was proposed a CG algorithm using auxiliary vectors and transferring a sequential dependence between the computing of matrix-vector product and scalar products of vectors. In this approach, the latency of communications is replaced by additional calculations.

2 Pipelined Algorithm of the Conjugate Gradient Method

We consider now the pipelined version of the conjugate gradient method, which is mathematically equivalent to the classical form of the preconditioned CG method and has the same convergence rate.

Algorithm 1: Pipelined algorithm CGwO.

```
1  r = b − Ax;
2  u = M^{-1} r;
3  w = Au;
4  γ1 = (r, u); δ = (w, u);
   while ||r||2/||b||2 > ε do
5      m = M^{-1} w ;
6      n = Am;
       if (j = 0) then
7          β = 0;
       else
8          β = γ1/γ0 ;
9      α = γ1/(δ − βγ1/α) ;
10     z = n + βz; w = w − αz; s = w + βs; r = r − αs;
11     p = u + βp; x = x + αp; q = m + βq; u = u + αq;
12     γ0 = γ1;
13     γ1 = (r, u); δ = (w, u);
```

In this algorithm, the modification of the vectors r_{j+1}, x_{j+1}, s_{j+1}, p_{j+1} and matrix-vector products provides pipeline computations. The computation of dot products (line 4) can be overlapped with the computation of the product by the preconditioner (line 3) and the matrix-vector product (line 3). However, the number of triads in the algorithm increases to eight, in contrast to three for the classic version and four in [13]. In this case, a parallel computation of triads and two dot products at the beginning of the iterative process and one synchronization point is possible.

The pipelined version CG presented in this work can be used with any preconditioner. There are two ways to organize computations in the preconditioned pipelined CG, which provide a compromise between scalability and the total number of operations [15]. Thus, the CG pipeline scheme is characterized by a different order of computations, the presence of global communication, which can overlap with local computations, such as matrix-vector product and operations with a preconditioner, and the possibility of organizing asynchronous communications.

The two variants of the conjugate gradient method were compared: the classical scheme and the pipelined one. Table 1 presents the results of numerical experiments where the execution time of a sequential version of the classical CG and the CGwO pipelined scheme (Algorithm 1) executed on the CPU and GPU are shown. Note that in the variants for the GPU, joint computation of all dot products of vectors in one kernel function was implemented, independently of each other. For this, when starting the CUDA kernel, the dimension of the `Grid` hierarchy of CUDA threads was set in two-dimensional form: 3 sets of blocks, each for performing computations on its own pair of vectors. This allowed us to reduce the number of exchanges between the CPU and GPU memory, combining all the resulting scalars in one communication.

Matrices from the SuiteSparse Matrix Collection (formerly the University of Florida Sparse Matrix Collection, https://sparse.tamu.edu/) were used in the test computations. The right hand side vector was formed as a row-wise sum of matrix elements. Thus, the solution of the system $Ax = b$, dimension $N \times N$ (with the number of nonzero elements nnz) is a vector $x = (1, 1, \ldots, 1)^T$.

For systems of equations of small dimension, the solution time on the CPU according to the classical CG scheme is significantly less than the GPU execution time for the same number of iterations (see Table 1). For large systems, the costs of synchronization and forwarding between the CPU and GPU overlap with the speed of the GPU. In the pipelined version of CGwO, the computational execution costs on the GPU are reduced almost threefold for all the considered systems of equations only due to the reduction of exchanges between the GPU and the CPU in the computation of dot products.

3 CG with the Combined Use of CPU and GPU

Let us consider the application of the Algorithm 1 for the parallel solution of super-large systems of equations on computing nodes, each of which contains several CPUs and GPUs. To solve SLAEs on several GPUs, we construct a block pipelined algorithm for the conjugate gradient method. On heterogeneous platform, data exchange between different GPUs within the same computing node is carried out with OpenMP technology, and the exchange between different computing nodes is carried out by MPI technology.

