Comparison study between modern and conventional programming languages in numerical astrophysics

Yeong-Bok Bae[1], Young-Hwan Hyun[2], Jinho Kim[2], Whansun Kim[3], Young-Min Kim[4], Jae-Joon Lee[2], Sang Hoon Oh[3], Miok Park[1], Chan Park[1]

[1]Institute for Basic Science, [2]Korea Astronomy and Space Science Institute,
[3]National Institute for Mathematical Sciences, [4]Ulsan National Institute of Science and Technology

Abstract

Fortran, C, and C++ have been a choice of languages for performance-oriented programming. While high-level languages, such as Python, have gained popularity with the rise of machine-learning applications, high-level languages are often slower than low-level languages although they do provide ways to improve their performance. We explore the features of many languages with a focus on single-core performance. As a simple example, we have implemented the multi-grid method in multiple languages including Fortran, C, C++, Python, Mathematica, Julia, Rust, etc. Multi-grid is a widely used numerical method to solve the elliptic partial differential equation such as the Poisson equation using hierarchical grid levels. In order to compare the performance, we measure the wall clock time of v-cycles as a function of grid resolution in each language. We compare the performance and discuss the difficulties of implementing and limits for each language.

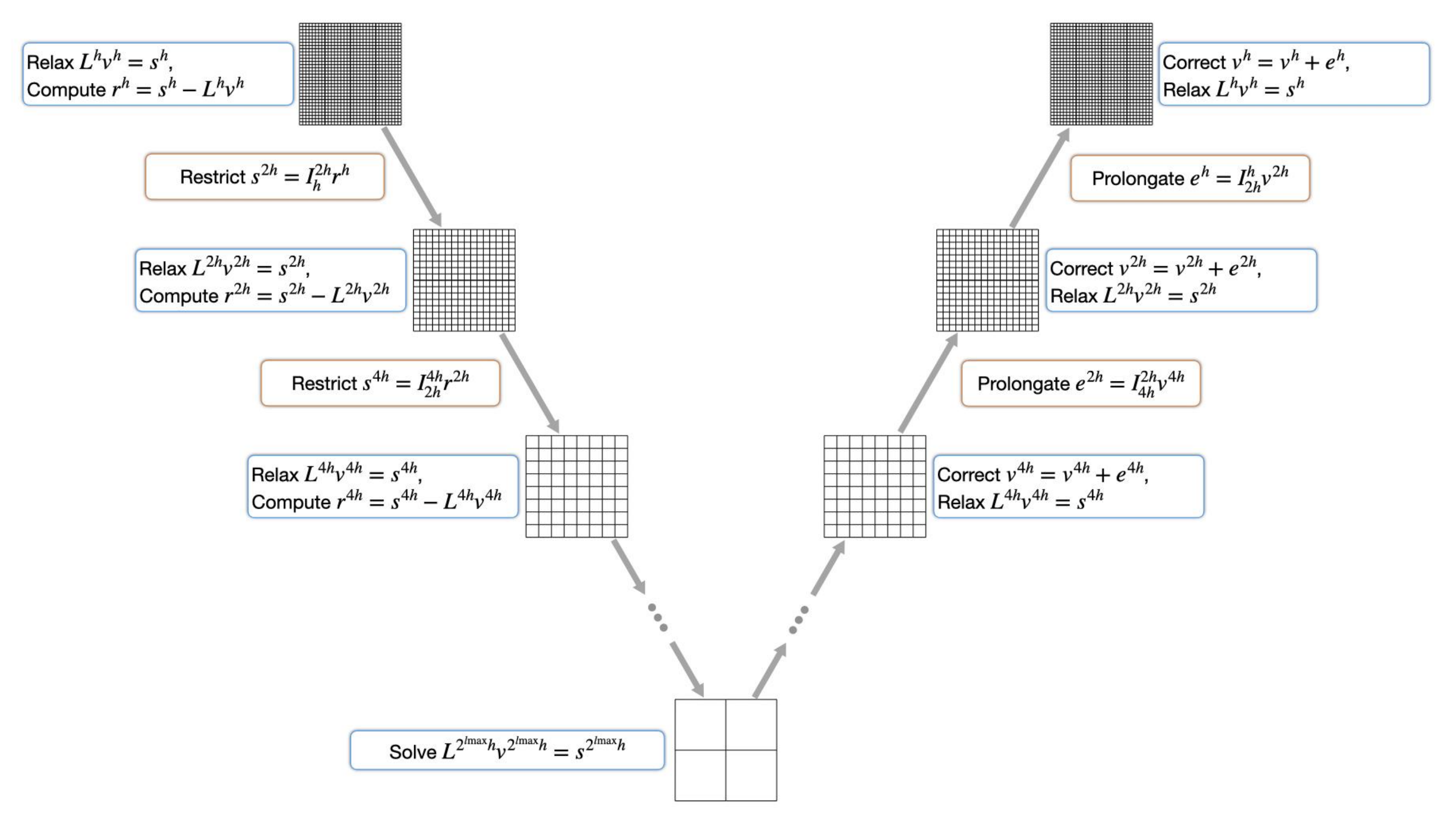
Motivation

