Supercomputing Frontiers and Innovations

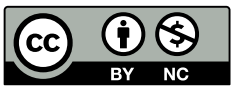
2018, Vol. 5, No. 2

Scope

- Enabling technologies for high performance computing
- Future generation supercomputer architectures
- Extreme-scale concepts beyond conventional practices including exascale
- Parallel programming models, interfaces, languages, libraries, and tools
- Supercomputer applications and algorithms
- Distributed operating systems, kernels, supervisors, and virtualization for highly scalable computing
- Scalable runtime systems software
- Methods and means of supercomputer system management, administration, and monitoring
- Mass storage systems, protocols, and allocation
- Energy and power minimization for very large deployed computers
- Resilience, reliability, and fault tolerance for future generation highly parallel computing systems
- Parallel performance and correctness debugging
- Scientific visualization for massive data and computing both external and in situ
- Education in high performance computing and computational science

Editorial Board

Contents



DOI: 10.14529/jsfi180201

Deep Analysis of Job State Statistics on Lomonosov-2 Supercomputer

Dmitry A. Nikitenko[1], *Vadim V. Voevodin*[1], *Sergey A. Zhumatiy*[1]

It is a common knowledge that the increasingly growing capabilities of HPC systems are always limited by a number of efficiency related issues. The reasons can be very different: hardware failures, incorrect job scheduling, peculiarities of algorithm, chosen programming technology specifics, etc. Most of these issues can be detected after precise analysis, but is a very resourceful way to study every application run. Therefore we performed less complicated analysis of the whole supercomputer job flow. In this paper we share our experience of analyzing user applications job states assigned by the SLURM resource manager that is used on the Lomonosov-2 system at Supercomputing center of Lomonosov Moscow State University. The statistics on job states was collected and it revealed that the ratio of correctly finished jobs (with the COMPLETED state) was rather low. The jobs owners were asked if the distribution of their jobs states is normal regarding their applications. This user feedback was processed, and some new ways of efficiency gain were revealed as the result.

Keywords: HPC, supercomputer, parallel computing, efficiency analysis, job state.

Introduction

Supercomputers are constantly getting more and more powerful – the current #1 supercomputer in the world, Summit, has more than 180 PFlop/s in peak [1]. Such trend is dictated by the desire to solve increasingly complex computational problems that arise in various scientific areas. But in order to achieve this goal it is necessary not only to increase the performance of computing systems, but also to ensure high efficiency of their functioning. And this is a big problem in modern systems, since the efficiency of their usage is usually low [2].

This problem is directly related to the low efficiency of individual parallel applications running on the supercomputer. One of the main reasons for applications not being effective is quite a significant ratio of incorrect application launches. There are various reasons for that: incorrect input data or wrong input parameters specified in the application, issues with the file system or interconnect switches, MPI-related issues, compute node fails, etc. All these problems decrease the useful utilization of supercomputers, so it is natural for the management of supercomputer centers to try minimizing the number of such problems.

In the Research computing center of Moscow State University (RCC MSU) we decided to conduct a study of these issues on the Lomonosov-2 supercomputer installed in our supercomputer center. One of the basic information that can be used to estimate the correctness of supercomputer job flow is the states of finished job runs. We collected the statistics on the states of all finished jobs during several months of Lomonosov-2 functioning and found out that the ratio of correctly finished jobs (with the COMPLETED state) is surprisingly low. So it was decided to ask the users about the reasons of such seemingly unproductive behavior, concerning their jobs.

This paper is devoted to the deep analysis of job state statistics on Lomonosov-2 supercomputer based on the communication with the users.

The rest of the paper is organized as follows. Section 1 describes our previous research concerning efficiency analysis of particular parallel applications and the supercomputer usage in

[1]Research Computing Center of Lomonosov Moscow State University, Moscow, Russian Federation

general, as well as related works performed by other scientists. The process of collecting data for the job state analysis is described in Section 2. Section 3 is devoted to the analysis of collected data as well as the results and useful ideas developed during this study. Last section contains conclusion and our plans for future work.

1. Background and Related Work

The efficiency of any supercomputer is affected by many factors. Our team has a long experience in optimizing the efficiency of our supercomputers. The first important step is to understand the supercomputer behavior in general and identify the key factors that cause inefficiency. Then it is possible to eliminate or at least to reduce the impact of the found factors and evaluate the results of taken actions.

Our RCC MSU team uses the approach described in [3], which enables us to collect the most detailed information on both the hardware part [4] and the computing job [5]. The collected data is analyzed both in automatic and manual modes. The system monitoring data gives information on how well the supercomputer is loaded. But a high load does not always mean high efficiency – if a large amount of resources is spent on jobs that did not produce a result, then this time was wasted.

Similar statistics can be found, for example, in [6], where various aspects of computing jobs behavior on the Blue Waters supercomputer are considered, although there is no data on the distribution of job exit states. A more interesting analysis is given in the article [7], which deals with distributions of jobs of the Kraken supercomputer, including such characteristics as the frequency of cancellation of jobs and the accuracy of setting their the time limits. Another existing solution is based on using XDMoD tool designed for managing of HPC systems [8, 9].

In these papers, a lot of analyzing results of the job behavior on the largest supercomputers are given, but no conclusions or assumptions are made on how to improve their efficiency. In our work we propose methods for increasing the efficiency of the supercomputer based on the results achieved in communication with the users of Lomonosov-2 supercomputer.

2. Collecting Information about Job States

We store all the data collected by monitoring system on running and finished jobs, as well as accounting data from SLURM manager, in our single PostgreSQL database. Therefore we only need to filter out needed information and analyze it accordingly. In this study, we analyzed all jobs finished since January 1, 2018 till May 28, 2018. Our goal was to find cases and users whose jobs mostly seems as “abnormal” according to the exit states of their jobs. For each user, a total amount of cpu-hours for his jobs was computed and compared with the amount of cpu-hours for his jobs with particular exit states. We used the following criteria in order to detect users with “abnormal” behavior:

- (amount of cpu-hours for user jobs with COMPLETED state) / (Total amount of cpu-hours for this user) < 0.5;
- *(The same for jobs with CANCELLED or FAILED state)* > 0.2;
- *(The same for jobs with TIMEOUT state)* > 0.2.

If at least one of these criteria was fulfilled, user was asked via email to explain the reasons for such big ratio of jobs with not COMPLETED state. We sent personalized statistics about jobs in emails to help users to better understand our interest. We chose thresholds for these

criteria empirically, using our experience and expectations. This process was repeated for each of 3 main partitions of Lomonosov-2 supercomputer, because different partitions have specific limits and different sets of jobs running on them. Therefore, several users were emailed more than once with questions about different partitions.

We detected 97 cases of criteria triggering, each case was a unique combination of user name and supercomputer partition. Since we are interested in analyzing overall statistics on the behavior of the whole supercomputer, there is no need to analyze small cases, so we filtered out ones with less than 20 000 cpu-hours. It resulted in 65 cases for 53 different users; an email was sent for each of these cases. It should be noted that these users cover the majority of the Lomonosov-2 usage – these users have consumed 85% of cpu-hours provided by the supercomputer during the considered time period.

Most of the users received only 1 email, but there were 8 users that received 2 emails (concerning 2 different partitions) and 2 users was asked questions about all our partitions, therefore each received 3 emails.

In found cases all 3 criteria were triggered 23 times, only 2 criteria – 26 times and 17 times only one criteria was triggered. We sent questions about small ratio of COMPLETED states (first criteria) 47 times, as well as 36 and 52 questions about big ratio of CANCELLED+FAILED and TIMEOUT states, accordingly.

3. Analysis of Job State Information

After collecting the information on the job states from the users, the next step is to analyze the obtained data. We received answers from 39 users out of 53 (74% users responded). The responded users were responsible for 56.5% of all CPU hours consumed by all users during the considered time period (between Jan 1 and May 28). It seems to be a good result since there were no obligations to provide feedback answering our questions.

We grouped all the answers from the users based on the job state (TIMEOUT, CANCELLED and FAILED states) and the reason for such job behavior. Further the statistics for each state is described.

3.1. TIMEOUT Job State

We received the most number of answers about big ratio of TIMEOUT job state – 32 users responded. The majority of these users (21 of 32) noticed that such behavior was absolutely normal for their jobs since the current time limit for job execution was not enough for them. In this case a user ran the job till the time limit and created a control point, so he could continue the calculation from that point in the next job run.

Many of users intentionally going for the time limit showed us that we needed to distinguish such behavior from unintentionally exiting with TIMEOUT job state, since the former was a normal job behavior and the latter could be an indicator of incorrect or inefficient job execution. So, after analyzing this information, we decided to use special option specified for the SLURM resource manager that would be used in Lomonosov-2 supercomputer. Specifying this option, users can explicitly mark that this new job is planned to reach TIMEOUT limit. This will help us in future to divide such cases, leading to more accurate statistics.

Next, 7 users mentioned that they had not correctly estimated the time needed for job execution; 2 of them added that they had already fixed that issue, and 4 of them said that the problem was most likely related to the issues with the used software package.

Another 4 users admitted that the TIMEOUT job state was due to inefficient software implementation bad parallelization, outdated software, packages working surprisingly inefficiently on Lomonosov-2 supercomputer. This is a good opportunity for optimization since it will help users to conduct experiments faster, and to increase the overall efficiency of the supercomputer as well.

The last 7 of 32 users said that the reason for many of their jobs to be finished with TIMEOUT job state was that they evaluated new methods and approaches. In this case it is hard to deal with these issues, since the problem is not technical but more of semantic kind.

3.2. CANCELLED Job State

We received response from 24 users about big ratio of jobs with CANCELLED job state. 10 of them said that, as in the case of TIMEOUT state, this situation was completely normal. It was dictated by the specifics of such jobs: it was impossible to predict when the calculation is completed in advance, so users had to specify bigger time limit. And when users decided that the needed computational goal was achieved, they manually canceled the job, since after that moment there would be no useful work done. Since such job behavior is normal, it would be useful to divide this scenario from other types of cancelled jobs; but as for now, there is no simple way to accurately identify it.

Other types of jobs with CANCELLED job state are the following. 13 users stated that they needed debugging job launches, but the reasons were different – it could be rather a test launch in order to check the correctness of planned experiment or an evaluation of new method or approach. The main optimization that can be done in this situation is to reduce the number of test launches in the main working partitions of the supercomputer – in our case, a special test partition should be used. Since it seems that in some cases users do not use this test partition on purpose, it is probably necessary to adjust the quotas and time limits for this partition to better suit the needs of users.

It is worth mentioning separately the interesting response of the last user. This user said that, due to the quotas on the maximum number of jobs running on the Lomonosov-2 supercomputer, he sometimes needed to cancel less priority jobs in order to be able to launch more priority jobs. This situation requires a more detailed discussion with the user – maybe this could be a signal to system administrators to adjust the quotas and policies.

3.3. FAILED Job State

The information about failed jobs is usually the most useful for us since it covers most of the situations that we can potentially fix. We received answers from 19 users which can be grouped into 5 different types.

5 users claimed that their jobs failed due to different system problems like compute node fails or issues with MPI library. This is exactly the situation where we as system administrators need to take prompt actions. Modern supercomputers are very big and complex, so such issues are going to happen in any way, therefore our goal is to eliminate such problems as quickly as

possible and take prophylactic measures trying to minimize the frequency of occurrence of such errors.

The problems on the user software side were the main reasons of failed jobs, as stated by the biggest group of the responders (7 out of 19). As in the case of inefficient implementation mentioned in subsection 3.1, it is desirable to interact with such users in order to detect the root causes of these problems and to correct and optimize their applications, which will lead to the increase of supercomputer useful utilization and help users to conduct experiments more effectively.

Next, 2 users stated that there were some system problems that led to failed jobs, but at the moment they were fixed by the supercomputer support team. In such case there is no need to take any action.

As mentioned by 3 more users, the big ratio of failed jobs in their cases was caused by the need for many test launches. As in the case of cancelled jobs, we should try to reduce the number of such launches by convincing users to use test partitions created specifically for this goal.

The last type of responses is quite unusual in our opinion. 2 users said the FAILED job state was a completely normal situation that could appear in software they were using. On the one hand, this situation should be fixed because FAILED job state should be only used in case of some errors occurred, but here it is used to label normal behavior. On the other hand, this is usually quite difficult to change the behavior of large and complex user software, especially with some ready-to-use packages involved. In any case, such information is a subject for further detailed study and discussion with the users.

Conclusions and Future Work

Many of the jobs with TIMEOUT and CANCELLED states appeared to be inefficient by the application nature and implementation peculiarities, that was proved by our users – jobs owners. So this illustrates the necessity for system support to work on improving the efficiency of supercomputer applications together with their owners.

It is revealed, that the ratio of COMPLETED state jobs which are usually considered as successful runs, is lower than can be expected. Some jobs just cannot finish with such state because of their specifics, for example, the jobs that compute until they meet timeout and are marked as TIMEOUT by state. It means that there is at least a good way to provide a SLURM option for the users that can mark the job as considered to be run until the timeout. This would allow subtracting these jobs from suspicious jobs category. Then, it would be reasonable to adjust currently used job statistics, splitting the TIMEOUT and CANCELLED categories in two groups each.

Many user applications are based on common packages and libraries. It seems a good idea to check the correlation of all revealed cases with the usage of these popular software packages and libraries. This would allow us detecting similar problems that experience different users and workgroups. This work is planned to be done using XALT tool designed for collecting job-level information about used libraries, modules and executables [10].

Another interesting way of future research related to the obtained results is to automate the process of getting user feedback. This can be developed as a special service of currently used Octoshell system [11] which is used for our supercomputer center workflow management. It could provide reports and surveys with statistics of the job states, enriched with more detailed data on

job resource utilization, revealed efficiency-related anomalies [12], used software packages and libraries, etc.

Acknowledgements

The results are obtained with the financial support of the Russian Foundation for Basic Research (grants No. 17-07-00664 and 17-07-00719). The research is carried out using the equipment of the shared research facilities of HPC computing resources at Lomonosov Moscow State University.

References

1. TOP500 List – June 2018 – TOP500 Supercomputer Sites. `https://www.top500.org/list/2018/06/`

2. Voevodin, V., Voevodin, V.: Efficiency of Exascale Supercomputer Centers and Supercomputing Education. In: High Performance Computer Applications: Proceedings of the 6th International Supercomputing Conference in Mexico (ISUM 2015). pp. 14–23. Springer, Cham (2016), DOI: 10.1007/978-3-319-32243-8_2

3. Nikitenko, D., Stefanov, K., Zhumatiy, S., Teplov, A., Shvets, P., Voevodin, Vad.: System monitoring-based holistic resource utilization analysis for every user of a large HPC center. Algorithms and Architectures for Parallel Processing, LNCS. 10049. 305–318. Springer (2016). DOI: 10.1007/978-3-319-49956-7_24

4. Stefanov, K., Voevodin, V., Zhumatiy, S., Voevodin, V.: Dynamically Reconfigurable Distributed Modular Monitoring System for Supercomputers (DiMMon). Procedia Computer Science 66, 625–634 (2015), DOI: 10.1016/j.procs.2015.11.071

5. Nikitenko, D., Antonov, A., Shvets, P., Sobolev, S., Stefanov, K., Voevodin, V., Voevodin, V., Zhumatiy, S.: JobDigest Detailed System Monitoring-Based Supercomputer Application Behavior Analysis. In: Supercomputing. Third Russian Supercomputing Days, RuSCDays 2017, Moscow, Russia, September 25–26, 2017, Revised Selected Papers. pp. 516–529. Springer, Cham (2017), DOI: 10.1007/978-3-319-71255-0_42

6. Jones, M.D., White, J.P., Innus, M., DeLeon, R.L., Simakov, N., Palmer, J.T., Gallo, S.M., Furlani, T.R., Showerman, M., Brunner, R., Kot, A., Bauer, G., Bode, B., Enos, J., Kramer, W.: Workload Analysis of Blue Waters (2017), `http://arxiv.org/abs/1703.00924`

7. You, H., Zhang, H.: Comprehensive workload analysis and modeling of a petascale supercomputer. In: Workshop on Job Scheduling Strategies for Parallel Processing. pp. 253–271. Springer (2012), DOI: 10.1007/978-3-642-35867-8_14

8. Furlani, T.R., Schneider, B.L., Jones, M.D., et al.: Using XDMoD to facilitate XSEDE operations, planning and analysis In:Proceedings of the Conference on Extreme Science

and Engineering Discovery Environment: Gateway to Discovery, ACM, p. 46, (2013), DOI: 10.1145/2484762.2484763

9. Palmer, J.T., Gallo, S.M., Furlani, T.R., et al.: Open XDMoD: A tool for the comprehensive management of high-performance computing resources. In: Computing in Science & Engineering, vol. 17, no. 4, pp. 52–62. IEEE (2015), DOI: 10.1109/MCSE.2015.68

10. Agrawal, K., Fahey, M.R., McLay, R., Doug, J.: User environment tracking and problem detection with XALT. In: Proceedings of the First International Workshop on HPC User Support Tools, pp. 32–40. IEEE press (2014), DOI: 10.1109/HUST.2014.6

11. Nikitenko, D., Voevodin, Vl., Zhumatiy, S.: Resolving frontier problems of mastering large-scale supercomputer complexes. In: ACM International Conference on Computing Frontiers (CF'16), pp. 349–352. ACM New York (2016), DOI: 10.1145/2903150.2903481

12. Voevodin, Vl., Voevodin, Vad., Shaikhislamov, D., Nikitenko, D.: Data mining method for anomaly detection in the supercomputer task flow. In: Numerical Computations: Theory and Algorithms, The 2nd International Conference and Summer School, Pizzo calabro, Italy, June 20–24, 2016. AIP Conference Proceedings, vol. 1776, pp. 090015-1–090015-4 (2016), DOI: 10.1063/1.4965379

DOI: 10.14529/jsfi180202

Scalability Evaluation of Cimmino Algorithm for Solving Linear Inequality Systems on Multiprocessors with Distributed Memory*

Irina M. Sokolinskaya[1], *Leonid B. Sokolinsky*[1]

The paper is devoted to a scalability study of Cimmino algorithm for linear inequality systems. This algorithm belongs to the class of iterative projection algorithms. For the analytical analysis of the scalability, the BSF (Bulk Synchronous Farm) parallel computation model is used. An implementation of the Cimmino algorithm in the form of operations on lists using higher-order functions *Map* and *Reduce* is presented. An analytical estimation of the scalability boundary of the algorithm for cluster computing systems is derived. An information about the implementation of Cimmino algorithm on lists in C++ language using the BSF program skeleton and MPI parallel programming library is given. The results of large-scale computational experiments performed on a cluster computing system are demonstrated. A conclusion about the adequacy of the analytical estimations by comparing them with the results of computational experiments is made.

Keywords: system of linear inequalities, iterative algorithm, projection algorithm, Cimmino algorithm, parallel computation model, bulk synchronous farm, scalability estimation, speedup, parallel efficiency, cluster computing systems.

Introduction

The problem of solving systems of linear inequalities arise in numerous fields. As examples, we can mention linear programming [1, 2], image reconstruction from projections [3], image processing in magnetic resonance imaging [4], intensity-modulated radiation therapy (IMRT) [5]. At the present time, a lot of methods for solving systems of linear inequalities are known, among which we can mark out a class of self-correcting iteration methods that allow efficient parallelization. In this field, pioneer works are papers [6, 7], in which the Agmon–Motzkin–Schoenberg relaxation method for solving systems of linear inequalities was proposed. The relaxation method belongs to the class of projection methods, which use the operation of orthogonal projection onto a hyperplane in Euclidean space. One of the first iterative algorithms of projection type was the Cimmino algorithm [8], intended for solving systems of linear equations and inequalities. Cimmino algorithm had a great influence on the development of computational mathematics [9]. A considerable number of papers have been devoted to the generalizations and extensions of the Cimmino algorithm (for example, see [3, 10–13]).

In many cases, systems of linear inequalities arising in the solution of practical problems can involve up to tens of millions of inequalities and up to hundreds of millions of variables [2]. In this case, the issue of developing scalable parallel algorithms for solving large-scale systems of linear inequalities on multiprocessor systems with distributed memory becomes very urgent. When one creates parallel algorithms for large multiprocessor systems, it is important at an early stage of the algorithm design (before coding) to obtain an analytical estimation of its scalability. For this purpose, one can use various models of parallel computation [14]. Nowadays, a large number of different parallel computation models are known. The most famous models among them are PRAM [15], BSP [16] and LogP [17]. Each of these models generated a large

*The article is recommended for publication by the Program Committee of the International Scientific Conference "Russian Supercomputing Days 2018".

[1]South Ural State University, Chelyabinsk, Russian Federation

family of parallel computation models, which extend and generalize the parent model (see, e.g., [18–20]). The problem of developing new parallel computation models is still important today. The reason is that it is impossible to create a parallel computation model, which is good in all respects. To create a good parallel computation model, the designer must restrict the set of target multiprocessor architectures and class of algorithms. In paper [21], the parallel computation model BSF (Bulk Synchronous Farm) intended for cluster computing systems and iterative algorithms was proposed. The BSF model makes it possible to predict the scalability boundary of an iterative algorithm with great accuracy before coding. An example of using the BSF model is given in [22].

The purpose of this article is to investigate the scalability of the Cimmino algorithm for solving large-scale systems of linear inequalities on multiprocessor systems with distributed memory by using the BSF parallel computation model. The rest of the article is organized as follows. Section 1 gives a formal description of the Cimmino algorithm. In Section 2, the representation of the Cimmino algorithm in the form of operations on lists using higher-order functions *Map* and *Reduce* defined in the Bird–Meertens formalism is constructed. Section 3 is dedicated to an analytical investigation of the scalability of the Cimmino algorithm on lists using the BSF model cost metrics; the equations for estimating the speedup and parallel efficiency are given; the boundary of the algorithm scalability depending on the problem size is calculated. In Section 4, a description of the implementation of the Cimmino algorithm on lists in C++ language using the BSF algorithmic skeleton and the MPI parallel programming library is presented; a comparison of the results obtained analytically and experimentally is given. In conclusion, the obtained results are summarized and directions for further research are outlined.

1. Cimmino Algorithm for Inequalities

Let us consider the system of linear inequalities

$$l_i(x) = \langle a_i, x\rangle - b_i \leqslant 0 \qquad (i = 1, \ldots, m), \tag{1}$$

where $\langle a_i, x\rangle$ is the Euclidean inner product of a_i and x in $\mathbb{R}^n$, $b_i \in \mathbb{R}$. To avoid triviality, we assume $m \geqslant 2$. We also assume that the system (1) is consistent. It is necessary to find a solution of the system of linear inequalities (1). To solve this problem, it is convenient to use a geometric language. Thus, we look upon $x = (x_1, \ldots, x_n)$ as a point in n-dimensional Euclidean space $\mathbb{R}^n$, and each inequality $l_i(x) \leqslant 0$ as a half-space P_i. Therefore, the set of solutions of system (1) is the convex polytope $M = \bigcap_{i=1}^{m} P_i$. Each equation $l_i(x) = 0$ defines a hyperplane H_i:

$$H_i = \{x \in \mathbb{R}^n | \, \langle a_i, x\rangle = b_i\}\,. \tag{2}$$

Let the orthogonal projection of $x \in \mathbb{R}^n$ onto the hyperplane $H_i \subset \mathbb{R}^n$ be denoted by $\pi_{H_i}(x)$. The orthogonal projection $\pi_{H_i}(x)$ can be calculated by the following equation:

$$\pi_{H_i}(x) = x + \frac{b_i - \langle a_i, x\rangle}{\|a_i\|^2} a_i, \tag{3}$$

where $\|\cdot\|$ is the Euclidean norm. Let us define the orthogonal reflection of x with respect to hyperplane H_i as follows:

$$\rho_{H_i}(x) = \pi_{H_i}(x) - x = \frac{b_i - \langle a_i, x\rangle}{\|a_i\|^2} a_i. \tag{4}$$

The *Cimmino algorithm for equally weighted inequalities* consists of the following steps:

Step 1: $k := 0$; $x_0 := \mathbf{0}$.

Step 2: $x_{k+1} := x_k + \frac{\lambda}{m} \sum_{i=1}^{m} \rho_{H_i}(x_k)$.

Step 3: If $\|x_{k+1} - x_k\|^2 < \varepsilon$ then go to Step 5.

Step 4: $k := k + 1$; go to Step 2.

Step 5: Stop.

Cimmino's method starts with an arbitrary point x_0 in $\mathbb{R}^n$ as an initial approximation, and then calculates at each step the centroid of a system of masses placed at the reflections of the previous iterate with respect to the hyperplanes $H_1, \ldots, H_m$ defined by the system of inequalities. This centroid is taken as the new iterate:

$$x_{k+1} = x_k + \frac{\lambda}{m} \sum_{i=1}^{m} \rho_{H_i}(x_k). \tag{5}$$

In equation (5), λ is a *relaxation parameter*. It is known [10] that for $0 < \lambda < 2$ the iteration process (5) converges to a point belonging to the polytope M.

2. Cimmino Algorithm in the Form of Operations on Lists

In order to obtain analytical estimations of an algorithm using the cost metrics of the BSF model, it must be represented in the form of operations on lists using higher-order functions *Map* and *Reduce* defined in the Bird–Meertens formalism [23]. The higher-order function Map applies the given function $F : \mathbb{A} \to \mathbb{B}$ to each element of the given list $[a_1, \ldots, a_m]$ and returns a list of results in the same order:

$$Map(F, [a_1, \ldots, a_m]) = [F(a_1), \ldots, F(a_m)]. \tag{6}$$

The higher-order function *Reduce* reduces the given list $[b_1, \ldots, b_m]$ to a single value by iteratively applying the given binary associative operation $\oplus : \mathbb{B} \times \mathbb{B} \to \mathbb{B}$ to each pair of elements:

$$Reduce(\oplus, [b_1, \ldots, b_m]) = b_1 \oplus \ldots \oplus b_m. \tag{7}$$

In the context of the Cimmino algorithm, we define the list L_{map} as follows:

$$L_{map} = [i_1, \ldots, i_m], \tag{8}$$

where $i_k \in \{1, \ldots, m\}$ and $i_k \neq i_l$ for $k \neq l$ $(k, l = 1, \ldots, m)$. In other words, L_{map} – is the list of numbers of inequalities (1) ordered in an arbitrary way. For an arbitrary point $x \in \mathbb{R}^n$, let us define the function $F_x : \{1, \ldots, m\} \to \mathbb{R}^n$ as follows:

$$F_x(i) = \rho_{H_i}(x) \tag{9}$$

for all $i \in \{1, \ldots, m\}$. In other words, the function $F_x(i)$ calculates the orthogonal reflection of x with respect to the hyperplane H_i. For an arbitrary point $x \in \mathbb{R}^n$, let us define the list $L^{(x)}_{reduce} \subset \mathbb{R}^n$ as follows:

$$L^{(x)}_{reduce} = [F_x(i_1), \ldots, F_x(i_m)]. \tag{10}$$

The list $L^{(x)}_{reduce}$ holds orthogonal reflections of the point x with respect to the hyperplanes $H_1, \ldots, H_m$ in the order determined by the list L_{map}. Thus, the list $L^{(x)}_{reduce}$ is obtained from the list L_{map} by applying to it the higher-order function Map using as a parameter the function F_x:

$$L^{(x)}_{reduce} = Map(F_x, L_{map}). \tag{11}$$

Let us define the binary associative operation $\oplus : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$ as follows:

$$x \oplus y = x + y \tag{12}$$

for all $x, y \in \mathbb{R}^n$. In this case, the $\oplus$ operator performs the conventional composition of vectors. Then the sum of orthogonal reflections of the point x can be obtained by applying to the list $L^{(x)}_{reduce}$ the higher-order function *Reduce* using as a parameter the vector composition operation $\oplus$:

$$\sum_{i=1}^{m} \rho_{H_i}(x) = Reduce(\oplus, L^{(x)}_{reduce}). \tag{13}$$

Now we can write the *Cimmino algorithm in the form of operations on lists*:

Step 1: $k := 0$; $x_0 := \mathbf{0}$; $L_{map} := [1, \ldots, m]$.

Step 2: $L^{(x_k)}_{reduce} := Map(F_{x_k}, L_{map})$.

Step 3: $s := Reduce(\oplus, L^{(x_k)}_{reduce})$.

Step 4: $x_{k+1} := x_k + \frac{\lambda}{m} s$.

Step 5: If $\|x_{k+1} - x_k\|^2 < \varepsilon$ then go to Step 7.

Step 6: $k := k + 1$; go to Step 2.

Step 7: Stop.

The BSF model assumes that the algorithm is executed by a computing system consisting of one master-node and K worker-nodes ($K > 0$). Step 1 of the algorithm is performed by both the master and the workers during the initialization of the iterative process. Step 2 (*Map*) is performed only on the worker-nodes. Step 3 (*Reduce*) is performed on the worker-nodes and partially on the master-node. Steps 4–6 are performed only on the master-node. The BSF model assumes that all arithmetic operations (addition and multiplication) as well as comparison operations on floating-point numbers take the same time τ_{op}.

3. Analytical Evaluation of Scalability

Let us introduce the following notation for the scalability evaluation of the Cimmino algorithm:

c_s	:	the quantity of float numbers transferred from the master to one worker;
c_{map}	:	the quantity of arithmetic operations performed in the *Map* step (Step 2 of the algorithm);
c_a	:	the quantity of arithmetic operations required to calculate the sum of two vectors;

c_r : the quantity of float numbers transferred from one worker to the master;

c_p : the quantity of arithmetic and comparison operations performed by the master in Steps 4 and 5 of the algorithm.

Let us calculate the indicated values. At the beginning of each iteration, the master sends to all the workers the current approximation x_k, which is a vector of dimension n. Hence:

$$c_s = n. \tag{14}$$

Let us calculate the number of arithmetic operations performed in the *Map* step. For each element of the list L_{map}, one vector is calculated by equation (4). Note that the values of $\|a_i\|^2$ $(i = 1, \ldots, m)$ do not depend on x_k, and therefore can be calculated in advance at the initialization stage. Taking this into account, the quantity of operations for calculating one orthogonal reflection of the point x_k is $3n + 1$. Multiplying this value by the number of inequalities, we obtain

$$c_{map} = m(3n + 1). \tag{15}$$

During the execution of *Reduce* step, the list L_{reduce} consisting of m vectors is divided into equal parts, each of them assigned to a single worker. Everywhere below we assume that $K \leqslant m$. For simplicity we assume that m is a multiple of number of workers K. The composition of vectors of dimension n requires n arithmetic operations. Hence:

$$c_a = n. \tag{16}$$

After execution of *Reduce* step, each worker sends the resulting vector to the master. Thus:

$$c_r = n. \tag{17}$$

The execution of Step 4 requires $2n$ operations (we assume the constant value of λ/m to be computed in advance). The execution of Step 5 requires $3n - 1$ arithmetic operations and one comparison operation. It follows the equation:

$$c_p = 5n. \tag{18}$$

Let us designate the time spent by the worker to perform one arithmetic operation as τ_{op}, and designate the time spent for transferring a single float number across the network excluding latency as τ_{tr}. In that way, we get the following values for the cost parameters of the BSF model [21] in the case of the Cimmino algorithm:

$$t_s = n\tau_{tr}; \tag{19}$$

$$t_{map} = m(3n + 1)\tau_{op}; \tag{20}$$

$$t_a = n\tau_{op}; \tag{21}$$

$$t_r = n\tau_{tr}; \tag{22}$$

$$t_p = 5n\tau_{op}. \tag{23}$$

Equation (19), obtained on the basis of (14), gives an estimation of the time t_s spent by the master to transfer a message to one worker excluding latency. Equation (20) is obtained using the equation (15). According to the BSF model cost metric, t_{map} denotes the total time spent by a single worker to process the entire *Map* list. Equation (21) obtained using equation (16) calculates the time t_p spent by a processor node on adding two vectors of dimension n. Equation (22), obtained on the basis of (17), gives an estimation of the time t_r spent by the master to transfer a message to one worker excluding latency. Equation (23) obtained using equation (18) calculates the time t_p spent by the master on the following actions: calculating the next approximation and checking of the stopping criterion. In accordance with this metric, the time for solving the problem by a system consisting of one master and one worker ($K = 1$) can be estimated as follows:

$$\begin{aligned} T_1 &= 2L + t_s + t_r + t_p + t_{Map} + lt_a \\ &= 2(L + \tau_{tr} n) + \tau_{op} \left(5n + m(3n+1) + (m-1)n\right). \end{aligned} \tag{24}$$

The time of solving the problem by a system composed of one master and K workers can be estimated by the following equation:

$$\begin{aligned} T_K &= K\left(2L + t_s + t_r + t_a\right) + \frac{t_{Map} + lt_a}{K} - t_a + t_p \\ &= 2K(L + \tau_{tr} n + \tau_{op} n) + \tau_{op} \left(\frac{m(3n+1) + (m-1)n}{K} + 4n\right). \end{aligned} \tag{25}$$

For $m \to \infty$, the equations (24) and (25) asymptotically tend to the following estimations:

$$T_1 = 2(L + \tau_{tr} n) + \tau_{op} \left(5n + m(3n+1) + mn\right); \tag{26}$$

$$T_K = 2K(L + \tau_{tr} n + \tau_{op} n) + \tau_{op} \left(\frac{m(3n+1) + mn}{K} + 4n\right). \tag{27}$$

On the basis of equations (26) and (27) we can write the equation for speedup a in the form of a function of K:

$$a(K) = \frac{T_1}{T_K} = \frac{2(L + \tau_{tr} n) + \tau_{op} \left(5n + m(3n+1) + mn\right)}{2K(L + \tau_{tr} n + \tau_{op} n) + \tau_{op} \left(\frac{m(3n+1)+mn}{K} + 4n\right)}. \tag{28}$$

To determine the scalability boundary of the Cimmino algorithm in accordance with the procedure described in [21], let us deduce the derivative $a'(K)$ and solve the equation

$$a'(K) = 0. \tag{29}$$

Using simple algebraic transformations, from equation (28), we can deduce the following equation for the derivative of speedup:

$$\begin{aligned} a'(K) = &\left(2(L + \tau_{tr} n) + \tau_{op} \left(5n + m(3n+1) + mn\right)\right) \cdot \\ &\cdot \frac{\frac{m(3n+1)+mn}{K^2} \tau_{op} - 2(L + n\tau_{tr}) - \tau_{op} n}{\left(2K(L + \tau_{tr} n + \tau_{op} n) + \tau_{op} \left(\frac{m(3n+1)+mn}{K} + 4n\right)\right)^2}. \end{aligned} \tag{30}$$

Let us solve the equation

$$\begin{gathered}(2(L+\tau_{tr}n)+\tau_{op}(5n+m(3n+1)+mn))\cdot \\ \cdot\frac{\frac{m(3n+1)+mn}{K^2}\tau_{op}-2(L+n\tau_{tr})-\tau_{op}n}{\left(2K(L+\tau_{tr}n+\tau_{op}n)+\tau_{op}\left(\frac{m(3n+1)+mn}{K}+4n\right)\right)^2}=0.\end{gathered} \tag{31}$$

Dividing both sides of equation (31) by the positive quantity

$$2(L+\tau_{tr}n)+\tau_{op}(5n+m(3n+1)+mn)$$

and multiplying by the positive quantity

$$\left(2K(L+\tau_{tr}n+\tau_{op}n)+\tau_{op}\left(\frac{m(3n+1)+mn}{K}+4n\right)\right)^2$$

we obtain the equation

$$\frac{m(3n+1)+mn}{K^2}\tau_{op}-2(L+n\tau_{tr})-\tau_{op}n=0,$$

which implies

$$K=\sqrt{\frac{(m(3n+1)+mn)\,\tau_{op}}{2(L+n\tau_{tr})+n\tau_{op}}}.$$

Thus, equation (31) has the only root

$$K_0=\sqrt{(m(3n+1)+mn)\,\tau_{op}/(2(L+n\tau_{tr})+n\tau_{op})}$$

on the interval $[1,+\infty)$. It is easy to see that the derivative $a'(K)$ calculated by the equation (30) takes only positive values in the interval $[1,K_0)$ and only negative values in the interval $(K_0,+\infty)$. Therefore, the point K_0 is the maximum of the function $a(K)$ on the interval $[1,+\infty)$. It follows that the maximum of speedup is obtained at the point K_0. Thus, in accordance with the BSF model, the boundary K_{max} of the scalability of the Cimmino algorithm is determined by the following equation:

$$K_{max}=\sqrt{\frac{(m(3n+1)+mn)\,\tau_{op}}{2(L+n\tau_{tr})+n\tau_{op}}}. \tag{32}$$

Let us simplify equation (32). For $n,m\to\infty$, we have

$$(m(3n+1)+mn)\,\tau_{op}\approx O(mn) \tag{33}$$

and

$$2(L+n\tau_{tr})+n\tau_{op}\approx O(n). \tag{34}$$

Substituting the right-hand sides of equations (33) and (34) into (32), we obtain

$$K_{max}=\sqrt{\frac{O(mn)}{O(n)}},$$

which is equivalent to

$$K_{max}=\sqrt{O(m)}. \tag{35}$$

In that way, the boundary of the scalability of the Cimmino algorithm on lists increases in proportion to the square root of the number m of inequalities. In conclusion of this section, let us write the equation for estimating the parallel efficiency e as a function of K. Considering equation (28), we have

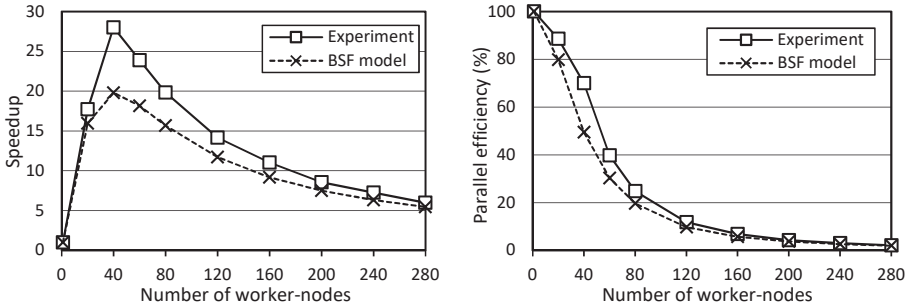
$$e(K) = \frac{a(K)}{K} = \frac{2(L + \tau_{tr} n) + \tau_{op} (5n + m(3n + 1) + mn)}{2K^2(L + \tau_{tr} n + \tau_{op} n) + \tau_{op} (m(3n + 1) + mn + 4nK)}. \quad (36)$$

4. Numerical Experiments

In order to verify the analytical results, we implemented the Cimmino algorithm in C++ language using the BSF algorithmic skeleton and the MPI parallel programming library. The source code of this program is freely available on Github, at `https://github.com/leonid-sokolinsky/BSF-Cimmino`. The system of inequalities was taken from the model scalable linear-programming problem *Model-n* given in [24]. In this system, the number of inequalities is $m = 2n + 2$, where n is the dimension of the space. We investigated the speedup and parallel efficiency of the Cimmino algorithm on the supercomputer "Tornado SUSU" [25]. The calculations were performed for the dimensions 1 500, 5 000, 10 000 and 16 000. At the same time, we plotted the curves of speedup and parallel efficiency for these dimensions using equations (28) and (36). For this, the following values in seconds were determined experimentally: $L = 1.5 \cdot 10^{-5}$, $\tau_{op} = 2.9 \cdot 10^{-8}$ and $\tau_{tr} = 1.9 \cdot 10^{-7}$. The results are presented in Fig. 1–4. In all cases, the analytical estimations were very close to experimental ones. Moreover, the performed experiments show that the boundary of the BSF-program scalability increases in proportion to the square root of the number m of inequalities. It was analytically predicted by the equation (35).