For example, consider a node containing a central eight-core processor and two graphics accelerators. The number of OpenMP threads is selected by the number of available CPU cores. The first two OpenMP threads are responsible

for exchanging data and running on two GPUs. Threads 2–6 provide computations on the CPU and can perform computations on a block of the SLAE matrix. The last thread provides data exchange with other computing nodes by MPI.

3.1 Matrix Partitioning

To divide the matrix A into blocks, we construct the graph $G_A(V, E)$, where $V = \{i\}$ is the set of vertices associated with the row index of the matrix (the number of vertices is equal to the number of rows of the matrix A); $E = \{(i, j)\}$ is the set of edges. Two vertices i and j are considered to be connected if the matrix A has a nonzero element with indices i and j. The resulting graph is divided into subgraphs whose number is d. For example, to split a graph, you can use the [16] layer-by-layer partitioning algorithm, which reduces communication costs due to the need to exchange only with two neighboring computing nodes. After that, each vertex of the graph is assigned its own GPU or CPU. On each computing unit, the vertices are divided into internal and boundary. The latter are connected with at least one vertex belonging to another subgraph.

After partitioning, each block A_k of the original matrix A contains the following submatrices:

- $A_k^{[i_k,i_k]}$ – matrix associated with the internal vertices;
- $A_k^{[i_k,b_k]}$, $A_k^{[b_k,i_k]}$ – matrices associated with the internal and boundary vertices;
- $A_k^{[b_k,b_l]}$ – matrix associated with the boundary vertices of the k-th and l-th blocks.

Then the matrix A can be written in the following form:

$$A = \begin{pmatrix} A_1^{[i_1,i_1]} & A_1^{[i_1,b_1]} & \cdots & 0 & 0 \\ A_1^{[b_1,i_1]} & A_1^{[b_1,b_1]} & \cdots & 0 & A_1^{[b_1,b_d]} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & A_d^{[i_d,i_d]} & A_d^{[i_d,b_d]} \\ 0 & A_d^{[b_d,b_1]} & \cdots & A_d^{[b_d,i_d]} & A_d^{[b_d,b_d]} \end{pmatrix}.$$

We divide the matrix-vector product $n = Am$ into two components by using the obtained partition:

$$n_k^b = A_k^{[b_k,i_k]} m_k^i + \sum_{l=1}^{l \le d} A_k^{[b_k,b_l]} m_l^b, \qquad n_k^i = A_k^{[i_k,i_k]} m_k^i + A_k^{[i_k,b_k]} n_k^b. \tag{1}$$

Here k corresponds to the computing device. The block representation of the vectors involved in the algorithm is inherited from the matrix partitioning. For example, the vector m has the form $m^T = \left(m_1^i, m_1^b, \ldots, m_k^i, m_k^b, \ldots, m_d^i, m_d^b\right)$. The implementation of the matrix-vector product reduces the cost of communication between blocks at each iteration of conjugate gradient method. To perform this operation, an exchange of vectors m_k^b is required, the size of which is less than the dimension of the initial vector m.

The partitioning of the preconditioner M is carried out in a similar way.

3.2 Block Pipelined Algorithm

The matrix blocks were mapped on the available CPU and GPU with the block partitioning of the matrix and vectors. The number and size of blocks let on to map the load in accordance with the performance of the computing units, including the allocation of several blocks to one.

Algorithm 2: Block algorithm CGwO performed on k-th device

Data: Matrix partitioning into blocks $A_k^{[i_k,i_k]}$, $A_k^{[i_k,b_k]}$, $A_k^{[b_k,i_k]}$, $A_k^{[b_k,b_l]}$.

