


The Roofline Analysis of Special Relativistic Hydrodynamics Coarray Fortran Code

*Igor M. Kulikov*¹ , *Igor G. Chernykh*¹, *Dmitry A. Karavaev*¹,
*Vladimir G. Prigarin*¹, *Anna F. Sapetina*¹, *Ivan S. Ulyanichev*¹,
*Oleg R. Zavyalov*¹

© The Authors 2023. This paper is published with open access at SuperFri.org

Our previous papers are dedicated to the development of the first code for computational astrophysics using Coarray Fortran technology. The main result of the study of the developed code is the achievement of weak scalability at the level of MPI implementations, which allows to fully concentrate on using Coarray Fortran for developing new program codes. Coarray Fortran is based on the MPI directives, and helps software developer to create simple code without Send/Receive or synchronization commands. At the same time, such scalability can be achieved due to the weak implementation of the sequential part of the program code, which is characterized by frequent cache misses, inefficient memory usage and poor overall performance. In this article, we propose a method for analyzing program code performance using the roofline analysis. We used Intel Advisor software package from Intel oneAPI toolkit. High performance and efficient work with the memory of both individual key procedures and the code as a whole are demonstrated.

Keywords: HPC analysis, Coarray Fortran, high performance computing, roofline analysis.

Introduction

Many astrophysical phenomena, such as relativistic jets in active galactic nuclei [1], are characterized by relativistic velocities. Relativistic jets play an essential role in several important astrophysical processes, such as star formation, galactic binaries interaction, microquasars, active galaxies, and quasars physics. The main tool for studying relativistic jets is the mathematical modeling using high-performance computing systems [2]. We can find a lot of astrophysical papers dedicated to new astrophysical codes. But most of codes cannot be used for numerical simulation of real big problems, because they are not suitable to run on high-performance computing clusters. There are different strategies to parallelize computational code. It depends on mathematical and numerical models. In our case, because of the hydrodynamical approach, one of the best ways to maximize the performance is to optimize efficiency on each node using OpenMP and deep vectorization. Then we need to optimize data exchange between computational nodes of a cluster. In our research, we are using Coarray Fortran, which is based on MPI technology but is more reliable for software development. The most important advantage of the MPI 3.0 standard is the effective implementation of one-way communications, allowing to move to one of the most promising parallel programming models PGAS (Partitioned Global Address Space). Computational experiments on continuum mechanics models [3–5] have shown that Coarray Fortran code has the same scalability as MPI code. At the same time, the complexity of code development is noticeably reduced, which leads to the development of new libraries [7] and language extensions [8]. We should also note that the multidimensionality of the decomposition of calculations does not affect the performance of the code [9, 10]. Based on the Coarray Fortran technology, we have developed a new code for the numerical solution of equations of special relativistic hydrodynamics [6]. In this paper, we analyze the code's performance using Intel Advisor software from the Intel OneAPI toolkit.

¹Institute of Computational Mathematics and Mathematical Geophysics SB RAS, Novosibirsk, Russian Federation

In the next section, we briefly describe the structure of the developed code. The third section is devoted to the analysis of software implementation using The Roofline Analysis. The Roofline Analysis is very important for optimizing the code's performance by vectorization of loops inside a program. The fourth section will present the simulation results. The fifth section formulates the conclusion.

1. Special Relativistic Hydrodynamics Coarray Fortran Code

The mathematical apparatus and parallel implementation are described in detail in [6]. We will focus on the structure of the code presented in Fig. 1. The figure shows an enlarged block

