Silesian J. Pure Appl. Math.
vol. 8, is. 1 (2018), 49–56

Tomasz JANUSZEK[1], Mariusz PLESZCZYŃSKI[2]

[1]Faculty of Applied Mathematics, Silesian University of Technology, Gliwice, Poland

[2]Institute of Mathematics, Silesian University of Technology, Gliwice, Poland

COMPARATIVE ANALYSIS OF THE EFFICIENCY OF JULIA LANGUAGE AGAINST THE OTHER CLASSIC PROGRAMMING LANGUAGES

Abstract. In this paper we analyze the performance of five popular programming languages. The efficiency analysis involves the comparison of elapsed time needed for executing the same computations when the degree of complexity rises. In this paper we compare C#, Python, R, Wolfram and Julia programming languages.

1. Introduction

In present times every engineer or mathematician that applies the queen of sciences for solving technical problems (as well as for verifying some theoretical hypothesis) helps himself by using a computer. The degree of complexity of some problems may be so high that even for a very efficient computing device it may take weeks to solve a problem. Therefore it is necessary to choose suitable programming language for a specific problem. The decision may depend on many factors, one of them is the utility of the particular language. Some of the programming

2010 Mathematics Subject Classification: 6504, 6804, 68N15.
Keywords: computational complexity, programming languages, Julia language.
Corresponding author: M. Pleszczyński (mariusz.pleszczynski@polsl.pl).
Received: 20.09.2018.

languages are dedicated for some specific type of problems, but their intensive evolution causes that in recent times every engineering problem can be solved with the aid of any such language. Therefore the language utility is not the most important element any more (the selection of particular language depends more on the programmer's habit). The main criterion in choosing the programming language became its efficiency. In this paper we will compare the time needed for performing specific task in the specified language. We will divide the investigated tasks into two categories:

a) concerning the speed of executing the arithmetic operations,

b) concerning the speed of executing the arithmetic operations connected with the simultaneous memory read/write operations.

2. Languages characteristics

We shortly describe the main features and usage of languages used in our paper.

2.1. C# language

C# language [1, 6] was made by Microsoft company in 2000. The purpose of making the new programming language was a need to create the objective oriented equivalent of C++ dedicated to Microsoft .NET platform. Anders Hejlsberg was a leader of the development team and he stated that C# language was, in fact, based on C++. However Java creators complained that it is a copy of their programming language. Today C# is mainly used for creating desktop applications for Windows systems using WPF subsystem, but it is increasingly used for web applications development using ASP.NET framework and mobile applications as well. Characteristic feature of C# is LINQ (Language Integrated Query) framework, which allows the programmers to create SQL queries and perform the operations on data collections by using the built-in expressions. Because of incompatibility of .NET framework with other systems, Microsoft created Core version of .NET that allows the development of multi-platform applications. Due to fact that C# and .NET are not open-source, Microsoft team created the open-source project called Mono, which gained the appreciation from programmers from around the world.

2.2. Julia language

Development of this programming language started in 2009. The team of independent programmers: Jeff Bezanson, Stefan Karpinski, Viral B. Shah and Alan Edelman worked for 3 years before announcing the first version of Julia in 2012 on their website [9].

The aim of creating new programming language was the necessity of merging multi-purpose, high efficiency language for both numerical analysis and computational science. Nowadays this language is widely used for any purposes related to data processing, calculations and mathematics in general. It combines dynamic types, multiple dispatch, built-in package manager and possibility to call Python and C-family languages functions freely. Although Julia is not so popular in Poland, it is used worldwide by the well-known organizations e.g. Federal Reserve Bank, Aviva or Celeste project [13].

From August 2018 the Julia version 1.0.0 is finally released, so we could use it in this paper.

2.3. Wolfram language

Wolfram language in 1.0 version was announced in 1989 as a programming language used in computing system Mathematica. It was conceived by English physicist and mathematician Stephen Wolfram, the founder of Wolfram Research company, which is in charge of Mathematica and Wolfram language development till today. Its purpose is focused on symbolic computation, which is impossible to maintain for most of the mathematical programs, and is fully controlled by the specifying arguments like computation precision. Mathematica possesses a huge base of mathematical algorithms and fully supports the paralleled calculations. It also offers various methods of data visualization [8, 14].

2.4. Python language

Idea of creation such programming language came into being in 1980, but emerged nine years later, in 1989. Creator of Python language, Guido van Rossum, wanted to form a language capable of using many programming paradigms, as well as dynamic programming. But the most important trait of this language was and will be the clean code [5].

Nowadays we hear about Python in the context of machine learning [12] and quantum computations. Immense community of programmers is developing many packages [10] that makes coding easier and faster. The packages database contains

over 130 000 packages for every known field of informatics e.g. database operating, data analysis, networking or even applications development. Pros of this language are: using any programming paradigm whilst coding, huge package repository, ease of learning and code understanding, good code structuring.