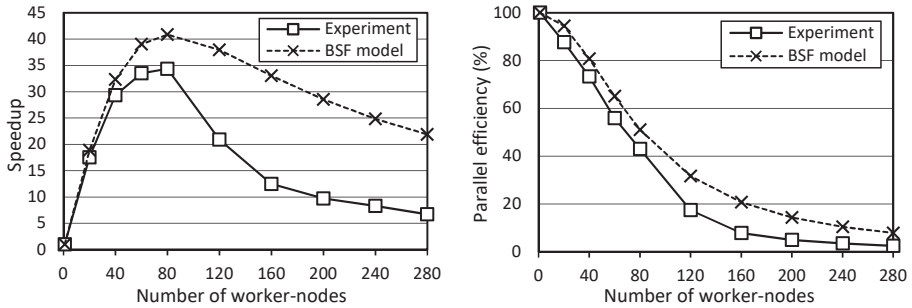
Conclusion

In this paper, the scalability and parallel efficiency of the iterative Cimmino algorithm used to solve large-scale linear inequality systems on multiprocessor systems with distributed memory were investigated. To do this, we used the BSF (Bulk Synchronous Farm) parallel computation model based on the "master-slave" paradigm. The BSF-implementation of the Cimmino algorithm in the form of operations on lists using higher-order functions *Map* and *Reduce* is described. A scalability boundary of the BSF-implementation of the Cimmino algorithm is obtained. This estimation tells us the following. If space dimension n is greater than or equal to the number m of inequalities, then the boundary of the scalability of the Cimmino algorithm on lists increases in proportion to the square root of the number m of inequalities. So, we may conclude that the Cimmino algorithm on lists is scalable well. Also, the equations for estimating the speedup and parallel efficiency of the Cimmino algorithm on lists are obtained. The implementation of the Cimmino algorithm in C++ language using the BSF algorithmic skeleton and the MPI parallel programming library was performed. This implementation is freely available on Github, at `https://github.com/leonid-sokolinsky/BSF-Cimmino`. On a cluster computing system, the large-scale experiments were conducted to obtain the actual speedup and parallel efficiency curves for systems having number of variables 1 500, 5 000, 10 000, 16 000 and the number of inequalities 3 002, 10 002, 20 002, 32 002, respectively. The results of the experiments showed that the BSF model predicts the boundary of the scalability of the Cimmino algorithm on lists with high accuracy. As future research directions, we intend to do the following:



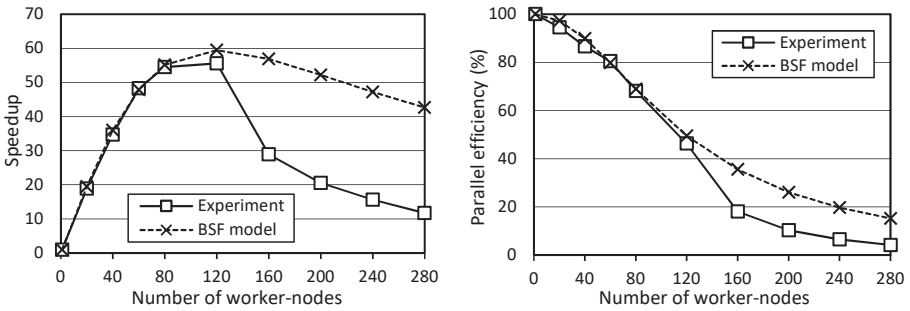
(a) Speedup (b) Parallel efficiency

Figure 1. Experiments for $n = 1\,500$ and $m = 3\,002$



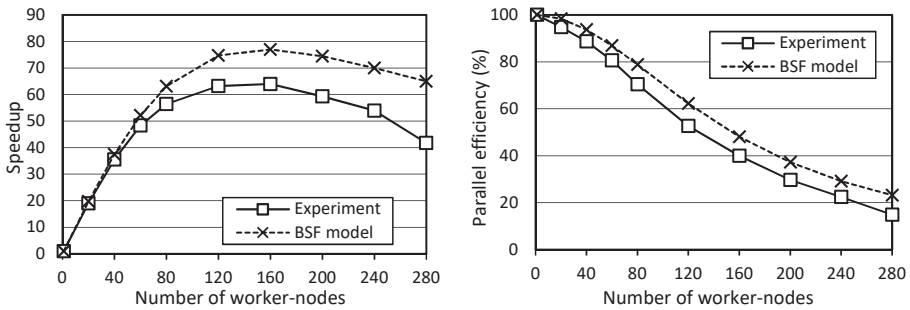
(a) Speedup (b) Parallel efficiency

Figure 2. Experiments for $n = 5\,000$ and $m = 10\,002$



(a) Speedup (b) Parallel efficiency

Figure 3. Experiments for $n = 10\,000$ and $m = 20\,002$



(a) Speedup (b) Parallel efficiency

Figure 4. Experiments for $n = 16\,000$ and $m = 32\,002$

1) apply the Cimmino algorithm to implement the Qwest phase of the NSLP algorithm [2], designed to solve large-scale non-stationary linear programming problems;

2) carry out computational experiments to solve large-scale linear programming problems on a cluster computer system under the conditions of dynamically changing the input data.

Acknowledgments

The study was partially supported by the RFBR according to research project No. 17-07-00352-a, by the Government of the Russian Federation according to Act 211 (contract No. 02.A03.21.0011) and by the Ministry of Science and Higher Education of the Russian Federation (government order 2.7905.2017/8.9).

References

1. Cottle, R.W., Pang, J.-S., Stone, R.E.: The Linear Complementarity Problem. Society for Industrial and Applied Mathematics (2009)

2. Sokolinskaya, I., Sokolinsky, L.B.: On the Solution of Linear Programming Problems in the Age of Big Data. In: Parallel Computational Technologies, PCT 2017. Communications in Computer and Information Science, vol. 753, pp. 86–100. Springer, Cham (2017), DOI: 10.1007/978-3-319-67035-5_7

3. Censor, Y., Elfving, T., Herman, G.T., Nikazad, T.: On Diagonally Relaxed Orthogonal Projection Methods. SIAM Journal on Scientific Computing 30, 473–504 (2008), DOI: 10.1137/050639399

4. Zhu, J., Li, X.: The Block Diagonally-Relaxed Orthogonal Projection Algorithm for Compressed Sensing Based Tomography. In: 2011 Symposium on Photonics and Optoelectronics, SOPO. pp. 1–4. IEEE (2011), DOI: 10.1109/SOPO.2011.5780660

5. Censor, Y.: Mathematical optimization for the inverse problem of intensity-modulated radiation therapy. In: Palta, J.R., Mackie, T.R. (eds.) Intensity-Modulated Radiation Therapy: The State of the Art. pp. 25–49. Medical Physics Publishing, Madison, WI (2003)

6. Agmon, S.: The relaxation method for linear inequalities. The Canadian Journal of Mathematics 6, 382–392 (1954), DOI: 10.4153/CJM-1954-037-2

7. Motzkin, T.S., Schoenberg, I.J.: The relaxation method for linear inequalities. The Canadian Journal of Mathematics 6, 393–404 (1954), DOI: 10.4153/CJM-1954-038-x

8. Cimmino, G.: Calcolo approssimato per le soluzioni dei sistemi di equazioni lineari. La Ric. Sci. XVI, Ser. II, Anno IX, 1. 326–333 (1938)

9. Benzi, M.: Gianfranco Cimmino's Contributions to Numerical Mathematics. In: Atti del SemiSeminario di Analisi Matematica, Dipartimento di Matematica dell'Universitá di

Bologna (Ciclo di Conferenze in Ricordo di Gianfranco Cimmino, 2004). pp. 87–109. Tecnoprint, Bologna, Italy (2005)

10. Censor, Y., Zenios, S.A.: Parallel Optimization: Theory, Algorithms, and Applications. Oxford University Press, New York (1997)

11. Censor, Y., Elfving, T.: New methods for linear inequalities. Linear Algebra Appl. 42, 199–211 (1982), DOI: 10.1016/0024-3795(82)90149-5

12. Censor, Y.: Sequential and parallel projection algorithms for feasibility and optimization. In: Censor, Y., Ding, M. (eds.) Proc. SPIE 4553, 25 Sept. 2001. Visualization and Optimization Techniques, pp. 1–9. International Society for Optics and Photonics (2001), DOI: 10.1117/12.441550

13. Kelley, C.T.: Iterative Methods for Linear and Nonlinear Equations. Society for Industrial and Applied Mathematics, Philadelphia (1995)

14. Bilardi, G., Pietracaprina, A.: Models of Computation, Theoretical. In: Encyclopedia of Parallel Computing. pp. 1150–1158. Springer US, Boston, MA (2011), DOI: 10.1007/978-0-387-09766-4

15. JaJa, J.F.: PRAM (Parallel Random Access Machines). In: Encyclopedia of Parallel Computing. pp. 1608–1615. Springer US, Boston, MA (2011), DOI: 10.1007/978-0-387-09766-4_23

16. Valiant, L.G.: A bridging model for parallel computation. Communications of the ACM 33, 103–111 (1990), DOI: 10.1145/79173.79181

17. Culler, D., Karp, R., Patterson, D., Sahay, A., Schauser, K.E., Santos, E., Subramonian, R., von Eicken, T.: LogP: towards a realistic model of parallel computation. In: Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming, PPOPP'93. pp. 1–12. ACM Press, New York, New York, USA (1993), DOI: 10.1145/155332.155333

18. Forsell, M., Leppanen, V.: An extended PRAM-NUMA model of computation for TCF programming. In: Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2012. pp. 786–793. IEEE Computer Society, Washington, DC, USA (2012), DOI: 10.1109/IPDPSW.2012.97

19. Gerbessiotis, A.V.: Extending the BSP model for multi-core and out-of-core computing: MBSP. Parallel Computing 41, 90–102 (2015), DOI: 10.1016/j.parco.2014.12.002

20. Lu, F., Song, J., Pang, Y.: HLognGP: A parallel computation model for GPU clusters. Concurrency and Computation Practice and Experience 27, 4880–4896 (2015), DOI: 10.1002/cpe.3475

21. Sokolinsky, L.B.: Analytical Estimation of the Scalability of Iterative Numerical Algorithms on Distributed Memory Multiprocessors. Lobachevskii Journal of Mathematics 39, 571–575 (2018), DOI: 10.1134/S1995080218040121

22. Sokolinskaya, I., Sokolinsky, L.B.: Scalability Evaluation of NSLP Algorithm for Solving Non-Stationary Linear Programming Problems on Cluster Computing Systems. In: Supercomputing, RuSCDays 2017. Communications in Computer and Information Science, vol. 793, pp. 40–53. Springer, Cham (2017). DOI: 10.1007/978-3-319-71255-0_4

23. Cole, M.I.: Parallel programming with list homomorphisms. Parallel Processing Letters 05, 191–203 (1995), DOI: 10.1142/S0129626495000175

24. Sokolinskaya, I., Sokolinsky, L.: Revised Pursuit Algorithm for Solving Non-stationary Linear Programming Problems on Modern Computing Clusters with Manycore Accelerators. In: Supercomputing, RuSCDays 2016. Communications in Computer and Information Science, vol. 687, pp. 212–223. Springer, Cham (2016), DOI: 10.1007/978-3-319-55669-7_17

25. Kostenetskiy, P.S., Safonov, A.Y.: SUSU Supercomputer Resources. In: Proceedings of the 10th Annual International Scientific Conference on Parallel Computing Technologies, PCT 2016. CEUR Workshop Proceedings, vol. 1576, pp. 561–573 (2016)

DOI: 10.14529/jsfi180203

On the Inversion of Multiple Matrices on GPU in Batched Mode*

Nikolay M. Evstigneev[1]***, Oleg I. Ryabkov***[1]***, Eugene A. Tsatsorin***[2]

In this research we are considering the benchmarking of batched matrix inversion and solution of linear systems. The problem of multiple matrix inversion with the same fill sparsity is usually considered in problems of fluid mechanics with chemistry. In this case the system is stiff, and an implicit method is required to solve the problem. The core of such method is the multiple matrix inversion. We benchmark different methods based on cuSPARSE and MAGMA libraries and CPU LAPACK version depending on the matrix filling. We also provide our own experimental code that implements GaussJordan elimination on GPU using register shuffle. It is shown that the fastest method is the QR matrix inversion for single precision calculations. We also show that the suggested Gauss–Jordan elimination method looks promising being about 8–10 times faster than cuSPARSE QR method. We also demonstrate the application of batch solvers in the coupled reactive flow problem.

Keywords: QR algorithm, LU Matrix Inversion, Batched Solver, Matrix solver, GPU Batched Solver.

Introduction

In many applications, such as astronomy, chemistry and approximate preconditioning design (e.g. block Jacobi preconditioning in implicit Discontinuous Galerkin methods), one must find solutions of many small linear systems of equations. Let us consider one situation that is very common in CFD where chemical or plasma-chemical reactions are essential and included into multicomponent system of equations, e.g. see [4]. Chemical reactions are governed by systems of ODEs, typically of small or medium size $M \sim \mathcal{O}(10)$, and the problem complexity is multiplied by the discretization of the CFD problem size $N \sim \mathcal{O}(10^6)$. These systems of ODEs are stiff, and implicit methods must be applied to find numerical solutions, e.g. Rosenbrock method [16] is a very popular choice. This leads to the solution of the following linear systems:

$$\mathtt{A}_j \mathbf{x}_j = \mathbf{b}_j, j = 1, ..., N, \tag{1}$$

where $\mathtt{A}_j \in \mathbb{R}^{M \times M}$ are matrices of the numerical method for systems of chemical reactions, $\mathbf{x}_j \in \mathbb{R}^M$ are the vectors of unknowns (concentrations) and $\mathbf{b}_j \in \mathbb{R}^M$ are the right hand sides. Methods for the solution of this problem type are called batch methods. In this paper we refer to the size N as **batch size** or simply **batch** and M as **matrix size**. Matrices $\mathtt{A}_j$ are, in general, nonsymmetric, nonsingular and usually sparse with filling up to 50%.

We aimed at multiple GPU architecture to be used for the solution of (1). There is no communication between systems (1), so we can analyse performance on a single GPU device and assume linear scaling of the problem for multiple GPUs. There are some papers related to the problem. In [1] authors give design and implementation of batched matrix-matrix multiplication on GPUs. It is shown that for relatively small matrices (8×8) one achieves performance of

*The paper is recommended for publication by the Program Committee of the International Scientific Conference "Parallel Computing Technologies (PCT) 2018".

[1]Federal Research Center "Informatics and Control", Institute for System Analysis, Russian Academy of Science, Moscow, Russian Federation

[2]Lomonosov Moscow State University, Moscow, Russian Federation

80 GLFOPS in single precision, while for rather big matrices 32×32 a performance of 260 GFlops is achieved, both on k40 GPUs. Analysis of symmetric matrices of linear systems is performed in [8] during the solution of the problem (1). A comparison of MKL LAPACK and MAGMA library [14]. It is stated, that 80% of the practical *dgemm* peak of the machine is achieved with the self-written code, while MAGMA achieves only 75%, and finally, in terms of energy consumption MAGMA is outperformed by 1.5 times in performance-per-watt for larger matrices. However MAGMA is assumed to be a fairly good alternative, since now MAGMA is extended to cover the batched routines. Batched matrices LU decomposition is discussed in [7]. Batched mode is compared with the streamed one, and it is shown that the premire is superior. A batched LU factorization for GPUs is proposed that uses a multi-level blocked right looking algorithm. It preserves the data layout but minimizes the penalty of partial pivoting. As a result 2.5 speedup is achieved, compared to the alternative CUBLAS solution on a K40c GPU. Batched matrix matrix multiplication for matrices size smaller then 32×32 are provided in [15], where MAGMA library is compared with cuBLAS and MKL. Very good results are reported for MAGMA library with peak performance of 1000 GFlops for Tesla P100 GPUs in double precision. Another new paper is [2] where MAGMA is compared with CUBLAS for the solution of million linear systems. It is shown that MAGMA is an efficient library and is significantly faster than CUBLAS for the considered problems, scoring speedups between 4.3 – 16.8 in single precision and between 3.4 – 14.3 in double precision. Performance is around 650–800 GFlops in single precision for P100 NVIDIA GPU and matrices 16×16.

All these results give great insights into performance, but we found no good comparison of libraries that are designed to be used in batch mode, except from MAGMA library. Besides, there is no comparison in terms of wall time which is what a user is looking for in the first place when trying to speed-up the problem with GPU usage. One can estimate wall time from provided floating point operation per second but it is difficult for complex algorithms, especially those that are using sparse matrix format or relay on non-naive algorithms. One can also use **cuSOLVER** NVIDIA CUDA library [5] to perform batch solution of many small linear systems. The goal of the paper is to perform as many tests as possible, related to the problem (1), and give an insight on using different libraries for future reference and solution strategies using modern compute capability of relitively cheap GPUs.

The paper is laid out as follows. In the first section we provide the benchmark problem and metrics that we collect. We describe libraries that we are using and library routines that we are testing. Here we also describe used hardware. The second section contains brief explanation our code implementation of Gauss–Jordan method that uses register shuffle. The third section contains results that we obtained during benchmarking. This section is divided into three subsections: analysis of libraries, analysis of Gauss–Jordan method and analysis of newer MAGMA library which, we belive, demonstrates abnormal behaviour. The final fourth section we demonstrate the application example of batch linear solver in reactive gas dynamics flow. Then, the conclusion follows.

1. Benchmark Problem and Metrics

In the paper we use four different methods to solve the problem (1). The CPU version is LAPACK routines **SGETRS** for single precision and **DGETRS** for double precision. The OpenMP is used to divide the stream and run independent solvers.

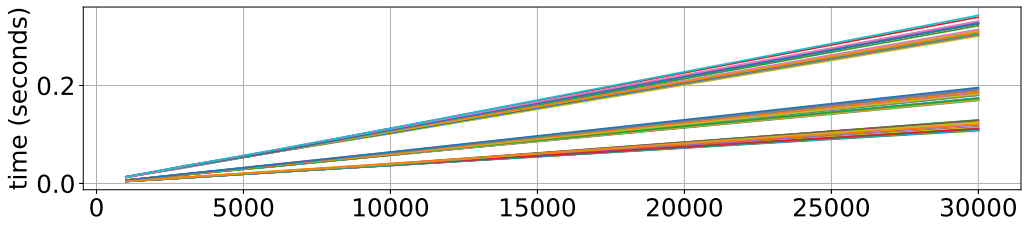


Figure 1. Time for inversion of all matrices on CPU using 1, 2 and 4 threads as function of batch size. Different colors correspond to different matrices

Next we use MAGMA library by calling **magma_(S/D)gesv_batched** routine that executes the linear solution of systems on a GPU in single or double precision. MAGMA uses full matrix storage and we refer to magma library below as *magma_float* and *magma_double*, respectively, for single and double precision call.

We also test two cuSOLVER libraries. The first one is using QR matrix decomposition and is called by **cusolver(S/D)pXcsrqrsvBatched** routine in single or double precision. Some additional preparations must be made that analyse system matrices connectivity graph and transfer reordering permutations to increase the efficiency on the device, for more information we refer a reader to the manual [5]. The second one is called refactored solver that uses LU matrix decomposition and is called by **cusolverRfBatchSolve** routine. Note that some additional preparations must be made on a host part of the program. Besides, the interface of the call is assumed to be only in double precision (with respect to CUDA Toolkit Version 8.0). Both routines use sparse CRS matrix storage format. We refer to these libraries as *QR_float*, *QR_double* and *RF_double*, respectively.

Our test is performed in the following manner. We generate a set of matrices with sizes $M = \{4, 5, 8, 9, 10, 11, 14, 16\}$ with random sparsity patterns having filling 10%, 25% and 50% of all matrix elements, except diagonal elements. We also fill diagonal elements in such a way, that matrices are invertable. It is checked on the stage of matrix generation. We assume that pivoting can be used to these systems to increase the stability of system solution to perturbations. We use the following set of batch sizes: $N = \{1, 3, 6, 10, 30, 60, 100, 300, 600, 1000\} \cdot 10^3$. We also check the performance of two different GPUs with single and double precision calculations, where possible. The *test set* is generated as a tensor product of all possible configurations. For each test in the *test set* we make 10 runs of the code with different generated matrices that share the same sparsity pattern, and execution time is averaged. Performance is measured in FLOPS obtained from **nvprof** utility. All tests are generated, executed and logged by an automated Python script.

We used CPU device INTEL XEON E5-2609 2.4 GHz, 4 cores and two GPU devices. The first one designated **Device 0** is a GTX TITAN X NVIDIA card with 12 GB RAM, 11 TFLOPS peak single precision, 1/32 multiplier for double precision and 5.2 compute capability. The second **Device 1** is a GTX TITAN Black NVIDIA card with 6 GB RAM, 5.1 TFLOPS peak single precision, 1/3 multiplier for double presision and 3.5 compute capability.

2. Gauss–Jordan Elemiation Using Register Shuffle

The code is based on simple Gauss–Jordan elemiation algorithm. A distinctive feature of the implementations is the application of register shuffle that is supported from compute capability

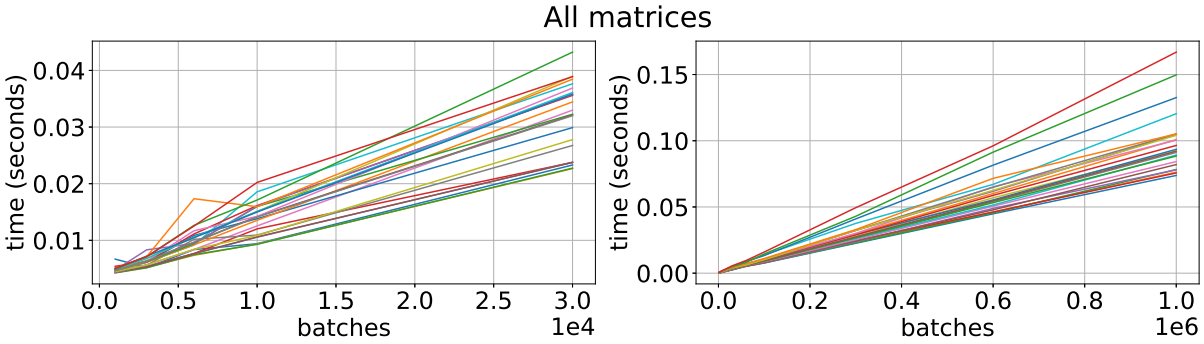


Figure 2. Time for inversion of all matrices on GPU using magma_float (left) and QR_float as function of batch size on Device 0. Different colors correspond to different matrices

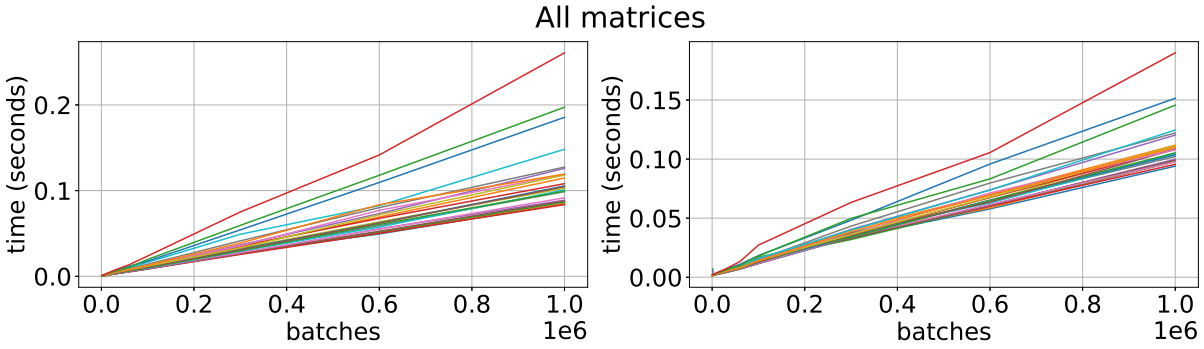


Figure 3. Time for inversion of all matrices on GPU using QR_double (left) and RF as function of batch size on Device 0. Different colors correspond to different matrices

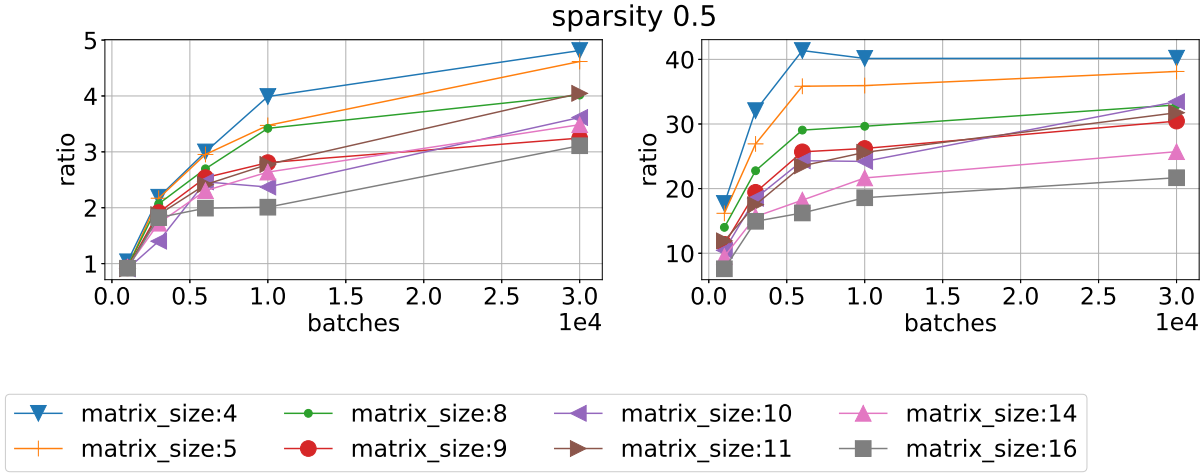


Figure 4. Ratio of mean time execution vs batch size using sparsity 0.5 for different matrix sizes. CPU/magma_float left and CPU/QR_float right on Device 0

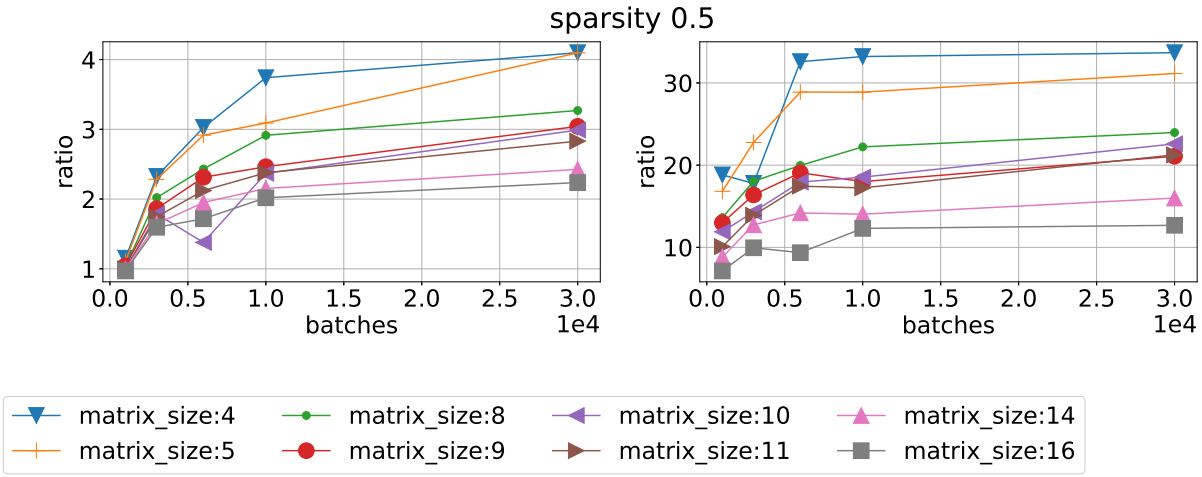


Figure 5. Ratio of mean time execution vs batch size using sparsity 0.5 for different matrix sizes. CPU/magma_float left and CPU/QR_float right on Device 1

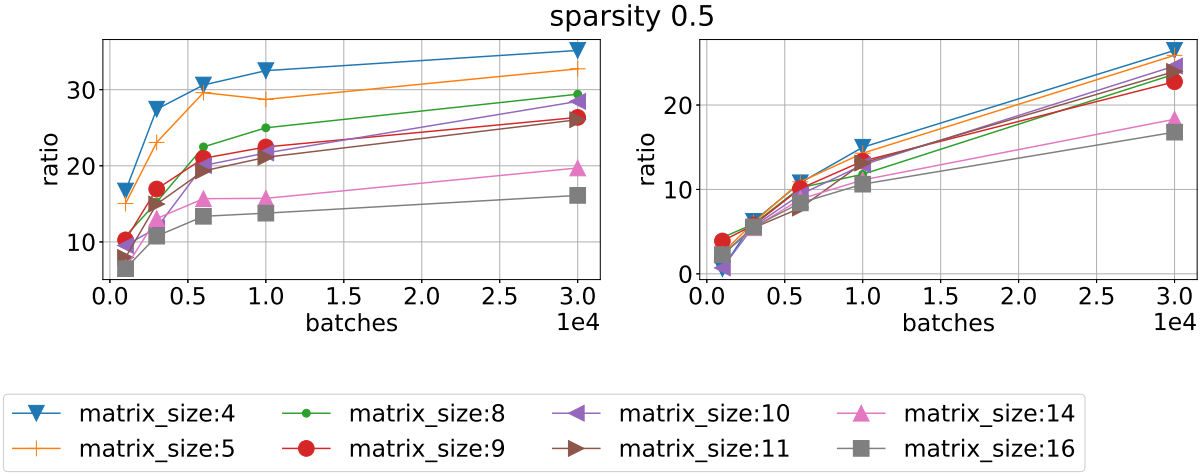


Figure 6. Ratio of mean time execution vs batch size using sparsity 0.5 for different matrix sizes. CPU/QR_double left and CPU/RF right on Device 0

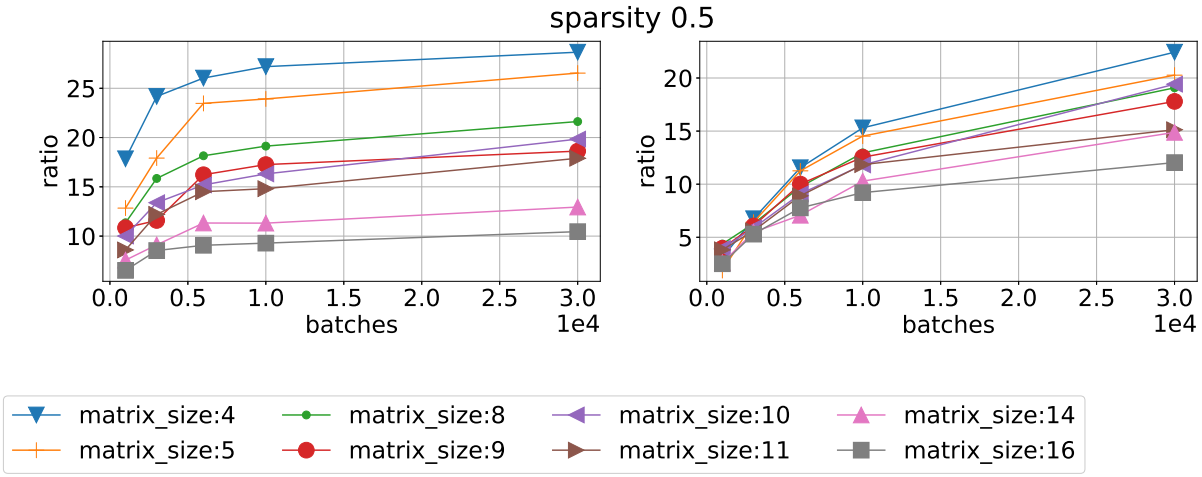


Figure 7. Ratio of mean time execution vs batch size using sparsity 0.5 for different matrix sizes. CPU/QR_double left and CPU/RF right on Device 1

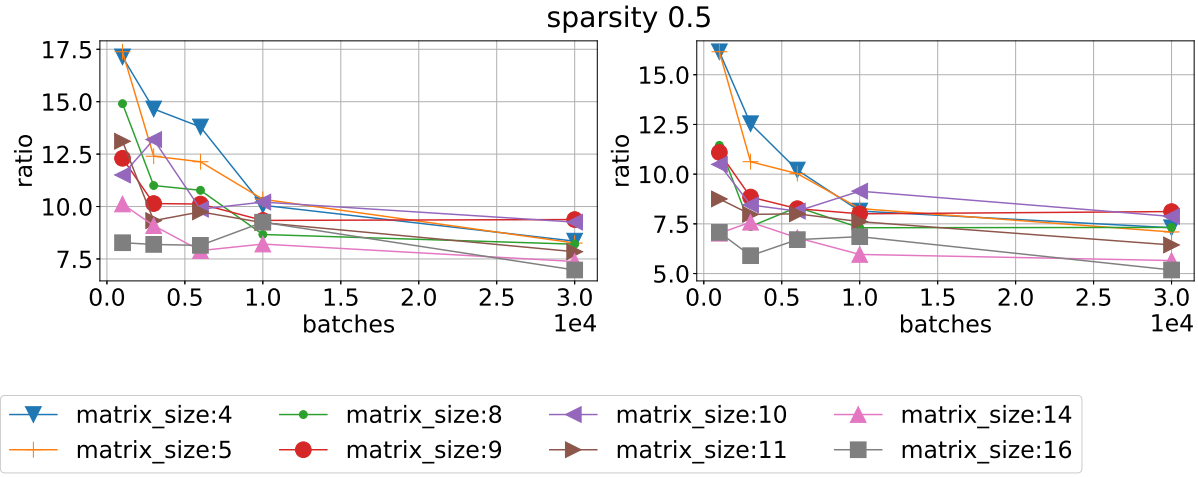


Figure 8. Ratio of mean time execution vs batch size using sparsity 0.5 for different matrix sizes. magma_float/QR_float left and magma_double/QR_double right on Device 0

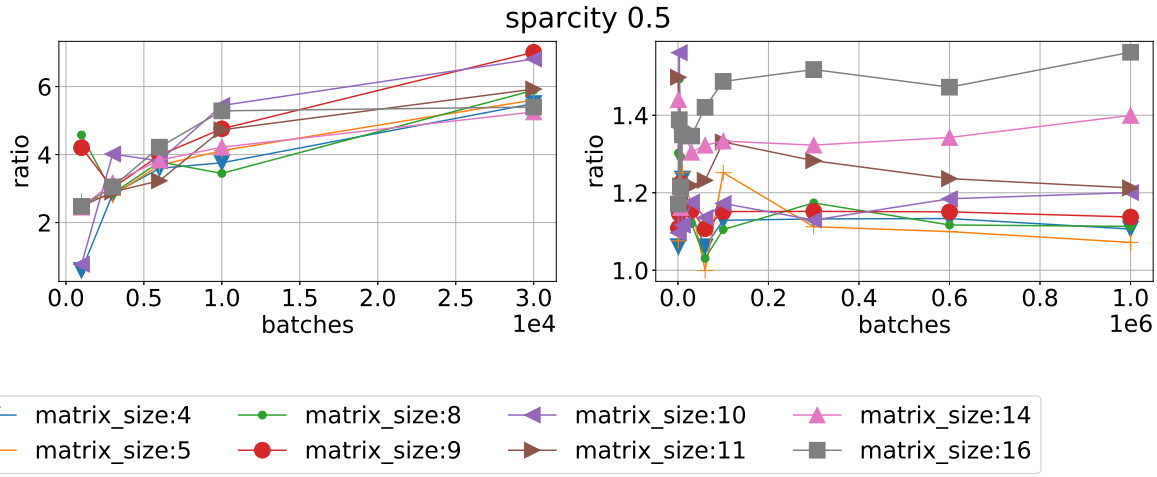


Figure 9. Ratio of mean time execution vs batch size using sparsity 0.5 for different matrix sizes. magma_float/RF left and QR_double/QR_float right on Device 0

3.0 and above. It allows us to share data between threads that are part of the same warp. It insures a speedup of about 3 times to the shared memory access speed but limits current implementation in using only single precision arithmetics (double precision requires spliting into two 32b registers) and matrix size $M \leq W$, where W is a warp size. For more information see [3, 6]. This code is now being tested and is in alpha version, we designate this code as **shuffleGJ**. It is also benchmarked against best results of selected libraries in the end, and performance Gflops are provided.

