IJISET - International Journal of Innovative Science, Engineering & Technology, Vol. 4 Issue 6, June 2017
ISSN (Online) 2348 – 7968 | Impact Factor (2016) – 5.264
www.ijiset.com

An Empirical Study of Programming Languages from the Point of View of Scientific Computing

Dharmendra Singh

Department of Computer Science, ARSD College (University of Delhi), New Delhi – 21, India

Abstract

Scientific Computing is a multidisciplinary activity and can be treated nowadays as the "third pillar of science". Computation is critically important for the success of many processes/programs and time critical situations. This work aims to find the suitability of a subset of programming languages from the viewpoint of numerical computation. Here we keep our focus on some mainstream and popular languages such as FORTRAN, C++, Python, Java, MATLAB, Mathematica and Julia. The key issues that need attention, language suitability towards scientific computing will be examined and various features of these languages will be discussed. Writing efficient scientific code requires architectural details of the machine on which computation is to be done.

Keywords: *Scientific Computation, FORTRAN, C++, Java, MATLAB, Julia, Python, Programming Languages, Verification, Performance Evaluation.*

1. Introduction

Solving real world problems requires physical phenomena to be modeled in the form of mathematical structured problems and requires a notation and syntactic structure for their description in the form of explicit procedure or algorithm. Such systems of notation are called programming languages [1]. Initially the programming was done in machine language, was scientific oriented and demanded awareness with internal architecture but later on with the advent of high level programming there was a shift in paradigm. Abstraction in the form of separation of concern achieved in some sense and languages became more user oriented.

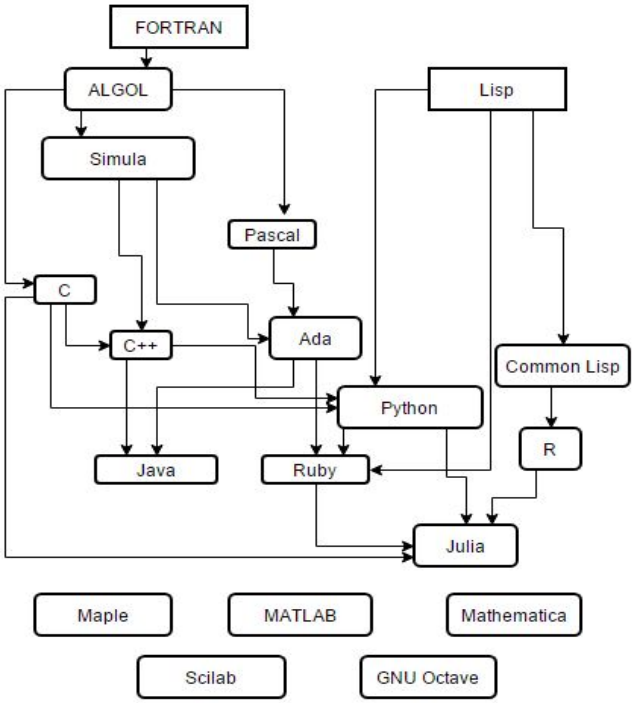
Scientific computing is challenging in the sense that it requires knowledge of scientific discipline to build the model as well as insight into language syntax to do efficient programming. Many computer models consist of three conceptual aspects [21]:
1. An abstract mathematical model (e.g., a system of partial differential equations);

2. A solution strategy (e.g., discretisation and approximation);

3. An implementation of this strategy (e.g., concrete, iterative code over arrays).

The condition number (Sensitivity measurement with respect to small changes in input) depends on the problem, not the algorithm used to solve it. The accuracy of a solution is limited by the condition number of the problem. Results can be unstable if it has an ill-conditioned sub problem [2]. Generally good programming practices and rules are avoided because it requires knowledge about software engineering principles and scientific programmers generally are scientist not the professional ones. Scientific code that performs well, is portable, and extensible takes a lot of programming effort [4]. So we need a language that takes less time in computation but still maintains the accuracy of the solution.

Fig. 1. Various languages and packages used for scientific computation.



1.1 Key issues in scientific computing

The goal of a scientific computation is rarely the exact answer, but a result that is as accurate as needed. There are four primary ways in which error is introduced into a computation [2]:

(i) Round off error.
(ii) Truncation error.
(iii) Termination of iterations.
(iv) Statistical error.