Modern programming languages, represented by Python are widely used in various fields these days. They are also adopted in astrophysics and being used in diverse data-driven work such as data process, statistics, machine learning, visualization, etc. They provide user-friendly interfaces and many scientific packages that can be accessed easily. They make tasks very simple and enable users can save their time. On the other hand, they are evaluated to be behind the classical programming languages in performance. In this small project, we would like to check whether the modern programming languages are also useful even in the numerical astrophysics, in which Fortran and C/C++ has been conventionally used, by solving simple elliptic equation with multigrid method as follows.

Problem Setting

- Our aim is to solve the Poisson's equation to obtain the gravitational potential of the spherically symmetric star in hydrostatic equilibrium.
  $\nabla^2\Phi = 4\pi\rho.$
- We assume the spherically symmetric system with compactified radial coordinate which covers spatial infinity as follows,
  $$\frac{d^2\Phi}{ds^2} + \frac{2}{s}\frac{d\Phi}{ds} = 4\pi\rho r_s(1-s)^{-4}, \text{ where } r = r_s\frac{s}{1-s} \text{ and } r_s \text{ is a size of star.}$$
- For the density profile, we use the Lane-Emden solution with $n = 1,\ K = 100.$
  And the central density is $\rho_c = 1.28 \times 10^{-3}$.
  $$\rho = \rho_c\frac{\sin{(r/\alpha)}}{r/\alpha} = \rho_c\frac{\sin\left(\pi\frac{s}{1-s}\right)}{\pi\frac{s}{1-s}}, \text{ where } \alpha^2 = 50/\pi.$$
- Then the gravitational potential is expressed as
  $$\Phi(s) = \begin{cases} -4\pi\alpha^2\rho_c\left[1 + \frac{\sin\left(\frac{\pi s}{1-s}\right)}{\frac{\pi s}{1-s}}\right], & \text{for } s \le \frac{1}{2} \\ -4\pi\alpha^2\rho_c\frac{1-s}{s}, & \text{for } s > \frac{1}{2} \end{cases}$$