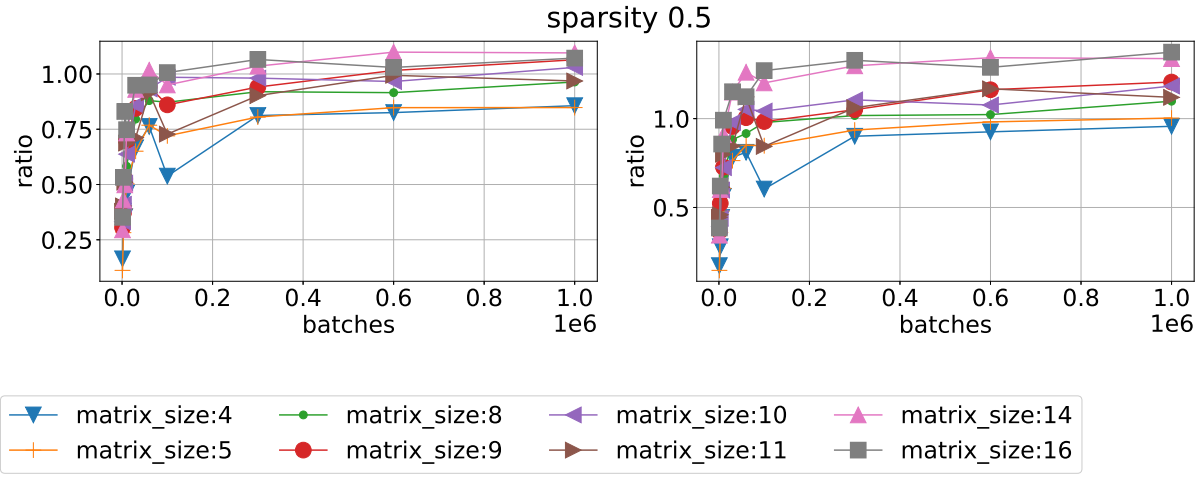


Figure 10. Ratio of mean time execution vs batch size using sparsity 0.5 for different matrix sizes. QR_float/RF left and QR_double/RF right on Device 1

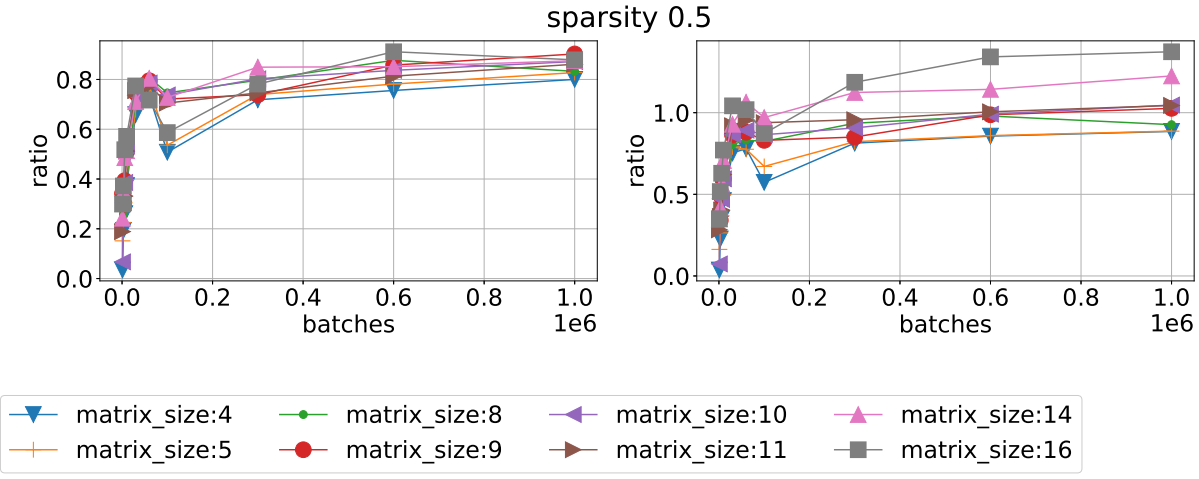


Figure 11. Ratio of mean time execution vs batch size using sparsity 0.5 for different matrix sizes. QR_float/RF left and QR_double/RF right on Device 0

3. Results

3.1. Libraries Analysis

All results are brought into figures that represent sections and projections of multidimensional data from the *test set*. All figures are self–explanationary, but we comment on some of most essential results. In Fig. 1 we can see the time needed to solve systems of equations for all matrices as function of batch size (up to 30,000 matrices) using 1,2 and 4 threads with single precision. We can see linear scaling with batch size since all matrices are treated as dense and have no dependence on sparsity. Analogous results by different libraries on Device 0 are provided in Fig. 2, 3 showing significant reduction of time. Interesting to note that batched RF method having no interface with single precision is almost as efficient as QR_float. Results are a little bit different on Device 1. We checked the occupancy of GPU RAM. QR_float and RF use about the same amount of RAM, for example it requires 1845 MB using QR_float and 3590 MB using RF for $N = 1 \cdot 10^6, M = 11$, and sparsity 0.5. MAGMA requires way more memory due to some internal memory allocation instructions, for example magma_float takes about 831 MB for $N = 3 \cdot 10^4, M = 11$ and about 5264 MB for $N = 2.4 \cdot 10^5$. This memory is dynamically allocated during the solution process and can cause exception of insufficient memory despite that memory for all matrices storage is sufficient, so we limit MAGMA batch size up to 30,000.

Speed ups against CPU 4-thread code for different libraries and different devices are shown in Fig. 4–7. We can see that MAGMA speed up varies from 5 to 3 times for Device 0 and 4 – 2 times for Device 1. QR_float achieves speed up 40 times for small matrices and 20 for big matrices on Device 0 and 35 – 15 times for Device 1. RF is about the same results, lower by approximately 10% and QR_double lower by another 10%. We can see that RF library has a narrower spread between matrix sizes.

We then benchmark one library against another in terms of execution time. Results for MAGMA library are presented in Fig. 8 and 9 (left). One can see that other libraries are faster, so taking memory demands into account we scratch out MAGMA from comparison. We also compare QR_float and QR_double with RF on different devices. We can see in Fig. 10 that RF version is more efficient on Device 1 compared with both QR_double and QR_float. However, QR_float is more efficient on Device 0, see Fig. 11.

Further investigation is conducted in term of speed dependence from sparsity and matrix size for all batches. For this test we check only Device 0 because we found that the difference in scaling on these axes is negligible. One see the scaling of QR_float on sparsity 0.1 in Fig. 12 and sparsity 0.5 in Fig. 13. The factor of speed loss is about 2 3 times for the matrix size increase from 4 to 16 when sparsity is 0.5 which implies better performance on bigger matrices. This effect is even stronger when sparsity is 0.1 and also for RF library, see Fig. 14 and 15. Another dependence on sparsity is given in terms of matrix size and provided for QR_float in Fig. 18, 19, and for RF in Fig. 16, 17.

Analysis of performance in terms of floating point operations is provided in the Tab. 1. One can see that MAGMA has very small performance results in the batch mode, except using magma_float on Device 0 with 267.78 GLFOPS. This is, probably, due to the usage of new device features of compute capability 5.2. However, timing for this library mode remains almost the same. Notice, that when MAGMA is executed as magma_float, some calculations are performed in double precision, and visa versa. Best flops performance (41.17 GLFOPS) is achieved on QR_float on matrix size $M = 16$. We also checked asymptotics by considering $M = 128$. We can

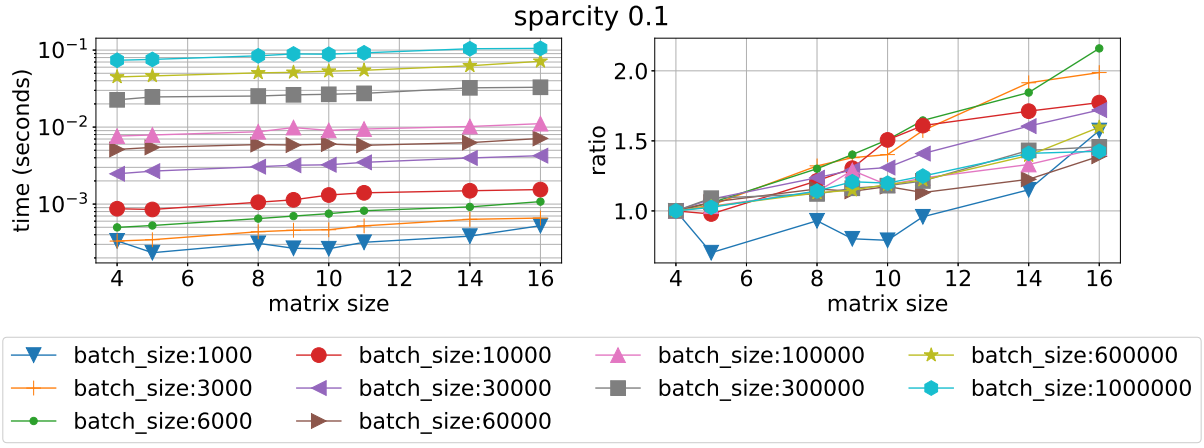


Figure 12. Mean time execution (left) and ratio of time execution to the smallest matrix vs matrix size for different batch sizes and sparsity 0.1 using QR_float on Device 0

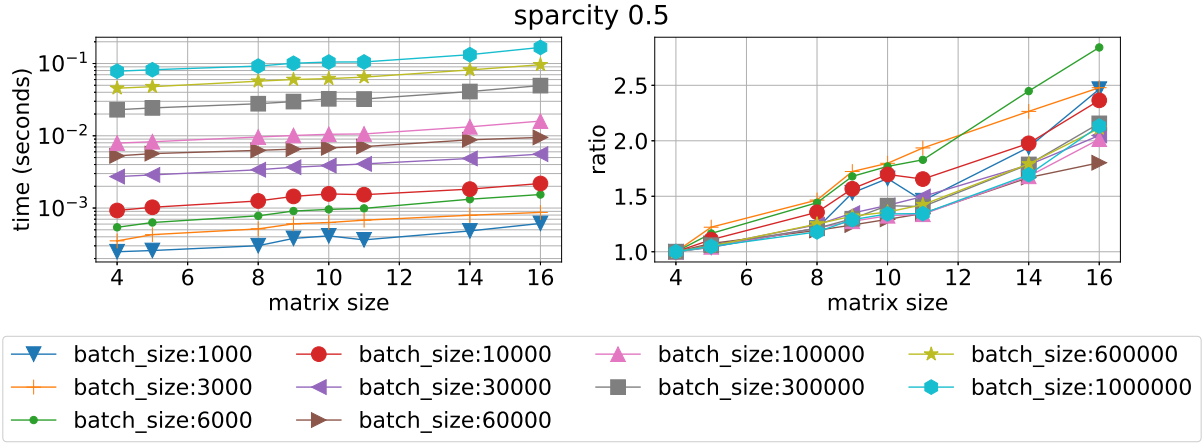


Figure 13. Mean time execution (left) and ratio of time execution to the smallest matrix vs matrix size for different batch sizes and sparsity 0.5 using QR_float on Device 0

see, that QR_float achieves maximum performance of 39.12 GFlops, so we can assume that the metrics is correct for $M = 16$. RF method also uses some single precision calculations but bulk part of all calculations is done in double precision with maximum of 20.35 GFlops.

3.2. Shuffle Gauss–Jordan Analysis

We test shuffleGJ method for $M = \{11, 16\}$ with sparsity 0.5. Ratio of time execution is shown in Fig. 20 for Device 0 and in Fig. 21 for Device 1. One can clearly see that the suggested method outperforms libraries on about 20 times for $M = 11$ and about 8–10 times for $M = 16$. This twofold decrease of performance is related to the algorithm requirements for matrix size in the shuffleGJ method. Still this gives us about 400 times acceleration compared to 4 threaded CPU version. We also calculated flops for this method that is provided in Tab. 1. One can see that we managed to achieve about 11.3% performance compared to the maximum theoretical performance of the device.

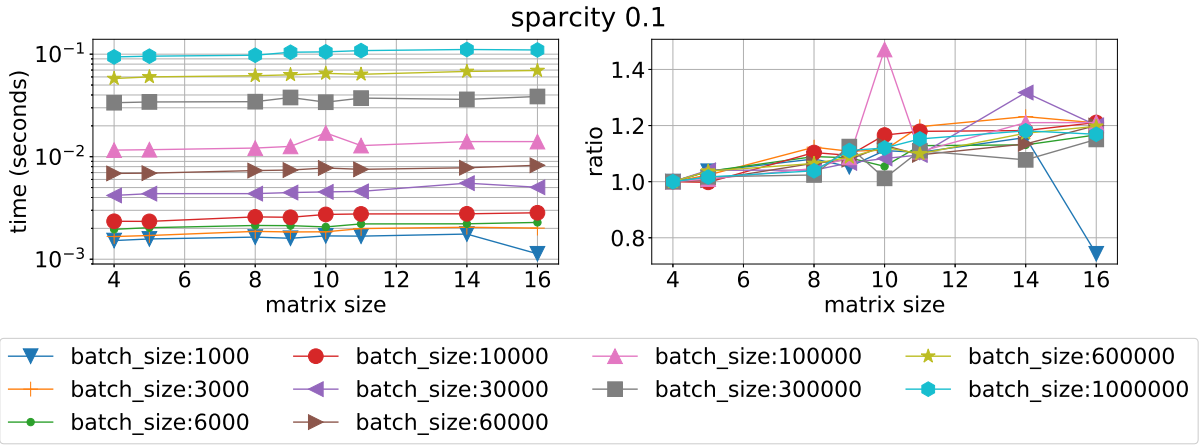


Figure 14. Mean time execution (left) and ratio of time execution to the smallest matrix vs matrix size for different batch sizes and sparsity 0.1 using RF on Device 0

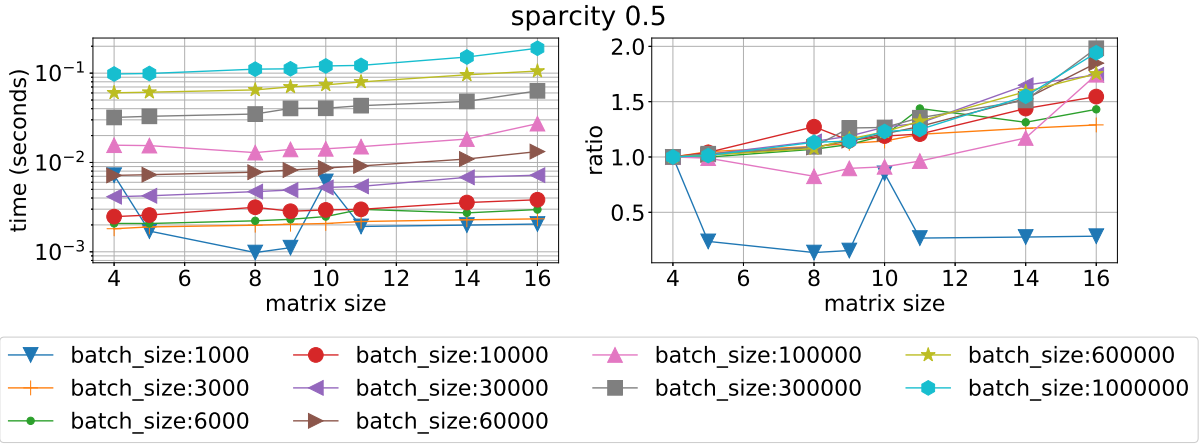


Figure 15. Mean time execution (left) and ratio of time execution to the smallest matrix vs matrix size for different batch sizes and sparsity 0.5 using RF on Device 0

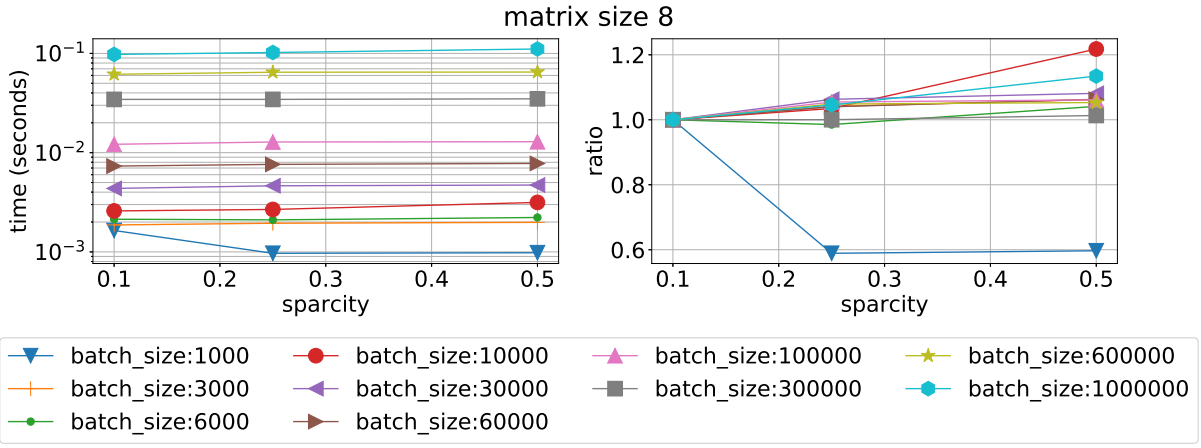


Figure 16. Mean time execution (left) and ratio of time execution to the smallest sparsity vs sparsity fill for different batch sizes and matrix size 8 using RF on Device 0

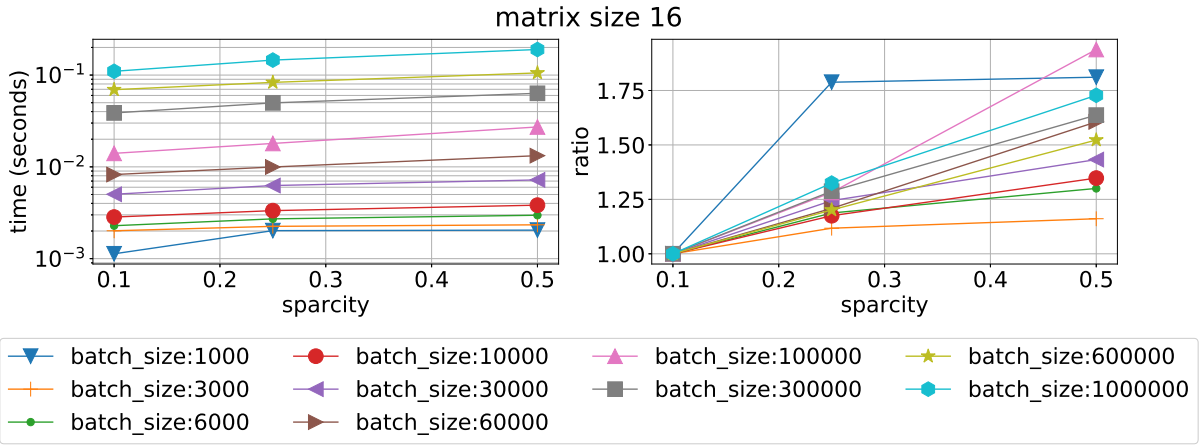


Figure 17. Mean time execution (left) and ratio of time execution to the smallest sparsity vs sparsity fill for different batch sizes and matrix size 16 using RF on Device 0

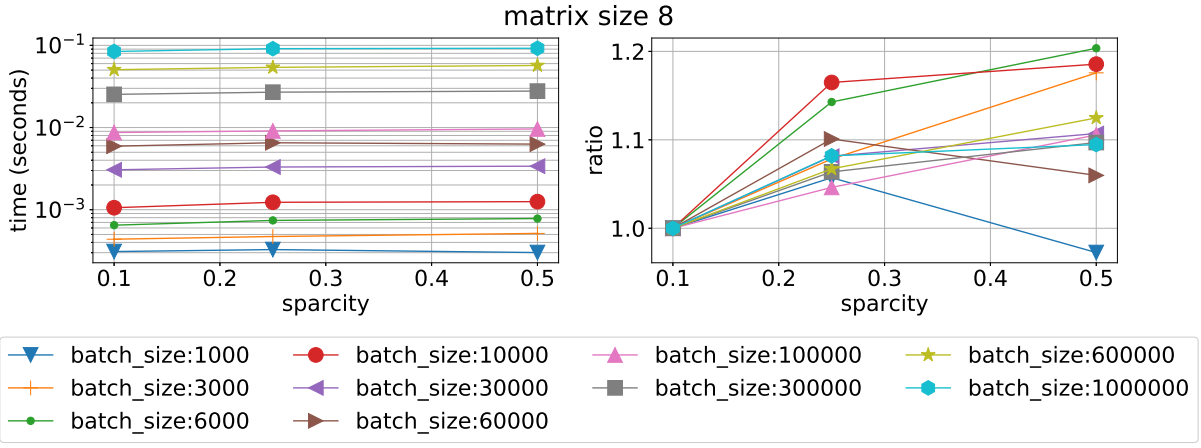


Figure 18. Mean time execution (left) and ratio of time execution to the smallest sparsity vs sparsity fill for different batch sizes and matrix size 8 using QR_float on Device 0

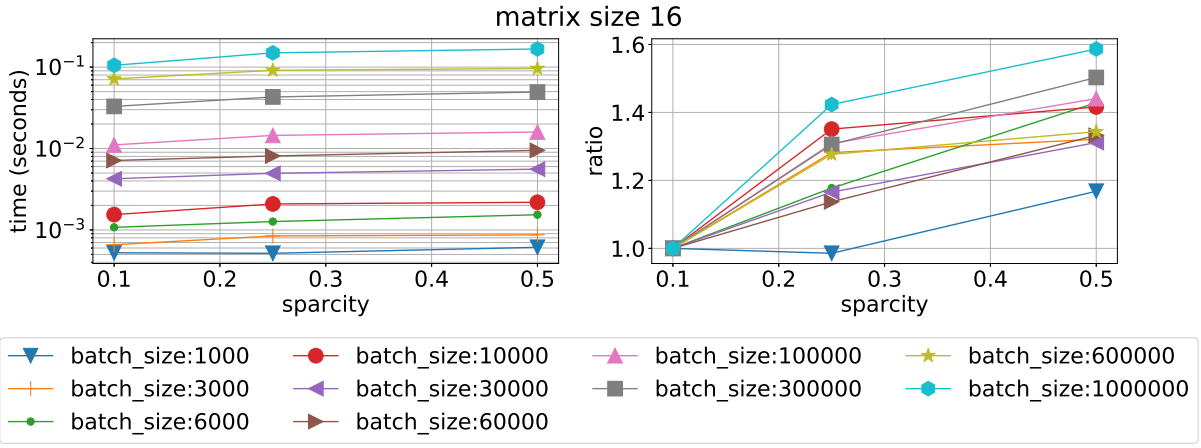


Figure 19. Mean time execution (left) and ratio of time execution to the smallest sparsity vs sparsity fill for different batch sizes and matrix size 8 using QR_float on Device 0

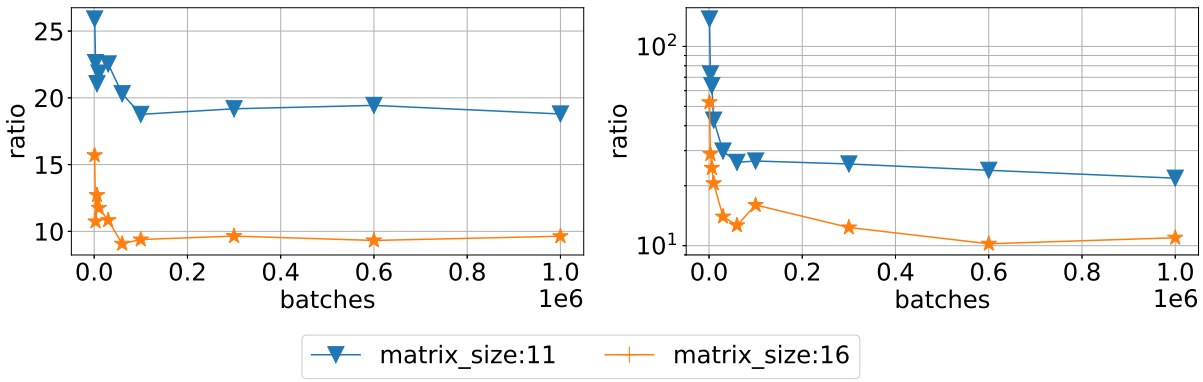


Figure 20. Ratio of time execution of QR_float/shuffleGJ (left) and RF/shuffleGJ (right) as function of batch size for sparsity 0.5 on Device 0

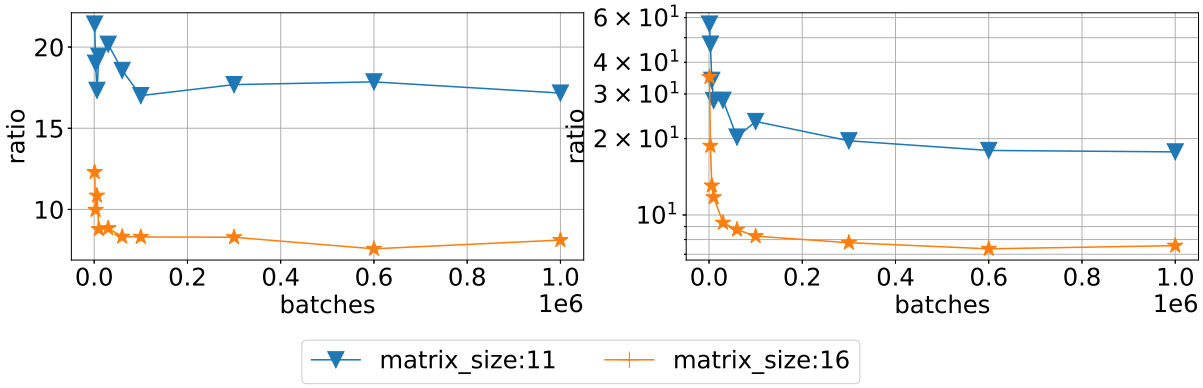


Figure 21. Ratio of time execution of QR_float/shuffleGJ (left) and RF/shuffleGJ (right) as function of batch size for sparsity 0.5 on Device 1

Table 1. Performance of different batch solver implementations on selected problems with sparsity 0.5 and one million batch size

solver	device	matrix_size	Gflops(float)	Gflops(double)
QR_float	0	4	4.85	0
QR_float	0	11	25.34	0
QR_float	0	16	41.17	0
QR_float	1	4	3.73	0
QR_float	1	11	15.16	0
QR_float	1	16	23.69	0
QR_double	0	4	0.18	7.97
QR_double	0	11	0.24	24.73
QR_double	0	16	0.119	28.04
QR_double	1	4	0.071	6.21
QR_double	1	11	0.105	15.51
QR_double	1	16	0.084	19.58
RF_double	0	4	0.3317	4.2
RF_double	0	11	0.497	14.5
RF_double	0	16	0.33	20.35
RF_double	1	4	0.136	3.45
RF_double	1	11	0.244	10.36
RF_double	1	16	0.234	14.5
magma_float	0	11	*267.78*	*0.14*
magma_float	1	11	9.22	*0.148*
magma_double	0	11	*0.139*	9.09
magma_double	1	11	*0.102*	7.31
QR_float	0	128	38.79	0
QR_float	1	128	39.12	0
shuffleGJ	0	11	955.49	0
shuffleGJ	0	16	1251.88	0
shuffleGJ	1	11	544.25	0
shuffleGJ	1	16	595.38	0

Table 2. Performance of MAGMA 2.3 library for LU factorization and solution of linear systems on matrices 16×16 with sparsity 0.5. Asterisk indicates tests with self written code, other tests use standard MAGMA library tests. Symbol '—' means that the problem does not fit in device memory

routine	batch size $\cdot 10^3$	Gflops	time, ms
sgetrf_batched (LU)	100	282.21	0.93
sgetrf_batched_nopiv (LU)	100	2.08	30.03
sgesv_batched (Solves)	100	14.67	20.0
sgetrs_batched* (Solves with LU)	100	12.4	23.05
sgetrf_batched (LU)	200	253.14	2.07
sgetrf_batched_nopiv (LU)	200	10.39	50.35
sgesv_batched (Solves)	200	19.68	31.0
sgetrs_batched* (Solves with LU)	200	18.3	33.86
sgetrf_batched (LU)	500	258.09	5.07
sgetrf_batched_nopiv (LU)	500	31.18	41.95
sgesv_batched (Solves)	500	22.63	72.9
sgetrs_batched* (Solves with LU)	500	26.7	33.86
sgetrf_batched (LU)	1000	307.26	8.51
sgetrf_batched_nopiv (LU)	1000	83.37	31.95
sgesv_batched (Solves)	1000	—	—
sgetrs_batched* (Solves with LU)	1000	—	—

3.3. Notes on MAGMA Library

We thank an anonymous reviewer for pointing out on the efficiency of newly released MAGMA 2.3 library (at the time of this research submission in October, 2017 latest MAGMA version was 2.2) for LU decomposition in batch mode. We performed tests on MAGMA library and confirmed that it achieves efficiency up to 308 GFlops (Device 0) for batched LU decomposition in single precision using **magma_sgetrf_batched** call with approximately 9.0 ms for one million matrices sized 16×16. This is outstanding result compared to **CUBLAS** native NVIDIA library. However, in this papaer we are interested in batch solution of linear systems. So we tested **magma_sgesv_batched** and **magma_sgetrs_batched** routines and obtained results close to the ones we obtained for MAGMA 2.2, see Tab. 2 for Device 0. For these tests we used simple programs that called these routines and compiled tests that are available in MAGMA library. Notice, that routine with no pivoting for LU decomposition takes substantial amount of time, compared to the standard LU decomposition with pivoting. This behaviour is abnormal and must be investigated.

In the results above we see, that only obtaining LU factors is efficient. One can't just take solver from MAGMA and solve batched linear systems out of box as it can be done for NVIDIA cuSPARSE libraries, at least on our hardware. Our recipe for MAGMA library is to use very efficient LU decomposition and then manually perform solution of linear systems (using batched triangular solver). One must take care, though, because arrays in GPU memory for MAGMA

calls are not optimally located for 1D indexing and one must introduce 2D grid in order to perform efficient solution of linear systems with sequential swaps in shared memory. This is the scope of the future work. More tests are required since results in paper [2] show performance of up to 800 GFlops for batch inversion of small matrices on P100 GPUs. Still, even LU MAGMA efficiency can't outperform our implementation of Gauss–Jordan method, compare GFlops in Tab. 1 and Tab. 2 for matrices 16×16 on Device 0.

4. Application Example

We are considering a standard chemical reaction flow benchmark problem, called ZND [9, 17, 18]. The problem is formulated for compressible perfect gas equations with chemical reaction. Detonation wave is propotaged with constant velocity D. The wave has the following structure: gas shock wave is propotaged, followed by reaction domain. Detonation reaction velocity is behind the shock wave. Governing equations are given bellow:

$$\begin{cases} \rho_t + \nabla \cdot (\rho \mathbf{u}) = 0, \\ (\rho \mathbf{u})_t + \nabla \cdot (\mathbf{u} \otimes (\rho \mathbf{u})) + \nabla p = \mathbf{0}, \\ \rho E_t + \nabla \cdot (\mathbf{u}(\rho E + p)) = 0, \\ (\rho Y_j)_t + \nabla \cdot (\rho Y_j \mathbf{u}) = \dot{\omega}_j. \end{cases} \tag{2}$$
$$\rho E = \tfrac{1}{2}\rho \mathbf{u}^2 + \rho e, j = \overline{1, M}, \sum_{j=1}^{M} Y_j = 1.$$

The chemical source term is given by:

$$\dot{\omega}_1 = -\dot{\omega}_2 = \begin{cases} -\rho Y_1 A \exp\left(\frac{-E_{act}}{\mathcal{R}T}\right), T \geq T_{ign}, \\ 0, T < T_{ign}. \end{cases} \tag{3}$$

Above equations are coupled by the equation of state:

$$\begin{gathered} p = \rho \widetilde{R} T, \\ h = C_p T + h_0, h_0 = \textstyle\sum_{j=1}^{M} h_{0,j} Y_j, \\ C_p = \frac{\gamma}{\gamma - 1} \widetilde{R}. \end{gathered} \tag{4}$$

Here ρ is density of gas mixture, ρY_j is the mass fraction of gas species j, $M = 2$ is the number of species, A is the Arrhenius frequency factor, $\mathbf{u}$ is the velocity vector, p is the pressure, E is the total energy density, h is the specific enthalpy, e is the specific internal energy, $\dot{\omega}_j$ is the reaction rate of species j, T is temperature, $\mathcal{R} = 8.31451$ is the universal gas constant, $\widetilde{R} = 1$ is the specific gas constant, γ is the specific heat ratio of the gas mixture, $h_{0,j}$ is the reference enthalpy of formation for the species j. In all calculations we set $T = 1$ and $T_{ign} = 1.01$. The fluid dynamics is solved using discontinuous Galerkin method [10], and the chemical part is solved using Crank–Nicolson method.

We define two domains – 1D segment and 2D plane. For the 1D segment we set boundary conditions as supersonic outflow conditions on the right and subsonic outflow conditions on the left. For the 2D plane we add two boundary conditions of sleep walls on top and bottom.

Initial conditions are defined as:

- $x \geq 0$: $(\rho, \rho u_x, \rho u_y, \rho E, \rho Y_1, \rho Y_2)^T = (\rho Y_1^*, -\rho Y_1^* D, 0, \rho E, \rho Y_1^*, 0)^T$;
- $x < 0$:

$$\frac{dY_1(x)}{dx} = -\frac{\dot{\omega}(Y_1(x))}{\rho(Y_1(x)) Y_1^* D}. \tag{5}$$

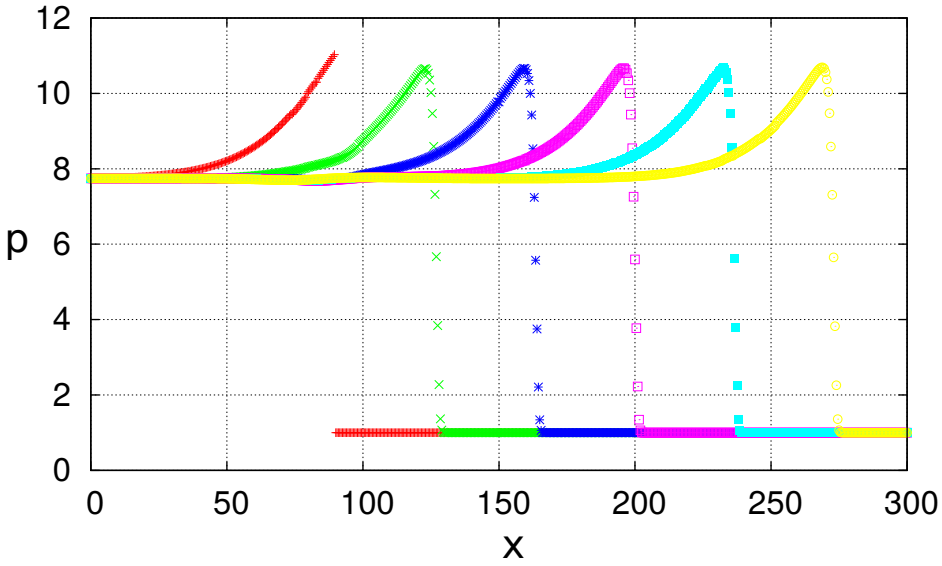


Figure 22. Pressure distributions for different time steps (10 seconds per step). Propagation velocity $D_{num} \sim 3.664275$ m/s

Here Y_1 is burned species, and $Y_2 = 1 - Y_1$ is unburned species, $Y_1^* = 1$ is a constant value. All other values for initial conditions are calculated in accordance with [11] as functions of $Y_1(x), \Delta h_{0,2}, D$. The Arrhenius frequency factor A is obtained from the given half–reaction length of a detonation wave as:

$$L_{1/2} = -\rho_2 D \int_{1/2}^{1} \frac{1}{\dot{\omega}(z)} dz. \tag{6}$$

The initial value problem (5) and value of A cannot be solved analytically, but the solution to these problems can be computed numerically for any given accuracy.

Three tests are performed – 1D comparison of computed velocity of detonation wave D, stability and instability of detonation wave for different initial D and 2D unstable detonation wave propagation. First results of the detonation wave propagation are presented in Fig. 22 for parameters $\gamma = 1.4$, $L_{1/2} = 12.5448$ m. One can observe that the obtained velocity $D_{num} \sim 3.664275$ m/s is close to the reference propagation velocity $D = 3.66931$ m/s. The other test verifies the stability of the detonation wave under provided value of D. We use parameter f to define the propagation velocity $D = \sqrt{f} D_{CJ}$ from the analytical speed D_{CJ} given by the Chapman – Jouguet theory [11]. The results demonstrate that for $L_{1/2} = 1$, $\gamma = 1.4$ and different values of parameter f one obtains different stability properties for the detonation wave. The results in Fig. 23 fully agree with reference results from [12].

The third test demonstrates spatial heterogeneity of the detonation wave (Fig. 24), calculated for the same parameter values as the previous test. Again, we can check these results with reference data from [12].

To check the performance of the chemical solver, we use the **RF** batch solver for the ODE batch Crank–Nicolson method. The performance results are presented in the Tab. 3. We can observe that the overall performance is satisfactory, though the chemical solver sightly degradates the performance.