Errors can introduce at any point of time during the development process so verification is performed via mathematical analyses for checking correctness of developed software. Correctness of the underlying model is the process of checking that an implementation is free from programming errors. Validation means conforms to the already established standards or results. Without proper verification, it is difficult to tell whether incorrect results are due to an invalid model or program errors, or both. Even worse are results which appear correct due to program errors compensating for an invalid model. Type systems has been used to aid program verification by recognising and preventing large classes of programming errors [19, 20]. Historically, scope and fidelity of simulations were primary concerns between scientific community and then increasing the performance and efficiency. Domain-specific frameworks have become increasingly popular as environments in which a variety of applications in a given scientific domain can be constructed [3]. Other considerations for scientific computing include interface provided by language for embedding code, diversified data types for computing needs, portability and platform support for parallelism etc.

2. Overview of scientific programming language characteristics

Although a number of packages are available for computation purposes but still there is a need to develop programs according to the domain specific requirements. One can use scripting languages for some particular work and general purpose languages for performance related issues. Scientific computing requires various features to be desirable and built-in support provided by language makes them suitable for scientific programming. Some of the features that can enhance the suitability of language for computation intensive task include [8]:

- Support for multi-dimensional array.
- Specialized array representations.
- Support for complex data type.
- Parallelism support.

Apart from these features computation oriented systems often judged on the basis of performance, memory usage, robustness, error miniaturization. Language adherence to these issues makes them good candidate for scientific computational systems. Language should also provide portability, extensibility and productivity. Any platform for scientific programming should be able to use the efficient legacy code from other languages.

3. Language selected for the study and their brief overview

According to the well-cited Indexes such as TIOBE, Redmonk, IEEE Spectrum, CodeEval etc. of programming language popularity shows that languages like Java, C++, Python, MATLAB, Fortran and Julia are much popular among programmers worldwide. Most programming languages have both compiled implementations and interpretated implementations and the line making distinction between these two classes is very blurred but for the sake of simplicity we specified them as compiled and interpreted. Fortran and C++ both approved by the ISO standard committee. Although Java does not standardized but presiding by public acceptance and market forces. Java and C++ exceptions are a significant improvement over error codes returned by functions, but mixed-language programming became complicated [22].

3.1 Compiled Languages:

A pure compiler reads the source text of a program in a one go reports any error if present, and translates it into machine code. Rather than analyzing each expression each time they encounter it, compilers do the analysis once, but record the actions an interpreter would take at that point in the program. In effect, a compiler is a weird kind of interpreter [25].

i. **FORTRAN:** The original version of FORTRAN, FORTRAN I, was developed in 1957 by the IBM Corporation for the 704 computer. The language was designed by John Backus and his colleagues reduced the cost of scientific application development significantly by providing a notation closer to the scientific programming domain [5]. Fortran has been evolving over time to include the most recent proven ideas and concepts garnered from other programming languages [6] while maintaining its efficiency. Coarray feature enables the programmer writing concurrent code to take

the advantage of multi-core architectures. Template feature was not included in the latest Fortran 2008 standard due to the cost of virtual function calls [17]. Fortran supports complex data type.

Due to the backward compatibility still it lacks in some modern language constructs. Fortran is static typed language. An article, published by ACM Queue, The Ideal HPC Programming Language [15], argues that Fortran is still one of the popular and primary languages in high performance computing (HPC).

ii. **C++:** C++ is a successor of C and simula like features designed by Bjarne Stroustrup in 1983. C++ is unbeatable in many aspects, supports static typing and built-in support for complex data type is provided through <complex.h> header file. It supports generic programming in the form of templates. Templates are C++'s way of providing for parametric polymorphism, which allows using the same code at multiple types. The C++ Extensions for Parallelism have added components to the C++ standard library. It provides support for multiple inheritance. It provides high degree of abstraction but consumes more memory. Automatic garbage collection feature is not provided by C++ and must be done by programmer explicitly.

iii. **Java:** Java supports static typing. Multi-dimensional arrays in java can be treated as array of arrays i.e. nested array but this representation has some disadvantages as well, such as: memory allocator is responsible for allocating rows of an array so they can be scattered throughout memory results in poor cache behavior; array aliasing makes code optimization difficult; nested array presents challenge for garbage collection and data parallelism. Unification representation problem in specialized arrays makes them hard to use as a primitive data type but language extensions such as: (i) Parameterized classes (ii) Overloading of the subscript operator (iii) Tuples and vector tuples and (iv) Method inlining can help in this regard. No built-in support is provided in java for complex data type but making a complex class [9] and using a smart compiler is a viable solution. Java uses thread as a basic mechanism for parallelism but requires careful analysis due to complicated memory model and suffers from classical problems such as deadlock prevention [10]. Java does not supports template for the same reason as Fortran.

iv. **Julia:** Julia is a dynamic typed language. Its execution speed is only between 2.64 and 2.70 times slower than the execution speed of the best C++ compiler. Julia is a new open-source high-performance programming language with a syntax very close to Matlab, Lisp-style macros, and many other modern features that it inherited from other languages, and it also uses a just-in-time compiler for speed based on the LLVM. Three particularly attractive features of Julia are [18]: 1. Julia's default typing system is dynamic but it is possible to indicate the type of certain values to avoid type-instability problems that often decrease speed in dynamically typed languages. 2. Julia can call C or Fortran functions without wrappers or APIs. 3. Julia has a library that imports Python modules and provides wrappers for all of the functions on them.