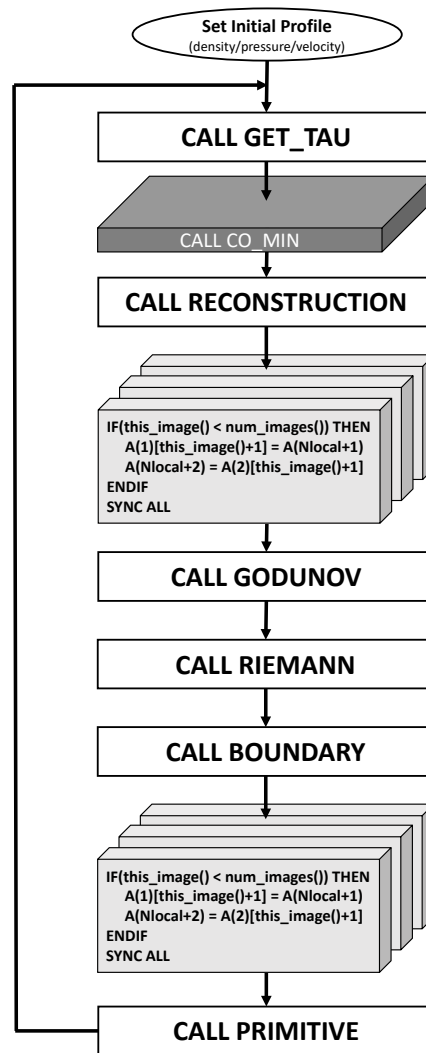


Figure 1. Code structure

diagram of the software implementation of the main computing cycle. The **GET_TAU** function calculates the time step in each subdomain. Then we use the Coarray Fortran reducing function **CO_MIN** to calculate the global minimum time step. At the next step, the **RECONSTRUCTION** function is called for a piecewise parabolic representation of physical variables (here we do not provide the entire list of functions). Its implementation requires an exchange of

overlap areas using Coarray Fortran. At the next stage, the **GODUNOV** function is called to implement the numerical method, which uses the **RIEMANN** and **BOUNDARY** functions. The latter function also requires the exchange of overlap areas. Then the physical variables are restored in the **PRIMITIVE** function.

2. Roofline Analysis

In our research, we used the Roofline model [11] to provide performance estimates of our astrophysical code running on a compute node based on two Intel Xeon Scalable 6248R processors with 24 cores each. Each node has 192 GB DDR4 RAM. For our tests, we did not use any data output for maximum performance. Roofline analysis was made by Intel Advisor [12] software from Intel oneAPI [13] software package. This software calculates performance values for each function of our code as well as the most compute-bound function. We can also compare performance values with peak performance values (scalar peak, single precision peak performance values, double precision vector peak, double precision FMA peak) of processors.

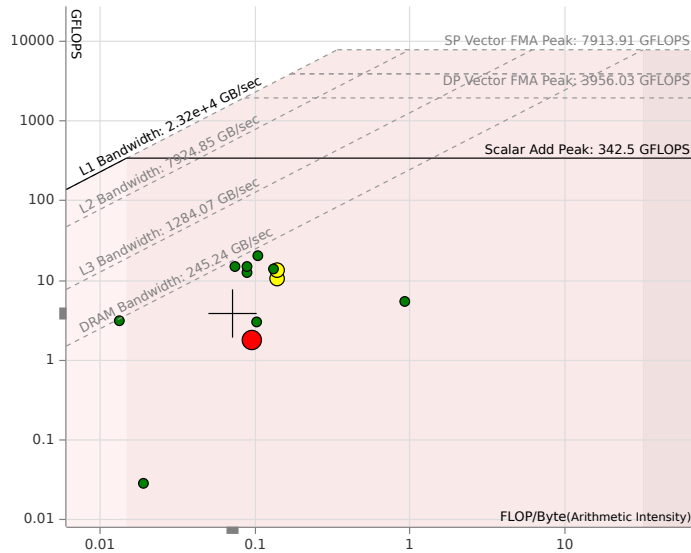


Figure 2. Roofline analysis results. Autovectorization by compiler turned off. Computational node: 2x Intel Xeon 6248R, 192 GB DDR4 RAM

This analysis can help to improve the performance of each function in developed code because Intel Advisor shows loops that can be autovectorized and loops whose structure should be improved for autovectorization. Why do we need to think about loop vectorization every second during the development of high-performance computing software? The peak performance of modern x86 processors from Intel or AMD as well as modern ARM processors is based on the CPU vector units. We can see that the scalar add peak performance for our compute node is about 342 GFLOPS and the dual precision vector FMA peak is about 3956 GFLOPS. The difference is about 10 times. Each core of the Intel Xeon Scalable Gold/Platinum processor has AVX512 registers and FMA mathematics instructions. These instructions can multiply and add eight double precision values at one CPU cycle. It means that we should help the compiler build vectorized loops for maximum performance. In this section, we will show the importance of auto-vectorization for code performance.

Figure 2 shows the results of roofline analysis for our astrophysical code without any compiler vectorization optimizations. The red dot is the the performance of the building parabola function

of the PPML method. Green and yellow dots are the performance of the other functions such as Riemann solver or part of a Godunov scheme. The total performance of the code is equal to 3.79 GFLOPS. The performance of the function with the longest calculation time (PPML method realization) is equal to 1.8 GFLOPS. Also, we can see that all functions in this optimization case are limited by the DRAM memory bandwidth.