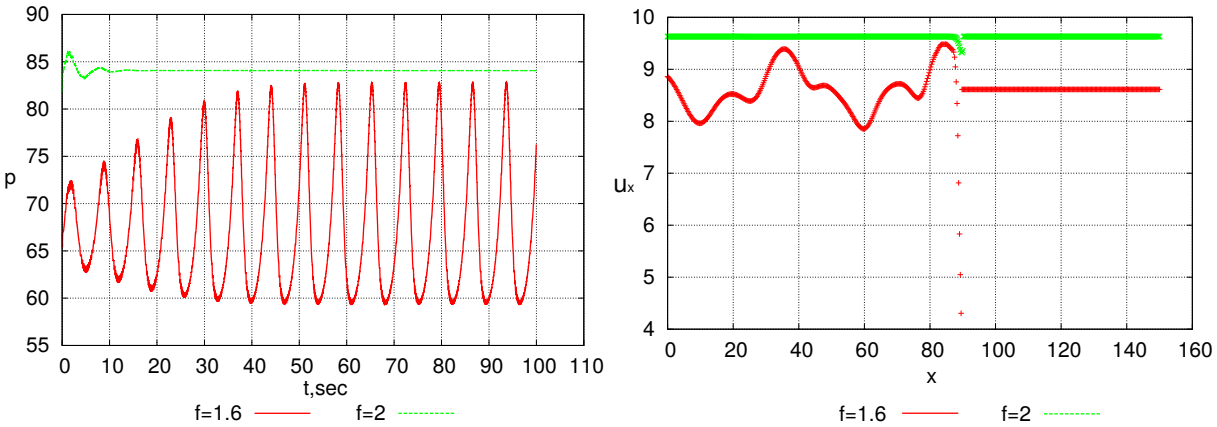


Figure 23. Temporal (left) and spatial (right) evolution of the detonation wave as function of initial detonation velocity parameter f

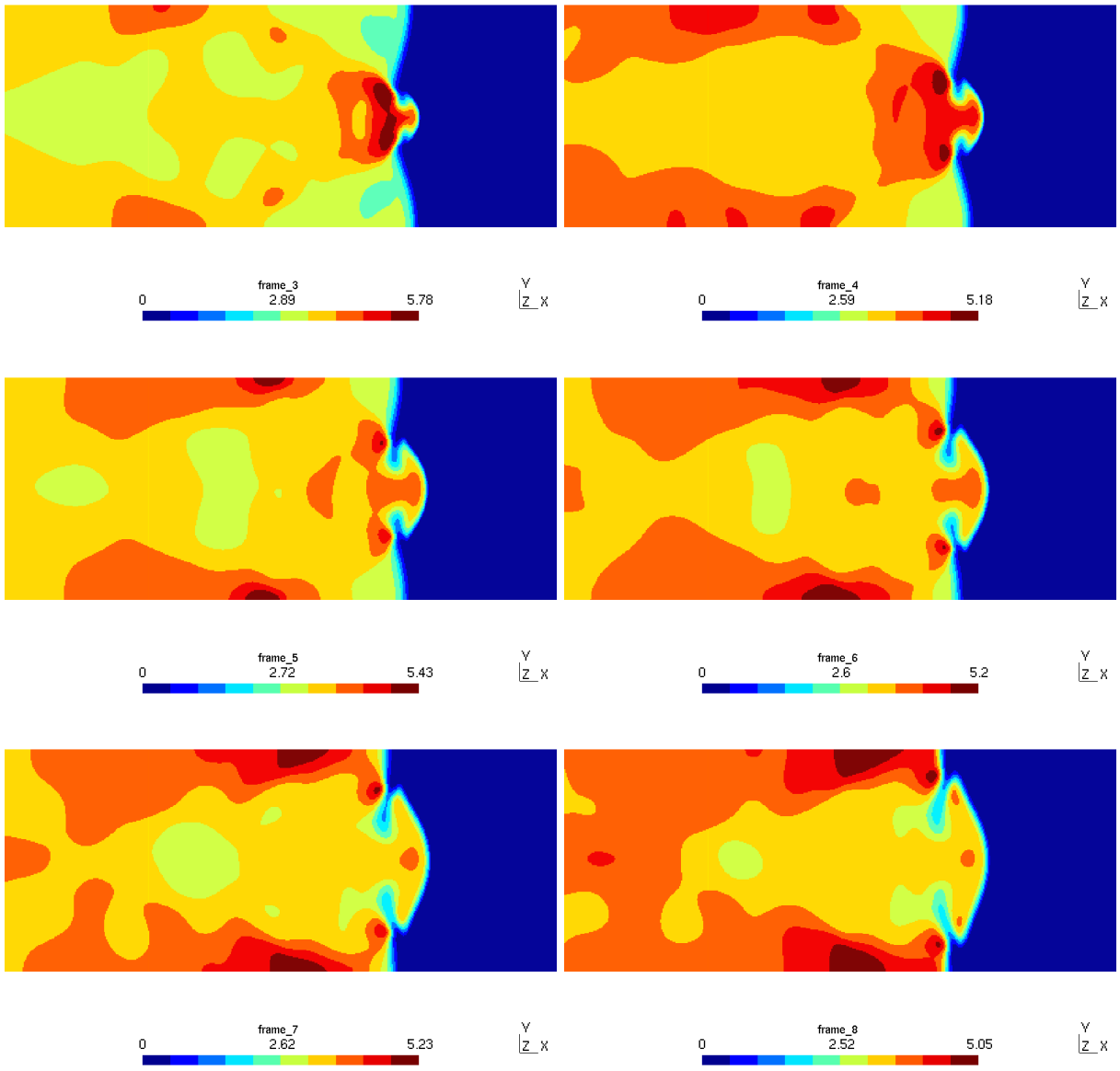


Figure 24. Spatial evolution of ρY_1 for different time steps in the reference frame moving with velocity D

Table 3. Acceleration of the 2D ZND problem for reactive flow (chem) and pure gas dynamics (no chem).

solver	time per step (ms)	acceleration
CPU, 1 thread, chem	8,708.158	1.00
GPU, Device 0, chem	46.339	187.92
GPU, Device 1,chem	65.381	133.19
CPU, 1 thread, no chem	5,922.279	1.00
GPU, Device 0, no chem	27.935	212.00
GPU, Device 1, no chem	43.538	136.03

Conclusion

First, we wish to note that we do not recommend using MAGMA batch library call for the solution of the problem (1) at least for library version 2.2. It is clear from all metrics that we collected, especially from Tab. 1. We also notice that it is very inefficient to use MAGMA solution routines for both MAGMA 2.2. and MAGMA 2.3. However, one can benefit from using very efficient MAGMA LU factorisation and then solve the system manually. This combination may look promising if one is ready to implement a self-written routine. Also, MAGMA may work much more efficiently on cutting edge GPUs (VOLTA architecture) so one must try MAGMA on these GPUs as well.

For the available libraries we can conclude the following. It is beneficial to use RF method if your code uses double precision arithmetics and QR_float if you are using single precision for our GPUs. This is valid for our hardware where GPUs have poor double precision performance. Both these methods perform graph connectivity analysis on CPU of provided matrices before calling batched routine for the solution of the problem on GPU. So these methods would require additional CPU work if your matrices connectivity is changing from one execution to anther. Note that the GFlops achieved by both of these methods is about 0.3 – 0.5% of the peak GFlops performance of GPUs.

If your GPUs support compute capability 3.0 and higher we can recommend using shuffle Gauss–Jordan method as an alternative to libraries for batch solution of linear systems. Achieved GLOPS and accelerations look promising. Debugging of this code and using it to solve plasma–chemistry on GPU is our next goal. Note, that we did not test this implementation on new NVIDIA GPUs (Volta architecture) and can't extrapolate these results.

In the last section we demonstrate a successful application of the batched solver in the coupled gas dynamics reacting to flow ZND problem.

Benchmark source codes are availablable at github [13] under GPL licence.

Acknowledgments

This work is supported by RFBR grant no. 17-07-00116 and by subprogram 0063–2016–0018 of the program III.3 ONIT RAS.

References

1. Abdelfattah, A., Haidar, A., Tomov, S., Dongarra, J.: Performance, Design, and Autotuning of Batched GEMM for GPUs. In: Kunkel, J.M., Balaji, P., Dongarra, J. (eds.) High Performance Computing, pp. 21–38, Springer International Publishing, Cham (2016), DOI: 10.1007/978-3-319-41321-1_2

2. Abdelfattah, A., Haidar, A., Tomov, S., Dongarra, J.: Factorization and Inversion of a Million Matrices using GPUs: Challenges and Countermeasures. Procedia Computer Science 108, 606–615, (2017), DOI: 10.1016/j.procs.2017.05.250

3. Anzt, H., Dongarra, J., Flegar, G., Quintana-Orti, E.S.: Batched Gauss-Jordan Elimination for Block-Jacobi Preconditioner Generation on GPUs. In: PMAM'17 Proceedings of the 8th International Workshop on Programming Models and Applications for Multicores and Manycores, 04–08 USA–February, Austin, TX, pp. 1–10 (2017), DOI: 10.1145/3026937.3026940

4. Asaithambi, R., Muppidi, S., Mahesh, K.: A numerical method for DNS of turbulent reacting flows using complex chemistry. 42nd AIAA Fluid Dynamics Conference and Exhibit, AIAA 2012–3252, (2012), DOI: 10.2514/6.2012-3252

5. cuSOLVER CUDA Toolkit Documentation. http://docs.nvidia.com/cuda/cusolver/index.html, accessed: 2018-04-01

6. Demouth, J.: Shuffle: Tips and Tricks. GPU Technology conference, (2013). http://on-demand.gputechconf.com/gtc/2013/presentations/S3174-Kepler-Shuffle-Tips-Tricks.pdf, accessed: 2018-04-01

7. Dong, T., Haidar, A., Luszczek, P., Harris, J.A., Tomov, S., Dongarra, J.: LU Factorization of Small Matrices: Accelerating Batched DGETRF on the GPU. In: IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS), (2014), DOI: 10.1109/HPCC.2014.30

8. Dong, T., Haidar, A.,Tomov, S., Dongarra, J.: A Fast Batched Cholesky Factorization on a GPU. In: Proc of 43-rd International Conference on Parallel Processing, 432–440 (2014), DOI: 10.1016/j.jocs.2016.12.009

9. Doring, W.: On detonation processes in gases. Annals of Physics 43, 421–436, (1943), DOI: 10.1002/andp.19434350605

10. Evstigneev, N.M., Ryabkov, O.I.: On The Development of High-Order Discontinuous Galerkin Method on 3D Unstructured Grid for Hyperbolic and Parabolic Problems Using Graphics Processors. In: Short Articles and Posters of the XI International Conference on Parallel Computational Technologies (PCT'2017), Kazan, 3–7 April 2017, pp. 63–77. Chelyabinsk, Publishing Center of the South Ural State University (2017)

11. Fickett, W., Davis, W.C.: Detonation, Theory and Experiment. Dover Publications (2000)

12. Geßner, T.: Dynamic Mesh Adaption for Supersonic Combustion Waves modeled with Detailed Reaction Mechanisms. Doctoral Dissertation, Universitat Freiburg im Breisgau (2001)

13. GIT authors repository. https://github.com/oryabkov/cuda_batch_linsolvers_test.git, accessed: 2018-04-01

14. MAGMA Library documentation. icl.cs.utk.edu/magma, accessed: 2018-04-01

15. Masliah, I., Abdelfattah, A., Haidar, A., Tomov, S., Baboulin, M., et al.: High-Performance Matrix-Matrix Multiplications of Very Small Matrices. 2nd International Conference on Parallel and Distributed Computing (Euro–Par 2016), Aug 2016, Grenoble, France. Springer, Lecture Notes in Computer Science, vol. 9833, pp. 659–671, (2016), DOI: 10.1007/978-3-319-43659-3

16. Rosenbrock, H.H.: Some general implicit processes for the numerical solution of differential equations. The Computer Journal 5(4), 329–330, (1963), DOI: 10.1093/comjnl/5.4.329

17. von Neumann, J.: Theory of detonation waves. In: A. J. Taub, editor, John von Neumann, Collected Works, vol. 6. Macmillan, New York (1942)

18. Zeldovich, Y. B.: On the theory of the propagation of detonation in gaseous systems. Journal of Experimental and Theoretical Physics, 10, 542–568 (1940). Engl. transl.: NACA TM 1261 (1960)

DOI: 10.14529/jsfi180204

Parallel Numerical Algorithm for Solving Advection Equation for Coagulating Particles

Sergey A. Matveev[1,3]***, Rishat R. Zagidullin***[2]***, Alexander P. Smirnov***[2,3]***, Eugene E. Tyrtyshnikov***[2,3]

In this work we present a parallel implementation of numerical algorithm solving the Cauchy problem for equation of advection of coagulating particles. This equation describes time-evolution of the concentration $f(x, v, t)$ of particles of size v at the point x at the time-moment t. Our numerical algorithm is based on use of total variation diminishing (TVD) scheme and perfectly matching layers (PML) for approximation of advection operator along spatial coordinate x and utilization of the fast numerical method for evaluation of coagulation integrals exploiting low-rank decomposition of coagulation kernel coefficients and fast FFT-based implementation of convolution operation along particle size coordinate v. In our work we exploit one-dimensional domain decomposition approach along spatial coordinate x because it allows to avoid use of parallel FFT implementations which are very expensive in terms of data exchanges and have poor parallel scalability. Moreover, locality of finite-difference operator from TVD-scheme along x coordinate allows to obtain good scalability even for computing clusters with slow network interconnect due to modest volumes of data necessary for synchronization exchanges between times integration steps.

Keywords: aggregation equations, parallel algorithms, low-rank matrices, convolution.

Introduction

Coagulation and fragmentation processes stand in the basement of a wide class of physical phenomena starting from micro-polymer chains growth [6, 17] and finishing at the scale of stars formation from interstellar dust [5, 8]. The very first model of aggregation was suggested by Smoluchowski in 1916 [24] and generalized by Hans Muller [18] into the form of the following integro-differential equation:

$$\frac{\partial f(t,v)}{\partial t} = \frac{1}{2}\int\limits_0^v K(u, v-u) f(t,u) f(t, v-u) du - f(t,v) \int\limits_0^\infty K(v,u) f(t,u) du \qquad (1)$$

This equation describes dynamics of concentration $f(t, v)$ of the particles of size v per unit volume. The first term in the right-hand side corresponds to growth of concentration due to coalescence of the aggregates of sizes v and $v - u$. The second term describes decrease of the concentration due to their coalescence of the particles of size v with other particles. Kernel coefficients

$$K(v, u) = K(u, v) \geq 0$$

correspond to rates of aggregation process and have to be derived for each concrete application. If the initial conditions $f(t = 0, v)$ are known, then Cauchy problem for the coagulation equation is defined and can be solved numerically.

Under natural assumptions [8] the solution of the Cauchy problem for semi-infinite size domain $v \in [0, \infty)$ can be approximated by its finite part $v \in [0, V_{\max}]$. In fact, the class

[1] Skolkovo Institute of Science and Technology, Moscow, Russian Federation
[2] Lomonosov Moscow State University, Faculty of Computational Mathematics and Cybernetics, Moscow, Russian Federation
[3] Marchuk Institute of Numerical Mathematics RAS, Moscow, Russian Federation

of mathematical models of potential interest is extremely wide and may include description of various physical effects: inception and sinks of particles of concrete sizes [2], unary fragmentation process [1] due to instabilities of big clusters and binary collisional fragmentation process [5], and many others [3, 7, 12, 14, 20]. Even though the list of phenomena for application of spatially homogeneous aggregation and fragmentation equations is really huge [25, 26], there is an even broader class of spatially inhomogeneous models [19, 22]. In this work we concentrate on analysis of advection-coagulation equation [28]:

$$\frac{\partial f(t,x,v)}{\partial t} + c(v)\frac{\partial f(t,x,v)}{\partial x} = \frac{1}{2}\int\limits_0^v K(u,v-u)f(t,x,u)f(t,x,v-u)du - \\ -f(t,x,v)\int\limits_0^\infty K(u,v)f(t,x,u)du, \tag{2}$$

where function $f(t,x,v)$ corresponds to concentration per unit volume of particles of size v at the point with coordinate x at moment t. Coagulating particles are transported along the axis of nonnegative coordinate x with velocity $c(v)$. The velocities of the coalescence processes are again determined by the values of function $K(u,v)$. In the similar manner with classical coagulation equation one needs initial conditions $f(t=0,x,v)$ and one boundary condition e.g. $f(t,x=0,v)$ for definition of the Cauchy problem for advection-coagulation equation and its further numerical investigation.

Recently, we proposed an efficient numerical method solving the Cauchy problem for this equation [28] and demonstrated its efficiency for the concrete examples of modelling problems. We revisit description of this algorithm in Section 1.

Nevertheless, even modest simulations presented in our previous report required quite a lot of CPU-time. In this work we propose a parallel implementation of this algorithm. We should also emphasize, that there exists a special parallel algorithm allowing to perform evaluation of aggregation and fragmentation sums of the similar structure as coagulation integrals. Even though, a reasonable speedup of calculations was presented at [16], the parallel scalability of the algorithm is relatively poor [16]. This fact lies in motivation to exploit one-dimensional domain decomposition approach along spatial coordinate x for advection-coagulation equation but not along axis v and not two-dimensional domain decomposition.

In Section 2 we present a detailed description of novel parallel algorithm. Section 3 is devoted to tests of performance of the proposed algorithm at different computing clusters and *Lomonosov* supercomputer. We obtain speedup of calculations more than by 300 times. This allows us consider a broader class of problems of potential interest for mathematical modelling and studies of advection-coagulation processes. In conclusions we discuss the presented results and possible directions for further generalization of the presented ideas.

1. Numerical Method, Revisited

In this section we revisit description of the fast numerical algorithm for advection-coagulation equation from work [28]. First of all, we use an explicit Euler time-integration

scheme with time step Δt for solution of the Cauchy problem:

$$\frac{f^{n+1} - f^n}{\Delta t} = A(f^n) + S(f^n),$$

where $A(f)$ is an approximation of the advection operator and $S(f)$ approximates Smoluchowski integrals. Let us denote M as number of grid nodes along particle size axis v with grid step Δv and parameters $v_1 = 0$, $v_M = M \cdot \Delta v$ and N as number of spatial grid nodes along x with step Δx, $x_1 = 0$, $x_N = N \cdot \Delta x$. Finally, we denote $f^n_{i,j}$ as value of numerical solution at grid node (x_j, v_i) at the moment $n \cdot \Delta t$.

For $S(f)$ we use skeleton decomposition of coagulation kernel and fast algorithms of linear algebra with overall algorithmic complexity of solution scheme to be $O(MR \log M)$ at each grid node along x axis [13]. Whereas without utilization of these ideas the complexity becomes $O(M^2)$. We discuss this method with more details in Section 2.2. We get significant acceleration for evaluation of Smoluchowski integrals if the rank R of the coagulation kernel is a modest number. For approximation of the advection part $A(f)$ we exploit well-known TVD scheme [10] and PML layers [4] allowing us to keep monotonicity of numerical method. Detailed relations for advection part are presented in Section 2.3.

1.1. Handling Smoluchowski Integrals

Skeleton decomposition of coagulation kernel

$$K(u, v) = \sum_{\alpha=1}^{R} a_\alpha(u) b_\alpha(v)$$

helps accelerate computations by reduction of the algorithmic complexity of the numerical evaluation of Smoluchowski integrals [27, 29].

With use of skeleton decomposition we can perform transformation of the first integral:

$$\int_0^v K(u, v-u) f(t, x, u) f(t, x, v-u) du = \int_0^v \sum_{\alpha=1}^{R} a_\alpha(u) b_\alpha(v-u) f(t, x, u) f(t, x, v-u) du = \\ = \sum_{\alpha=1}^{R} \int_0^v a_\alpha(u) f(t, x, u) b_\alpha(v-u) f(t, x, v-u) du.$$

Following this idea we also transform the second integral:

$$\int_0^\infty K(u, v) f(t, x, u) du \approx \int_0^{V_{\max}} \sum_{\alpha=1}^{R} a_\alpha(u) b_\alpha(v) f(t, x, u) du = \sum_{\alpha=1}^{R} b_\alpha(v) \int_0^v a_\alpha(u) f(t, x, u) du.$$

Transformation of integrals written as sum of R lower-triangular convolutions, hence, with the use of FFT algorithm, total algorithmic complexity of the numerical integration using the relations above yields to $O(MR \log M)$ operations at each from N points of spatial grid. Hence, total cost of evaluation of Smoluchowski integrals is $O(NMR \log M)$

1.2. Approximation of Advection Part

For approximation of the advection equation we use the following TVD-scheme.

$$\frac{f_j^{n+1} - f_j^n}{dt} = A(f^n) = -\frac{c}{2dx}(C_j f_{j+1}^n + (2 - C_j - C_{j-1})f_j^n - (2 - C_{j-1})f_{j-1}^n) + \\ + \frac{c^2 dt}{2dx^2}(C_j f_{j+1}^n - (C_j + C_{j-1})f_j^n + C_{j-1} f_{j-1}^n),$$

where C_j is a flux limiter function that ensures that no artificial oscillations occur and has to satisfy the following restrictions

$$\frac{f_j^n - f_{j-1}^n}{f_{j+1}^n - f_j^n} \geq 0 \rightarrow 0 \leq C_j \leq 2$$

$$\frac{f_j^n - f_{j-1}^n}{f_{j+1}^n - f_j^n} < 0 \rightarrow C_j = 0.$$

For simplicity of notations we revisit this scheme ignoring volume index i because at each grid point along size coordinate v_i we perform these operations along spatial axis x. In fact, there are a lot of possible choices for C_j which were introduced by many authors [10, 11]. In our work, we use the one that is called "Monotonized Central Flux Limiter" [10].

$$C_j = \max(0, \min(2r_j, 0.5(1 + r_j), 2)), \text{where } r_j = \frac{f_j - f_{j-1}}{f_{j+1} - f_j}$$

The last thing that has to be done with respect to numerical solution of advection equation is incorporation of PML to emulate the absence of the right boundary condition in the grid. Hence, we modify the advection equation into following:

$$\frac{\partial f}{\partial t} = -c\frac{\partial f}{\partial x} - c\sigma(x)f,$$

where

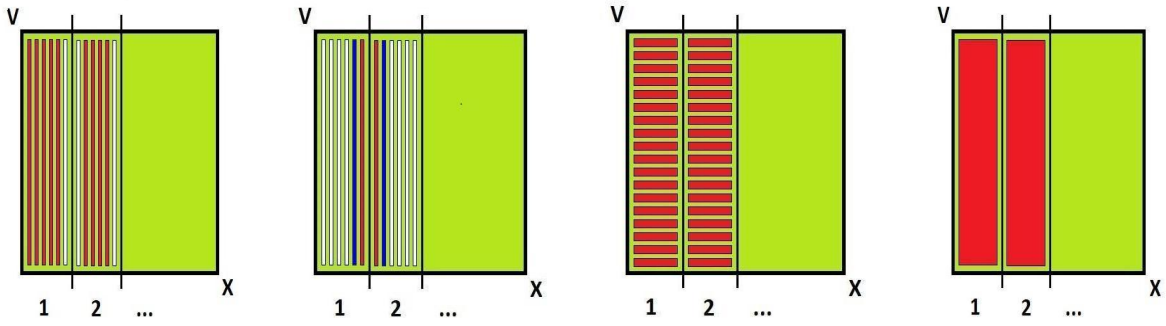
$$\sigma(x) = \begin{cases} \dfrac{(m+1)W \ln 10}{d} \left(\dfrac{x-l}{d}\right)^m & x \in [l; l+d], \\ 0 & \text{elsewhere,} \end{cases}$$

where l is width of calculation domain along axis x without PML, d is width of PML and W, m are PML parameters and formal assumption $f|_{x=l+d} \equiv 0$.

2. Parallel Algorithm

There are two possible ways of parallel implementation of our numerical method with use of p processors. First of all, we can perform simultaneous calculation by allocation of fixed particle size intervals to different processors. This is one-dimensional domain decomposition along axis v. But this option turns out to be bad due to quite poor parallel scalability of the algorithm for evaluation of the Smoluchoswki coagulation integrals what was reported in [16].

The reason lies in the fact that fast calculation of coagulation convolution typed integrals via FFT has a very limited resource of speedup on real parallel systems. Here we refer to work [9] where authors demonstrated that main problem in parallel execution of one-dimensional FFT is



(a) Computation of $S(f^n)$ (b) Data exchange (c) Computation of $A(f^n)$ (d) Calculation of f^{n+1}

Figure 1. One parallel time-integration step with use of the suggested parallel algorithm. Areas bordered by black lines represent the grid areas allocated to separated processors

problematic due to extremely high communication costs (comparable with computational). Such bad parallel scalability of evaluation of the coagulation integrals also leads to loss of motivation to exploit two-dimensional domain decomposition along both v and x coordinates in application to our problem.

However another option of execution of parallel calculations is to distribute them along the spatial coordinate x. Such distribution of data among processors corresponds exactly to uniform one-dimensional domain decomposition into the segments along spatial coordinate x. This approach allows to obtain very modest requirements in terms of number of communications – it is necessary just to synchronize the boundaries of spatial segments corresponding to neighboring processes for evaluation of advection operator with use of TVD-scheme stencil. In other words, we can allocate integrals at different points of the physical space to different processors and then transmit the data at the borders of the corresponding intervals. Thus, on the input each from p processes has set of grid nodes X and values of $f^n(x_i, v_j)$ at this domain $X \times (v_1, \ldots, v_M)$. Total complexity of time-integration step for each processor is $O\left(\frac{NRM \log M}{p}\right)$ operations and requires $O(pM)$ data elements for $O(p)$ data exchange operations.

We present our approach informally in Fig. 1. It shows the structure of the parallel numerical scheme and the areas of the grid which communicate with each other. Areas bordered by black lines represent the grid areas allocated to separated processors. Red stripes indicate how calculations are performed. Blue stripes pose organization of data transmission between the neighboring processes.

During stage corresponding to Fig. 1a each process evaluates the coagulation integrals along vertical axis v (lines 1-3 the pseudocode) and keeps two additional vectors for the boundaries assignment (presented as white stripes). Stage from Fig. 1b corresponds to synchronization of the particle size distributions along axis v (from blue stripes into red of the neighboring process) for the edge coordinates x between the processes (lines 4-5 of the pseudocode). It is necessary for the correct work of the TVD-scheme stencil. During stage from Fig. 1c the advection operator is evaluated along axis x for each particle size v (horizontal stripes corresponding to lines 6-10). All in all, during stage from Fig. 1d we provide calculation of the time-step result for both coordinates v and x (lines 11-16 of the pseudocode).

However, we should mention that grid parameter affects the parallel the efficiency of presented algorithm. Number of grid points along spatial coordinate x must be greater or equal to number of used processes. Otherwise there will be a lack of opportunity to distribute the data among the processors. We assume that the best parallel performance can be achieved in case of uniform domain decomposition when number of grid points along x can be divisible by a number of used processes. In the next section we present benchmarks of the proposed algorithm.

Algorithm 1 Operations performed by each process

for $i \in X$ **do**
    Compute $S(f^n(x_i; v_1 \ldots, v_M))$ at x_i along v ▷ Requires $O\left(\frac{NRM \log M}{p}\right)$ operations
end for
Exchange data with left and right neighbors
and assign boundary conditions ▷ Each process sends and receives $2M$ elements
for $i \in X$ **do** ▷ loop among grid points corresponding to process
    for $j \in V$ **do**
        Apply advection operator $A(f^n(x_i; v_j))$ ▷ Requires $O\left(\frac{NM}{p}\right)$ operations
    end for
end for
for $i \in X$ **do** ▷ loop among grid points corresponding to process
    for $j \in V$ **do**
        $f^{n+1}(x_i; v_j) = f^n(x_i; v_j) + \Delta t \cdot (\ S(f^n(x_i; v_j)) + A(f^n(x_i; v_j))\ )$ ▷ Requires $O\left(\frac{NM}{p}\right)$ operations
    end for
end for
return $f^{n+1}(x_i; v_j)$ ▷ defined for $X \times (v_1, \ldots, v_M)$

3. Benchmarking Parallel Implementation

In our numerical tests we investigated the Cauchy problem for the advection-coagulation equation with zero-value initial conditions and the following left boundary condition:

$$f(t, x = 0, v) = e^{-v^2}. \tag{3}$$

Thus, we model a constant flow of particles from one side of the grid to the other.

Also, calculations differ with respect to coagulation kernels. Tests were conducted for constant kernel

$$K(u, v) \equiv 1,$$

which rank $R = 1$ and for ballistic coagulation kernel

$$K(u, v) = (u^{\frac{1}{3}} + v^{\frac{1}{3}})^2 \sqrt{\frac{1}{u} + \frac{1}{v}}$$

with greater values of ranks. Grid parameters were set as $\Delta x = \Delta v = 10^{-2}$ and $\Delta t = 0.005$.

Scalability of parallel implementation was tested on several different computing clusters, so we assume that the results are robust. Moreover, our calculations also differ with respect to mesh and coagulation kernel. Surprisingly, even for a system with slow interconnect (see Tab. 4

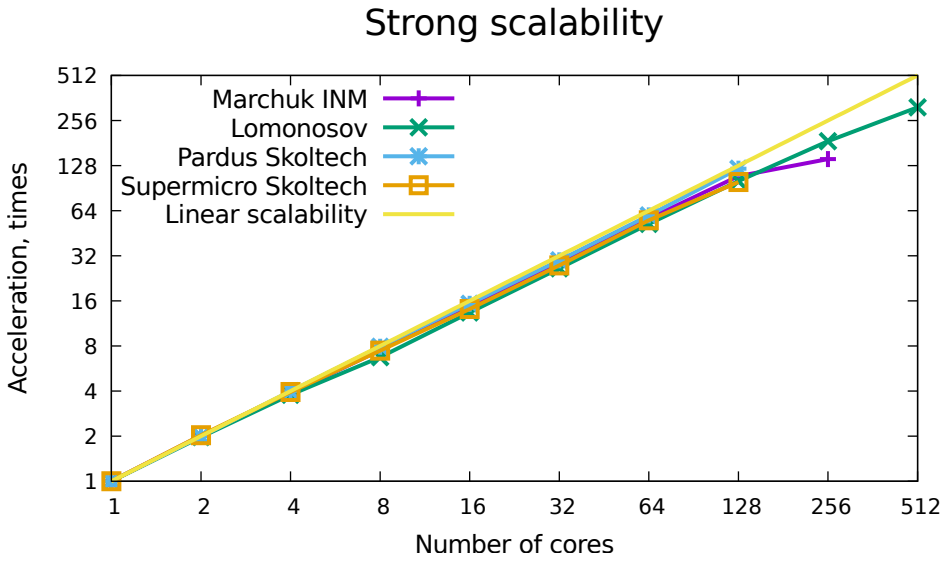


Figure 2. Strong scalability of the proposed parallel algorithm for different clusters. Visualized data corresponds to results presented in Tab. 1 – 4. We obtain almost linear speedup if number of processors is less than or equal to 128

Table 1. Supercomputer *Lomonosov*, tests for *regular4* partition with *40 Gb/s QDR* interconnect. Constant coagulation kernel, $R = 1$, $N = M = 8192$, 32 time-integration steps

p	Time, s	Speedup
1	34.476	1
2	17.468	1.97
4	9.126	3.778
8	5.124	6.728
16	2.576	13.384
32	1.306	26.398
64	0.66	52.236
128	0.342	100.807
256	0.185	186.357
512	0.11	313.418

with results at *Supermicro* cluster of Skoltech *1 Gb/s Ethernet*) and for heterogeneous cluster of Marchuk Institute of Numerical Mathematics (see Tab. 2) we obtain a reasonable speedup of calculations. Besides the fact that we obtain a good scalability of the algorithm for different clusters, we also see its good performance for different problem sizes (for each of Tab. 1 – 4 we tried a different problem size).

Tables 1 – 4 and Fig. 2 indicate that the scalability of parallel program is almost linear at the amount of nodes per dimension lower than or equal to 128. If the number of processors goes beyond that number, the payoffs from parallel implementation slowly decrease due to

Table 2. Cluster of Marchuk Institute of Numerical Mathematics RAS; test for heterogeneous cluster consisting of many different models of CPUs with *32 Gb/s QDR* interconnect. Ballistic coagulation kernel, $R = 12$, $N = M = 2048$, 32 time-integration steps

p	Time, s	Speedup
1	133.43	1
2	66.112	2.018
4	34.49	3.869
8	17.775	7.507
16	8.997	14.83
32	4.618	28.893
64	2.353	56.706
128	1.227	108.745
256	0.943	141.495

Table 3. *Pardus* cluster of Skoltech with *56 Gb/s FDR* interconnect. Ballistic coagulation kernel, $R = 14$, $N = M = 4992$, 32 time-integration steps

p	Time, s	Speedup
1	1202.947	1
2	604.34	1.99
4	301.304	3.992
8	152.223	7.903
16	78.681	15.289
32	40.3	29.85
64	20.175	59.626
128	9.848	122.151

Table 4. *Supermicro* cluster of Skoltech with *1 Gb/s Ethernet* interconnect. Ballistic coagulation kernel, $R = 14$, $N = M = 8064$, 32 time-integration steps

p	Time, s	Speedup
1	2373.189	1
2	1168.678	2.03
4	601.246	3.947
8	318.474	7.452
16	168.127	14.115
32	85.922	27.62
64	42.952	55.252
128	23.838	99.555

increasing costs of data exchanges between processors after each time integration step. Speedup also saturates with satisfactory results when the size of problems is not large enough (see Tab. 2).

Conclusions

In this work we propose a parallel implementation of the numerical method solving advection-coagulation equation from work [28]. The numerical scheme is parallelized along spatial axis x which promotes a good speedup of calculations. In the presented benchmarks we show that our scheme of finding the numerical solution of the problem might be accelerated by hundreds of times giving an almost linear speedup with respect to the amount of used CPU cores.

There is also an opportunity for additional increase of performance of the presented method. As far as coalescence integrals are evaluated sequentially at each processor their calculation can be additionally accelerated by use of GPU and Cuda technology. We will apply our ideas to more intricate multicomponent coagulation models [7, 15, 23] requiring special implementations of multidimensional convolutions based on utilization of low-rank tensor decompositions [15, 21].

Acknowledgements

This work is supported by the Program 1 of the Presidium of the Russian Academy of Sciences “Fundamental mathematics and its applications” and by Program 26 “Fundamental foundations of design of algorithms and software for prospective and high-performance computing systems”. We also would like to acknowledge Center for Computational and Data-Intensive Science and Engineering of Skolkovo Institute of Science and Technology for comprehensive support of their Pardus cluster which we used during preparation of our benchmarks.

References

1. Aloyan, A.: Dynamics and kinematics of gas impurities and aerosols in the atmosphere. A Textbook (2002)

2. Ball, R., Connaughton, C., Jones, P., Rajesh, R., Zaboronski, O.: Collective oscillations in irreversible coagulation driven by monomer inputs and large-cluster outputs. Physical review letters 109(16), 168304 (2012), DOI: 10.1103/PhysRevLett.109.168304

3. Ball, R., Connaughton, C., Stein, T.H., Zaboronski, O.: Instantaneous gelation in Smoluchowskis coagulation equation revisited. Physical Review E 84(1), 011111 (2011), DOI: 10.1103/PhysRevE.84.011111

4. Berenger, J.P.: A perfectly matched layer for the absorption of electromagnetic waves. Journal of Computational Physics 114(2), 185–200 (1994), DOI: 10.1006/jcph.1994.1159

5. Brilliantov, N., Krapivsky, P., Bodrova, A., Spahn, F., Hayakawa, H., Stadnichuk, V., Schmidt, J.: Size distribution of particles in saturns rings from aggregation and fragmentation. Proceedings of the National Academy of Sciences 112(31), 9536–9541 (2015), DOI: 10.1073/pnas.1503957112

6. Brilliantov, N., Bodrova, A., Krapivsky, P.: A model of ballistic aggregation and fragmentation. Journal of Statistical Mechanics: Theory and Experiment 2009(06), P06011 (2009), DOI: 10.1088/1742-5468/2009/06/P06011

7. Chaudhury, A., Oseledets, I., Ramachandran, R.: A computationally efficient technique for the solution of multi-dimensional PBMs of granulation via tensor decomposition. Computers & Chemical Engineering 61, 234–244 (2014), DOI: 10.1016/j.compchemeng.2013.10.020

8. Galkin, V.: Smoluchowski equation. Fizmatlit, Moscow (2001), (in Russian)

9. Gupta, A., Kumar, V.: The scalability of fft on parallel computers. Parallel and Distributed Systems, IEEE Transactions on 4(8), 922–932 (1993), DOI: 10.1109/71.238626

10. Leer, B.V.: Towards the ultimate conservative difference scheme. iv. a new approach to numerical convection. Journal of Computational Physics 23(3), 276–299 (1977), DOI: 10.1016/0021-9991(77)90095-X

11. Lyra, P.R.M., Morgan, K., Peraire, J., Peir, J.: TVD algorithms for the solution of the compressible euler equations on unstructured meshes. International Journal for Numerical Methods in Fluids 19(9), 827–847, DOI: 10.1002/fld.1650190906

12. Matveev, S.A., Krapivsky, P.L., Smirnov, A.P., Tyrtyshnikov, E.E., Brilliantov, N.V.: Oscillations in aggregation-shattering processes. Phys. Rev. Lett. 119, 260601 (Dec 2017), DOI: 10.1103/PhysRevLett.119.260601

13. Matveev, S., Smirnov, A., Tyrtyshnikov, E.: A fast numerical method for the cauchy problem for the Smoluchowski equation. Journal of Computational Physics 282, 23–32 (2015), DOI: 10.1016/j.jcp.2014.11.003

14. Matveev, S., Stadnichuk, V., Tyrtyshnikov, E., Smirnov, A., Ampilogova, N., Brilliantov, N.: Anderson acceleration method of finding steady-state particle size distribution for a wide class of aggregationfragmentation models. Computer Physics Communications 224, 154–163 (2018), DOI: 10.1016/j.cpc.2017.11.002

15. Matveev, S., Zheltkov, D., Tyrtyshnikov, E., Smirnov, A.: Tensor train versus Monte Carlo for the multicomponent Smoluchowski coagulation equation. Journal of Computational Physics 316, 164–179 (2016), DOI: 10.1016/j.jcp.2016.04.025

16. Matveev, S.A.: A parallel implementation of a fast method for solving the smoluchowski-type kinetic equations of aggregation and fragmentation processes. Vychislitel'nye Metody i Programmirovanie 16(3), 360–368 (2015), (in Russian)

17. Mirzaev, I., Byrne, E.C., Bortz, D.M.: An inverse problem for a class of conditional probability measure-dependent evolution equations. Inverse Problems 32(9), 095005 (2016), DOI: 10.1088/0266-5611/32/9/095005

18. Müller, H.: Zur allgemeinen theorie ser raschen koagulation. Fortschrittsberichte über Kolloide und Polymere 27(6), 223–250 (1928), DOI: 10.1007/BF02558510

19. Okuzumi, S., Tanaka, H., Kobayashi, H., Wada, K.: Rapid coagulation of porous dust aggregates outside the snow line: A pathway to successful icy planetesimal formation. The Astrophysical Journal 752(2), 106 (2012), DOI: 10.1088/0004-637X/752/2/106

20. Piskunov, V.: Analytical solutions for coagulation and condensation kinetics of composite particles. Physica D: Nonlinear Phenomena 249, 38–45 (2013), DOI: 10.1016/j.physd.2013.01.008

21. Rakhuba, M.V., Oseledets, I.V.: Fast multidimensional convolution in low-rank tensor formats via cross approximation. SIAM Journal on Scientific Computing 37(2), A565–A582 (2015), DOI: 10.1137/140958529

22. Sabelfeld, K.: A random walk on spheres based kinetic monte carlo method for simulation of the fluctuation-limited bimolecular reactions. Mathematics and Computers in Simulation (2016), DOI: 10.1016/j.matcom.2016.03.011

23. Smirnov, A., Matveev, S., Zheltkov, D., Tyrtyshnikov, E.: Fast and accurate finite-difference method solving multicomponent Smoluchowski coagulation equation with source and sink terms. Procedia Computer Science 80, 2141–2146 (2016), DOI: 10.1016/j.procs.2016.05.533

24. von Smoluchowski, M.: Drei vortrage uber diffusion, brownsche bewegung und koagulation von kolloidteilchen. Zeitschrift fur Physik 17, 557–585 (1916)

25. Sorokin, A., Strizhov, V., Demin, M., Smirnov, A.: Monte-Carlo modeling of aerosol kinetics. Atomic Energy 117(4), 289–293 (2015), DOI: 10.1007/s10512-015-9923-7

26. Stadnichuk, V., Bodrova, A., Brilliantov, N.: Smoluchowski aggregation–fragmentation equations: Fast numerical method to find steady-state solutions. International Journal of Modern Physics B 29(29), 1550208 (2015), DOI: 10.1142/S0217979215502082

27. Tyrtyshnikov, E.E.: Incomplete cross approximation in the mosaic–skeleton method. Computing 64(4), 367–380 (2000), DOI: 10.1007/s006070070031

28. Zagidullin, R.R., Smirnov, A.P., Matveev, S.A., Tyrtyshnikov, E.E.: An efficient numerical method for a mathematical model of a transport of coagulating particles. Moscow University Computational Mathematics and Cybernetics 41(4), 179–186 (Oct 2017), DOI: 10.3103/S0278641917040082

29. Zheltkov, D.A., Tyrtyshnikov, E.E.: A parallel implementation of the matrix cross approximation method. Vychislitel'nye Metody i Programmirovanie 16(3), 369–375 (2015), (in Russian)

DOI: 10.14529/jsfi180205

Generation of Multiple Turbulent Flow States for the Simulations with Ensemble Averaging

Boris I. Krasnopolsky[1]

The paper deals with the problem of improving the performance of high-fidelity incompressible turbulent flow simulations on high performance computing systems. The ensemble averaging approach, combining averaging in time together with averaging over multiple ensembles, allows to speedup the corresponding simulations by increasing the computing intensity of the numerical method (flops per byte ratio). The current paper focuses on further improvement of the proposed computational methodology, and particularly, on the optimization of procedure to generate multiple independent turbulent flow states.