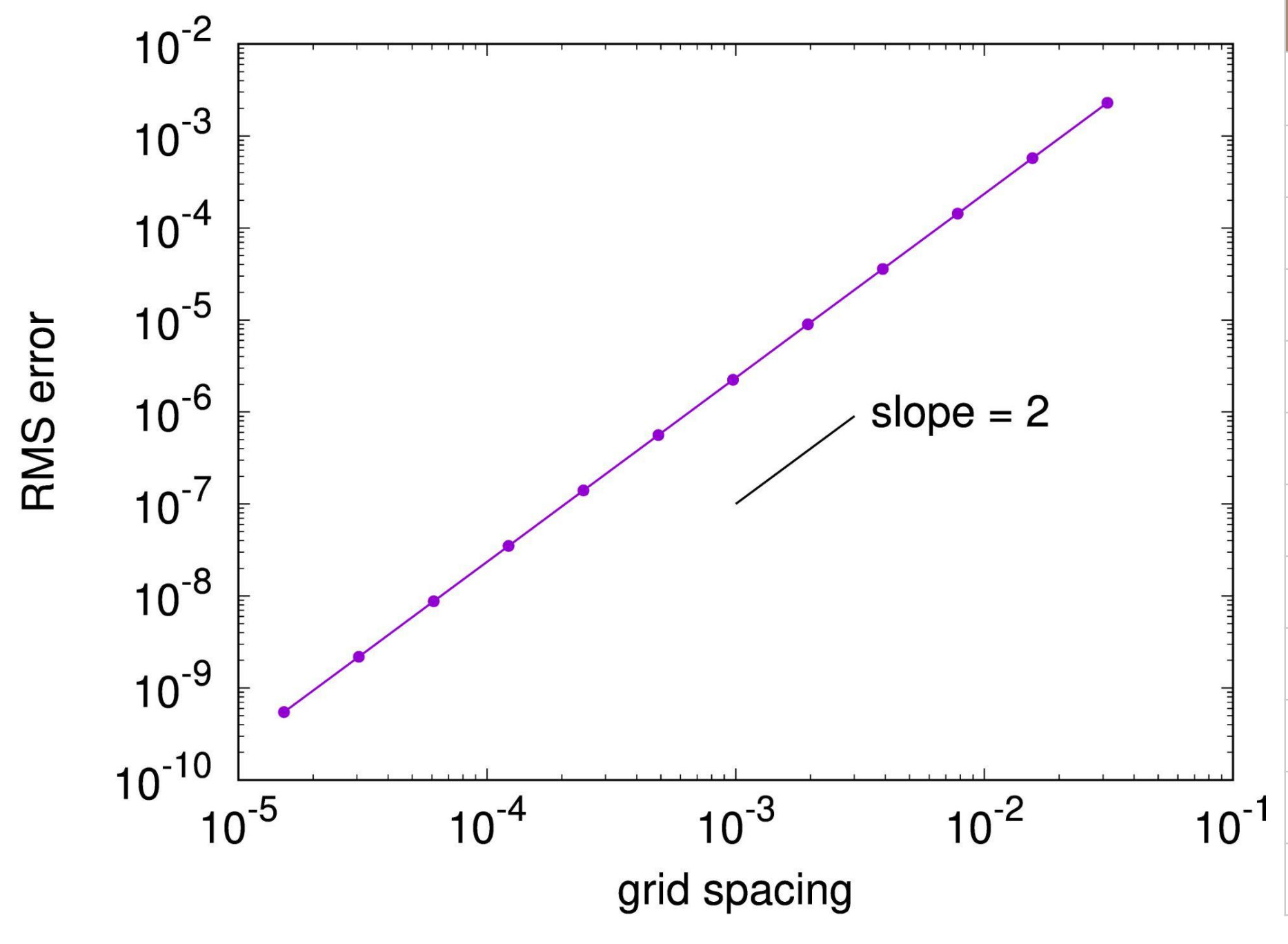
About Multigrid


Relax $L^hv^h = s^h$,
Compute $r^h = s^h - L^hv^h$
Restrict $s^{2h} = I_h^{2h}r^h$
Relax $L^{2h}v^{2h} = s^{2h}$,
Compute $r^{2h} = s^{2h} - L^{2h}v^{2h}$
Restrict $s^{4h} = I_{2h}^{4h}r^{2h}$
Relax $L^{4h}v^{4h} = s^{4h}$,
Compute $r^{4h} = s^{4h} - L^{4h}v^{4h}$
Solve $L^{2^{lmax}h}v^{2^{lmax}h} = s^{2^{lmax}h}$
Correct $v^{4h} = v^{4h} + e^{4h}$,
Relax $L^{4h}v^{4h} = s^{4h}$
Prolongate $e^{2h} = I_{4h}^{2h}v^{4h}$
Correct $v^{2h} = v^{2h} + e^{2h}$,
Relax $L^{2h}v^{2h} = s^{2h}$
Prolongate $e^h = I_{2h}^{h}v^{2h}$
Correct $v^h = v^h + e^h$,
Relax $L^hv^h = s^h$

Multigrid is one of the optimal technique for solving elliptic partial differential equation by adopting hierarchical grid system to accelerate the convergence of relaxation method as shown in Figure 1.

Numerical Code Validation

- Our designed order of accuracy is 2.