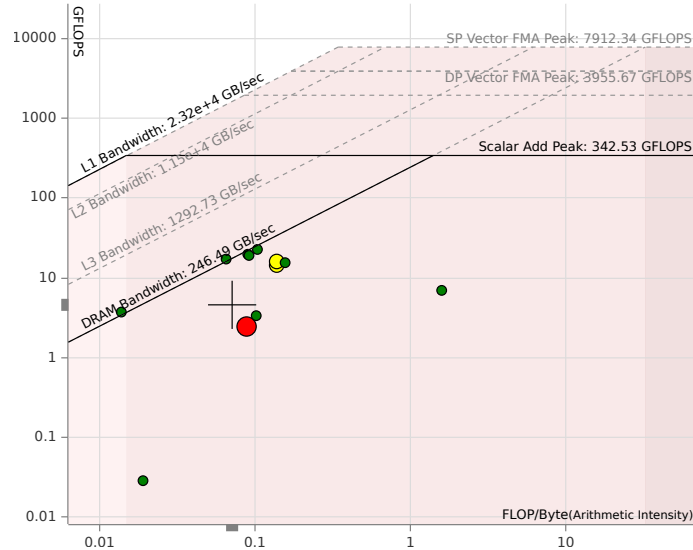


Figure 3. Roofline analysis results. AVX2 autovectorization by compiler. Computational node: 2x Intel Xeon 6248R, 192 GB DDR4 RAM

Figure 3 shows the results of roofline analysis for our code with AVX2 auto-vectorization optimizations made by Intel’s compiler. The red dot is the the performance of the building parabola function of the PPML method. Green and yellow dots are the performance of the other functions such as Riemann solver or part of a Godunov scheme. The total performance of the code is equal to 4.61 GFLOPS. The performance of the function with the longest calculation time is equal to 2.5 GFLOPS. This is the same function as shown in Fig. 2. We can see that the Riemann solver bandwidth in this optimization case became a little bit better than the DRAM memory bandwidth. AVX2 optimized code works with the math-related intrinsic functions which can use 256-bit double precision vectors containing 4 doubles. AVX2 provides instructions that fuse multiplication and addition by using FMA intrinsics which also works with 256-bit double precision vectors. These vector instructions are suitable for most Intel and AMD processors. But if you have the latest Intel Xeon Scalable processors you should use AVX512 vectorization for better performance. The next figure will show the advantages of AVX512 vector registers.

Figure 4 shows the results of roofline analysis with AVX512 auto-vectorization optimizations made by Intel’s compiler. The red dot is the the performance of the building parabola function of the PPML method. Green and yellow dots are the performance of the other functions such as Riemann solver or part of a Godunov scheme. The total performance of the code is equal to 7.73 GFLOPS. The performance of the function with the longest calculation time is equal to 3.66 GFLOPS. This is also the same function as shown in Figs. 2–4. We can see that the Riemann solver bandwidth in this optimization case is about 435 GB/sec with a performance of about 28.4 GFLOPS. AVX512 optimized code works with the math-related intrinsic functions which can use 512-bit double precision vectors containing 8 doubles. AVX512 provides instructions that fuse multiplication and addition by using FMA intrinsics which also works with 512-bit

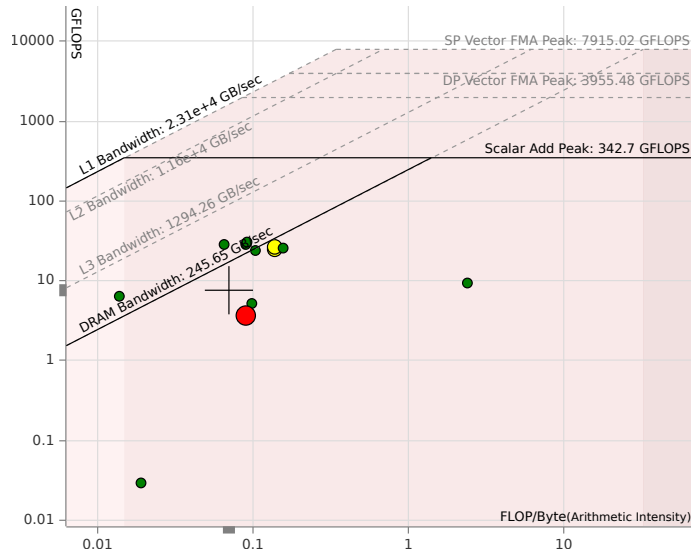


Figure 4. Roofline analysis results. AVX512 autovectorization by compiler. Computational node: 2x Intel Xeon 6248R, 192 GB DDR4 RAM

double precision vectors. The average estimated speed-up of vectorized code compared to the scalar version is equal to 5.5 times. And this is not the best result. Intel Advisor suggests some optimizations that can help build faster code. These optimizations are to add data padding, vectorize serialized functions, and convert some functions to Fortran SIMD-enabled functions.