Keywords: Incompressible turbulent flow, Generalized sparse matrix vector multiplication, Multiple right-hand sides, Ensemble averaging.

Introduction

The high-fidelity simulations of turbulent flows for the complex geometries are among the typical applications for the high performance computing (HPC) systems. These applications are characterized by the high computational costs, long simulation times, and low computational efficiency, not exceeding several percent of the peak performance of HPC system. This leads to the situation when the simulations may take months to complete even using modern compute systems.

The present paper focuses on the problem of improving the performance and efficiency of the high-fidelity turbulent flow simulations. An approach to model incompressible turbulent flows, combining conventional time averaging with the ensemble averaging has been proposed in [1, 4]. This approach allows to parallelize the simulations in time and replace the single simulation with long time averaging by multiple simulations with much shorter time averaging intervals. The paper [4] suggests to perform each of these simulations independently and to obtain the simulation speedup by increasing the amount of utilized compute resources. This approach increases the corresponding computational costs for the simulation, but allows to speedup the calculations beyond the strong scalability limit. Alternatively, the paper [1] suggests to perform the simulations in a single run and rearrange the computations in order to utilize the operations with blocks of vectors. The multiple solutions of system of linear algebraic equations (SLAE) for the pressure Poisson equation can be combined in a single solution of SLAE with multiple right-hand sides (RHS). The memory traffic reduction when solving the SLAE with multiple RHS compared to multiple solutions of SLAE with single RHS vectors allows to increase the computational intensity of the algorithm (flops per byte ratio), and as a result, to speedup the turbulent flow simulations by a factor of 1.5–2.

Both simulation scenarios mentioned above are the limiting cases of the generalized algorithm, combining several independent runs and several flow states inside each run. The corresponding estimates and recommendations to choose the optimal simulation scenario have been proposed in [2].

While the results presented in [1, 4] demonstrate the proof of concept for the suggested approach to parallelize the simulations in time, several questions affecting overall simulation

[1]Institute of Mechanics, Lomonosov Moscow State University, Moscow, Russian Federation

speedup are not discussed in detail in these publications. Among them is the problem of generation of multiple uncorrelated turbulent flow states to start averaging in time. The simplest approach used in the mentioned publications assumes the simulation of flow transition for multiple different initial flow states. These calculations, however, produce significant computational overhead thus reducing the efficiency of the suggested methodology. The current paper focuses on the problem of improving the efficiency of the proposed ensemble averaging methodology, and particularly, on the aspects of efficient generation of multiple initial turbulent flow states. An alternative strategy to generate initial turbulent flow states based on introduction of random perturbations to the single flow state at different moments in time is examined. The efficiency of the methodology is analyzed in terms of the total time to compute m uncorrelated flow states, that can be used as initial flow states for time averaging.

The paper is organized as follows. Section 1 contains theoretical estimates reflecting the influence of generation of multiple turbulent flow states on the overall efficiency of the ensemble averaging methodology. A brief description of the computational codes used for the experimental evaluation is presented in Section 2. Section 3 is focused on the detailed description of the procedure used to generate multiple turbulent flow states and its experimental comparison with the basic one, including the full simulation of the transition interval for each of flow states. Finally, the Conclusion section concludes the paper.

1. Theoretical Estimates

The simplest, but computationally expensive way to generate multiple independent turbulent flow states comprises of modelling of transition to the turbulence for the flows with different initial states (e.g. laminar flows with different perturbations). The choice of different initial flow states guarantees the turbulent flow states finally to become uncorrelated. This statement is a result of exponential divergence of the turbulent flow trajectories in the phase state, and the corresponding divergence growth rate is determined by the Lyapunov exponent [6, 8].

The corresponding simulation of transition to the turbulence for multiple flow states includes two characteristic time scales: (i) the time scale to perform transition to the statistically steady turbulent flow state, T_{turb}, and (ii) the decorrelation time scale when two different turbulent flow states become uncorrelated, T_{corr}. The basic strategy to generate multiple turbulent flow states can be optimized depending on the ratio of these time scales. In case the transition time scale exceeds the decorrelation time scale, $T_{turb} \gg T_{corr}$, the procedure of generation multiple turbulent flow states can be obviously improved. For example, the simulation of the transition interval can be started for the single flow state. Then, the small perturbations are introduced at the bifurcation point, T_B, to generate multiple uncorrelated flow states. The bifurcation point is reasonable to be chosen close to the end of the transition interval, T_T, with the limitation $T_T - T_B \gtrsim T_{corr}$ (Fig. 1).

The contribution of the optimized procedure to generate multiple turbulent flow states on the overall simulation speedup can be expressed with simple theoretical estimates. Let us consider the expected simulation speedup for the ensemble averaging approach with the suggested procedure to generate multiple initial flow states. Following the methodology suggested in [1], the simulation speedup P_m can be expressed as:

$$P_m = \frac{\mathcal{T}_1}{\mathcal{T}_m}, \tag{1}$$

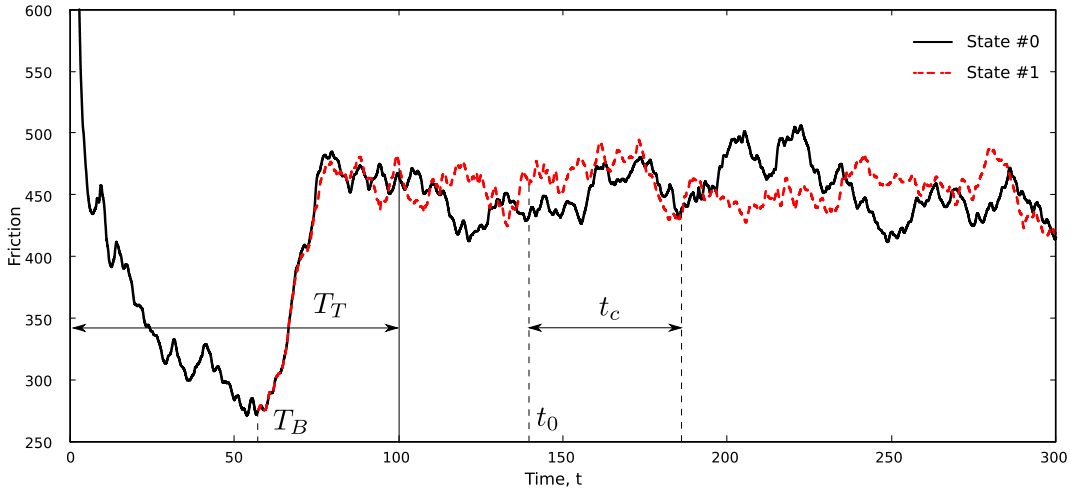


Figure 1. Evolution of the friction at the channel walls for the problem of modelling turbulent flow in a channel with a matrix of wall-mounted cubes. Results are presented for two flow states; the second state is produced as a result of evolution of random perturbations introduced in T_B moment in time

where $\mathcal{T}_1$ and $\mathcal{T}_m$ are the times to simulate the problem using averaging over single and m flow states respectively. The time to solve the problem with averaging over single flow state is expressed as:

$$\mathcal{T}_1 = \frac{T_T + T_A}{\tau} t_1, \tag{2}$$

where τ is the integration step, t_1 is the time to simulate single integration step, and T_A is the overall time averaging interval. The simulation time with multiple flow states includes the simulation with single flow state for the initial interval T_B and the subsequent simulation of interval $T_T - T_B + T_A/m$ with m flow states:

$$\mathcal{T}_m = \frac{T_B}{\tau} t_1 + \frac{(T_T - T_B + T_A/m)}{\tau} t_m. \tag{3}$$

Here, t_m is the time to simulate single integration step with m flow states. Substituting these expressions into (1) and introducing the parameters $\beta = T_A/T_T$ and $\gamma = (T_T - T_B)/T_T$, one can obtain the following relation:

$$P_m = \frac{1+\beta}{(1-\gamma) + (\beta + m\gamma)\left(\frac{mt_1}{t_m}\right)^{-1}}. \tag{4}$$

Using the estimate for the simulation times ratio based on the memory traffic reduction, proposed in [1], the final form of the estimate is obtained:

$$P_m = \frac{5m(1+\beta)}{5m(1-\gamma) + (5m - 3\theta(m-1))(\beta + m\gamma)}. \tag{5}$$

The expression (5) is the generalization of the basic estimate in [1] on the case $\gamma \neq 1$.

The suggested estimate allows to analyse the influence of the computational costs reduction to perform transition for multiple flow states on the overall performance gain when using the method combining averaging in time with the ensemble averaging to model turbulent flows. The corresponding performance gain distributions presented in Fig.2, show that the parameter γ strongly affects on the simulation speedup. The reduction of computational overhead to generate

multiple turbulent flow states leads to three major changes: (i) increase of the expected overall simulation speedup, (ii) increase of the optimal number of modelled turbulent flow states, and (iii) extension of the range of applicability for the proposed ensemble averaging methodology towards $\beta \sim 1$.

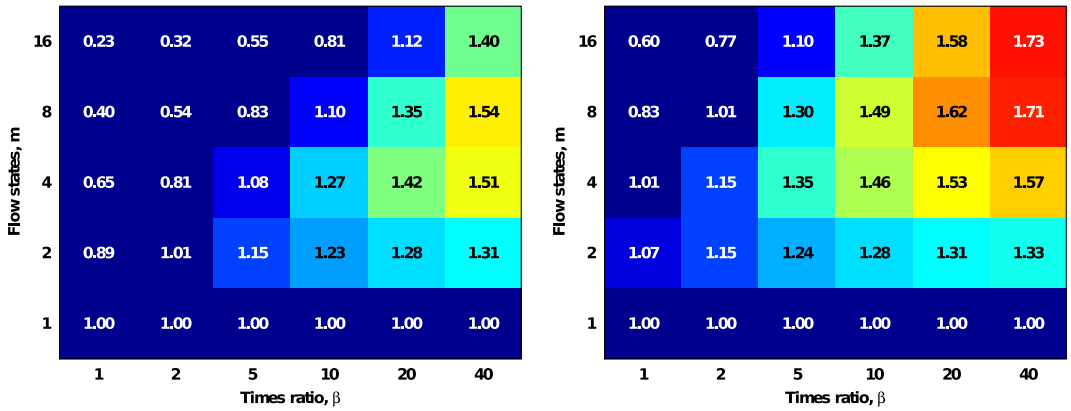


Figure 2. Expected overall simulation speedup for $\gamma = 1$ (left) and $\gamma = 0.25$ (right)

2. Validation

The estimate (5) vividly demonstrate that reduction of computational costs to construct multiple independent flow states allows to significantly improve the range of applicability of the ensemble averaging methodology and the overall simulation speedup. The suggested approach to generate multiple turbulent flow states is based on fast decorrelation of different flow states compared to the transition interval. In practice, however, the length of both transition and decorrelation intervals is problem specific, and reliable estimates can only be obtained from some preliminary simulations.

2.1. Computational Codes

The corresponding computational codes to perform the simulations with multiple turbulent flow states, including the SLAE solver and "in-house" application for direct numerical simulation of turbulent flows have been developed. The SLAE solver implements the Krylov-subspace iterative methods, accelerated by the algebraic multigrid preconditioner. The "in-house" DNS code is based on finite difference scheme for structured grids [7] providing second-order spatial discretization together with third-order Runge-Kutta scheme for time integration. The further details on these codes can be found in [1, 3].

2.2. Test Problem

The current paper uses the problem of modelling turbulent flow in a channel with a matrix of wall-mounted cubes [5] as a validation example. The problem statement assumes the isothermal flow of incompressible viscous fluid, governed by Navier-Stokes and continuity equations, in a rectangular domain containing single dedicated cube (Fig. 3). The corresponding equations are supplemented with periodic boundary conditions on streamwise and spanwise directions, and no-slip conditions on the rigid walls. The constant flow rate is preserved with the Reynolds number $\mathrm{Re}_b = U_b h/\nu = 3854$, where U_b is the bulk velocity, h is the cube height, and ν is

the kinematic viscosity of the fluid. The integration interval for this problem is set to $T = 2100$ with transition interval $T_T = 100$ and averaging interval $T_A = 2000$ time units, i.e. $\beta = T_A/T_T = 20$. The simulations presented in this paper are performed on structured orthogonal non-uniform computational grid consisting of 2.32 mln. cells. All the simulations are calculated on "Lomonosov-2" supercomputer and use 32 compute nodes per each run.

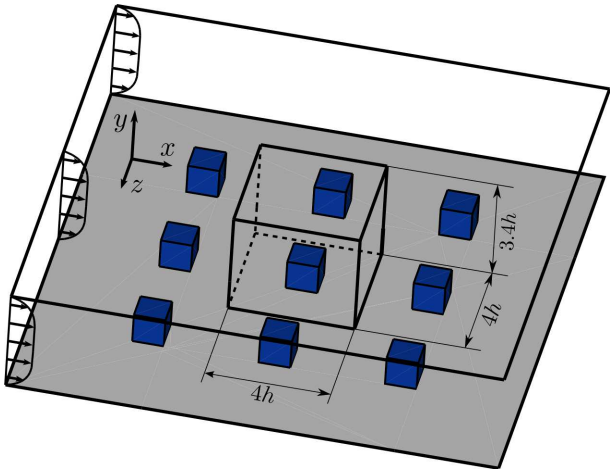


Figure 3. Sketch of the computational domain

2.3. Validation Procedure

The several simulations are performed to validate the procedure of generation multiple independent turbulent flow states with the introduction of random perturbations during the modelling of transition interval. The simulations differ in the choice of the bifurcation point. The cross-correlation functions are used to ensure the decorrelation of the corresponding flow states. The corresponding velocity components time series are collected in the control points around the cube. The cross-correlation function for two time series X and Y is defined as:

$$\rho_{XY} = \frac{cov(X,Y)}{var(X)\,var(Y)}, \tag{6}$$

where

$$cov(X,Y) = \frac{1}{t_c}\int_{t_0}^{t_0+t_c} (X(t)-\bar{X})(Y(t)-\bar{Y})dt,$$
$$\bar{X} = \frac{1}{t_c}\int_{t_0}^{t_0+t_c} X(t)dt,$$
$$var(X) = cov(X,X).$$

The time series X and Y are the corresponding subsets of the velocity components profiles, defined by the starting point t_0 and the length of the interval t_c (Fig. 1).

3. Numerical Results

A series of simulations has been performed to validate the proposed optimized procedure of generation multiple turbulent flow states. The first series comprised of modelling transition interval to generate decorrelated initial turbulent flow states at the end of transition interval.

Initially, the simulation is performed with single flow state. Using the obtained intermediate data sets the simulation is restarted at $T_B = 77$ to generate 4 decorrelated flow states. The corresponding simulation time, including the time to simulate single flow state over the interval T_B and 4 flow states from the bifurcation point to the end of transition interval is equal to 71 min. (Tab. 1). This result indicates about 1.9 times speedup for the generation of multiple turbulent flow states compared to basic methodology where the perturbations are introduced at the beginning of transition interval.

Table 1. The time to generate multiple turbulent flow states with various simulation scenarios, results in min

Flow states, m	T_B	1 flow state	m flow states	Total
1	–	53	–	53
4	0	–	132	132
4	77	41	30	71

The second test series includes simulation of several full scale DNS runs to model flow over a matrix of wall mounted cubes. This includes the run with single flow state, the run with 4 flow states and two runs with restarts from the bifurcation point $T_B = 77$ for 4 and 8 flow states. The corresponding compute times are summarized in Tab. 2. Using the expression (5) one can obtain the expected simulation speedup with the optimized procedure of generation multiple turbulent flow states compared to conventional DNS run. With basic procedure to generate initial turbulent flow states, the optimal number of flow states is equal to $m = 4$, and the speedup by a factor of 1.42 is expected. The obtained simulation speedup is 1.38, which is close to predicted one[2]. For the simulations with bifurcation point $T_B = 77$, corresponding to $\gamma = 0.23$, performance gain for 4 flow states of about 1.53 is expected, and the optimal number of states is equal to 8 with the speedup by a factor of 1.63. These results correlate with the full scale simulation results, which are equal to 1.49 and 1.57 correspondingly.

Table 2. The computational times to simulate the turbulent flow with various number of simultaneously modelled flow states, results in min

Flow states, m	1 flow state	m flow states	Total
1	1088	–	1088
4	–	790	790
4	41	688	729
8	41	654	695

The time series for the velocity components monitored in 8 control points around the cube are analyzed to verify the obtained initial turbulent flow states are uncorrelated. The corresponding cross-correlation function distributions are evaluated for the time length $t_c = 100$. The distributions presented in Fig. 4, demonstrate that the turbulent flows become uncorrelated before the end of transition interval, $T_T = 100$.

[2]The presented simulation times obtained in the current test session are systematically 10-15% higher than the ones published in [1]. The reason for the observed difference will be further investigated.

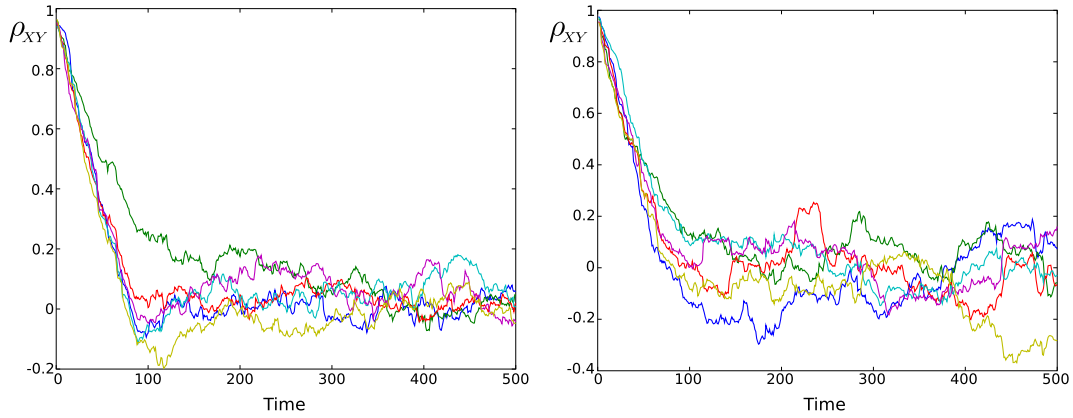


Figure 4. Cross-correlation distributions for the streamwise velocity in different control points. Results for the simulation with 4 flow states

Conclusion

The current paper focuses on further improvement of the algorithm to model turbulent flows based on ensemble averaging, proposed earlier in [1]. The problem of reducing the computational overhead when generating multiple initial turbulent flow states is investigated. The new procedure to generate uncorrelated flow states based on introduction of random perturbations during the modelling transition to the statistically steady turbulent flow state is proposed. The corresponding theoretical estimates demonstrate that the optimized procedure for generation multiple turbulent flows extends the range of applicability and improves the overall performance gain for the ensemble averaging approach. The corresponding numerical experiments performed for the problem of modelling the turbulent flow in a channel with a matrix of wall-mounted cubes are in agreement with suggested theoretical estimates and demonstrate additional 15% overall speedup for the ensemble averaging methodology.

Acknowledgements

The presented work is supported by the RSF grant No. 18-71-10075. The research is carried out using the equipment of the shared research facilities of HPC computing resources at Lomonosov Moscow State University.

References

1. Krasnopolsky, B.: An approach for accelerating incompressible turbulent flow simulations based on simultaneous modelling of multiple ensembles. Computer Physics Communications 229, 8–19 (2018), DOI: 10.1016/j.cpc.2018.03.023

2. Krasnopolsky, B.: Optimal strategy for modelling turbulent flows with ensemble averaging on high performance computing systems. Lobachevskii Journal of Mathematics 39(4), 533–542 (2018), DOI: 10.1134/S199508021804008X

3. Krasnopolsky, B., Medvedev, A.: Acceleration of large scale OpenFOAM simulations on

distributed systems with multicore CPUs and GPUs. In: Parallel Computing: On the Road to Exascale. Advances in Parallel Computing, vol. 27, pp. 93–102. Amsterdam, Netherlands (2016), DOI: 10.3233/978-1-61499-621-7-93

4. Makarashvili, V., Merzari, E., Obabko, A., Siegel, A., Fischer, P.: A performance analysis of ensemble averaging for high fidelity turbulence simulations at the strong scaling limit. Computer Physics Communications 219, 236–245 (2017), DOI: 10.1016/j.cpc.2017.05.023

5. Meinders, E., Hanjalić, K.: Vortex structure and heat transfer in turbulent flow over a wall-mounted matrix of cubes. International Journal of Heat and Fluid Flow 20(3), 255–267 (1999), DOI: 10.1016/S0142-727X(99)00016-8

6. Nastac, G., Labahn, J.W., Magri, L., Ihme, M.: Lyapunov exponent as a metric for assessing the dynamic content and predictability of large-eddy simulations. Physical Review Fluids 2(9), 094606 (Sep 2017), DOI: 10.1103/PhysRevFluids.2.094606

7. Nikitin, N.: Finite-difference method for incompressible Navier-Stokes equations in arbitrary orthogonal curvilinear coordinates. J. Comput. Phys. 217(2), 759–781 (2006), DOI: 10.1016/j.jcp.2006.01.036

8. Nikitin, N.: Disturbance growth rate in turbulent wall flows. Fluid Dynamics 44(5), 27–32 (2009), DOI: 10.1134/S0015462809050032

DOI: 10.14529/jsfi180206

High Performance Computing with Coarse Grained Model of Biological Macromolecules

Emilia Lubecka[1,2], *Adam Sieradzan*[2], *Cezary Czaplewski*[2], *Paweł Krupa*[3], *Adam Liwo*[2]

The Unified Coarse Grained Model of biological macromolecules (UCGM) that is being developed in our laboratory is a model designed to carry out large-scale simulations of biological macromolecules. The simplified chain representation used in the model allows to obtain 3-4 orders of magnitude extention of the time-scale of simulations, compared to that of all-atom simulations. Unlike most of the other coarse-grained force fields, UCGM is a physics-based force field, independent of structural databases and applicable to treat non-standard systems. In this communication, the efficiency and scalability of the new version of UCGM package with Fortran 90, with two parallelization levels: coarse-grained and fine-grained, is reported for systems with various size and oligomeric state. The performance was tested in the canonical- and replica exchange MD mode, with small- and moderate-size proteins and protein complexes (20 to 1,636 amino-acid residues), as well as with large systems such as, e.g., human proteosome 20S with size over 6,200 amino-acid residues, which show the advantage of using coarse-graining. It is demonstrated that, with using massively parallel architectures, and owing to the physics-based nature of UCGM, real-time simulations of the behavior of subcellular systems are feasible.

Keywords: Unified Coarse Grained Model, UNRES force field, molecular dynamics simulations, fine-grained parallelization, coarse-grained parallelization.

Introduction

The knowledge of the three-dimensional structure of macrobiomolecules is crucial for understanding many biological processes. Most biologically important molecules are too large to handle for the classical simulation tools. Therefore, coarse-grained models and force fields have become useful in these studies [37]. Reduced representations of molecules allow saving computational cost of a few orders of magnitude in comparison with full-atomistic simulations. Various types of coarse-grained models such as the SICHO and CABS models have been developed [13, 14]. These knowledge-based protein models, developed in the Kolinski group, use only three degrees of freedom per residue. Compared to coarse-grained models of proteins [12], coarse-grained models of polysaccharides have a relatively short history. Most of them use three or four interaction centers per sugar ring [26, 28, 34], including the polysaccharide component of the well-known MARTINI force field [24]. The MARTINI force field allows to simulate also other important biomolecules: lipids, proteins [29], DNA etc. [27]

The Unified Coarse Grained Model of biological macromolecules (UCGM) [19] that is being developed in our laboratory is a physics-based model, designed to carry out large-scale simulations of biological macromolecules: proteins (the UNRES model [20]), nucleic acids (the NARES-2P model [7]), polysaccharides (the SUGRES-1P model), and lipid membranes. The chain representation in UCGM is reduced to two interaction sites (united peptide groups and united side chains in UNRES, united phosphate groups and united sugar-base groups in NARES-2P) or one interaction site (united sugar rings in SUGRES-1P). In UCGM the effective energy function was derived via generalized-cumulant expansion [17] of the potential of mean force of

[1]Institute of Informatics, Faculty of Mathematics, Physics and Informatics, University of Gdańsk, Gdańsk, Poland
[2]Faculty of Chemistry, University of Gdańsk, Gdańsk, Poland
[3]Institute of Physics, Polish Academy of Sciences, Warsaw, Poland

biopolymer chains [21, 36]. Therefore, it reproduces the structure, dynamics, and thermodynamics of biopolymers very well despite the significant reduction of the number of interaction sites.

UNRES, which is the most advanced component of UCGM and has the longest history of development, has succeeded in physics-based protein-structure predictions [8, 15], investigation of protein-folding [38] and studying biological processes, such as amyloid formation [33], and iron-sulfur cluster biogenesis [30]. NARES-2P [7] can reproduce the structures and thermodynamics of small DNA and RNA molecules in free molecular-dynamics simulations and has also recently been applied with success to model the stability of telomere sequences of DNA [35]. The preliminary version of SUGRES-1P [19] can reproduce the helical structure of amylose and the sheet-like structure of cellulose.

The article is organized as follows. Section 1 describes the UNRES model, force field, and coarse-grained molecular dynamics used in our study. In Section 2, we present details of implementation of UNRES with Fortran 90. In Section 3, we report the results and their analysis. The Conclusion section summarizes our study.

1. Theory

1.1. UNRES Model and Force Field

In the UNRES model (Fig. 1) [19], a polypeptide chain is represented as a sequence of α-carbon (C^α) atoms connected by virtual bonds, with united side chains (SC) attached to them (with different $C^\alpha \cdots SC$ bond lengths, dsc) and united peptide groups (p) positioned in the middle between the two consecutive C^αs. The *SC*s and *p*s are the only interacting sites, while the C^αs only assist in geometry definition. Backbone geometry of the simplified polypeptide chain is defined by the $C^\alpha \cdots C^\alpha \cdots C^\alpha$ virtual-bond angles θ (θ_i has the vertex at C_i^α) and the $C^\alpha \cdots C^\alpha \cdots C^\alpha \cdots C^\alpha$ virtual-bond-dihedral angles γ (γ_i has the axis passing trough C_i^α and C_{i+1}^α). The local geometry of the ith side-chain center is defined by the zenith angle α_i (the angle between the bisector of the respective angle θ_i and the $C_i^\alpha \cdots SC_i$ vector) and the azimuth angle β_i (the angle of counter-clockwise rotation of the $C_i^\alpha \cdots SC_i$ vector about the bisector from the $C_{i-1}^\alpha \cdots C_i^\alpha \cdots C_{i+1}^\alpha$ plane, starting from C_{i-1}^α). The energy function consists of local (including virtual-bond-deformation, virtual-bond-angle bending, virtual-bond-torsional and double torsional, and virtual-side-chain rotamer potentials), pairwise site-site interaction potentials, and correlation (multibody) potentials [20]. The solvent is implicit in the force field. UNRES was derived based on the potential of mean force (PMF) of polypeptide chains in water, which was factorized into Kubo cluster-cumulant functions [17], which are, in turn, identified with specific effective energy terms [21, 36]. Expansion of the cluster-cumulant functions into Kubo cluster cumulants [17] yielded approximate analytical expressions for the energy terms, including the correlation terms [21, 36]. The force field was calibrated with solution structural data of 7 traning proteins with various types of structure, by using the maximum-likelihood method recently developed in our laboratory [16].

Because of PMF-based origin of UNRES, the force field has the sense of free energy restricted to a given coarse-grained conformation rather than the potential energy. The temperature dependence was implemented in our earlier work [22]; it affects mostly the correlation terms, thereby contributing to weakening the regular structure elements with increasing temperature.

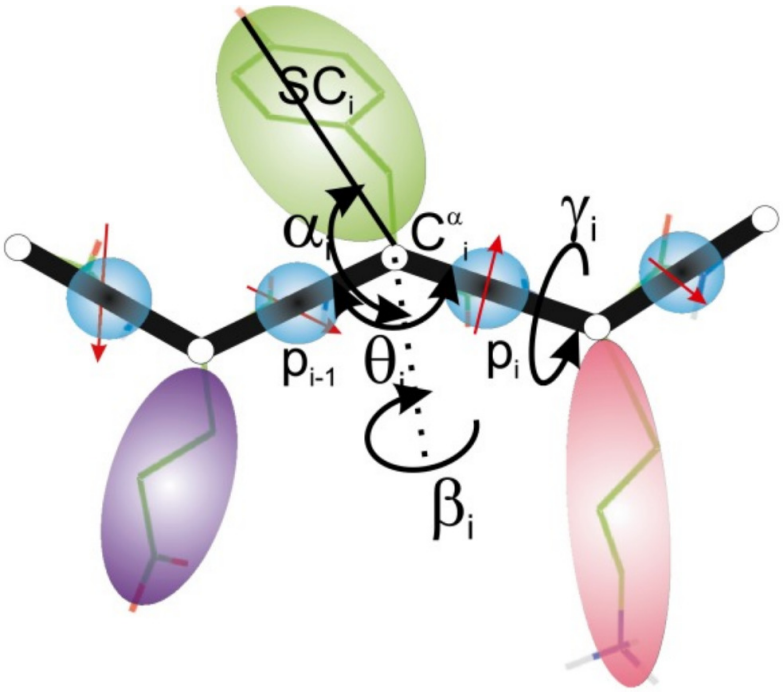


Figure 1. UNRES model of polypeptide chains, with united peptide-groups (p)(blue circles), united side chains (SC)(ellipsoids) and α carbons (C^{α})(small open circles)

1.2. Molecular Dynamics and Replica Exchange Molecular Dynamics with UNRES

Molecular dynamics is the core of the main conformational-search engine with UNRES. The equations of motion are Langevin equations because of implicit solvent treatment; however, for faster search, the simple Berendsen thermostat [2] (which originates from application of the Langevin equations to a system) is often used [9, 10]; the Nosé–Hoover and Nosé–Poincaré thermostats that enable more strict control of temperature conservation were also implemented [11]. Because the virtual bonds are treated as stretchable rods, it was natural to select the $C^{\alpha}\cdots C^{\alpha}$ and $C^{\alpha}\cdots$SC virtual-bond vectors as variables. With such a representation, the inertia matrix is a full, albeit constant, matrix. Recently, by selecting the C^{α} and SC coordinates as variables, we reduced it to a pentadiagonal form (J. Sans and A. Liwo, to be submitted), this resulting in linear and not quadratic scaling the memory requirements with system size. A version of adaptive multiple-time-step velocity-Verlet algorithm was developed to integrate the equations of motion [9, 31].

For improved conformational search, replica exchange (REMD) [6] and multiplexed replica exchange (MREMD) [32] extensions of MD were implemented with UNRES [4]. In the MREMD method, a series of independent MD trajectories are run, each at a different temperature, with synchronization every pre-defined (usually 10,000 or 20,000) number of steps. At each synchronization point, the energies of the conformations from consecutive trajectories are compared, and exchange of temperature occurs following the Metropolis criterion. This approach gives a chance for high-energy conformations stuck at low temperatures to escape from high-energy regions because of overcoming energy barriers upon heating and, conversely, to low-energy conformations simulated at high temperatures to cool down and achieve even lower energy. The

MREMD method differs from REMD in that more than one trajectory is simulated at a given temperature.

2. Implementation

2.1. Parallelization

UNRES was parallelized on two levels: coarse-grained and fine-grained [23]. At the coarse-grained level, each MD trajectory is handled by a given task or task group. This means trivial parallelization scheme when many canonical MD trajectories are run, or task-farm-type parallelization scheme for REMD/MREMD runs, with the master processor controlling the synchronization and exchange of replicas and the slave processors running only MD trajectories between the synchronization points. The REMD/MREMD UNRES jobs cannot be run in a serial mode, and the number of cores per job must be equal to the number of trajectories if coarse-grained-parallelization only is executed or a multiple of the number of trajectories for fine-grained parallelization. Because the management work is a small fraction of trajectory computations and the speed of calculations of all trajectories is virtually the same, the master processor also computes an MD trajectory. The efficiency of coarse-grained parallelization is close to 100 %.

To treat large systems, fine-grained parallelization was developed [23] in which energy and force evaluation was split between the tasks handling a given trajectory. The task-group master distributes coordinates to the slave tasks and collects the energy and force components from them; it also does its share of energy and force calculations.

2.2. Implementation with Fortran 90

Originally, UNRES was written in Fortran 77. Because of this, the code started to be more and more obscure as new functions were added to the package. Therefore, recently [25] we have rewritten the code in Fortran 90, grouping subroutines and data structures into clearly defined modules, which can also be shared by the programs for post-processing UNRES results (WHAM, which executes the weighted-histogram analysis method [18] given the results of REMD/MREMD simulations in order to compute ensemble averages and probabilities of conformations and CLUSTER to group the conformations into families [22]). This approach enabled us to reduce the redundancy in the code to a great extent. Moreover, the code takes advantage of dynamic memory allocation, this making memory usage more efficient, especially for 2-,3- and 4-dimensional arrays. The source code in the UNRES package with Fortran 90 has the following directory structure:

- unres (main program modules, source code with all UNRES functionalities);
  - data (modules containing arrays declarations that correspond to COMMON block files in the UNRES package with Fortran 77);
- wham (weighted-histogram analysis method source code);
- cluster (cluster analysis source code);
- xdrfpdb (format conversion programs source code).

Because the postprocessing programs require a negligible fraction of compute time compared to that required for simulations with the main UNRES module, only the performance of UNRES will be discussed in this paper.

3. Results and Discussion

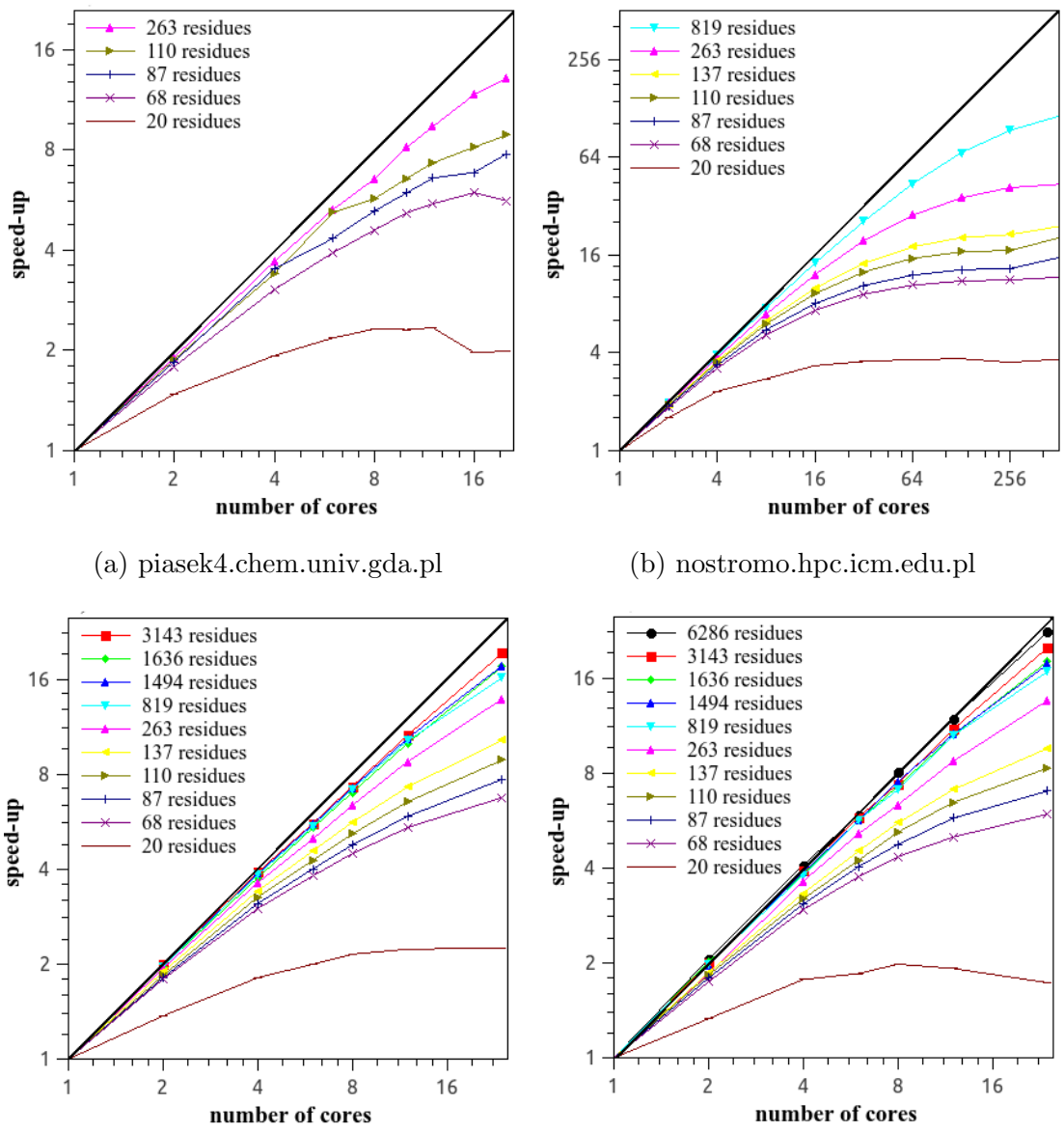
We tested the scalability of the Fortran 90 implementation of UNRES with proteins of different size and on various hardware platforms, which included our local Beowulf cluster composed of 20-core Dell nodes Intel Xeon CPU E5-2640 v4 2.40 GHz connected with Gigabit Ethernet, at Faculty of Chemistry, University of Gdańsk (piasek4.chem.univ.gda.pl), the 3214-node cluster, each node containing a 12-core Intel Xeon E5 v3 2.3 GHz processor (35,568 cores total), with Infiniband connection, at the Centre of Informatics - Tricity Academic Supercomputer & networK (CI TASK) in Gdańsk (tryton-ap.gv.kdm.task.gda.pl), the 1084-node Cray supercomputer, each node containing 2 Intel Xeon E5-2690 v3 12-core processors with Cray Aries connection (okeanos.hpc.icm.edu.pl), and the 1024-node IBM Blue Gene/Q supercomputer, each node containig 16 4-thread PowerPC A2 cores with 5D Torus connectivity (nostromo.hpc.icm.edu.pl), at the Interdisciplinary Centre for Mathematical and Computer Modelling, University of Warsaw.