Julia implements a distributed-memory, parallel execution model based on one-sided message passing [23]. Julia is an open-source language and supports multi-programming paradigm very well and gaining popularity among scientific community rapidly. Julia allows co-routines. User defined types in Julia are as fast and compact as built-ins. Provided LISP like macros and other meta programming abilities [24]. It provides support for complex data type.

3.1 Interpreted Languages

A pure interpreter reads the source text of a program line by line, analyzes it, and executes it as it goes. This is usually very slow--the interpreter spends a lot of time analyzing strings of characters to figure out what they mean [25].

i. **MATLAB:** MATLAB is a multi-paradigm language.The first intended usage of MATLAB, also known as Matrix Laboratory, was to make LINPACK and EISPACK available to students without knowning the Fortran complexity. Steve Bangert and Jack Little, along with Cleve Moler, founded MathWorks in 1983 keeping in mind the potential of the software. Apart from its own language MATLAb provides built-in functions based on LAPACK and BLAS that are coded in Fortran for the sake of efficiency. MATLAB supports dynamic typing system. MATLAB is much slower compared to C++ but performance can be enhanced with hybrid techniques.

MATLAB is more than a language it’s a complete package of various tools such as editor, extensive libraries for numerical computing and data visualization.

ii. **Python:** Python is a dynamically typed language and built-in support is provided for complex data type. Python allows inspection of the full state of the running program, introspection, and management of both

memory and variable typing, and easing unit testing [11, 12]. Python also supports generic programming. The numpy library [13] allows direct manipulation in Python of multi-dimensional arrays.

A key advantage in Python is the availability of numerous libraries due to open-source project. While Python modules for scientific computing are easy to use, they are more difficult to develop, because we must write binding code that connects Python to libraries of C functions and C++ classes. Before we launch Python, we must ensure that we have wrapped everything in those libraries that we might use [22].

iii. **Mathematica**: Wolfram language is the official language of Mathematica and initially was untyped but support for static typing has been extended for better code generation. Mathematica is more a computer algebra system package than a programming language. Support is provided for complex numbers and parallelism. It supports high-precision arithmetic by adopting GNU Multi-Precision Library. While taking advantage of language peculiarities mathematica performs better (only three times slower compared to C++) [18].

4. Evaluation Results and Discussions

In scientific computing multi paradigm approach is getting popular and trend is toward component based model and using libraries from the domain specific field but it comes with additional complexity but tradeoff between them seems to use them as an efficient method for developing software mostly due to performance issues [3]. Libraries hides the complexity of underlying high performance code and provides an abstract view that helps programmer to focus on problem rather than other issues.

When it comes to the criteria for selection of a language on merit basis in general then only relatively little high-quality objective information is available about the relative merits of different languages [7]. Compiled languages are less prone to runtime failures than interpreted languages due to type system adherence. Sometimes debates on programming languages are more religious than scientific. From an engineering viewpoint, programming languages comes with multiple tradeoffs that achieve certain desirable properties (such as speed) at the expense of others (such as simplicity) [14]. Scripting languages can reasonably be acceptable from productivity point of view for small projects that don't have much strict performance criteria over the conventional languages relative to the run time and memory consumption. Java slightly lacks in performance compared to C++.

5. Conclusions

The question of choosing an appropriate language for scientific language is subjective rather than objective because of a number of variables affects this decision. One should choose according to their comfertness with language, platform available, application aspects, performance issue, budget and of course time deadline for the project. A good language in the hand of less efficient programmer performs bad than non environment specific languages. A good language should provide abstractions, restrictions on code to prevent errors and aided verification via specifications. Language extensions can help to meet these considerations up to some extent.

There's no universally best language, for every scientific problem nor ever will be but it seems that Fortran will remain the choice of scientist for scientific computing providing better features and support without comprising its performance.

References

[1] F. P. Mathur, " A Brief Description and Comparison of Programming Languages FORTRAN, ALGOL, COBOL and LISP ", Technical Memorandum, JET propulsion laboratory California institute of technology, September 1972, 33-566.