$r = b$;
$u = M^{-1}r$;
// Parallel algorithm branches
// $(\texttt{CPU} \vee \texttt{GPU})_k$ // CPU
$w_k^i = A_k^{[i_k,i_k]} \cdot u_k^i + A_k^{[i_k,b_k]} \cdot u_k^b$; Assembly of the vectors u_k^b;
$w_k^b = A_k^{[b_k,b_k]} \cdot u_k^b + A_k^{[b_k,i_k]} \cdot u_k^i$; $w_h^b = \sum_{l=1,l\neq k}^{l\leq d} A_k^{[b_k,b_l]} \cdot u_k^b$;
Copying w_h^b on the $\texttt{GPU}_k$;
$w_k^b = w_k^b + w_h^b$;
$m = M^{-1}w$; Assembly of the vectors m_k^b;
$\gamma_{1k} = (r_k, u_k)$; $\delta_k = (w_k, u_k)$; Assembly $\delta = \sum_k \delta_k; \gamma_1 = \sum_k \gamma_{1k}$;
while $||r||_2/||b||_2 > \varepsilon$ **do**

$n_k^i = A_k^{[i_k,i_k]} \cdot m_k^i + A_k^{[i_k,b_k]} \cdot m_k^b$; $n_h^b = \sum_{l=1,l\neq k}^{l\leq d} A_k^{[b_k,b_l]} \cdot m_k^b$;
$n_k^b = A_k^{[b_k,b_k]} \cdot m_k^b + A_k^{[b_k,i_k]} \cdot m_k^i$; Copying n_h^b on the GPU_k;

$n_k^b = n_k^b + n_h^b$; $\beta = ((j = 0)\,?\,0 : \gamma_1/\gamma_0)$;
$z = n + \beta z$; $\alpha = \gamma_1/(\delta - \beta\gamma_1/\alpha)$;
$w = w - \alpha z$;
$q = m + \beta q$;
$s = w + \beta s$; Assembly of the vectors w_k^b;
$p = u + \beta p$;
$x = x + \alpha p$;
$r = r - \alpha s$; Assembly vectors m_k^b;
$u = u + \alpha q$;
$m = M^{-1}w$; Assembly $\delta = \sum_k \delta_k; \gamma_1 = \sum_k \gamma_{1k}$;
$\gamma_0 = \gamma_1$;
$\gamma_{1k} = (r_k, u_k)$; $\delta_k = (w_k, u_k)$;

Let us represent parallel block scheme of the method CGwO that is performed each k-th computing unit in the form of Algorithm 2. Two parallel branches of this algorithm are executed accordingly on the CPU and GPU/CPU. Operations performed in parallel are shown in one line of the algorithm. Vector operations on each computing unit occur in two stages, for internal and boundary nodes. The designations of the internal and boundary nodes for vectors are omitted, with the exception of the matrix-vector multiplication. Dot products are performed independently by each computing unit on its parts of vectors. The summation

of intermediate scalars occurs in parallel threads responsible for communication, which is the synchronization point at each iteration of the algorithm.

In block CGwO, compared to Algorithm 1, the preconditioning step has been moved (line 5 to line 21). This is done in order to combine vector operations on the computing unit and the assembly of the vector parts of the right hand side to perform matrix-vector multiplication in preconditioning. The 13 line on the right uses the ternary operator: if $j = 0$, then $\beta = 0$, in other cases $\beta = \gamma_1/\gamma_0$. The subscript h is used for vectors that are stored only in CPU memory.

Numerical experiments on the Algorithm 2 were carried out on heterogeneous platform with various configuration of computing nodes containing several CPUs and GPUs. In the general case, the parallel computing on several heterogeneous computing nodes containing one or more CPUs and several GPUs is implemented by the combination of several technologies: MPI, OpenMP and CUDA.