The proteins used for benchmark runs were as follows: tryptophan cage (20 residues, single chain, PDB code: 1L2Y), double-stranded RNA binding domain (68 residues, single chain, PDB code: 1STU), M-domain from the GAG polyprotein (87 residues, single chain, PDB code: 1A6S), endonuclease NucB (110 residues, single chain, PDB code: 5OMT), BID domain of Bep6 (137 residues, single chain, PDB code: 4YK1), CDI complex (263 residues, two chains, PDB code: 5HKQ), UDP-glucose glycoprotein glucosyltransferase (1,494 residues, single chain, PDB code: 5NV4), STRA6 receptor (819 residues, two chains, and 1,636 residues, four chains; PDB code: 5SY1) and human proteasome 20S (3,143 residues, 14 chains, and 6286 residues, 28 chains; PDB code: 4R3O).

The speed-ups vs. the number of cores for single-trajectory MD runs are displayed in Fig. 2, while the corresponding efficiency plots are shown in Fig. 3. These data pertain to single-trajectory MD runs, i.e., they illustrate the performance of the software in the fine-grained parallel runs. As expected, the speed up and efficiency increase with the number of residues in a system. For the smallest protein, only up to 20-times speed up can be achieved, while for large proteins over 50 % efficiency can be achieved with up to 24 cores or up to 128 cores with Blue Gene/Q. Except for Blue Gene/Q, which has fast interconnect, all cores used for fine-grained parallelization were contained in a single node. Use of cores from more than one node for fine-grained parallelization resulted in performance deterioration. For example, use of 48 fine-grained processors (2 nodes) on the Cray resulted in the same or lower speed up than that with 24 processors (a single node).

The fraction of communication time split into synchronization (barrier) and data exchange is plotted for the 5HKQ protein in Fig. 4 as a function of the number of fine-grained tasks. It can be seen that communication constitutes less than 10 % of the total wall-clock time and that the main fraction of it is spent on synchronization, which is understandable because the data are exchanged between the cores of the same hardware unit.

To test the efficiency of the parallel implementation of the code in full-blown calculation, replica-exchange runs consisting of 2 to 48 trajectories, single-trajectory runs taken as reference, were carried out for the 1A6S protein (87 residues). Each trajectory consisted of 400,000 steps and exchange of replicas occurred every 10,000 steps. The non-setup time (defined as the wall-clock time after subtracting that for data input, initializing variables, and computing the inertia matrix) efficiencies as functions of number of trajectories (coarse-grained tasks) and different numbers of cores per trajectory (fine-grained tasks) are shown in Fig. 5. It can be seen from the figure that the non-setup time and efficiency effectively depend only on the number of fine-



(a) piasek4.chem.univ.gda.pl

(b) nostromo.hpc.icm.edu.pl

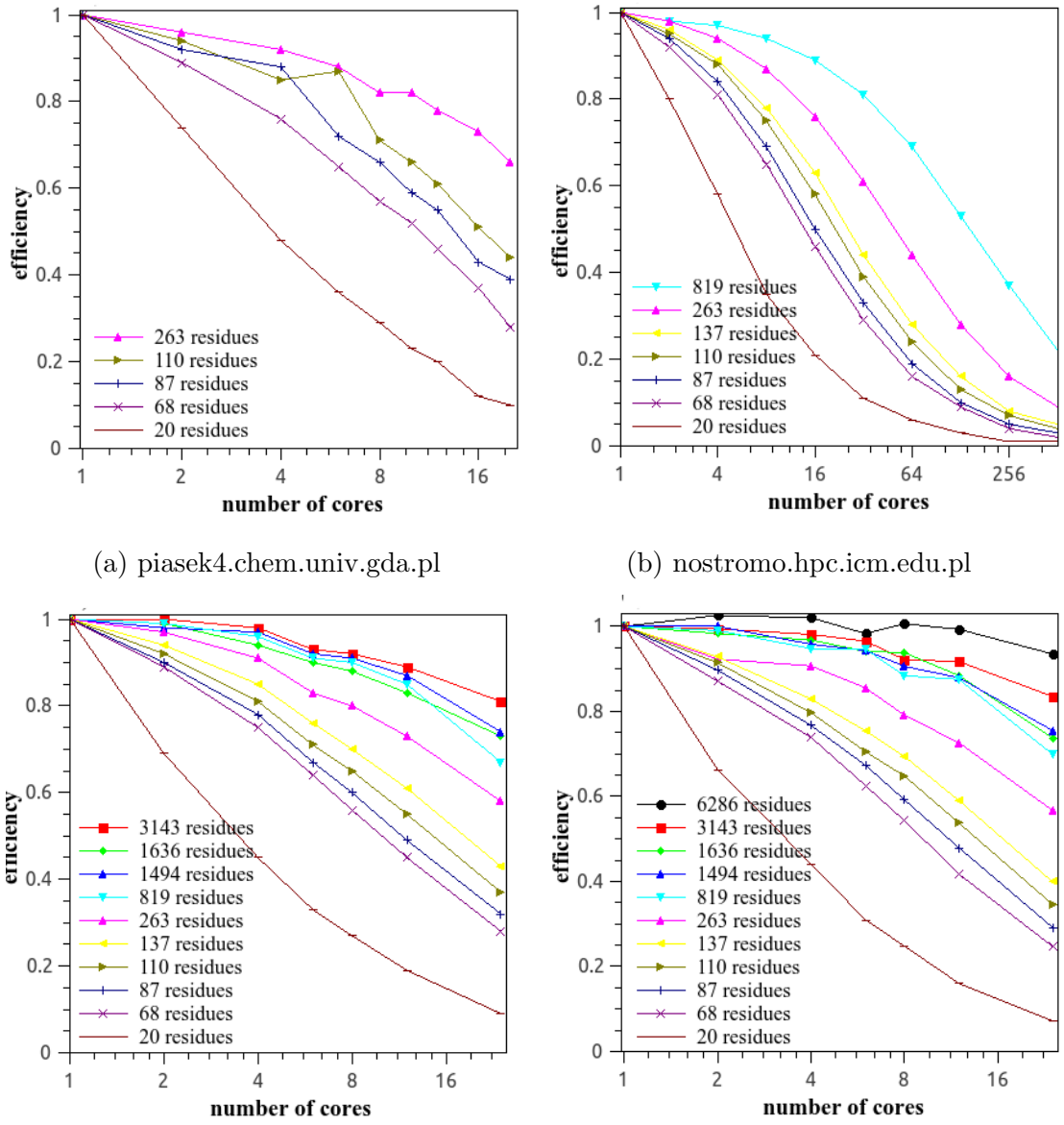
(c) okeanos.hpc.icm.edu.pl

(d) tryton-ap.gv.kdm.task.gda.pl

Figure 2. Plots of the speed-ups for canonical single-trajectory UNRES/MD runs obtained for proteins with various chain length vs. the number of processors on various hardware platforms. The logarithmic scale with base 2 is used on both axes

grained tasks per trajectory. This is because all MD trajectories are running at virtually the same compute speed, this providing good load balancing and data transmission occurs rarely (in the reported calculations only 40 times).

To compare UNRES with the all-atom simulations we have chosen one of the most popular molecular dynamics software packages, GROMACS [1, 3]. We have used the all-atom GROMACS 5.1.2 program run on tryton-ap.gv.kdm.task.gda.pl for 1L2Y, 1A6S, and 5OMT proteins, and 10,000 steps of molecular dynamics with amber03 force field [5]. The non-setup times for the above-mentioned proteins, with various numbers of processors are plotted in Fig. 6. As can be seen from this figure, the GROMACS computation time scales almost linearly with both number of processors and protein size. This confirmed that GROMACS is currently one of the most efficient and best load-balanced MD software (due, among other things, to the cut-



(a) piasek4.chem.univ.gda.pl

(b) nostromo.hpc.icm.edu.pl

(c) okeanos.hpc.icm.edu.pl

(d) tryton-ap.gv.kdm.task.gda.pl

Figure 3. Plots of the efficiencies for canonical single-trajectory UNRES/MD runs obtained for proteins with various numbers of residues vs. the number of processors on various hardware platforms. The logarithmic scale with base 2 is used on x-axis

off introduction on non-bonded interactions). As shown, UNRES provides a few times quicker simulations compared to GROMACS. Moreover, because the UNRES event-based time scale is at least 1,000 times longer than that of the all-atom approach, the effective speed-up with respect to GROMACS is even bigger. When the single-processor times are compared, the UNRES simulation time is 33 times lower for 1L2Y, 10 times lower for 1A6S, and 5 times lower for 5OMT. Summarizing, the simplified biopolymer-chain representation in UNRES results in several order of magnitude extension of the time- and size-scale of computations, compared to that of the all-atom simulations.

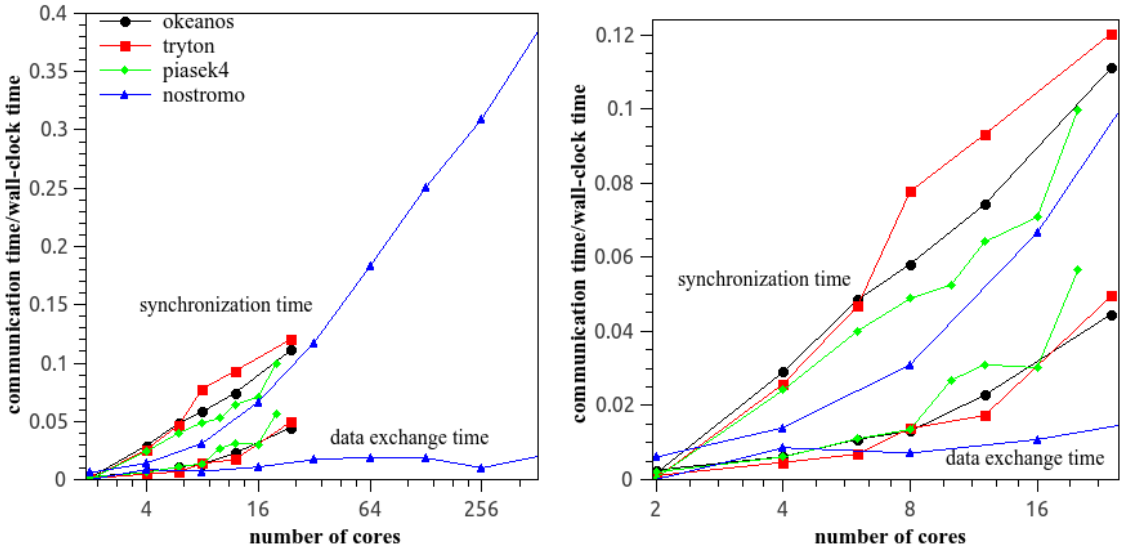
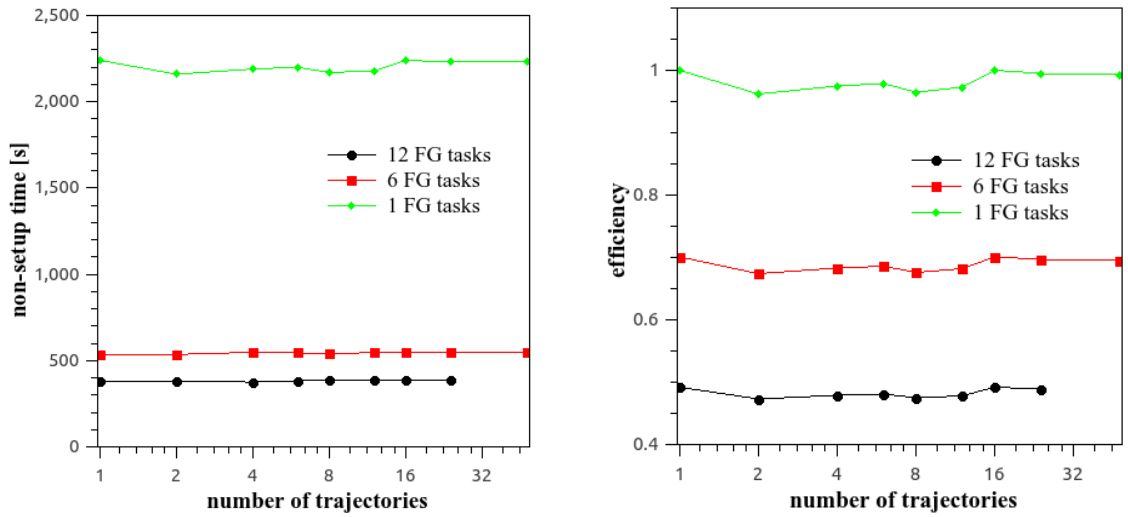


Figure 4. Plots of the fraction of the collective-communication time to the non-setup wall-clock time with a given number of cores in a single-trajectory UNRES/MD run for the 5HKQ protein on okeanos.hpc.icm.edu.pl, tryton-ap.gv.kdm.task.gda.pl, piasek4.chem.univ.gda.pl and nostromo.hpc.icm.edu.pl. The right panel (zoom) shows a higher-magnification view of the bottom left region in the left panel. The logarithmic scale with base 2 is used on x-axis



(a) non-setup times

(b) efficiencies

Figure 5. Plots of non-setup times and efficiencies in 40,0000 step MREMD simulations with replica exchange each 10,000 steps, numbers of trajectories from 1 (reference) to 48, and various numbers of fine-grain tasks per trajectory obtained for the 1A6S (87-residue) protein with tryton-ap.gv.kdm.task.gda.pl. The logarithmic scale with base 2 is used on x-axis. For a given point in the graph, the total number of cores used is obtained by multiplying the number of trajectories by the number of fine-grained tasks

Conclusion

In this work, we implemented the Fortran 90 version of the UNRES package for coarse-grained simulations of proteins and protein complexes on a number of parallel platforms and

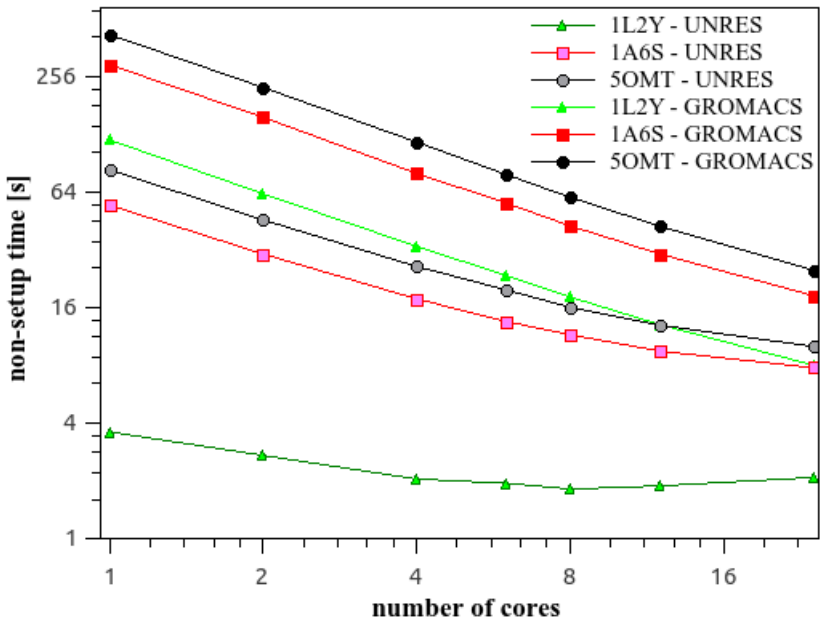


Figure 6. Plots of non-setup times in 10,000 steps MD simulations with UNRES and GROMACS, with various numbers of processors obtained for the 1L2Y (20-residue), 1A6S (87-residue) and 5OMT (110-residue) proteins with tryton-ap.gv.kdm.task.gda.pl. The logarithmic scale with base 2 is used on both axes

tested its parallel efficiency both in the fine-grained mode (energy and force parallelization in single-trajectory molecular dynamics calculation) and in the full-blown (coarse- and fine-grained) mode, in which multiple trajectories were run in replica exchange calculations with communication between the respective task groups every given number of steps, and parallelization of the computation of each trajectory. For proteins with chain length about 300 amino-acid residues or more over 50 % parallel efficiency is obtained. The efficiency does not deteriorate when replica-exchange calculations are carried out.

The parallel implementation of UNRES enables us to simulate the dynamics of quite large proteins in real time. For example, for the 263-residue 5HKQ protein, the time required for 10,000 time steps with 24 fine-grained tasks run on the Cray machine is 24 seconds, which translates to about 176 nanoseconds of molecular-dynamics time per day (24 hours of computations) with the 4.89 fs time step commonly set in UNRES runs. However, because UNRES simulation time is at least 1,000-fold extended with respect to that of all-atom simulations with explicit solvent [9], one day of UNRES simulations for this protein amounts to a 0.176 millisecond molecular dynamics. With this speed up, physics-based simulations of biologically important processes are now at hand.

More informations about UNRES force field and the source code are available (Description of UNRES program. http://www.unres.pl/unres, accessed: 2018-06-15; Downloads page. http://unres.pl/downloads, accessed: 2018-06-15; UNRES server version. http://unres-server.chem.ug.edu.pl./about, accessed: 2018-06-15).

Acknowledgements

Supported by grant BMN: 538-5300-B476-17 from the University of Gdańsk and grants UMO-2017/25/B/ST4/01026, UMO-2015/17/D/ST4/00509, and UMO-2-15/17/N/ST4/03935

from the National Science Centre of Poland (Narodowe Centrum Nauki). Computational resources were provided by (a) the Interdisciplinary Center of Mathematical and Computer Modeling (ICM) at the University of Warsaw under Grant No. GA71-23, (b) the Centre of Informatics - Tricity Academic Supercomputer & networK (CI TASK) in Gdańsk, and (c) our 796-processor Beowulf cluster at the Faculty of Chemistry, University of Gdańsk.

References

1. Abraham, M.J., Murtola, T., Schulz, R., Páll, S., Smith, J.C., Hess, B., Lindahl, E.: Gromacs: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. SoftwareX 1-2, 19–25 (2015), DOI: 10.1016/j.softx.2015.06.001

2. Berendsen, H.J.C., Postma, J.P.M., van Gunsteren, W.F., DiNola, A., Haak, J.R.: Molecular dynamics with coupling to an external bath. Journal of Chemical Physics 81(8), 3684–3690 (1984), DOI: 10.1063/1.448118

3. Berendsen, H.J.C., van der Spoel, D., van Drunen, R.: Gromacs: A message-passing parallel molecular dynamics implementation. Computer Physics Communications 91(1), 43–56 (1995), DOI: 10.1016/0010-4655(95)00042-E

4. Czaplewski, C., Kalinowski, S., Liwo, A., Scheraga, H.A.: Application of multiplexing replica exchange molecular dynamics method to the UNRES force field: Tests with α and $\alpha + \beta$ proteins. Journal of Chemical Theory and Computation 5(3), 627–640 (2009), DOI: 10.1021/ct800397z

5. Duan, Y., Wu, C., Chowdhury, S., Lee, M.C., Xiong, G., Zhang, W., Yang, R., Cieplak, P., Luo, R., Lee, T., Caldwell, J., Wang, J., Kollman, P.: A point-charge force field for molecular mechanics simulations of proteins based on condensed-phase quantum mechanical calculations. Journal of Computational Chemistry 24(16), 1999–2012 (2003), DOI: 10.1002/jcc.10349

6. Hansmann, U.H.E., Okamoto, Y.: Comparative study of multicanonical and simulated annealing algorithms in the protein folding problem. Physica A 212(3-4), 415–437 (1994), DOI: 10.1016/0378-4371(94)90342-5

7. He, Y., Maciejczyk, M., Ołdziej, S., Scheraga, H.A., Liwo, A.: Mean-field interactions between nucleic-acid-base dipoles can drive the formation of a double helix. Physical Review Letters 110(9), 098101 (2013), DOI: 10.1103/PhysRevLett.110.098101

8. He, Y., Mozolewska, M.A., Krupa, P., Sieradzan, A.K., Wirecki, T., Liwo, A., Kachlishvili, K., Rackovsky, S., Jagieła, D., Ślusarz, R., Czaplewski, C., Ołdziej, S., Scheraga, H.A.: Lessons from application of the UNRES force field to predictions of structures of CASP10 targets. Proceedings of the National Academy of Sciences U. S. A. 110(37), 14936–14941 (2013), DOI: 10.1073/pnas.1313316110

9. Khalili, M., Liwo, A., Jagielska, A., Scheraga, H.A.: Molecular dynamics with the united-residue model of polypeptide chains. II. Langevin and Berendsen-bath dynamics and tests on model α-helical systems. Journal of Physical Chemistry B 109(28), 13798–13810 (2005), DOI: 10.1021/jp058007w

10. Khalili, M., Liwo, A., Rakowski, F., Grochowski, P., Scheraga, H.A.: Molecular dynamics with the united-residue model of polypeptide chains. I. Lagrange equations of motion and tests of numerical stability in the microcanonical mode. Journal of Physical Chemistry B 109(28), 13785–13797 (2005), DOI: 10.1021/jp058008o

11. Kleinerman, D.S., Czaplewski, C., Liwo, A., Scheraga, H.A.: Implementations of Nosé – Hoover and Nosé –Poincaré thermostats in mesoscopic dynamic simulations with the united-residue model of a polypeptide chain. Journal of Chemical Physics 128(24), 245103 (2008), DOI: 10.1063/1.2943146

12. Kmiecik, S., Gront, D., Kolinski, M., Wieteska, L., Dawid, A.E., Kolinski, A.: Coarse-grained protein models and their applications. Chemical Reviews 116(14), 7898–7936 (2016), DOI: 10.1021/acs.chemrev.6b00163

13. Kolinski, A., Skolnick, J.: Reduced models of proteins and their applications. Polymer 45(2), 511–524 (2004), DOI: 10.1016/j.polymer.2003.10.064

14. Kolinski, A., Skolnick, J.: Assembly of protein structure from sparse experimental data: An efficient Monte Carlo model. Proteins: Structure, Function, and Bioinformatics 32(4), 475–494 (1998), DOI: 10.1002/(SICI)1097-0134(19980901)32:43.0.CO;2-F

15. Krupa, P., Mozolewska, M.A., Wiśniewska, M., Yin, Y., He, Y., Sieradzan, A.K., Ganzynkowicz, R., Lipska, A.G., Karczyńska, A., Ślusarz, M., Ślusarz, R., Giełdoń, A., Czaplewski, C., Jagieła, D., Zaborowski, B., Scheraga, H.A., Liwo, A.: Performance of protein-structure predictions with the physics-based UNRES force field in CASP11. Bioinformatics 32(21), 3270–3278 (2016), DOI: 10.1093/bioinformatics/btw404

16. Krupa, P., Hałabis, A., Żmudzińska, W., Ołdziej, S., Scheraga, H.A., Liwo, A.: Maximum likelihood calibration of the UNRES force field for simulation of protein structure and dynamics. Journal of Chemical Information and Modeling 57(9), 2364–2377 (2017), DOI: 10.1021/acs.jcim.7b00254

17. Kubo, R.: Generalized cumulant expansion method. Journal of the Physical Society Japan 17(7), 1100–1120 (1962), DOI: 10.1143/JPSJ.17.1100

18. Kumar, S., Rosenberg, J.M., Bouzida, D., Swendsen, R.H., Kollman, P.A.: The weighted histogram analysis method for free-energy calculations on biomolecules. I. The Method. Journal of Computational Chemistry 13(8), 1011–1021 (1992), DOI: 10.1002/jcc.540130812

19. Liwo, A., Baranowski, M., Czaplewski, C., Gołaś, E., He, Y., Jagieła, D., Krupa, P., Maciejczyk, M., Makowski, M., Mozolewska, M.A., Niadzvedtski, A., Ołdziej, S., Scheraga, H.A., Sieradzan, A.K., Ślusarz, R., Wirecki, T., Yin, Y., Zaborowski, B.: A unified coarse-grained model of biological macromolecules based on mean-field multipole-multipole interactions. Journal of Molecular Modeling 20(8), 2306 (2014), DOI: 10.1007/s00894-014-2306-5

20. Liwo, A., Czaplewski, C., Ołdziej, S., Rojas, A.V., Kaźmierkiewicz, R., Makowski, M., Murarka, R.K., Scheraga, H.A.: Simulation of protein structure and dynamics with the coarse-grained UNRES force field. In: Voth, G. (ed.) Coarse-Graining of Condensed Phase and Biomolecular Systems, chap. 8, pp. 1391–1411. CRC Press (2008), DOI: 10.1201/9781420059564.ch8

21. Liwo, A., Czaplewski, C., Pillardy, J., Scheraga, H.A.: Cumulant-based expressions for the multibody terms for the correlation between local and electrostatic interactions in the united-residue force field. Journal of Chemical Physics 115(5), 2323–2347 (2001), DOI: 10.1063/1.1383989

22. Liwo, A., Khalili, M., Czaplewski, C., Kalinowski, S., Ołdziej, S., Wachucik, K., Scheraga, H.A.: Modification and optimization of the united-residue (UNRES) potential energy function for canonical simulations. I. Temperature dependence of the effective energy function and tests of the optimization method with single training proteins. Journal of Physical Chemistry B 111(1), 260–285 (2007), DOI: 10.1021/jp065380a

23. Liwo, A., Ołdziej, S., Czaplewski, C., Kleinerman, D.S., Blood, P., Scheraga, H.A.: Implementation of molecular dynamics and its extensions with the coarse-grained UNRES force field on massively parallel systems: Towards millisecond-scale simulations of protein structure, dynamics, and thermodynamics. Journal of Chemical Theory and Computation 6(3), 583–595 (2010), DOI: 10.1021/ct9004068

24. Lopez, C.A., Rzepiela, A., de Vries, A.H., Dijkhuizen, L., Hunenberger, P.H., Marrink, S.J.: Martini coarse-grained force field: Extension to carbohydrates. Journal of Chemical Theory and Computation 5(12), 3195–3210 (2009), DOI: 10.1021/ct900313w

25. Lubecka, E.A., Liwo, A.: New UNRES force field package with FORTRAN 90. TASK Quarterly 20(4), 399–407 (2016), DOI: 10.17466/tq2016/20.4/n

26. Markutsya, S., Devarajan, A., Baluyut, J., Windus, T.L., Gordon, M.S., Lamm, M.H.: Evaluation of coarse-grained mapping schemes for polysaccharide chains in cellulose. Journal of Chemical Physics 138(21), 214108 (2013), DOI: 10.1063/1.4808025

27. Marrink, S.J., Tieleman, D.P.: Perspective on the Martini model. Chemical Society Reviews 42(16), 6801–6822 (2013), DOI: 10.1039/C3CS60093A

28. Molinero, V., Goddard III, W.A.: M3b: a coarse-grain force field for molecular simulations of malto-oligosaccharides and their water mixtures. Journal of Physical Chemistry B 108(4), 1414–1427 (2004), DOI: 10.1021/jp0354752

29. Monticelli, L., Kandasamy, S.K., Periole, X., Larson, R.G., Tieleman, D.P., Marrink, S.J.: The Martini coarse-grained force field: Extension to proteins. Journal of Chemical Theory and Computation 4(5), 819–834 (2008), DOI: 10.1021/ct700324x

30. Mozolewska, M.A., Krupa, P., Scheraga, H.A., Liwo, A.: Molecular modeling of the binding modes of the iron-sulfur protein to the Jac1 co-chaperone from Saccharomyces cerevisiae by all-atom and coarse-grained approaches. Proteins: Structure, Function, and Bioinformatics 83(8), 1414–1426 (2015), DOI: 10.1002/prot.24824

31. Rakowski, F., Grochowski, P., Lesyng, B., Liwo, A., Scheraga, H.A.: Implementation of a symplectic multiple-time-step molecular dynamics algorithm, based on the united-residue mesoscopic potential energy function. Journal of Chemical Physics 125(20), 204107 (2006), DOI: 10.1063/1.2399526

32. Rhee, Y.M., Pande, V.S.: Multiplexed-replica exchange molecular dynamics method for protein folding simulation. Biophysical Journal 84(2), 775–786 (2003), DOI: 10.1016/S0006-3495(03)74897-8

33. Rojas, A., Liwo, A., Browne, D., Scheraga, H.A.: Mechanism of fiber assembly; treatment of Aβ-peptide peptide aggregation with a coarse-grained united-residue force field. Journal of Molecular Biology 404(3), 537–552 (2010), DOI: 10.1016/j.jmb.2010.09.057

34. Samsonov, S.A., Bichmann, L., Pisabarro, M.T.: Coarse-grained model of glycosoaminoglycans. Journal of Chemical Information and Modeling 55(1), 114–124 (2015), DOI: 10.1021/ci500669w

35. Sieradzan, A.K., Krupa, P., Wales, D.J.: What makes telomeres unique? Journal of Physical Chemistry B 121(10), 2207–2219 (2016), DOI: 10.1021/acs.jpcb.6b08780

36. Sieradzan, A.K., Makowski, M., Augustynowicz, A., Liwo, A.: A general method for the derivation of the functional forms of the effective energy terms in coarse-grained energy functions of polymers. I. Backbone potentials of coarse-grained polypeptide chains. Journal of Chemical Physics 146(12), 124106 (2017), DOI: 10.1063/1.4978680

37. Takada, S., Kanada, R., Tan, C., Terakawa, T., Li, W., Kenzaki, H.: Modeling structural dynamics of biomolecular complexes by coarse-grained molecular simulations. Accounts of Chemical Research 48(12), 3026–3035 (2015), DOI: 10.1021/acs.accounts.5b00338

38. Zhou, R., Maisuradze, G.G., Suñol, D., Todorovski, T., Macias, M.J., Xiao, Y., Scheraga, H.A., Czaplewski, C., Liwo, A.: Folding kinetics of WW domains with the united residue force field for bridging microscopic motions and experimental measurements. Proceedings of the National Academy of Sciences U.S.A. 111(51), 18243–18248 (2014), DOI: 10.1073/pnas.1420914111

DOI: 10.14529/jsfi180207

3D Problems of Rotating Detonation Wave in a Ramjet Engine Modeled on a Supercomputer

***Valeriy F. Nikitin*[1,2], *Yurii G. Filippov*[1], *Lyuben I. Stamov*[1,2], *Elena V. Mikhalchenko*[1,2]**

A rotating detonation engine (RDE) combustion chamber was modeled in the work numerically using 3D geometry. The RDE is a new type of engines capable to create higher thrust than the traditional ones, which are based on the combustible mixture deflagration process. In the numerical experiment, different scenarios of the engine performance were obtained. The calculations were made at a compact supercomputer APK-5 with a peak performance of 5.5 Tera Flops.

Keywords: Mathematical Modeling, Detonation, Deflagration, RDE, Ramjet.

Introduction

The optimization of modern engines based on the traditional design is now close to its technological limit. The engines performance may be increased only with the use of radically new technical solutions [1]. One of those solutions is the development of detonation engines; we deal with an engine with a rotating detonation [2]. The rotating detonation engine is being investigated extensively during the last decades; see works [3, 4].

The numerical modeling of the processes in a combustion chamber is an important stage in the investigation of its design and further usage. The mathematical model includes equations for multicomponent gas mixture, it considers chemical reactions and turbulent transport of mass, momentum and energy. In order to resolve such events as detonation cells development and thin wave con-figurations, one should have a rather thin computational mesh, and it contributes to the complexity. Therefore, the problem should be solved using big computational resources, high precision approximation schemes, and effective parallelization methods: OpenMP, MPI, CUDA, etc. In order to solve such problems, the authors have created a parallel code with the AUSM [5], the MUSCL [6] methods incorporated. The current research uses the OpenMP version of the computer program.

1. Mathematical Model

The mathematical model contains the governing equations (differential and algebraic), boundary and initial conditions. The details of the numerical realization together with the computational mesh design and the placement of variables on the mesh, is a subject of the numerical model.

1.1. The Balance Equations

To model a multicomponent gas mixture, we use the following system of balance equations:

$$\frac{\partial \rho_k}{\partial t} + \frac{\partial}{\partial x_j}\left(\rho_k u_j\right) - \frac{\partial J_{kj}}{\partial x_j} = \dot{\omega}_k, \tag{1}$$

[1]Lomonosov Moscow State University, Moscow, Russia
[2]SRISA RAS, Moscow, Russia

$$\frac{\partial \rho u_i}{\partial t} + \frac{\partial}{\partial x_j}(\rho u_i u_j) + \frac{\partial p}{\partial x_i} - \frac{\partial \tau_{ij}}{\partial x_j} = 0, \tag{2}$$

$$\frac{\partial E_T}{\partial t} + \frac{\partial}{\partial x_j}((E_T + p)u_j) - \frac{\partial}{\partial x_j}(J_{Tj} + u_i \tau_{ij}) = \dot{Q}. \tag{3}$$

Here ρ_k is partial density of a species k, J_{kj} are vector components of the species k diffusion flux, $\dot{\omega}_k$ is the intensity of the species k origination in chemical reactions, ρ is the gas mixture density, u_j are vector components of gas velocity, p is the mixture pressure, τ_{ij} are the tensor components of viscous and turbulent stresses, E_T is total energy of the gas volume unit consisting of thermal, chemical, kinetic and turbulent energy, J_{Tj} are vector components of the energy diffusion flux, $\dot{Q}$ is the heat flux from an external source.

1.2. Additional Algebraic Relations

Density of mixture is a sum of partial densities, and it is useful to introduce two types of concentrations: mass shares of species, and molar densities:

$$\rho = \sum_{k=1}^{N_C} \rho_k, \quad Y_k = \frac{\rho_k}{\rho}, \quad X_k = \frac{\rho_k}{W_k}. \tag{4}$$

Here: Y_k is the mass share of the species k, X_k is the molar density (in terms of many works on chemical kinetics it is named a molar concentration), W_k is the molar mass of a species.

The pressure p is defined as the spherical part of the stresses tensor, with the opposite sign. It is a sum of thermal pressure $\hat{p}$ of perfect gases mixture, and an additive corresponding to turbulent pulsations, which is modeled by means of the turbulent energy per mass unit K:

$$p = \hat{p} + \frac{2}{3}\rho K, \quad \hat{p} = R_G T \sum_{k=1|}^{N_C} X_k. \tag{5}$$

The total energy of a volume unit is the following sum:

$$E_T = E + \rho\frac{u^2}{2} + \rho K, \quad u^2 = u_j u_j. \tag{6}$$

The total energy E_T is therefore the sum of internal (thermal and chemical) energy, kinetic, and turbulent energy. The internal energy of a volume unit is modeled as follows:

$$E = \sum_{k=1}^{N_C} X_k E_k = \sum_{k=1}^{N_C} X_k \left(\hat{H}_k(T) - 1\right). \tag{7}$$

Here, E_k is an internal energy of a species mole, $\hat{H}_k$ is dimensionless enthalpy of a species containing the formation enthalpy at a reference temperature T_{ref} (chemical energy). Those functions are the basis of the species thermodynamic description; for many species, they are either tabulated, or approximated with polynomials. In the current research, they are taken from [7]; their format (two temperature interval) is described in [8] and [9]. We joined those temperature intervals into a single, and obtained the polynomial coefficients using the linear regression analysis based on the least squares technique.

1.3. Chemical Kinetics

In the current research, the chemical sources $\dot{\omega}_k$ depend on temperature T and the set of molar densities X_k; the sum of those sources is zero due to the law of mass conservation in chemical reactions:

$$\dot{\omega}_k = W_k \hat{\omega}_k(T, X_j), \quad \sum_{k=1}^{N_C} \dot{\omega}_k = 0. \tag{8}$$

Here $\hat{\omega}_k$ is the intensity of a species mole origination in a volume unit.

Some more strict laws for chemical interactions exist, e.g. the conservation of mass for each element. Those laws are considered in the chemical mechanism, and sometimes can reduce the computation effort and increase the precisity. A general form for the chemical sources is usually complex, and it consists of many nonlinear terms; a typical expression is as follows:

$$\hat{\omega}_k = \sum_{r=1}^{N_R} \nu_{rk} \omega_r, \quad \omega_r = M_r(X_j) \left[k_{Fr}(M_r, T) \prod_{i=1}^{N_C} X_i^{\alpha_{ri}} - k_{Br}(M_r, T) \prod_{i=1}^{N_C} X_i^{\beta_{ri}} \right]. \tag{9}$$

Here, ω_r is the reaction r speed (intensity), ν_{rk} is an algebraic stoichiometric coefficient for a species k in the reaction r, this coefficient is positive for the species being produced, and negative for those being consumed. M_r is the influence coefficient of non-changing components, which is equal to unity in the lack of such an influence, k_{Fr} is the direct reaction speed coefficient; it usually depend on temperature only, but for some falloff reactions it depends on M_r, k_{Br} is the reverse reaction speed, α_{rk} are degrees for species in a direct reaction (usually but not always they are non-zero for the input species), β_{rk} are the degrees for the reverse reactions.

In case of elementary reactions, the degrees for species in the expression (9) are the same with the input and output stoichiometric coefficients. The reverse reaction speed coefficients were calculated to provide the reach of the dynamical chemical equilibrium in the case of zero external fluxes and constant density and internal energy; the following expression was used:

$$k_{Br} = k_{Fr} \exp \left[\sum_{k=1}^{N_C} \nu_{rk} \left(\dot{H}_k(T) - \dot{S}_k(T) - 1 \right) \right] \left(\frac{R_G T}{p_{ref}} \right)^{\nu_r}, \quad \nu_r = \sum_{k=1}^{N_C} \nu_{rk}. \tag{10}$$

The entropy dimensionless coefficients $\hat{S}_k$ are constructed like $\hat{H}_k$ from data in [7] using formulae from [9].