For collecting the performance data, we use the same technique as in [14]. We did not change the source of our astrophysical code during tests. We only change the target architecture by adding `-ax` compiler option with a set of processor’s instructions which can be used for target code.

3. Numerical Modeling

The formulation of the problem of the relativistic jet evolution can be found in [2]. Figure 5 shows the results of modeling the evolution of the galactic jet. From the simulation results it is clear that a shock wave moves forward, the speed of waves propagation corresponds to the speed of light. Behind the shock front, there is a shell that separates the shock front and the hot region where the maximum temperature is reached. The internal part of the flow has a cocoon and is limited by the contact surface. On the outer side of the cocoon, closer to the base, currents of the reverse flow type propagate, which in turn interact with the jet flow. The characteristic development time of Kelvin–Helmholtz-type instability at the base of the jet is 6000 years, which corresponds to the results of the computational experiment.

Conclusions

Last decade, our group developed codes for numerical simulation of different astrophysical problems. We developed different codes based on the CUDA toolkit, MPI, and OpenMP technologies as well as C++ or Fortran languages. We had some C++ implementations based on AVX512 intrinsics, and this code version has the best performance on Intel Xeon Scalable processors. But this code cannot be used on AMD processors till AMD starts to produce their AVX512-based CPUs. Our latest codes are based on the Coarray Fortran extension, which was started as an extension of Fortran 95/2003 for parallel processing. At this time, Coarray Fortran-

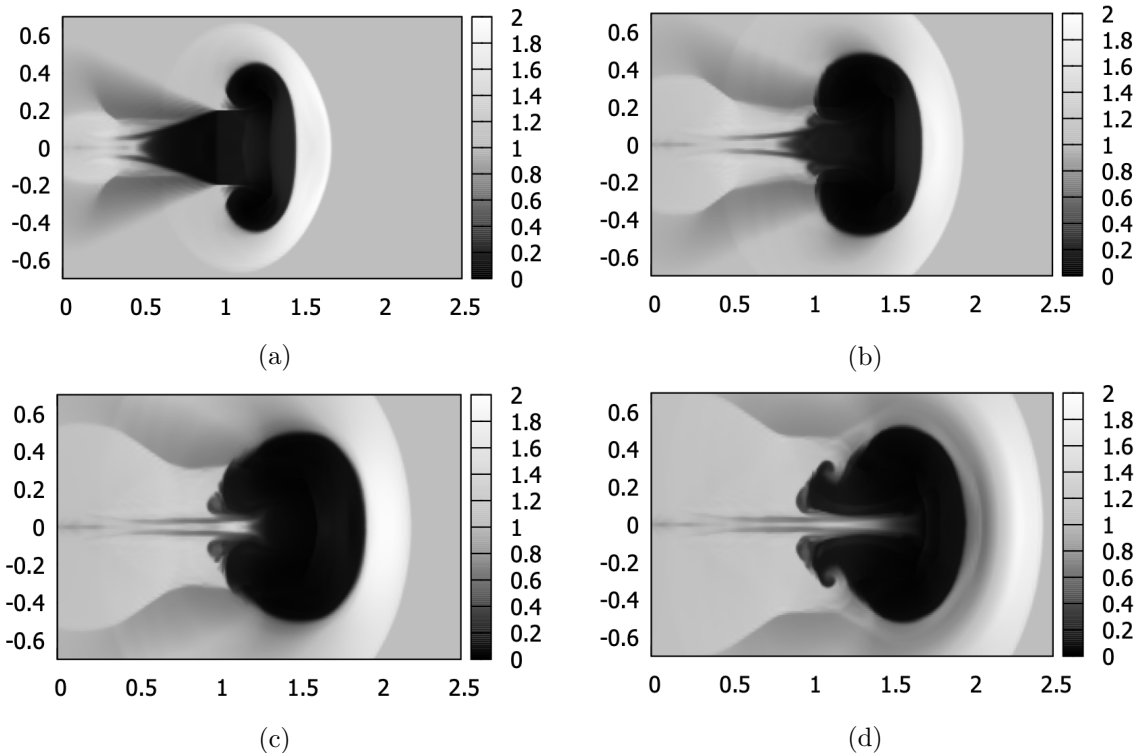


Figure 5. Gas density in the equatorial plane is 10^{-2} cm^{-3} at time points: 3000 years (a), 4500 years (b), 6000 years (c), 7500 years (d)

based codes have the same performance as the MPI codes. However, the creation of the program is much easier than that of the MPI code. Modern Fortran compilers understand this extension and can optimize codes. In this paper, we focused on the auto-vectorization results of the Intel Fortran compiler. We achieved two times the performance speed-up of our astrophysical code only by the compiler options. It is possible to speed up our code more in future with some recommendations from the Intel Advisor toolkit which was used for performance evaluation. The simulation results for the evolution of relativistic jet are in good accordance with the results from our earlier codes based on C++ MPI/OpenMP technologies. In our future works, we will continue to optimize our code, possibly for Advanced Matrix Extensions instruction set.