[2] D. Bindel and J. Goodman, " Principles of Scientific Computing", March 2009, http://www.cs.nyu.edu/courses/spring09/G22.2112-001/book/book.pdf

[3] D.E. Bernholdt, B.A. Allan, R. Armstrong, F. Bertrand, K. Chiu, T.L. Dahlgren, K. Damevski, W.R. Elwasif, T.G.W. Epperly, M. Govindaraju, D.S. Katz, J.A. Kohl, M. Krishnan, G. Kumfert, J.W. Larson, S. Lefantzi, M.J. Lewis, A.D. Malony, L.C. Mclnnes, J. Nieplocha, B. Norris, S.G. Parker, J. Ray, S. Shende, T.L. Windus and Shujia Zhou, " A Component Architecture for High-Performance Scientific Computing ", International Journal of High Performance Computing Applications 2006; 20: 163, DOI: 10.1177/1094342006064488

[4] P. Prabhu, T.B. Jablin, A. Raman, Y. Zhang, J. Huang H. Kim, N.P. Johnson, F. Liu, S. Ghosh, S. Beard, T. Oh, M. Zoufaly, D. Walker and D.I. August, "Survey of the Practice of Computational Science", Proceeding SC'11 State of the Practice Reports, Seattle, November 2011, DOI: 10.1145/2063348.2063374 (ACM).

[5] B.K. Szymanski, "Fortran programming language and Scientific Programming: 50 Years of mutual growth", Scientific Programming 15 (2007), IOS Press, 1-21

[6] C.D. Norton, V.K. Decyk, B.K. Szymanskicand and H.Gardner, "The transition and adoption to modern

programming concepts for scientific computing in Fortran ", Scientific Programming 15 (2007), IOS Press, 27-44
[7] L. Prechelt, "An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl for a search/string-processing program", Technical Report March 2000
[8] B. Carpenter, Y.-J. Chang, G. Fox, D. Leskiw, and X. Li, "Experiments with 'HPJava' ", Concurrency and Computation Practice and Experience, Vol 9 Issue 6, June 1997, pp 633-648.
[9] Java Grande Forum, http://www.vni.com/corner/garage/-grande/complex.htm
[10] H.J. Sips and K.V. Reeuwijk, "Java for Scientific Computation: Prospects and Problems", International Conference on Large-Scale Scientific Computing LSSC 2001, LNCS vol 2179, pp 236-243, DOI: 10.1007/3-540-45346-6_24
[11] J.K. Nilsen, X. Cai, Bjø rn Høyland, and H. P. Langtangen, " Simplifying the parallelization of scientific codes by a function-centric approach in Python", Computational Science & Discovery, 3:015003, 2010.
[12] H.P. Langtangen, Python Scripting for Computational Science, Springer 2009.
[13] T.E. Oliphant, "Guide to Numpy", Trelgol, 2008.
[14] S.Nanz, C.A. Furia, " A Comparative Study of Programming Languages in Rosetta Code", Proceedings of the 37th International Conference on Software Engineering, Vol 1, May 2015, pp 778-788.
[15] E.Loh, "The Ideal HPC Programming Language", ACM Queue, Vol 8 Issue 6, June 2010, DOI: 10.1145/1810226.1820518
[16] S. Alam, "Is Fortran Still Relevant? Comparing Fortran with Java and C++", International Journal of Software Engineering & Applications (IJSEA), Vol.5, No.3, May 2014.
[17] R.W. Numrich and J. Reid, "Co-array Fortran for Parallel Programming", ACM SIGPLAN Fortran Forum, vol 17 Issue 2, Aug. 1998, pp. 1 – 31
[18] S.B. Aruoba and J.F. Villaverdez, "A Comparison of Programming Languages in Economics ", Aug. 2014.
[19] T. Sheard, "Putting Curry-Howard to work", In Proceedings of the 2005 ACM SIGPLAN workshop on Haskell, 2005, pp. 74-85.
[20] M. Sulzmann and R. Voicu, "Language-based program verification via expressive types", Electronic Notes in Theoretical Computer Science, 174(7), pp. 129–147, 2007.
[21] D. Orchard, A. Rice, "A computational science agenda for programming language research", Procedia Computer Science Volume 29, ICCS 2014, pp. 713–727, DOI: 10.1016/j.procs.2014.05.064
[22] D. Hale, The Java and C++ platforms for scientific computing , CWP-547 pp 280-288.
[23] B. McLean, Julia: a programming language for scientific computing, January 2014.
[24] V. Sudheer, "Julia Awareness for Scientific Computing", https://www.quest-global.com
[25] Interpretation and Compilation, ftp://ftp.cs.utexas.edu/pub/garbage/cs345/schintro-v13/schintro_112.html