For each direct reaction, its coefficient is modeled with an extended Arrhenius formula:

$$k_{Fr} = A_r T^{B_r} \exp \left(-\frac{\Theta_r}{T} \right), \tag{11}$$

where the reaction activation temperature Θ_r is derived from the activation energy as $\Theta_r = E_{ar}/R_G$.

1.4. Turbulence Model and Transport

The current research uses the Wilcox ka-omega model [10]:

$$\frac{\partial \rho K}{\partial t} + \frac{\partial}{\partial x_j} (\rho K u_j) - \frac{\partial}{\partial x_j} \left((\mu + \sigma^* \mu_T) \frac{\partial K}{\partial x_j} \right) = \tau_{ij}^T \frac{\partial u_i}{\partial x_j} - \beta^* \rho K \omega, \tag{12}$$

$$\frac{\partial \rho \omega}{\partial t} + \frac{\partial}{\partial x_j} (\rho \omega u_j) - \frac{\partial}{\partial x_j} \left((\mu + \sigma \mu_T) \frac{\partial \omega}{\partial x_j} \right) = \alpha \frac{\omega}{K} \tau_{ij}^T \frac{\partial u_i}{\partial x_j} - \beta \rho \omega^2, \tag{13}$$

$$\mu_T = \rho \frac{K}{\omega}. \tag{14}$$

Here K is the kinetic energy of pulsations per mass unit, μ is the molecular viscosity of the gas mixture, μ_T is the turbulent (eddy) viscosity, τ_{ij}^T is the turbulent part of the stresses tensor, ω is the intensity of the turbulent energy decay (dissipation) far away from walls and in lack of the turbulent energy input sources, parameters

$$\alpha = \frac{5}{9}, \quad \sigma = \sigma^* = \frac{1}{2}, \quad \beta = \frac{3}{40}, \quad \beta^* = \frac{9}{100}$$

are constant parameters of the standard Wilcox model.

The turbulent part of the stresses tensor deviator is:

$$\tau_{ij}^T = \mu_T \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{2}{3} \frac{\partial u_k}{\partial x_k} \delta_{ij} \right) - \frac{2}{3} \rho K \delta_{ij}, \tag{15}$$

where δ_{ij} is the Kronecker symbol.

The molecular viscosity of the gas mixture is calculated using the pure species viscosity $\mu_k(T)$ and molar densities X_k as:

$$\mu = \sum_{k=1}^{N_C} \frac{\mu_k X_k}{\sum\limits_{j=1}^{N_C} \phi k j X_j}. \tag{16}$$

The effective mixture viscosity is lower than the weighted average due to the binary influence coefficients ϕ_{kj}. A simple method to compute them was published in [11]:

$$\phi_{kj} = \frac{1}{\sqrt{8}} \left(1 + \frac{W_k}{W_j} \right)^{-\frac{1}{2}} \left[1 + \left(\frac{\mu_k}{\mu_j} \right)^{\frac{1}{2}} \left(\frac{W_j}{W_k} \right)^{\frac{1}{4}} \right]^2. \tag{17}$$

A molecular viscosity is calculated using physical molecular constants taken from the database [12], and the method of computation was taken from [11].

To calculate the fluxes of mass and energy J_{kj} and J_{Tj}, and the stresses tensor deviator τ_{ij}, we used a model taking into account the turbulent transport calculated by means of the Wilcox model [10]. In most cases, the turbulent transport supersedes the molecular, and the last is made using simplified technique using constant Prandtl Pr and Schmidt Sc numbers hypothesis [13]:

$$J_{kj} = \left(\frac{\mu}{\mathrm{Sc}} + \frac{\mu_T}{\mathrm{Sc}_T} \right) \frac{\partial Y_k}{\partial X_j}, \tag{18}$$

$$J_{Tj} = \left(\frac{\mu}{\mathrm{Pr}} + \frac{\mu_T}{\mathrm{Pr}_T} \right) \frac{\partial h}{\partial X_j} + (\mu + \mu_T) \frac{\partial K}{\partial x_j}, \tag{19}$$

$$\tau_{ij} = (\mu + \mu_T) \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{2}{3} \frac{\partial u_k}{\partial x_k} \delta_{ij} \right) - \frac{2}{3} \rho K \delta_{ij}. \tag{20}$$

Here, Sc_T, Pr_T are constant turbulent Schmidt and Prandtl numbers, h is the enthalpy per mass unit; the last is computed as follows:

$$h = \frac{R_G T}{\rho} \sum_{k=1}^{N_C} X_k \hat{H}_k(T). \tag{21}$$

1.5. The Species List and the Kinetic Mechanism

Hydrogen, oxygen and nitrogen were the initial and the inflow mixture components. In the process of combustion, besides the main product, water vapor, numerous products (radicals) are originated; at high temperature they still persist in the mixture, and at lower temperature they decay. We used the following set of species:

$$\{H_2O, OH, H, O, HO_2, H_2O_2; H_2, O_2, N_2\}.$$

The research used a kinetic mechanism described in the Maas & Pope work [14] (1992). The mechanism consisted of 20 reversible elementary reactions.

2. Results

A model combustion chamber of a ramjet detonation engine was treated as a test. Geometrically, it is a hollow cylinder with the cylindrical internal body, which ends up as a cone. The fuel flow into the chamber through numerous injectors (premixed rich composition $[H_2] : [O_2] = 3 : 1$, stagnant pressure 10 bar, stagnant temperature 258 K, Mach number at each orifice 1). At the initial instance, the chamber is filled with air at 1 bar pressure and 300 K temperature. The ignition is made by means of an external energy source into a small spherical portion: $r_{ign} = 2.5$ mm after a delay of 10 μs from the initial instance, and during $t_{ign} = 1$ μs, and with power per volume as high as $Q = 20$ kW/mm^3.

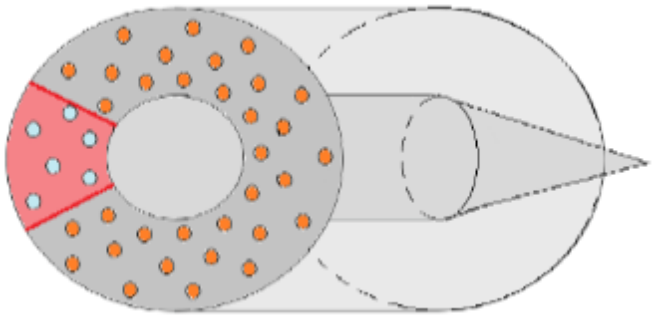


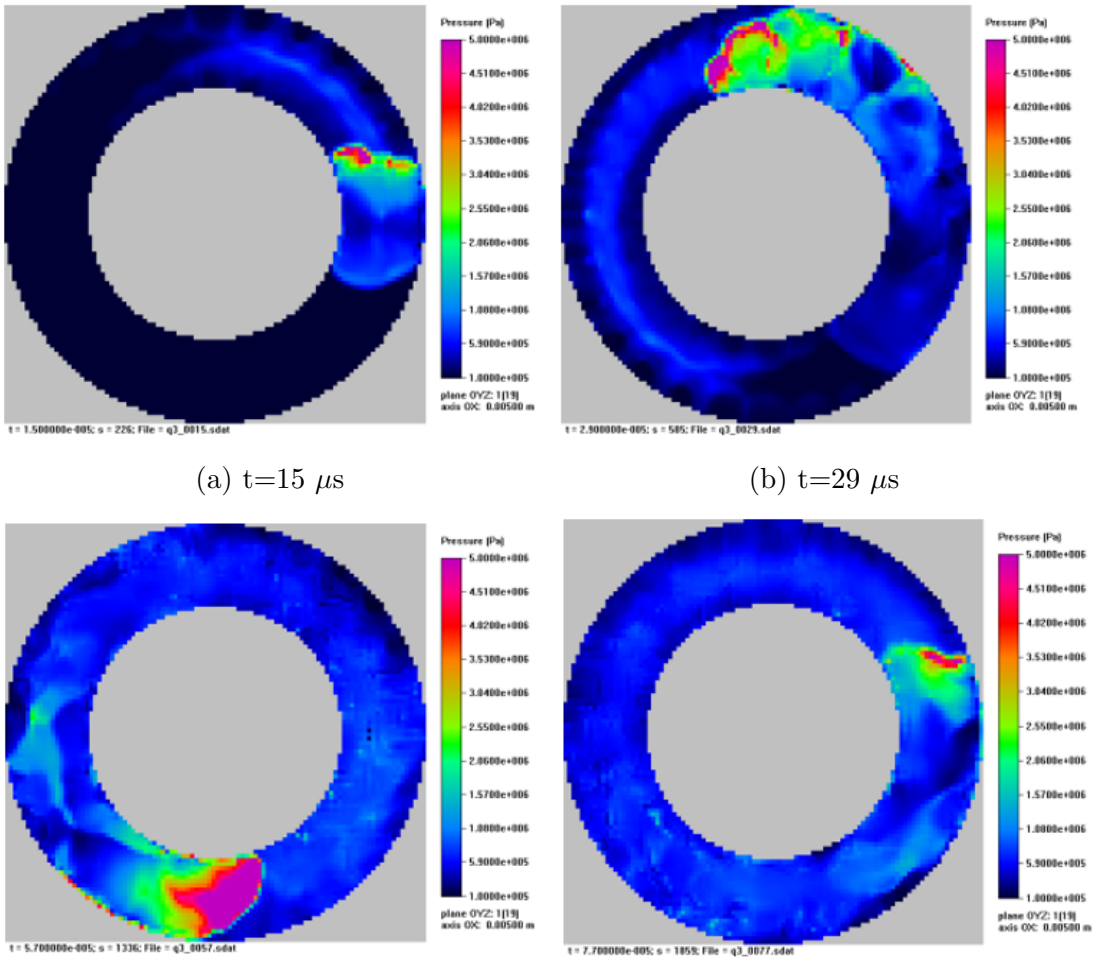
Figure 1. The combustion chamber geometry

The work area length was $L = 10$ cm, the maximal radius of the work area $R = 2.5$ cm, the inner body radius $R_b = 1.5$ cm, the inner body length without the terminating cone was $L_b = 3$ cm, the terminating cone length $L = 3$ cm, the number of injectors $N_r = 72$, their orifices radii were $r = 0.2$ cm. The test problem was solved on a structured cubic cells mesh made of $\approx 1.3 \cdot 10^6$ cells; the size of a cell was 0.5 mm.

In order to force detonation wave propagation in one direction, initially all the orifices were closed, then they are open in turn depending on their angular co-ordinate so that the whole ring is open by 30 μs from the beginning.

The Fig. 2 shows pressure in the ramjet engine combustion chamber for different times shown in a cross section OYZ at the distance of 0.5 cm from the injectors end.

One can see from the Fig. 2 that the detonation wave is formed after the ignition at 10 μs propagating in one direction (counter-clockwise) due to specific filling of the combustible mixture via the orifices (Fig. 2a). During the first circulation, its strength increases due to the increasing amount of fresh mixture ahead of it (Fig. 2b – Fig. 2c). After the detonation wave rotates once,



(a) t=15 μs (b) t=29 μs

(c) t=57 μs (d) t=77 μs

Figure 2. Pressure in the cross section

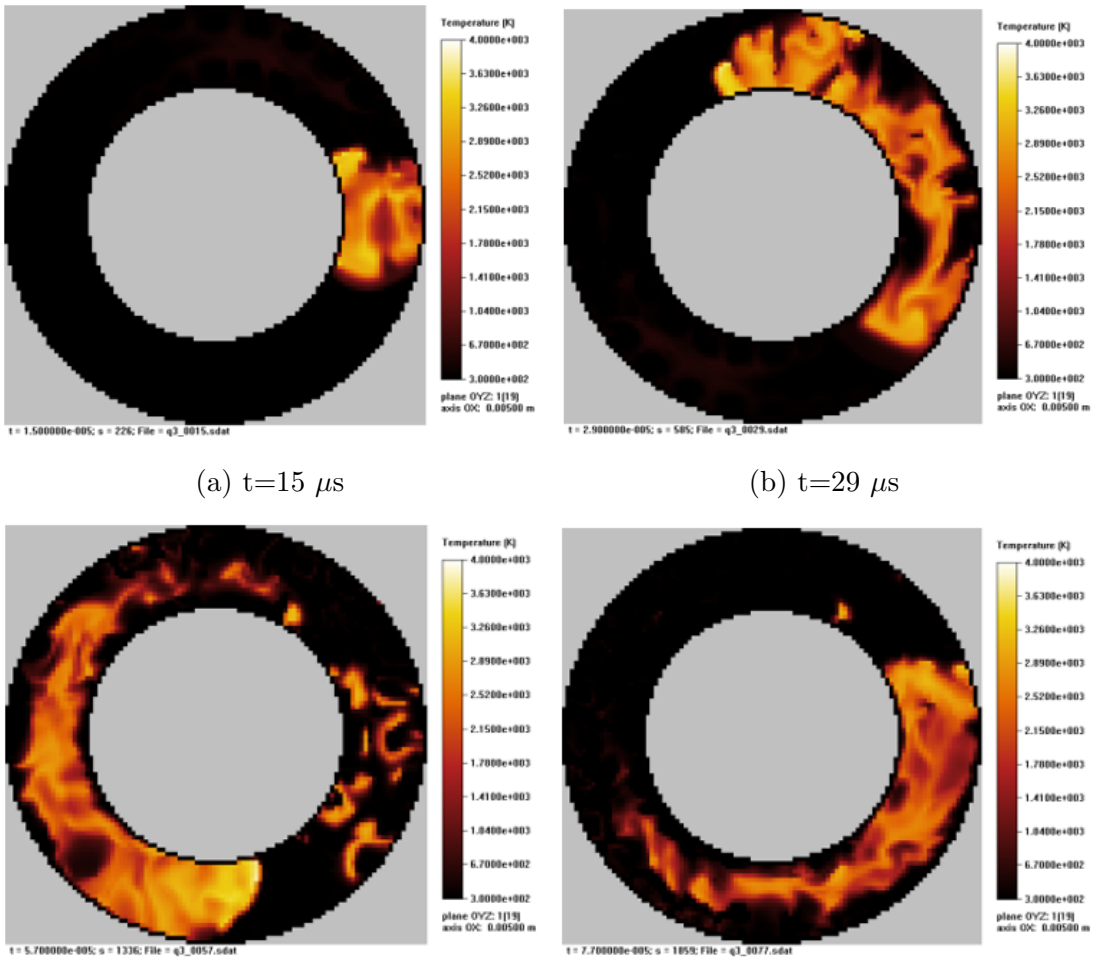
it goes weaker due to lower amount of combustible mixture ahead (Fig. 2d). Still it propagates without extinguishing.

The Fig. 3 shows temperature in this cross section at the same times.

The Fig. 3 shows that a portion of high temperature propagates counter-clockwise as the detonation wave, but the high temperature goes also clock-wise with lower velocity (Fig. 3a – Fig. 3b). Then, the temperature far behind the detonation wave decreases gradually due to the displacement of hot products of combustion with a fresh combustible mixture. This process is highly turbulent, and portions of hot products of combustion still remain in this cross section (Fig. 3c). When the detonation wave rotates once, the structure of the temperature field behind the wave in this cross section becomes more regular, but there still exist hot spots ahead of the wave; however, it propagates stably in the fresh mixture (Fig. 3d).

The Fig. 4 shows 3D distribution of pressure (Fig. 4a) and temperature (Fig. 4b), at $t = 57$ μs from the beginning of the process.

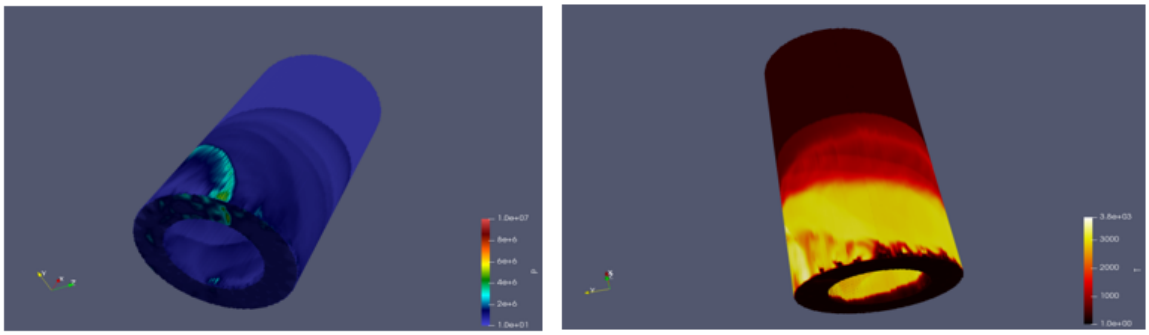
One can see the spiral-shaped pressure profile (Fig. 4a); the lower portion of it in the vicinity of the orifices is a detonation wave, higher, it transfers to a shock wave. Other shock waves of smaller amplitude could be seen, generated by the initial gas mixture inflow. The temperature field (Fig. 4b) also demonstrates this spiral; one can see that the portion of fresh (cold) mixture ahead of the detonation wave is rather thin; the hot products of combustion originated at the



(a) t=15 μs (b) t=29 μs

(c) t=57 μs (d) t=77 μs

Figure 3. Temperature in the cross section



(a) 3D view of pressure at $t = 57$ μs (b) temperature at $t = 57$ μs

Figure 4. 3D view of pressure and temperature at $t = 57$ μs

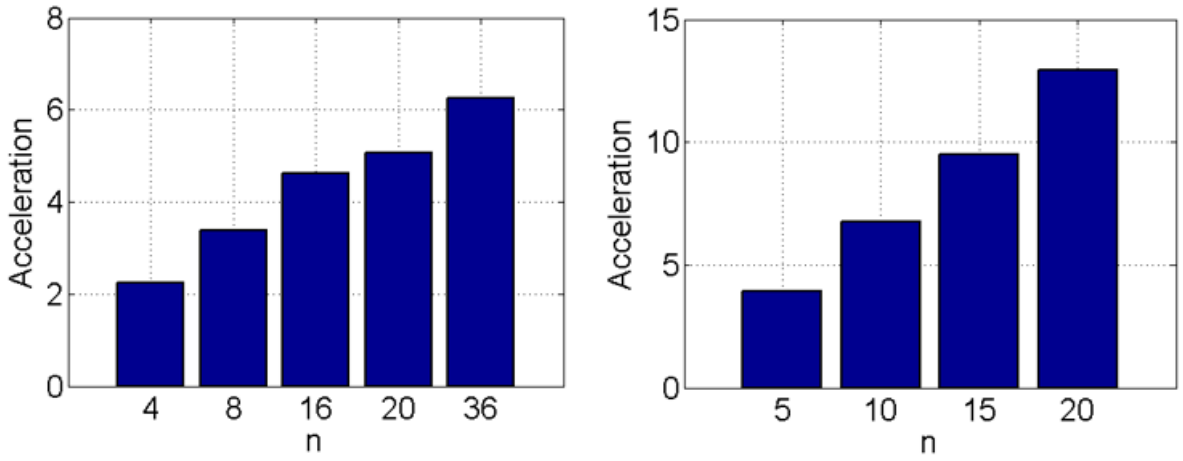
first circulation of the detonation wave are depicted in yellow there, and they occupy the lower 1/3 of the combustion chamber by this time; the gas portion depicted in red is air heated by the primary shock generated by the inflow of combustible gases. The upper 1/3 of the combustion chamber is filled with cold undisturbed air: the shock wave has not reached it by $t = 57$ μs from the beginning of the process.

For the given parameter set, we obtained a detonation wave rotating around the bottom of the combustion chamber. In the beginning, a stable detonation wave originates, at first circulation its strength increases due to the increasing amount of fresh micture ahead of it. Then, it decreases due to much lower fresh mixture because the previous mixture was consumed one circulation before, and the new portion of it is much fewer than the initial. In some time after 70 μs the primary detonation wavet splits into two, even 3 waves, which then join into a single detonation wave. After that, in some time the process repeats: lateral waves reflections from walls contribute to it. As the result, we have obtained a galloping regime of rotating detonation.

2.1. The Calculating System

The calculations were performed at a computational system with two INTEL Xeon E5-2650 processors with 8 computational cores and 16 threads in each, and on a computational node of a native supercomputer APK-5 [15]. The fig. 5 shows the results of a computational test on both: acceleration due to the number of OpenMP threads, against a single thread. The time stepping algorithm was organized into 4 main cycles by cells and faces; the last cycle assembling all the fluxes together and creating the new state of parameters. The promotion to the 2nd order approximation in time was made repeating this algorithm once more with some modifications; the details are shown in [16] and [17].

The modeled configuration included more than 9 million cells.



(a) acceleration for the server node (b) acceleration for the APK-5

Figure 5. Acceleration for the server node and for the APK-5

It is seen that for the number of processes given, the acceleration is near linear. This is due to the explicit numerical scheme used in calculations. We obtained more than 6 times acceleration on the server system, and more than 12 times acceleration on the computer APK-5. The APK-5 node acceleration is better due to high frequency of processors and its new architecture in comparison with the processors of the another server node.

In order to calculate the bigger devices, and/or to use better mesh refinement to resolve more details of the process, one should use all the computational nodes. To do this, it is worth to add the MPI parallelization paradigm into the computational algorithm.

Conclusions

The problem is computationally complex; the mathematical model includes multi-component gas dynamics, with the addition of the transport terms, the chemical kinetics, and the turbulence modeling. The following is obtained:

- The combustion chamber is a hollow cylinder; the inner body is cylindric, transfering to a cone. In case the fresh combustible mixture is inflown via the orifices which are open in turn, and not simultaneously, we obtained a detonation wave propagating in one direction.
- The detonation wave between two coaxial cylinders rotates consuming the fresh mixture. After one circulation, the thickness of the fresh mixture is rather small, due to a high velocity of detonation, and a small enough radius of circulation so that the time of one rotation was about 50 μs.
- Due to this, the strength of the detonation wave decreases after the first rotation. However, a stable inflow of cold fresh mixture causes the detonation wave to propagate even after the initial combustible mixture is consumed.
- Some hot spots initiate detonation in the fresh mixture ahead of the main wave so that it can split in two and even more other waves. The galloping detonation originated in some time.
- The numerical scheme was explicit, and the computational acceleration using OpenMP parallelization paradigm was nearly linear on both devices we have tested.

Acknowledgements

The work was made with the support of the RFBR grant no. 18-07-00889.

References

1. Bulat, P. V., Volkov, K. N.: Detonation jet engine. Part 2 construction features. International Journal of Environmental and Science Education (IJESE) 11(12), 5009–5019 (2016), https://istina.msu.ru/publications/article/42853937, accessed: 2018-05-15

2. Roy, G., Frolov, S.: Deflagrative and Detonative Combustion. Torus Press, Moscow (2010)

3. Wolanski, P., Kindracki, J., Fujiwara, T., Oka, Y., Shimauchi K.: An experimental study of rotating detonation engine. 20th International Colloquium on the Dynamics of Explosions and Reactive Systems 31 July – 5 August, 2005, Montreal, Canada (2005), http://www.icders.org/ICDERS2005, accessed: 2018-05-15

4. Frolov, S.M., Dubrovskii, A.V., Ivanov, V.S.: Three-dimensional numerical simulation of the operation of the rotating-detonation chamber. Russian Journal of Physical Chemistry B 6(2), 276–288 (2012), DOI: 10.1134/S1990793112010071

5. Liou, M.S.: A Sequel to AUSM: AUSM+. Journal of Computational Physics 129(2), 364–382 (1996), DOI: 10.1006/jcph.1996.0256

6. Hirsch, C.: Numerical computation of internal and external flows, 2nd Edition. Wiley (1990)

7. Marinov, N.M., Pitz, W.J., Westbrook, C.K., Hori, M., Matsunaga, N.: An experimental and kinetic calculation of the promotion effect of hydrocarbons on the NO-NO2 conversion in a flow reactor. Symposium (International) on Combustion 27(1), 389–396 (1998), DOI: 10.1016/S0082-0784(98)80427-X

8. Kee, R.J., Miller, J.A., Jefferson, T.H.: Chemkin: a general-purpose, problem-independent, transportable Fortran chemical kinetics code package. Sandia National Laboratories Report SAND80-8003 (1980)

9. CHEMKIN. A software package for the analysis of gas-phase chemical and plasma ki-netics. CHE-036-1. Chemkin collection release 3.6. Reaction Design, September 2000

10. Wilcox, D.C.: Turbulence modeling for CFD. DCW Industries, Inc. La Canada (1993)

11. Transport. A software package for the evaluation of gas-phase, multicomponent transport properties. TRA-036-1, CHEMKIN collection, release 2000

12. Connaire, M. O., Curran, H J., Simmie, J. M., Pitz, W. J., Westbrook, C.K.: A comprehensive modeling study of hydrogen oxidation. International Journal of Chemical Kinetics 36(11), 603–622 (2004), DOI: 10.1002/kin.20036

13. Smirnov, N.N., Nikitin, V.F., Alyari-Shourekhdeli, S.: Transitional regimes of wave propagation in metastable systems. Combustion, Explosion, and Shock Waves 44(5), 517–528 (2008), DOI: 10.1007/s10573-008-0080-3

14. Maas, U., Pope, S.: Simplifying chemical kinetics: Intrinsic low-dimensional manifolds in composition space. Combustion and Flame 88(3), 239–264 (1992), DOI: 10.1016/0010-2180(92)90034-M

15. APK-5 documentation. http://compcenter.org/super-evm-apk-5, accessed: 2018-05-15

16. Smirnov, N.N., Penyazkov, O.G., Sevrouk, K.L., et al.: Detonation onset following shock wave focusing. Acta Astronautica 135, 114–130 (2017), DOI: 10.1016/j.actaastro.2016.09.014

17. Betelin, V.B., Kushnirenko, A.G., Smirnov, N.N., et al.: Numerical investigations of hybrid rocket engines. Acta Astronautica 144, 363–370 (2018), DOI: 10.1016/j.actaastro.2018.01.009

DOI: 10.14529/jsfi180208

Numerical Simulations of Black Hole Accretion Flows

***Agnieszka Janiuk*[1], *Konstantinos Sapountzis*[1], *Jérémy Mortier*[1], *Ireneusz Janiuk*[2]**

We model the structure and evolution of black hole accretion disks using numerical simulations. The numerics is governed by the equations of general relativistic magneto-hydrodynamics (GRMHD). Accretion disks and outflows can be found at the base of very energetic ultra-relativistic jets produced by cosmic explosions, so called gamma-ray bursts (GRBs). Another type of phenomena are blazars, with jets emitted from the centers of galaxies.

Long-lasting, detailed computations are essential to determine the physics of these explosions, and confront the theory with potential observables. From the point of view of numerical methods and techniques, three ingredients need to be considered. First, the numerical scheme must work in a conservative manner, which is achieved by solving a set of non-linear equations to advance the conserved quantities from one time step to the next. Second, the efficiency of computations depends on the code parallelization methods. Third, the analysis of results is possible via the post-processing of computed physical quantities, and visualization of the flow properties. This is done via implementing packages and libraries that are standardized in the field of computational astrophysics and supported by community developers.

In this paper, we discuss the physics of the cosmic sources. We also describe our numerical framework and some technical issues, in the context of the GRMHD code which we develop. We also present a suite of performance tests, done on the High-Performance Computer cluster (HPC) in the Center for Mathematical Modeling of the Warsaw University.

Keywords: astrophysical flows, black hole accretion, hydrodynamics, numerical simulations, general relativistic MHD.

Introduction

Astrophysical black holes are ubiquitous in the Universe. They occupy centers of galaxies [28], they may be found in the binary systems with ordinary stars, where the streams of plasma lead to the phenomenon called a "microquasar" [23], and, finally, they may be responsible for the most extreme cosmic explosions – the gamma ray bursts [20]. In all these types of sources one common physical process is in work: accretion of matter onto a black hole. It is the most efficient of the known energy release mechanisms, which is orders of magnitude stronger than the nuclear fusion reactions that fuel ordinary stars. The gravitational potential energy in the field of the compact star is governed by its mass-to-radius ratio. Hence, per unit rest-mass energy of the gas fallen into the black hole, we can extract up to almost sixty per-cent of power available to be released in the form of the electromagnetic radiation [10].

Gamma ray bursts (GRBs) are electromagnetic transients observed from the most distant parts of the Universe. Their brightness detected by the human-made telescopes implies that the intrinsic power of the events is enormously large. During the collapse of a massive star into a black hole in a hyper-nova process a long duration burst ($t \sim 100 - 1,000$ seconds) can be observed. It is required that the progenitor star has enough angular momentum to form an accretion disk around a black hole. Short GRBs ($t \sim 0.01 - 2$ seconds) are associated with the coalescence of binary neutron stars, which form a black hole as a product of their merger. The transient jets of plasma are generated by central engine, which is composed of a newly

[1] Center for Theoretical Physics PAS, Warsaw, Poland
[2] Verifone Sp. Z.o.o., Warsaw, Poland

formed black hole, surrounded by a remnant disk. These jets are ultimately responsible for the electromagnetic gamma-ray emission, observed by our telescopes.

Another type of sources visible as the ultra-relativistic jets of plasma that emit very high energy radiation spectra, are called blazars. These sources are persistent in nature, and they do not originate in violent explosions. However, the mechanism of extracting energy from the rotating black hole is the same, and requires the magnetic fields to mediate the process. The famous Blandford–Znajek mechanism can work as a kind of cosmic "battery" [3] and requires that the accretion disk is supplied with a strong poloidal magnetic field. When the jet is pointing in the direction of the Earth, the observer detects phenomenon called a "blazar", where the highest energy flux is detected due to the boosting effect and collimation of the stream in a narrow cone around the line of sight [33]. Gamma-ray emission of blazars exhibits often a short-timescale variability that lasts from hours to days [1]. This means that the gamma-ray emission from the jet is not uniform and short time-scale variability suggest that it occurs close to the black hole. This effect is quite similar again to the GRB variability, albeit the timescales are now on the order of a millisecond.

Our work focuses on modeling the structure and evolution of the accretion disk at the base of the jet engine, which is composed of a highly magnetized plasma accreting onto a black hole. In order to construct a physical model of such disk, we need to solve the GRMHD equations. They are further supplemented by the equation of state (EOS) of the matter. Its form depends on the particular phenomenon and astrophysical scenario considered. In the quiescent centers of galaxies, and also in the persistent jet sources, such as blazars, the accreting matter is quite hot, but rarefied, and to a good approximation it can be described with an ideal gas EOS. In the GRBs explosive events, the EOS is more complex. Under the conditions of extremely high densities and temperatures, the nuclear reactions have to be taken into account. Furthermore, the matter is highly degenerated, and relativistic hot particles cannot form an ideal gas. The density and temperature are tied to the pressure and internal energy in a non-linear way. Finally, the nuclear processes may occur on the rates faster than the timescale required to establish the statistical equilibrium conditions in the gas.

All these physical complexities: magnetic fields, general relativity, nuclear reactions, pose a challenge to any kind of numerical scheme. Different codes have been proposed to cover both the microphysics of the fluids, governed by EOS, and the evolution of magnetized gas in the black hole gravitational potential, governed by the GRMHD equations.

The article is organized as follows. In Section 1 we define the physical equations thet are solved by our GRMHD code and we describe the conversion scheme. In Section 2 we present the tools used for visualisation of the simulation results. Section 3 is devoted to the problem of boundary conditions in MHD simulations. In Section 4 we discuss the parallelisation methods and compare the performance of the code when different techniques are used. Section 5 presents exemplary results of our simulations of a black hole, torus and jet system evolved with the GRMHD code, and the results of post-processing simulation with a nuclear reaction network code. In Section 6 we discuss our computations in the broader context of recent astronomical discoveries.

1. GRMHD Equations

From the steady state based on the analytical, equilibrium solution driven by the main physical parameters of the black hole accretion disk (namely, BH mass, its spin, and size and

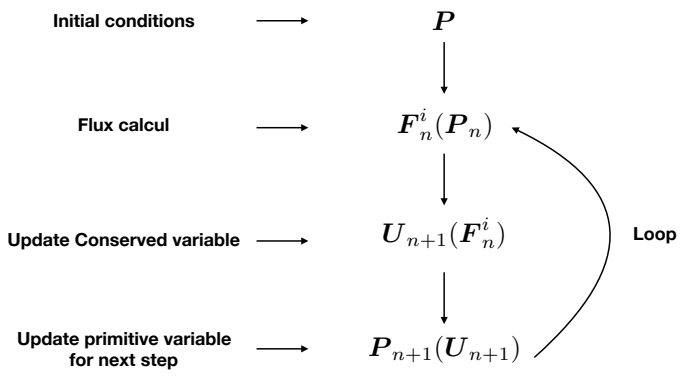


Figure 1. Time step of a conservative scheme of order 1 in time, n denotes the time step

the mean accretion rate in the torus), as well as the seed configuration of magnetic field, we follow the dynamical evolution. This is achieved by solving numerically the continuity Eq. (1), the four-momentum-energy conservation Eq. (2), and induction Eq. (3) equations in GR framework:

$$\frac{1}{\sqrt{-g}}\partial_\mu(\sqrt{-g}\ \rho u^\mu) = 0, \tag{1}$$

$$\partial_t(\sqrt{-g}\ T^t_{\ \nu}) = -\partial_i(\sqrt{-g}\ T^i_{\ \nu}) + \sqrt{-g}\ T^\kappa_\lambda \Gamma^\lambda_{\ \nu\kappa}, \tag{2}$$

$$\partial_t(\sqrt{-g}\ B^i) = -\partial_j(\sqrt{-g}\ (b^j u^i - b^i u^j)). \tag{3}$$

Here the stress tensor separates into gas and electromagnetic parts:

$$\begin{gathered} T^{\mu\nu} = T^{\mu\nu}_{gas} + T^{\mu\nu}_{EM} \quad , \\ T^{\mu\nu}_{gas} = \rho h u^\mu u^\nu + p g^{\mu\nu} = (\rho + u + p) u^\mu u^\nu + p g^{\mu\nu}, \\ T^{\mu\nu}_{EM} = b^2 u^\mu u^\nu + \frac{1}{2} b^2 g^{\mu\nu} - b^\mu b^\nu; b^\mu = u^*_\nu F^{\mu\nu}. \end{gathered} \tag{4}$$

Other symbols in Eq. (4) have their usual meaning: u^μ is the four-velocity of gas, u is internal energy, ρ is density, p denotes pressure, and b^μ is the magnetic four-vector. Note that in Eq. (3) B^i is the magnetic field three-vector, b^i is the spatial part of the magnetic field four-vector and u^i is the spatial part of the four-velocity. Finally, F is the Faraday tensor, and in the force-free approximation $E_\nu = u^\nu F^{\mu\nu} = 0$. The space-time metric $g_{\mu\nu}$ is described in Eq. (1) with determinant $g \equiv \mathrm{Det}(g_{\mu\nu})$ and $\Gamma^\lambda_{\ \nu\kappa}$ is the spatial connection.

1.1. Our Code for the High Accuracy Relativistic MHD

HARM (High Accuracy Relativistic Magneto-hydrodynamics) is a conservative shock capturing scheme, for evolving the equations of GRMHD, developed by C. Gammie et al. [11]. The integrated equations are of the form:

$$\partial_t \boldsymbol{U}(\boldsymbol{P}) = -\partial_i \boldsymbol{F}^i(\boldsymbol{P}) + \boldsymbol{S}(\boldsymbol{P}), \tag{5}$$

where $\boldsymbol{U}$ is a vector of "conserved variables", such as particle number density, or energy or momentum, $\boldsymbol{F}^i$ are the fluxes in finite control volume, and $\boldsymbol{S}$ is a vector of source terms. $\boldsymbol{U}$ is conserved in the sense that, if $\boldsymbol{S} = 0$, it depends only on fluxes at the boundaries. The vector $\boldsymbol{P}$ is composed of "primitive" variables, such as rest-mass density, internal energy density, velocity components, and magnetic field components, which are interpolated to model the flow within zones. $\boldsymbol{U}$ and $\boldsymbol{F}^i$ depend on $\boldsymbol{P}$. Conservative numerical schemes advance $\boldsymbol{U}$, then, depending on the order of the scheme, calculate $\boldsymbol{P}(\boldsymbol{U})$ once or twice per time step, as shown in Fig. 1.

Our version of the code works in 2D and 3D. It is fully parallelized using the Message Passing Interface (MPI) library (see Section 4 for parellelisation and performance test results) and the output of the simulation is dumped both in ASCII and Hierarchical Data Format (see Section 2).

1.2. Equation of State

The equation of state in the plasma is based on equilibrium of the nuclear reactions, which, when reached, defines the proton-to-baryon density ratio, and hence the pressure, internal energy and entropy of the gas.

In the hyper-accreting matter in the GRB central engine, temperatures are above $10^9 - 10^{10}$ K, and the plasma is fully ionized and composed of free nuclei, n, p, and electron-positron pairs, e^+, e^-. The chemical and pressure balance required by nuclear reactions between these species, namely the electron and positron capture on nuclei, and the β-decay. Reactions are in the form:

$$\begin{aligned}
p + e^- &\rightarrow n + \nu_{\rm e},\\
p + \bar{\nu}_{\rm e} &\rightarrow n + e^+,\\
p + e^- + \bar{\nu}_e &\rightarrow n,\\
n + e^+ &\rightarrow p + \bar{\nu}_{\rm e},\\
n &\rightarrow p + e^- + \bar{\nu}_{\rm e},\\
n + \nu_{\rm e} &\rightarrow p + e^-.
\end{aligned}$$

The rates for these reactions are given by appropriate integrals [27].

The electron neutrinos and anti-neutrinos are the source for cooling of the plasma, in addition to advective and radiative cooling (the latter is in fact negligible, due to the huge opacities for the photons). The above nuclear processes have been studied in numerous works, devoted to the neutron star Equation of State, and later on in the context of GRB central engines [5, 7, 14–18, 26].

Other neutrino emission processes that occur at lower rates are: electron-positron pair annihilation, bremsstrahlung, and plasmon decay:

$$\begin{aligned}
e^- + e^+ &\rightarrow \nu_{\rm i} + \bar{\nu}_{\rm i},\\
n + n &\rightarrow n + n + \nu_{\rm i} + \bar{\nu}_{\rm i},\\
\tilde{\gamma} &\rightarrow \nu_{\rm e} + \bar{\nu}_{\rm e}.
\end{aligned}$$

We calculate their rates numerically, with proper integrals over the distribution function of relativistic, partially degenerated species. These processes lead to formation of heavy lepton neutrinos, ν_τ and ν_μ.