Acknowledgements

This work was supported by the Russian Science Foundation (project No. 23-11-00014) <https://rscf.ru/project/23-11-00014/>.

This paper is distributed under the terms of the Creative Commons Attribution-Non Commercial 3.0 License which permits non-commercial use, reproduction and distribution of the work without further permission provided the original work is properly cited.

References

1. Sotomayor, P., Romero G.: Nonthermal radiation from the central region of super-accreting active galactic nuclei. *Astronomy & Astrophysics* 664, A178 (2022). <https://doi.org/10.1051/0004-6361/202243682>
2. Kulikov, I.: A new code for the numerical simulation of relativistic flows on supercomputers

- by means of a low-dissipation scheme. *Computer Physics Communications* 257, 107532 (2020). <https://doi.org/10.1016/j.cpc.2020.107532>
3. Reshetova, G., Cheverda, V., Khachkova, T.: A comparison of MPI/OpenMP and Coarray Fortran for digital rock physics application. In: *Parallel Computing Technologies. PaCT 2019. Lecture Notes in Computer Science*, vol. 11657, pp. 232–244 Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25636-4_19
 4. Reshetova, G., Cheverda, V., Khachkova, T.: Numerical experiments with digital twins of core samples for estimating effective elastic parameters. In: *Supercomputing. RuSC-Days 2019. Communications in Computer and Information Science*, vol. 1129, pp. 290–301. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36592-9_24
 5. Reshetova, G., Cheverda, V., Koinov, V.: Comparative efficiency analysis of MPI blocking and non-blocking communications with Coarray Fortran. In: *Supercomputing. RuSC-Days 2021. Communications in Computer and Information Science*, vol. 1510, pp. 322–336. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-92864-3_25
 6. Kulikov, I., Chernykh, I., Karavaev, D., *et al.*: A new parallel code based on a simple piecewise parabolic method for numerical modeling of colliding flows in relativistic hydrodynamics. *Mathematics* 10(11), 1865 (2022). <https://doi.org/10.3390/math10111865>
 7. Wang, Y., Li, Z.: GridFOR: a domain specific language for parallel grid-based applications. *International Journal of Parallel Programming* 44, 427–448 (2016). <https://doi.org/10.1007/s10766-014-0348-z>
 8. Kataev, N., Kolganov, A.: The experience of using DVM and SAPFOR systems in semi automatic parallelization of an application for 3D modeling in geophysics. *The Journal of Supercomputing* 75, 7833–7843 (2019). <https://doi.org/10.1007/s11227-018-2551-y>
 9. Shterenlikht, A., Cebamanos, L.: MPI vs Fortran coarrays beyond 100k cores: 3D cellular automata. *Parallel Computing* 84, 37–49 (2019). <https://doi.org/10.1016/j.parco.2019.03.002>
 10. Guo, P., Wu, J.: One-sided communication in Coarray Fortran: performance tests on TH-1A. In: *Algorithms and Architectures for Parallel Processing. ICA3PP 2018. Lecture Notes in Computer Science*, vol. 11337, pp. 21–33. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-05063-4_3
 11. The Roofline Model. https://en.wikipedia.org/wiki/Roofline_model (2023), accessed: 2023-10-25
 12. Intel Advisor tutorial. <https://www.intel.com/content/www/us/en/docs/advisor/tutorial-roofline/2021-1/run-a-roofline-analysis.html> (2021), accessed: 2023-10-25
 13. Intel oneAPI overview. <https://www.intel.com/content/www/us/en/developer/tools/oneapi/overview.html> (2023), accessed: 2023-10-25
 14. Chernykh, I., Vorobyov, E., Elbakyan, V., Kulikov, I.: The impact of compiler level optimization on the performance of iterative Poisson solver for numerical modeling of protostellar disks. In: *Supercomputing. RuSCDays 2021. Communications in Computer and Information Science*, vol. 1510, pp. 415–426. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-92864-3_32