Let us consider the software organization of computations using as example some cluster, which includes two computing nodes (8 CPU cores and 2 GPUs). Each computing node is associated with a parallel MPI process. In a parallel process, 9 parallel OpenMP threads are generated, which is one more than the available CPU cores. The eighth OpenMP thread is responsible for communications between different computing nodes (using MPI technology, vector assembly using the `Allgatherv` function, adding scalars `Allreduce`) and various GPUs. In the 2 Algorithm, the operations performed by this thread are presented to the right. Zero and first OpenMP threads are the host threads for one of the available GPU devices and are responsible for transfer data between the GPU-CPU (calls to asynchronous copying functions) and auxiliary computations. Each available GPU device (further considered as a computing unit) is associated with one of the parallel OpenMP threads, which is responsible for transferring data between the GPU and CPU (calls to asynchronous copy functions) and participates with the eighth treads in matrix-vector product on boundary vertices (lines 4, 9 right column). The remaining parallel threads (second to seventh) perform the calculations as a separate computing unit for their matrix block. The operations performed by computing units in the 2 Algorithm are shown on the left.

The preconditioning in lines 2, 7 and 21 implies the use of block matrix-vector multiplication of the form (1) considered above.

4 Numerical Experiments

The numerical experiments were performed on the heterogeneous partitions of cluster “Uran” on the computing nodes (CNs) of several types of Supercomputer center IMM UB RAS, Yekaterinburg, Russia. The cluster partitions with the following characteristics were used:

– partition “debug”: 4 CNs tesla [31–32,46–47] with two 8-cores CPU Intel Xeon E5-2660 (2.2 GHz), cache memory is 20 MB L3 cache, RAM is 96 GB and 8 GPU Tesla M2090 (6 GB per device), network is 1 Gb/s Ethernet.

– partition "tesla[21–30]": 10 CNs with two 6-cores CPU Intel Xeon X5675 (3.07 GHz), RAM is 192 GB, cache memory is 12 MB L3 cache and 8 GPU Tesla M2090 (6 GB per device), with network is Infiniband 20 Gb/s.
– partition "tesla[33–45]": 13 CNs with two 8-cores CPU Intel Xeon E5-2660 (2.2 GHz), cache memory is 20 MB L3 cache, RAM is 96 GB, and 8 GPU Tesla M2090 (6 GB per device), network is Infiniband 20 Gb/s.
– partition "tesla[48–52]": 5 CNs with two 8-cores CPU Intel Xeon E5-2650 (2.6 GHz), cache memory is 20 MB L3 cache, RAM is 64 GB and 3 GPU Tesla K40m (12 GB per device), network is Infiniband 20 Gb/s.

The results of comparing two algorithms of the conjugate gradient method on SLAEs containing test matrices are presented in Table 1. The results are given for several types of computing nodes using a single graphics accelerator.

Table 1. Statistics of the test problems. Problem names, dimensions (N), number of nonzeros (nnz), device type (DT) and problem analysis in terms of the timing in seconds.

Matrix	N	nnz	# iter.	DT	Time, s	
					CG	CGwO
Plat362	362	5786	991	M2090	6.88E-01	**3.07E-01**
				K40m	4.13E-01	3.12E-01
1138_bus	1138	4054	717	M2090	3.81E-01	**1.84E-01**
				K40m	5.31E-01	2.01E-01
				debug	6.82E-01	1.90E-01
Muu	7102	170134	12	M2090	2.64E-01	4.68E-03
				K40m	3.31E-01	**4.55E-03**
Kuu	7102	340200	378	M2090	4.31E-01	**1.31E-01**
				K40m	4.39E-01	1.35E-01
Pres_Poisson	14822	715804	661	M2090	6.72E-01	3.13E-01
				K40m	6.346E-01	**2.73E-01**
Inline_1	503712	36816342	5642	M2090	4.74E+01	5.17E+01
				K40m	**3.06E+01**	3.37E+01
Fault_639	638802	28614564	4444	M2090	3.83E+01	4.32E+01
				K40m	**2.44E+01**	2.77E+01
				debug	2.44E+01	2.77E+01
thermal2	1228045	8580313	2493	M2090	1.35E+01	1.82E+01
				K40m	**8.33E+00**	1.18E+01
G3_circuit	1585478	7660826	592	M2090	3.43E+00	4.32E+00
				K40m	**1.94E+00**	2.92E+00
Quenn_4147	4147110	399499284	8257	M2090	5.46E+02	5.78E+02
				K40m	**3.55E+02**	3.75E+02