The newest available Python version is 3.7.0 and this one is used in this paper.

2.5. R language

R language [2, 3] was conceived in 90s and its purpose was to provide free, easy and versatile coding language for statistical computing. Its name originated from the first letter of both creators first names: Robert Gentleman and Rossa Ihake. They were researchers at the University of Auckland in New Zealand. The high utility of R language led to its popularization, so in 1997 there were nearly one hundred mathematicians and computer scientists that worked on its improvements. In fact, R is an interpreted language and is based on thousands of supporting packages, which help in its usage. If needed, one can call functions from other programs libraries or create very complex and various charts.

R language exists in version 3.5.1 from July 2018, and this version is used in this paper.

3. Comparison analysis

In this section we will study and compare the efficiency of all mentioned programming languages by performing the operations described in introduction section 1. As an environment we use 64-bit Windows 7, when both systems use Intel®Core^{TM}i7-5820K and 16GB RAM.

3.1. Algorithm 1

In this particular task we do not use any built-in complex computation methods, only the nested loops and basic operations of arithmetic, that is the addition, subtraction, multiplication and division.

The operations are done accordingly to the pseudocode presented below:

```
FOR i in bounds from 1 to n
    FOR j in bounds from 1 to n
        FOR k in bounds from 1 to n
```

```
            COMPUTE (i - j) / (i + j)
        ENDFOR
    ENDFOR
ENDFOR
```

where for n we mean a given value e.g. $n = 1, 2, 3, \ldots, 100$.

For languages that start the array indexing from 0, we shifted each index by -1 and we add 2 in denominator. When operation occurs, we measure the execution time by using the best built-in processor time measurement functions in each language. Thus, we perform n^3 additions, n^3 subtractions and n^3 divisions, in total of $3n^3$ arithmetic operations. We collected the acquired data for different n. The results can be seen in Figure 1.

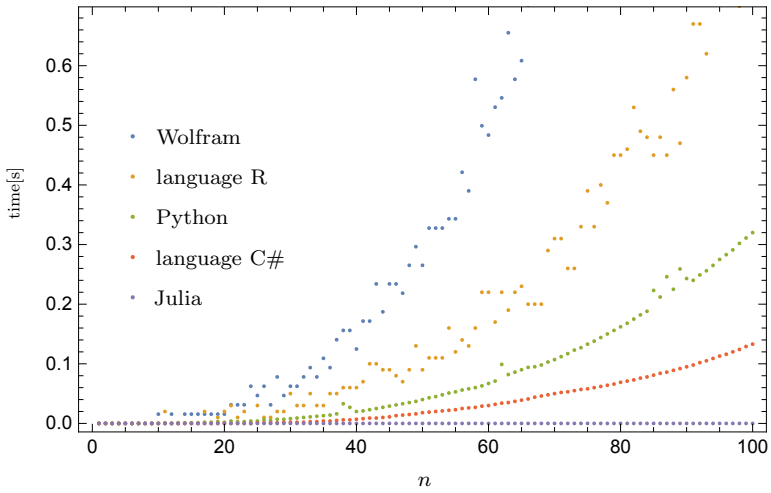

Fig. 1. Working times of programs in algorithm 1

3.2. Algorithm 2

In this task we multiply two square matrices A and B including n rows and columns. To measure only the real execution time of multiplication, we do not count any matrix generation as a part of this operation. Because of possibility that random matrix numbers may have impact on multiplication time, we fill both A and B matrices with one randomized value. Multiplication of matrices is executing according to the definition. Let C be the resulting matrix of multiplying matrices A and B. Then we can describe each element of matrix C by using the following formula:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj},$$

which will be summed up by using the loops for matrix rows and columns.

Therefore, we note that for each element of matrix C we need to execute n multiplications and $n-1$ additions. If A and B have got n rows and columns, we need to execute n^3 multiplications and $n^2(n-1)$ additions, which gives in total $n^2(2n-1)$ arithmetic operations. This means the polynomial computational complexity of rank n^3.

Similarly as for previous algorithm, we collected the acquired data for different n and visualized the results in Figure 2.