1.2.1. Pressure Components

In the EOS, contribution to the pressure is by the free nuclei and $e^+ - e^-$ pairs, helium, radiation and the trapped neutrinos:

$$P = P_{\rm nucl} + P_{\rm He} + P_{\rm rad} + P_\nu,$$

where $P_{\rm nucl}$ includes free neutrons, protons, and the electron-positrons:

$$P_{\rm nucl} = P_{\rm e-} + P_{\rm e+} + P_{\rm n} + P_{\rm p},$$

with

$$P_{\rm i} = \frac{2\sqrt{2}}{3\pi^2}\frac{(m_i c^2)^4}{(\hbar c)^3}\beta_i^{5/2}\left[F_{3/2}(\eta_{\rm i},\beta_{\rm i}) + \frac{1}{2}\beta_{\rm i}F_{5/2}(\eta_{\rm i},\beta_{\rm i})\right]. \quad (6)$$

Here in Eq. (6), $F_{\rm k}$ are the Fermi–Dirac integrals of the order k, and $\eta_{\rm e}$, $\eta_{\rm p}$ and $\eta_{\rm n}$ are the reduced chemical potentials, $\eta_i = \mu_i/kT$, is the degeneracy parameter (where μ_i is the standard chemical potential). Reduced chemical potential of positrons is $\eta_{\rm e+} = -\eta_{\rm e} - 2/\beta_{\rm e}$. Relativity parameters are defined as $\beta_{\rm i} = kT/m_{\rm i}c^2$. EOS is computed numerically by solving the balance of nuclear reactions [12, 16, 35].

To sum up, the proper description of the hyper-accretion in GRBs requires detailed treatment of microphysics. Based on the solutions for degenerate Fermi gas EOS, with $P(\rho, T)$ non-linear dependence, a non-trivial transformation between conserved variables and "primitives" in HARM due to GRMHD scheme.

The interpolation over the tables of EOS is done (using the Akima-spline method [2]) during the dynamical simulation at each and every time step. In order to save the computer power, we usually compute a small matrix of $4 \times 4 = 16$ points at each grid cell, whenever the value must be interpolated, and only then we store the table. To perform the interpolations with maximum efficiency, we use the multi-threading feature of the Linux operating system (with *pthread* command).

1.2.2. Neutrino Cooling

The effect important for the state of accretion disk is neutrino cooling. The neutrinos of three flavors are emitted via the above weak interactions, and the neutrino cooling rate is finally given by the two-stream approximation, and includes the scattering and absorptive optical depths for neutrinos of all three flavors:

$$Q_\nu^- = \frac{\frac{7}{8}\sigma T^4}{\frac{3}{4}} \sum_{i=e,\mu,\tau} \frac{1}{\frac{\tau_{{\rm a},\nu_{\rm i}}+\tau_{\rm s}}{2} + \frac{1}{\sqrt{3}} + \frac{1}{3\tau_{{\rm a},\nu_{\rm i}}}} \times \frac{1}{H} \quad [\text{erg s}^{-1}\ \text{cm}^{-3}],$$

as given by [7]. This expression assumes that neutrinos are thermalized. Ideally, neutrino transport should be accounted for (see e.g. [25]).

1.3. Conversion Scheme

The composition-dependent EOS is in our simulations coupled to the conservative scheme. The HARM (high-accuracy relativistic magneto-hydrodynamics) scheme solves equations Eq. 5 where $\mathbf{U}$ is a vector of "conserved" variables, (i.e. the number density, energy or momentum density in the coordinate frame), $\mathbf{F}^i$ are the fluxes, and $\mathbf{S}$ is a vector of source terms that do not involve derivatives of $\mathbf{P}$ and therefore do not affect the characteristic structure of the system. In non-relativistic MHD, both $\mathbf{P} \rightarrow \mathbf{U}$ and $\mathbf{U} \rightarrow \mathbf{P}$ have a closed-form solution. However, in GRMHD $\mathbf{U}(\mathbf{P})$ is a complicated, nonlinear relation. Inversion $\mathbf{P}(\mathbf{U})$ is calculated once or twice in every time step, numerically. The transformation between "conserved" (momentum, energy density) and "primitive" (rest mass density, internal energy) variables requires to solve a set of 5 non-linear equations. This inversion is complex for a non-adiabatic relation of the pressure with density. We are doing it numerically and interpolate over the table of pre-computed EOS results.

2. Visualization and Post-processing of Results

The code produces a set of outputs that can be divided into three categories. The Initialization Output is only produced once, before the integration begins, and contains the constant quantities of the simulation like the grid and coordinates, the metric and its determinant on the grid points. Some model parameters are also stored during the Initialization stage, e.g. the BH spin parameter, the adiabatic index, etc.

The Results Output contains the main results of the integration and stores the physical quantities of the flow (density, internal energy, contravariant velocity, magnetic field vector), while it contains also some other derived quantities of physical interest. Among the various options we used for the form of the dumping data (text, raw binary, HDF5, etc.), the HDF5 proved to be the most advantageous. The 1-D, and 2-D models of an ideal conducting fluid do not pose significant restrictions on the dumped data type. The situation alters when we assume a composition dependent EOS, or we proceed to the 3-D simulations where both file size and structure complexity increase dramatically. The hierarchical structure of the file makes it easy to locate specific quantities through a *POSIX*-like syntax.

The size of the dumped file or of the objects it includes is not limited, while the special extensions, *slib, zlib*, can be used to compress the resulting file. But the most significant features that HDF5 provided, were the performance of its collective (parallel) I/O driver and the portability of the data in both C/C++ and Python/IPython interfaces. The robust performance of the I/O driver was noticed in both of the HPC and local servers we used.

The post-processing and the visualization of the results for the 1-D, 2-D simulation is performed by the standard packages of the *Python3* language (*numpy, matplotlib/pylab, scipy*). With the extension of the *h5py* package, the import of the results is straightforward and sequentially the powerful routines of the python libraries are used for further manipulation. An exemplary plot (Fig. 2) was produced using the *matplotlib/pyplot* module. The parallelization of the above scripts proved to be a challenging task and we concluded that the *mpi4py* module is easier portable to a series of machines from our Desktops and 32-Intel-cores local server, up to the Cray XC40 of the Okeanos HPC, mostly because of the parallel *HDF5/h5py* and the *mpi4py* module compatibility. It is clear that the parallelization is essential especially for the 3D post-process where various manipulations, e.g. interpolation, coordinate transformations, have

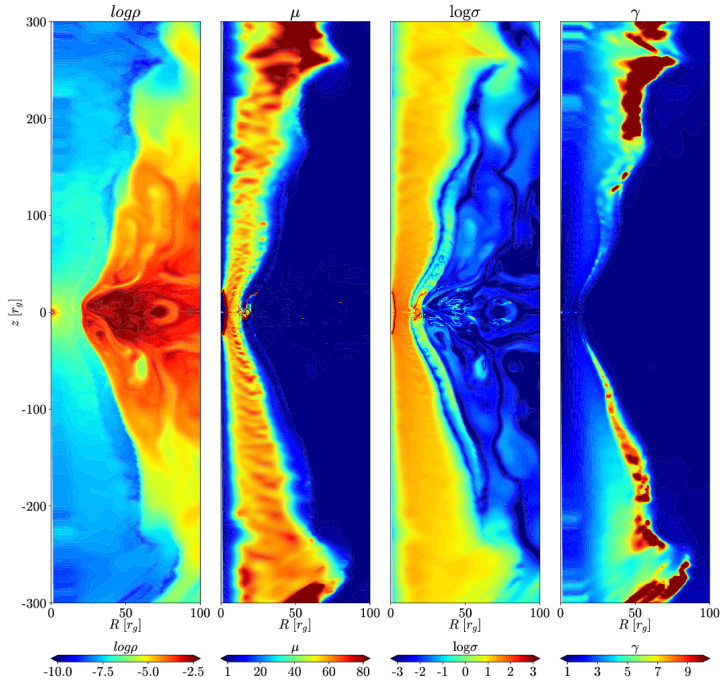


Figure 2. Space distribution of the density, magnetization, and Lorentz factor in the jet-outflow in the GRB. Example of the graphical visualization with *matplotlib/pyplot*

to be done on the points of extended grids. Once these calculations are completed, we use *PyVtk* module to produce a VTK file output and more advanced tools for the 3D visualizations, e.g. *VisIt*. An example plot is shown in Fig. 3.

The Debug Output is the final set of output produced during the run time. Beyond the simple execution messages and the validity of the $\vec{\nabla} \cdot \vec{B} = 0$ condition, the code performs a number of physical diagnostics during each step of integration and dumps the results through a series of binary and ppm graphic files (per process). The motivation for these tests is to help user to identify unphysical results and avoid time consuming calculations.

To sum up, the HDF5 format is characterized by:

- Robust and satisfactory performance of the MPI I/O process.
- High portability in many interfaces (C/C++ and Python).
- Easy to locate quantities through the POSIX type structure.

Furthermore, our Python main processing (cython under development) allows for:

- Calculation of various physical quantities (h5py).
- Building of VTK Cartesian Structured Grid (PyVTK).
- Parallel Python (mpi4py).
- 2D slices and analysis (numpy, scipy, etc.).

Finally, we take advantage of the open-source tool *VisIt* for visualization and 3D post-processing.

3. Boundary Conditions

The HARM code does not solve the equations of the GRMHD in the Boyer-Lindquist coordinates system, but rather on a modified version of the so called Kerr-Schild coordinate system (KS). The KS system is not singular on the black hole horizon and therefore, the matter can

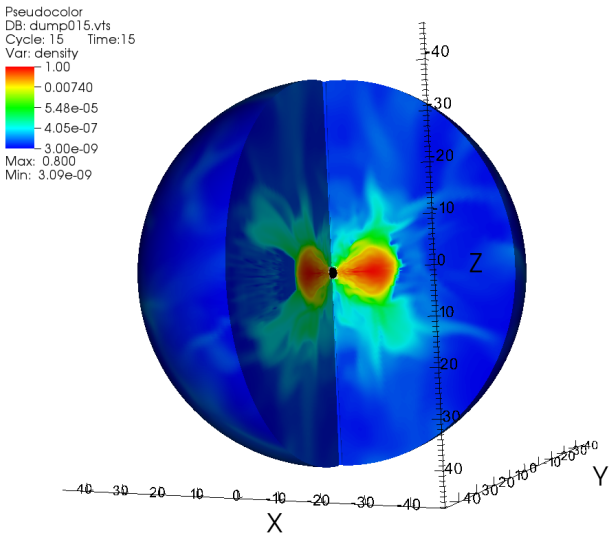


Figure 3. 3D simulation of the black hole-torus accretion system. The snapshot is taken at $t = 1,000M$. The x, y and z axes are in r_g units (gravitational radius). The black hole is represented with a black circle. Magnetic field is neglected in this simulation. The color scale shows the normalized density

accrete smoothly through this surface, and the evolution of the flow can be followed, see for example [9]. A further transformation applies on the radial component of the specific system using a logarithmic mapping [11]. As a consequence, our points distribution is denser close to the horizon, when we assume an equally spaced grid on the r-direction.

The boundary conditions that apply on the radial direction are the free inflow-outflow conditions, modified by a specific extrapolation schema that reduces the unphysical behavior. This behavior is induced mostly by the variation of the metric between the normal and the ghost cells [11], and the selected extrapolation was chosen such as to maximize the robustness of the code. In reality, the radial boundary conditions have negligible effect on the physics of the problem under the proper choice of the grid. The inner boundary is located inside the horizon, while due to the logarithmic spatial scale of the grid, the outer boundary can be set far away from the domain of interest.

The 2-dimensional simulations are, by definition, axisymmetric, i.e. the derivatives of quantities in the ϕ-direction are neglected; note however, that the vectors (velocity field, magnetic field) still have all the three components. In contrast for the 3-dimensional simulations the derivatives of the quantities are computed so the non-axisymmetric evolution is being followed properly. The periodic boundary condition is always used in the azimuthal direction and the flow quantities at $\phi_0 + 2\pi$ are the same as at ϕ_0 describing the smooth continuation of the solution between the neighboring domains.

The real technical problem is posed by the boundary condition at the z-axis, namely in the polar direction. Physically, the vertical axis is only the symmetry axis in the 2D flows, but it should not work as a real boundary in the 3D flows. In order to get some intuition on this effect, the reader can think of a Cartesian coordinate system. In such a case boundary conditions have to be set in an inner box, that will lay well inside the horizon, and an outer

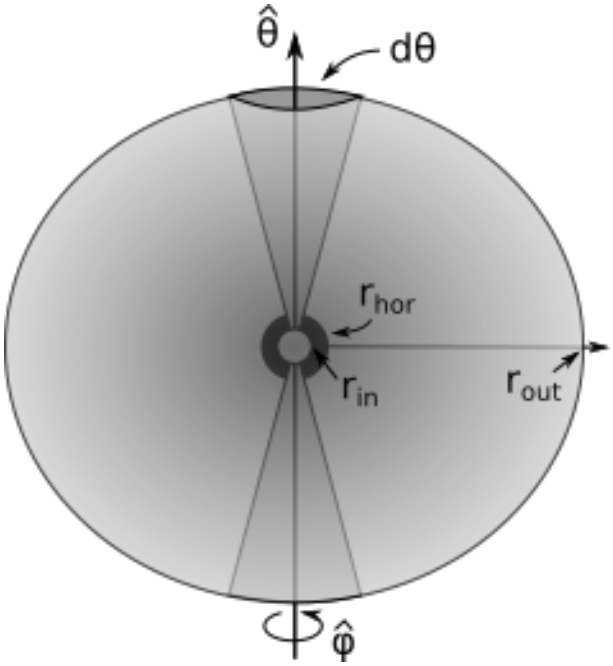


Figure 4. Illustration of the spherical coordinate system and boundary conditions problem

one which lays at great distances, but not on the axis of the black hole rotation (see Fig. 4). A technique to overcome the extra needed boundary conditions is to cancel in practice the axis existence by attaining the "ghost cells" variables from the values of the corresponding normal grid cells on the other side of the axis; notice though that the vectors $\phi-$ components must change sign because of the opposite direction of the $\phi-$unitary. Therefore the application of the above technique requires by our algorithm to connect the two correspond grid domains and sets further complexity in our MPI implementation.

4. Parallelization Methods in HARM Simulations

The HARM code works in 1, 2, and 3-Dimensions. In the latter case, the optimal time for any realistic simulation requires parallel computing.

In the simulation presented in Fig. (3), i.e. the non-magnetized, 3-dimensional case, the grid resolution was 192x192x192 points. The number of HPC nodes was N=64, the number of tasks per node was n=24. The number of run-time steps of integration was about 110000 and the total real time of computation was about 12.7 hours.

4.1. MPI in Practice

The distribution of the processes among the physical system directions is provided by the function *MPI_Dims_create(nprocs, 2, dims)*; currently it is 2D, but it can easily be generalized to 3D. According to this routine the divisors are set to be as close as possible using an appropriate divisibility algorithm, while *dims[i]* are ordered in decreasing order: $dims[0] \geq dims[1] \geq dims[2] \geq ....$

An alternative procedure incorporated also in our schema distributes the number of processes using two criteria. In order to reduce bottlenecks, the $N_x \times N_y$ grid is distributed to the $n_x \times n_y$ processes on each direction such as $N_x - n_x[\frac{N_x}{n_x}]$ processes loaded with $[\frac{N_x}{n_x}]$ points and $n_x([\frac{N_x}{n_x}]+1) - N_x$ processes with $[\frac{N_x}{n_x}]+1$ ones where the square brackets denote the integer part

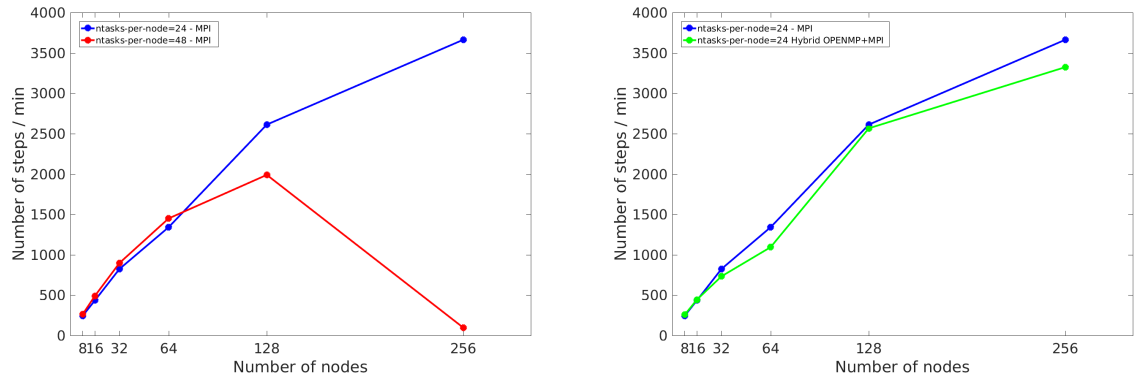


Figure 5. Performance test results on the Okeanos supercomputer. The code HARM-2D was run on 8-256 nodes, using MPI parallelization method (blue line), and Hybrid MPI+OpenMPI method (green line). Also, the pure MPI runs were made with the hyper-threading (on 2x24=48 threads; shown by red line)

of the division; the same holds for the y direction. We then perform an optimization routine requiring that the grid points load per each process is as balanced as possible, while in addition the necessary MPI communications between the neighboring domains are minimum. The two criteria are calculated with different weight allowing us to perform further optimization based on the specific integrating system and the machine specifications.

The assignment of the Cartesian grid to the MPI communicator is performed with the default MPI_Cart_create and MPI_Cart_coords routines that provide a specific rank per direction to every process. The latter is of primary importance for identifying the mirror to the rotation axis points and applying the proper boundary conditions (see Sec. 3).

4.2. Supercomputer Performance Tests

We used both the standard Message Passing Interface (MPI) technique, and also a more advanced, Hybrid parallelization method. The MPI+hyper-threading was also tested (cf. Fig. 5, red and blue lines). The computational grid was divided into slices, where every process works on its own area, and boundaries are exchanged when needed. The pure MPI program running on the cluster of 1024 nodes and using 24 threads per node creates 24,576 processes in total. It is a huge number, and the first expectation is that interprocess communication for boundaries exchange should decrease the overall performance.

The code occurred to be well-scalable for a uniform resolution, e.g. the number of the grid points in 3 dimensions is equal to $N_{\rm r} \times N_\theta \times N_\phi = 192 \times 192 \times 192$. For non-uniform grids, the dependence on $N_{\rm nodes}$ and $N_{\rm tasks/node}$ is not monotonic, due to the properties of function *MPI_Dims_create*. Another method was to use the MPI + OpenMP shared memory, i.e. a Hybrid approach [3]. Here, only one MPI process is created per node and called master. The parallel execution is done for every loop in the code in *fork-join* mode. It was rather straightforward (in comparison to MPI) to add OpenMP calls, using only several *pragma* statements. Our preliminary tests were aimed to check if pure MPI-HARM, running on N nodes with 24 cores each, will be more/less efficient with comparison to MPI+OpenMP hybrid solution running on

[3] http://bisqwit.iki.fi/story/howto/openmp/ and https://computing.llnl.gov/tutorials/openMP/

the same number of nodes. The results are in contradiction to the claims published in literature, which show that the hybrid solution usually works better.

In our code, there is one significant difference with respect to the common MPI usage. We have computational domain divided into pieces and every MPI process uses only small memory region. This gives a faster memory access and it might make a difference.

5. Models Specific to Black Hole Accretion Systems

Below, we present some exemplary results of our simulations. Then, in the next Section, we discuss their results in the context of the visualization and post-processing methods. These "technical" aspects are by no means trivial from the computational point of view, and proper analysis of the physics involved is tightly coupled with the post-processing demands. Finally, these simulations are at the limits of our computational resources, and fine numerical techniques have to be used to increase the efficiency of the simulations and code performance on the available computer clusters.

5.1. Magnetized Torus

For the purpose of better understanding of the black hole accretion and jets variability, we investigate the role of magnetically arrested disks (MADs) [32], as producer of turbulence in relativistic jet. MADs state occurs when the magnetic pressure force, pushes outward on the accretion disk gas. Because we need a certain amount of magnetic flux surrounding the black hole, to have enough magnetic pressure balancing the accretion, we need a large initial magnetized torus. To do this, we implement a thick disk model as an intial condition in HARM using the Chakrabarti's prescription [4]. In this model, the angular momentum distribution is chosen to have a power law distribution. Alternatively, the default disk model of Fishbone & Moncrief [8], is assuming the angular momentum to be constant in the disk. This model allows to create an initial torus configuration with a large amount of poloidal magnetic flux. The initial and evolved state of the trous, seeded by the poloidal magnetic field, is visualized in Fig. 6.

5.2. Ejection of Relativistic Jets

If the black hole starts rotate fast, the jet ejection is inevitable. The presence of magnetic fields and/or neutrino-antintineutrino pairs provide the mechanism for jets acceleration. The Blandford–Znajek process, which allows for the extraction of rotational energy of the black hole and transports it to the remote jets via magnetic fields, can be quantified in our simulations with the following expression

$$\dot{E} \equiv \int d\theta d\phi \, \sqrt{-g} \, T^{r}{}_{t}. \tag{7}$$

Here, in Eq. (7) the magnetic part of the stress-energy tensor is integrated over the black hole horizon. In addition, the magnetic fields transport angular momentum in the accretion disk and allow accretion. In Fig. 7 we show the resulting power available for the jets, as dependent on the black hole rotation spin.

The jets are accelerated at the vicinity of the black hole due to the central engine activity, and at large distances their kinetic energy has to be ultimately transferred to the emitted gamma

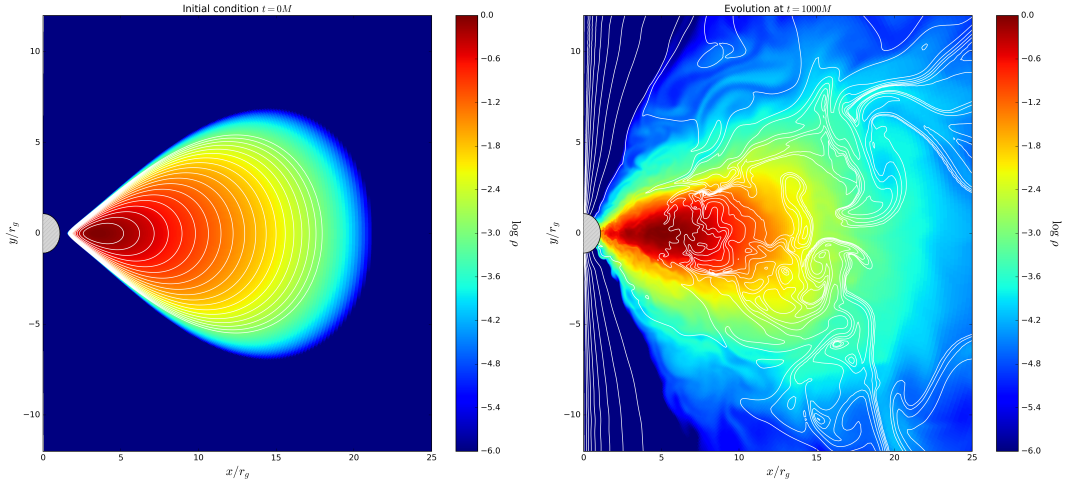


Figure 6. 2D simulation of a black hole-torus accretion system, with resolution of 512 x 512 grid points in the r and θ direction. The left panel shows the initial condition, while the right one presents a snapshot $t = 1,000M$. The x and y axes are in r_g units (gravitational radius). The black hole is represented with a dashed circle. Magnetic field lines are plotted in white contours, and the color scale shows the normalized density

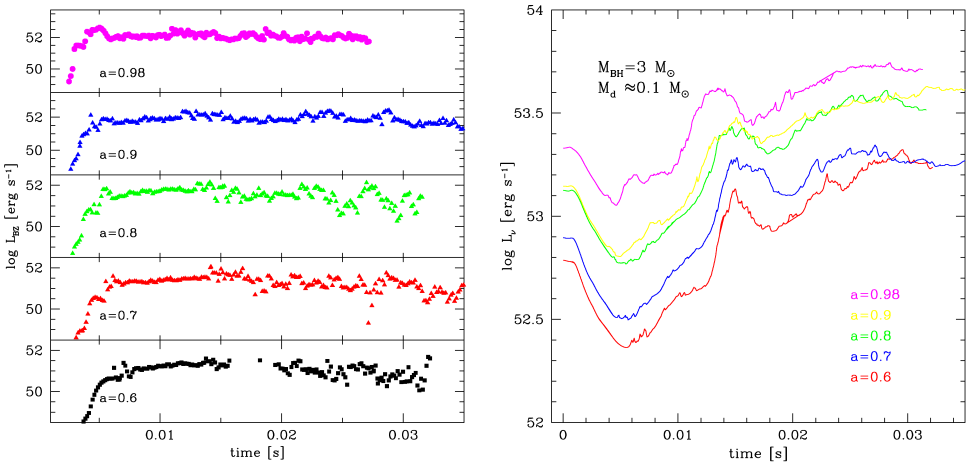


Figure 7. Power available to accelerate the relativistic jets, produced within the accretion torus in the GRB central engine. Several results are plotted, for a varying black hole spin parameter, a, as denoted in the figures. Two mechanisms are compared: Blandford–Znajek process (left) and neutrino anihillation (right). Note, that the latter has to be reduced by an efficiency factor, on the order of $\eta_\nu \sim 0.01$, due to the uncertainties in the neutrino pair annihilation process.

ray radiation. To achieve this, the jet speed, expressed with the dimensionless Lorentz factor,

$$\gamma = \frac{1}{\sqrt{1-(\frac{v}{c})^2}},$$

has to be on the order of 100. This is required to avoid so-called "compactness problem", since the observed gamma ray spectra in GRBs are non-thermal. If the Lorentz factors in jets were small, the huge optical depth due to electron-positron pair creation would rather produce thermal emission [24]. The Lorentz factors of the jet estimated at 'infinity' in our simulations can easily reach the values around 80-100 (see Fig. 2).

5.3. Formation of Heavy Nuclei in the Neutron-rich Plasma

In the hyper-accreting disk at the GRB central engine, the conditions in the degenerated Fermi gas allow for substantial overabundance of the neutrons over protons. This is quantified with so-called "electron fraction" ratio:

$$Y_e = \frac{1}{1+n_n/n_p},$$

which in the highly neutronized matter is much smaller than 0.5.

The electron fraction distribution, together with the density and temperature, serve as an input for the subsequent nuclear reaction network computations [13, 21, 22, 34]. The network allows for production of heavier isotopes (beyond Helium, and in fact beyond the Iron peak), due to the rapid capture of neutrons on the nuclei. The nuclear reactions may proceed with 1 (decays, electron-positron capture, photodissociacion), 2 (encounters), or 3 (triple alpha reactions) nuclei. Abundances of the isotopes are calculated under the assumption of nucleon number and charge conservation for a given density, temperature and electron fraction ($T \leq 1MeV$).

In the GRB engine, along with the abundant light elements such as Carbon, and then Silicon, Sulfur and Calcium isotopes, we also found copious amounts of Titanium, Iron, and Nickel. The X-ray emission originating from the radioactive decay of isotopes, such as ^{45}Ti, ^{57}Co, ^{58}Cu, ^{62}Zn, ^{65}Ga, ^{60}Zn, ^{49}Cr, ^{65}Co, ^{61}Co, ^{61}Cu, and ^{44}Ti, might give the signal in the 12-80 keV energy band.

The r-process elements have been found to enrich the interstellar gas, first in our Solar system, and recently in the circum-burst environment of several Gamma Ray Bursts [30, 31] As discussed now in the literature, the dynamical ejecta launched during the compact binary mergers may be responsible for the of r-process nuclei. In addition, the ejecta subsequently produced by an accretion disk formed after the merger, may add a contribution to this "kilonova" lightcurve [19]. Thus observational verification of our computations results will now be much more robust, thanks to the new data. The observed effect is discussed briefly below in Sec. 6.

6. Summary

Numerical modeling of black hole accretion flows in extremely high energetic systems, such as the gamma ray bursts, is essential from the point of view of the recent observational discoveries. The discovery of gravitational waves in 2015, which was awarded the Nobel Prize in Physics in 2017, boosted the research also in high energy astrophysics, while it is related to the "multi-messenger" astronomy. An example of the recently announced event is the kilonova signal, and the short gamma ray burst accompanied by the gravitational wave emission.

Kilonova effect

- Optical and near-infrared emission, powered by radioactive decay of r-process nuclei [19].
- Kilonova candidates can be distinguished from supernova by faster time evolution, fainter absolute magnitudes, and redder colors.
- Dynamical ejecta from compact binary mergers, $M_{\rm ej} \sim 0.01 M_\odot$, can emit about $10^{40} - 10^{41}$ erg/s in a timescale of 1 week.
- Subsequent accretion can provide bluer emission, if it is not absorbed by precedent ejecta [30].
- In the GRB 130603B afterglow, the excess NIR flux corresponds to absolute magnitude $M(J) \sim 15.35$ mag at $\sim$ 7d after the burst, consistent with the kilonova behavior. The lightcurve is in agreement with predicted r-process kilonova optical emission [31].

Electromagnetic counter part: GW 170816

- Gravitational waves were discovered with the detection of binary black hole mergers and they should also be detectable from lower mass neutron star mergers.
- NS-NS are predicted to eject material rich in heavy radioactive isotopes that can power a kilonova.
- The gravitational wave source GW170817 arose from a binary neutron star merger in the nearby Universe with a relatively well confined sky position and distance estimate.
- A rapidly fading electromagnetic transient in the galaxy NGC4993, is spatially coincident with GW170817 and a weak short gamma-ray burst [6, 29].

Conclusion

With our simulations, we found that the proper microphysics treatment in GRMHD simulations of hyper-accretion is essential for determining the disk structure: thickness, chemical composition of torus and its ejecta. Furthermore, we concluded that the neutrinos and Blandford–Znajek process have comparable role in powering the GRB jets. As the large scale jets in GRBs are concerned, the variability of these jets is related to the disk's magneto-rotational turbulence timescale. The ultimate Lorentz factors are found to be on the order of few 100. Thus, the late-time X-ray and high frequency radio emission can provide constraints on the properties of the disk-jet system for a particular source, e.g. GW+EM170817. Finally, the magnetically driven, low $Y_{\rm e}$ winds from accretion disks in GRB engine can provide power to kilonova emission, which was found in this source.

Acknowledgments

This work was supported by the grants 2012/05/E/ST9/03914, 2016/23/B/ST9/03114, and 2015/18/M/ST9/00541 from the Polish National Science Center. The computational resources were granted from the project GB 70-4 in the Interdisciplinary Center for Mathematical Modeling.

References

1. Abdo, A.A., et al.: Gamma-ray Light Curves and Variability of Bright Fermi-detected Blazars. The Astrophysical Journal 722(1), 520–542 (2010), DOI: 10.1088/0004-637x/722/1/520

2. Akima, H.: A New Method of Interpolation and Smooth Curve Fitting Based on Local Procedures. Journal of the ACM 17(4), 589–602 (1970), DOI: 10.1145/321607.321609

3. Blandford, R.D., Znajek, R.L.: Electromagnetic extraction of energy from Kerr black holes. Monthly Notices of the Royal Astronomical Society 179(3), 433–456 (1977), DOI: 10.1093/mnras/179.3.433

4. Chakrabarti, S.K.: The natural angular momentum distribution in the study of thick disks around black holes. The Astrophysical Journal 288(1), 1–6 (1985), DOI: 10.1086/162755

5. Chen, W.X.: Neutrino-cooled Accretion Disks around Spinning Black Holes. In: AIP Conference Proceedings. vol. 836, pp. 193–196. AIP (2006), DOI: 10.1063/1.2207888

6. Cowperthwaite, P., et al.: The Electromagnetic Counterpart of the Binary Neutron Star Merger LIGO/Virgo GW170817. II. UV, Optical, and Near-infrared Light Curves and Comparison to Kilonova Models. The Astrophysical Journal 848(2), L17–L30 (2017), DOI: 10.3847/2041-8213/aa8fc7

7. Di Matteo, T., Perna, R., Narayan, R.: Neutrino Trapping and Accretion Models for Gamma-Ray Bursts. The Astrophysical Journal 579(2), 706–715 (2002), DOI: 10.1086/342832

8. Fishbone, L., Moncrief, V.: Relativistic fluid disks in orbit around Kerr black holes. The Astrophysical Journal 207(1), 962–976 (1976), DOI: 10.1086/154565

9. Font, J.A., Ibáñez, J.M., Papadopoulos, P.: A "Horizon-adapted" Approach to the Study of Relativistic Accretion Flows onto Rotating Black Holes. The Astrophysical Journal 507(1), L67–L70 (1998), DOI: 10.1086/311666

10. Frank, J., King, A., Raine, D.: Accretion Power in Astrophysics: Third Edition. Cambridge University Press (2002), DOI: 10.1063/1.2815178

11. Gammie, C., McKinney, J., Tóth, G.: HARM: A Numerical Scheme for General Relativistic Magnetohydrodynamics. The Astrophysical Journal 589(1), 444–457 (2003), DOI: 10.1086/374594

12. Janiuk, A., Yuan, Y.F.: The role of black hole spin and magnetic field threading the unstable neutrino disk in gamma ray bursts. Astronomy and Astrophysics 509, A55 (2010), DOI: 10.1051/0004-6361/200912725

13. Janiuk, A.: Microphysics in the Gamma-Ray Burst Central Engine. The Astrophysical Journal 837(1), 39–50 (2017), DOI: 10.3847/1538-4357/aa5f16

14. Janiuk, A., Mioduszewski, P., Moscibrodzka, M.: Accretion and outflow from a magnetized, neutrino cooled torus around the gamma-ray burst central engine. The Astrophysical Journal 776(2), 105–116 (2013), DOI: 10.1088/0004-637X/776/2/105

15. Janiuk, A., Perna, R., Di Matteo, T., Czerny, B.: Evolution of a neutrino-cooled disc in gamma-ray bursts. Monthly Notices of the Royal Astronomical Society 355(3), 950–958 (2004), DOI: 10.1111/j.1365-2966.2004.08377.x

16. Janiuk, A., Yuan, Y., Perna, R., Di Matteo, T.: Instabilities in the time-dependent neutrino disk in gamma-ray bursts. The Astrophysical Journal 664(2), 1011–1025 (2007), DOI: 10.1086/518761

17. Kohri, K., Mineshige, S.: Can Neutrino-cooled Accretion Disks Be an Origin of Gamma-Ray Bursts? The Astrophysical Journal 577(1), 311–321 (2002), DOI: 10.1086/342166

18. Lee, W.H., Ramirez-Ruiz, E.: Accretion modes in collapsars – prospects for GRB production. The Astrophysical Journal 641(2), 961–971 (2005), DOI: 10.1086/500533

19. Li, L.X., Paczyński, B.: Transient events from neutron star mergers. The Astrophysical Journal Letters 507(1), L59–L62 (1998), DOI: 10.1086/311680

20. Mészáros, P.: Gamma-ray bursts. Reports on Progress in Physics 69(8), 2259 (2006), DOI: 10.1088/0034-4885/69/8/R01

21. Meyer, B.: WEBNUCLEO.org. In: Nuclei in the Cosmos (NIC XII). p. 96 (2012), `http://adsabs.harvard.edu/abs/2012nuco.confE..96M`, accessed: 2018-04-01

22. Meyer, B.S., Adams, D.C.: Libnucnet: A Tool for Understanding Nucleosynthesis. Meteoritics and Planetary Science Supplement 42, 5215 (2007), `http://adsabs.harvard.edu/abs/2007M%26PSA..42.5215M`, accessed: 2018-04-01

23. Mirabel, I.F., Rodriguez, L.F.: Sources of relativistic jets in the galaxy. Annual Review of Astronomy and Astrophysics 37(1), 409–443 (1999), DOI: 10.1146/annurev.astro.37.1.409

24. Paczynski, B.: Gamma-ray bursters at cosmological distances. The Astrophysical Journal 308(2), L43–L46 (1986), DOI: 10.1086/184740

25. Perego, A., et al.: Neutrino-driven winds from neutron star merger remnants. Monthly Notices of the Royal Astronomical Society 443(4), 3134–3156 (2014), DOI: 10.1093/mnras/stu1352

26. Popham, R., Woosley, S.E., Fryer, C.: Hyperaccreting Black Holes and Gamma-Ray Bursts. The Astrophysical Journal 518(1), 356–374 (1999), DOI: 10.1086/307259

27. Reddy, S., Prakash, M., Lattimer, J.M.: Neutrino Interactions in Hot and Dense Matter. Physics Letters B 58(1), 013009 (1997), DOI: 10.1103/PhysRevD.58.013009

28. Rees, M.M.J.: Black hole models for active galactic nuclei. Annual review of astronomy and astrophysics 22(1), 471–506 (1984), DOI: 10.1146/annurev.aa.22.090184.002351

29. Smartt, S.J.e.a.: A kilonova as the electromagnetic counterpart to a gravitational-wave source. Nature 551(7678), 75–79 (2017), DOI: 10.1038/nature24303

30. Tanaka, M.: Kilonova/macronova emission from compact binary mergers. Advances in Astronomy 2016, 6341974 (2016), DOI: 10.1155/2016/6341974

31. Tanvir, N.R., et al.: A "kilonova" associated with the short-duration γ-ray burst GRB 130603B. Nature 500, 547–549 (2013), DOI: 10.1038/nature12505

32. Tchekhovskoy, A.: Launching of Active Galactic Nuclei Jets. In: Contopoulos, I., Gabuzda, D., Kylafis, N. (eds.) The Formation and Disruption of Black Hole Jets. Astrophysics and Space Science Library, vol. 414, pp. 45–82 (2015), DOI: 10.1007/978-3-319-10356-3_3

33. Urry, C.M., Padovani, P.: Unified schemes for radio-loud active galactic nuclei. Publications of the Astronomical Society of the Pacific 107(715), 803–845 (1995), DOI: 10.1086/133630

34. Wallerstein, G., et al.: Synthesis of the elements in stars: forty years of progress. Review of Modern Physics 69(4), 995–1084 (1997), DOI: 10.1103/RevModPhys.69.995

35. Yuan, Y.F.: Electron-positron capture rates and a steady state equilibrium condition for an electron-positron plasma with nucleons. Physical Review D - Particles, Fields, Gravitation and Cosmology 72(1), 1–8 (2005), DOI: 10.1103/PhysRevD.72.013007