The matrices are ordered by increasing the order of the system of equations (N) and the number of nonzero elements (nnz). Bold indicates the best time to solve the system in each case. The pipelined algorithm CGwO showed a reduction in execution time on small SLAEs which are characterized by a small computing load, due to which a reduction in communications provides less time. Note that the classic CG algorithm was implemented based on CUBLAS, while the CGwO variant uses matrix and vector operations of its own GPU implementation.

For systems `Inline_1` and `Fault_639`, the execution time of the pipeline algorithm is 10 and 13.5% longer than the block version of CG, which is associated with additional vector operations that are not blocked by reduced communications. With a decrease in the number of iterations, for example, for solving a large system with `G3_circuit`) with an approximately equal number of equations with `thermal2`, the execution time of the CG and CgwO algorithms on one GPU increases slightly. For the system (`thermal2` and `G3_circuit`) the increase in costs becomes more significant.

Table 2 presents the results of the block variant of the algorithms for computing on several computing nodes for systems with small dimension matrices. Here are the results for 2 and 3 subdomains. Each subdomain was considered on a separate computing node. Communications were carried out using MPI technology. A significant influence of network characteristics on the performance of block methods can be seen in Table 2 for system of equations with matrix `1138_bus`. Computations for these SLAEs were performed at various computing nodes with different throughput and latency of the network. In numerical experiments on the CNs (partition "debug") connected by a Gigabit network, communication costs significantly increase the execution time of the CG algorithm.

Table 2. Time of solving by the block algorithms CG and CGwO on CPU/GPU, s.

Matrix/DT	CG/#blocks		CGwO/#blocks	
	2	3	2	3
`Plat362`/M2090	1.55E+00		**1.22E+00**	
/K40m	1.92E+00	1.56E+00	1.28E+00	1.31E+00
`1138_bus`/M2090	1.84E+00		**9.28E-01**	
/K40m	1.90E+00	1.85E+00	1.03E+00	1.04E+00
/debug	1.25E+01		**5.36E+00**	
`Muu`/M2090	6.12E-01		2.29E-01	
/K40m	6.59E-01	5.64E-01	2.89E-01	**2.88E-01**
`Kuu`/M2090	1.30E+00		**6.43E-01**	
/K40m	1.29E+00	1.36E+00	6.81E-01	7.95E-01
`Pres_Poisson`/M2090	1.55E+00		**9.57E-01**	
/K40m	1.60E+00	1.66E+00	1.02E+00	1.19E+00
`G3_circuit`/M2090	4.27E+00		3.99E+00	
/K40m	4.04E+00	3.510E+00	3.27E+00	**2.77E+00**

For example, in the variant `1138_bus` on the cluster partition "debug", the execution time of the pipeline algorithm is 3.6 times less (the line "debug" in Table 2 and any row in Table 1). Using the Infiniband 20 Gb/s communication network reduces the execution time for all presented systems of equations (lines "M2090" and "K40m").

When reducing the computational load, a decrease in the number of synchronization points and the consolidation of transfers per transaction is more pronounced. This shows a comparison of systems with matrices `Kuu` and `Muu`. Both systems have an equal number of equations and nonzero elements, but the conditionality of these matrices is significantly different and, as a consequence, the number of iterations in the conjugate gradient method is different. Table 1 shows that using the pipeline algorithm for the matrix `Muu` gives speedup by 70 times, compared with the matrix `Kuu`, where the speedup is only 2.8.