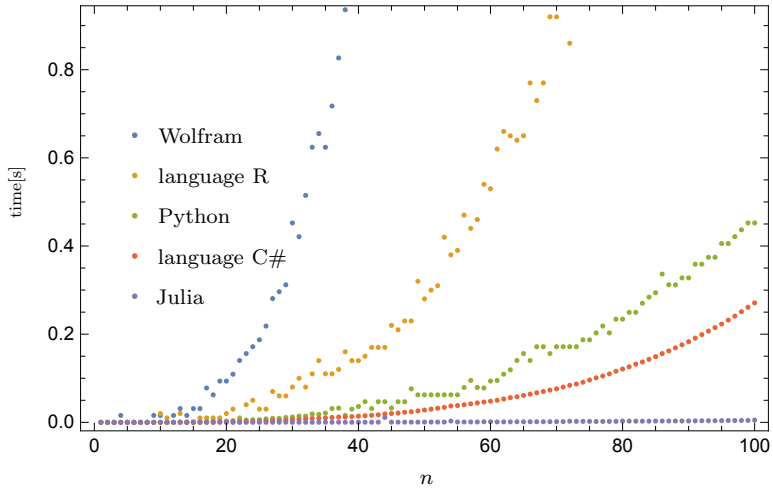

Fig. 2. Working times of programs in algorithm 2

Additionally, one should emphasize that in Figures 1 and 2 there are presented the exemplary results obtained for one series of tests. Such tests were executed many times and their averaged results do not significantly differ from the results presented in this paper, which confirms again the determination of efficiency hierarchy of the investigated programs. The standard deviations of results behave similarly, that is their values decrease when the algorithm efficiency increases, and this fact strengthens also the usefulness of the most efficient programs, especially of the Julia program.

4. Conclusions

Our results show that the slowest programming language, from among the ones taken into consideration, is Wolfram. In both examined cases it requires more time to execute the specified operations than any other language. However, low efficiency in case of these particular algorithms does not represent the overall performance of Wolfram. It remains still one of the best languages designed for symbolic computation.

We observe the better efficiency for R language, which managed significantly better in performing both algorithms, but still much worse in comparison with other programming languages. R is surely useful in data analysis and reaches outstanding results in statistical computing, notwithstanding it would be recommended to use other tools for tasks shown in this experiment.

Python appears to be included to the group of three leading languages. It is much more efficient than Wolfram and R, but the algorithms are still performed slower than by applying C# and Julia. The obtained results show that it is reasonable to use Python as a tool for such operations and it works great at its field.

Surprisingly, C# language was placed the second best language in this experiment, although it was not designed for the purposes investigated here. It may be caused by the close coupling between .NET Common Language Runtime and Windows 7 system. However, as it performed very good in the discussed cases, it may happen to get much worse results in more complex equations.

Concerning the whole set of investigated languages, the Julia language appeared to be the fastest and the more efficient and it declasses the other languages. The goal of this paper was to reveal the efficiency of this, still not to much popular, programming language, which is confirmed by the results presented in this elaboration, and to encourage the Readers to study and to apply this language. Some further investigations followed the ones executed in works [4, 7, 11], as well as the obtained results will be presented in the new paper which will be the continuation and extension of the subject discussed in the current paper.

References

1. Albahari J., Albahari B.: *C# 7.0 in a Nutshell: The Definitive Reference*. O'Reilly Media, Sebastopol 2017.
2. Biecek P.: *Survay for R package*. Oficyna wydawnicza GiS, Wrocław 2014 (in Polish).
3. Crawley M.J.: *The R book*, John Wiley & Sons, Chichester 2007.
4. Damaševičius R., Štuikys V.: *Metrics for evaluation of metaprogram complexity*, Comput. Sci. Inf. Sys. **7**, no. 4 (2010) 770–787.
5. Dawson M.: *Python Programming for the Absolute Beginner*. Course Technology, Boston 2010.
6. Evjen B., Hanselman S., Rader D.: *Professional ASP.NET 4 in C# and VB*. Wiley Publ., Indianapolis 2010.
7. Falola O., Misra S., Adewumi A., Damaševičius R.: *Evaluation and comparison of metrics for XML schema languages*. In: *Frontiers in Artificial Intelligence and Applications*, Mizera-Pietraszko J. et al. (eds.), IOS Press, Amsterdam 2017, 51–59.
8. Gliński H., Grzymkowski R., Kapusta A., Słota D.: *Mathematica 8*. Jacek Skalmierski Computer Studio, Gliwice 2012 (in Polish).
9. https://julialang.org/blog/2012/02/why-we-created-julia.
10. McKinley W.: *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. O'Reilly Media, Sebastopol 2018.
11. Misra S., Adewumi A., Fernandez-Sanz L., Damaševičius R.: *A suite of object oriented cognitive complexity metrics*. IEEE Access **6** (2018), 8782–8796.
12. Raschka S.: *Python: Machine Learning*. Packt Publ., Birmingham 2016.
13. Regier J., Pamnany K., Giordano R., Thomas R., Schlegel D., McAuliffe J., Prabhat: *Learning an astronomical catalog of the visible universe through scalable Bayesian inference*. arXiv:1611.03404.
14. Wolfram S.: *The Mathematica Book, Fifth Edition*, Wolfram Media, Champaign 2003.