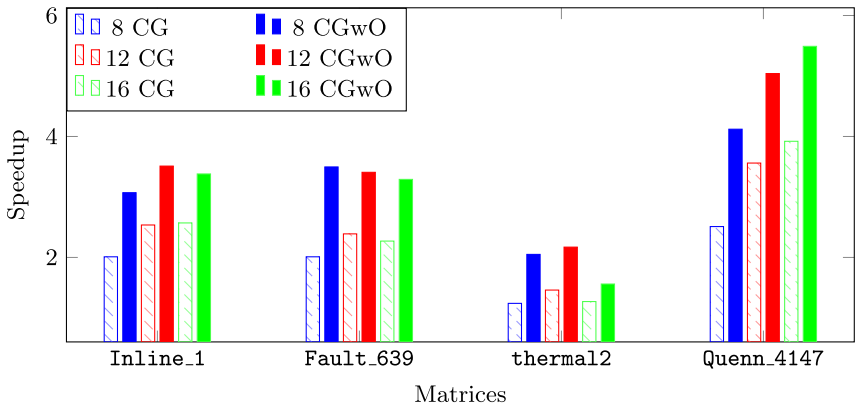

Fig. 1. Speedup of the block algorithms CG and CGwO

Figure 1 presents the results of accelerating the block algorithms of the conjugate gradient method, when divided into a larger number of blocks, accordingly 8, 12 and 16. To compute the speedup, parallel application was run repeatedly with different mapping of subdomains to several CPUs and GPUs. For example, in the case of 12 subdomains, variants were considered: 2 CPUs with 6 GPUs, 3 CPUs with 4 GPUs, 6 CPUs with 2 GPUs. The best time is shown.

The speedup was considered relative to the option on one GPU from Table 1. An application that implements this algorithm was executed in the exclusive mode of the computing node but not of the network.

As can be seen from the presented results, the pipelined CG shows the speedup greater than the classic version of conjugate gradient method. Wherein, for the largest of the considered matrices `Quenn_4147`, the speedup achieves 5.49 times, while the classical version gives 3.92 as maximum. For the strongly sparse matrix `thermal2`, block algorithms don't give high speedup (maximum is 1.56),

since the computational load depends mainly on the number of the nonzero elements.

An analysis of the results showed that reducing the data size due to the matrix partitioning and reducing the synchronization points slightly decrease the impact of communication costs on the total algorithm performance. Only the use of computing nodes connected by Infiniband allowed us to get speedup when computing on several computing nodes. The matrix partitioning into blocks allowed to decrease the execution time of the pipelined block algorithm in comparison with the conjugate gradients on one node on the matrices `Inline_1`, `Fault_639` by reducing the computational load on one GPU.

Large systems `thermal2`, `G3_circuit`, solved by the block of the CGwO algorithm, as well as the reduction in communications costs and synchronization points, do not overlap the increasing costs of additional vector operations.

5 Conclusion

The heterogeneous computing platforms containing and sharing CPU + GPUs provide an effective solution to a wider range of problems with high energy efficiency when CPU and GPUs are uniformly loaded.

The parallel implementation of the solution of systems of linear algebraic equations on a heterogeneous platform was considered. The performance of parallel algorithms for classical conjugate gradient method is significantly limited by synchronization points when using the CPU and GPU together. A pipelined algorithm of the conjugate gradient method with one synchronization point was proposed. Also, it is provided the possibility of asynchronous computations, load balancing between several GPUs located both on the same computing node and for a GPU cluster when solving systems of large-dimensional equations. To further increase the efficiency of calculations, it is supposed to study not only the communication load of the algorithms but also the distributing of the computational load between the CPU and GPU. To obtain more reliable evaluation of communications costs, it is necessary to conduct a series of computational experiments on supercomputer with a completely exclusive mode of operation and a large number of heterogeneous nodes.

The following conclusions can be drawn from the analysis of data obtained during numerical experiments: the use of a pipeline algorithm reduces communication costs, but increases computational ones. For systems of small sizes or with a small number of iterations, this reduces the execution time of the algorithm when using a single GPU. For systems of large dimensions, a reduction in execution time, in comparison with CG, is possible only with a sufficiently small partition of the matrix into blocks, in which the increased computing costs overlap the communication decrease.

The proposed block algorithms, in addition to reducing the execution time, allow solving large linear systems that requires memory resources not provided by one GPU or computing node. At the same time, the pipelined block algorithm reduces the overall execution time by reducing synchronization points and combining communications into one message.

References

1. Agullo, E., Giraud, L., Guermouche, A., Roman, J.: Parallel hierarchical hybrid linear solvers for emerging computing platforms. C. R. Mec. **333**, 96–103 (2011)
2. Gaidamour, J., Henon, P.: A parallel direct/iterative solver based on a Schur complement approach. In: IEEE 11th International Conference on Computational Science and Engineering, pp. 98–105. San Paulo (2008)
3. Giraud, L., Haidar, A., Saad, Y.: Sparse approximations of the Schur complement for parallel algebraic hybrid solvers in 3D. Numer. Math. **3**, 276–294 (2010)
4. Rajamanickam, S., Boman, E.G., Heroux, M.A.: ShyLU: a hybrid-hybrid solver for multicore platforms. In: IEEE 26th International Parallel and Distributed Processing Symposium, Shanghai, pp. 631–643 (2012)
5. Yamazaki, I., Rajamanickam, S., Boman, E., Hoemmen, M., Heroux, M., Tomov, S.: Domain decomposition preconditioners for communication-avoiding Krylov methods on a hybrid CPU/GPU cluster. In: Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis (SC14), pp. 933–944 (2014)
6. Kopysov, S., Kuzmin, I., Nedozhogin, N., Novikov, A., Sagdeeva, Y.: Scalable hybrid implementation of the Schur complement method for multi-GPU systems. J. Supercomputing **69**(1), 81–88 (2014). https://doi.org/10.1007/s11227-014-1209-7
7. Hestenes, M.R., Stiefel, E.: Methods of conjugate gradients for solving linear systems. J. Res. Nat. Bur. Stan. **49**(6), 409–436 (1952)
8. Jamal, A., Baboulin, M., Khabou, A., Sosonkina, M.A.: A hybrid CPU approach GPU for the parallel algebraic recursive multilevel solver pARMS. In: 2016 18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC). Timisoara, pp. 411–416 (2016)
9. Kasmi, N., Zbakh, M., Mahmoudi, S.A., Manneback, P.: Performance evaluation of StarPU schedulers with preconditioned conjugate gradient solver on heterogeneous (Multi-CPUs/Multi-GPUs) architecture. IJCSNS Int. J. Comput. Sci. Netw. Secur. **17**, 206–215 (2017)
10. Cornelis, J., Cools, S., Vanroose, W.: The Communication-Hiding Conjugate Gradient Method with Deep Pipelines. https://arxiv.org/pdf/1801.04728.pdf. Accessed 14 Apr 2020
11. D'Azevedo, E.F., Romine, C.H.: Reducing communcation costs in the conjugate gradient algorithm on distributed memory multiprocessors. Technical report ORNL/TM-12192, Oak Ridge National Lab (1992)
12. Linear Algebra. Springer, Singapore (2018). https://doi.org/10.1007/978-981-13-0926-7_7
13. Chronopoulos, A.T., Gear, C.W.: s-step iterative methods for symmetric linear systems. J. Comput. Appl. Math. **25**(2), 153–168 (1989)
14. Ghysels, P., Vanroose, W.: Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm. Parallel Comput. **40**(7), 224–238 (2014)
15. Gropp, W.: Update on libraries for blue waters. http://wgropp.cs.illinois.edu/bib-/talks/tdata/2011/Stream-nbcg.pdf. Accessed 14 Apr 2020
16. Kadyrov, I.R., Kopysov, S.P., Novikov, A.K.: Partitioning of triangulated multiply connected domain into subdomains without branching of inner boundaries. Uchenye Zap. Kazanskogo Univ. Ser. Fiz. Matematicheskie Nauki **160**(3), 544–560 (2018)