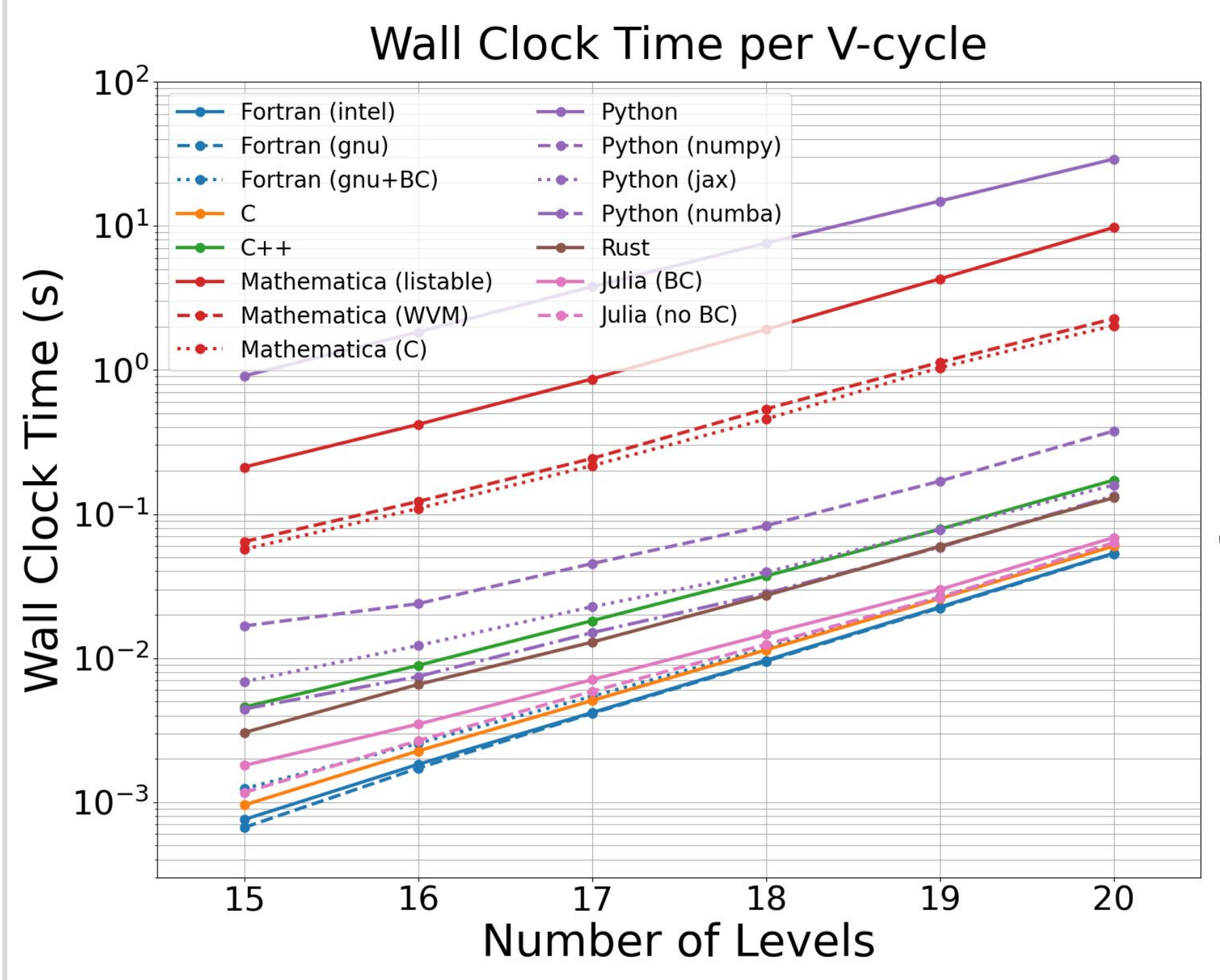
# of levels	# of grid points	grid spacing	RMS error	Convergence Rate
5	33	3.13E-02	2.29E-03	
6	65	1.56E-02	5.74E-04	1.99939
7	129	7.81E-03	1.43E-04	1.99959
8	257	3.91E-03	3.59E-05	1.99976
9	513	1.95E-03	8.97E-06	1.99987
10	1025	9.77E-04	2.24E-06	1.99993
11	2049	4.88E-04	5.61E-07	1.99997
12	4097	2.44E-04	1.40E-07	1.99998
13	8193	1.22E-04	3.50E-08	1.99999
14	16385	6.10E-05	8.76E-09	1.99999
15	32769	3.05E-05	2.19E-09	2.00003
16	65537	1.53E-05	5.47E-10	2.00186

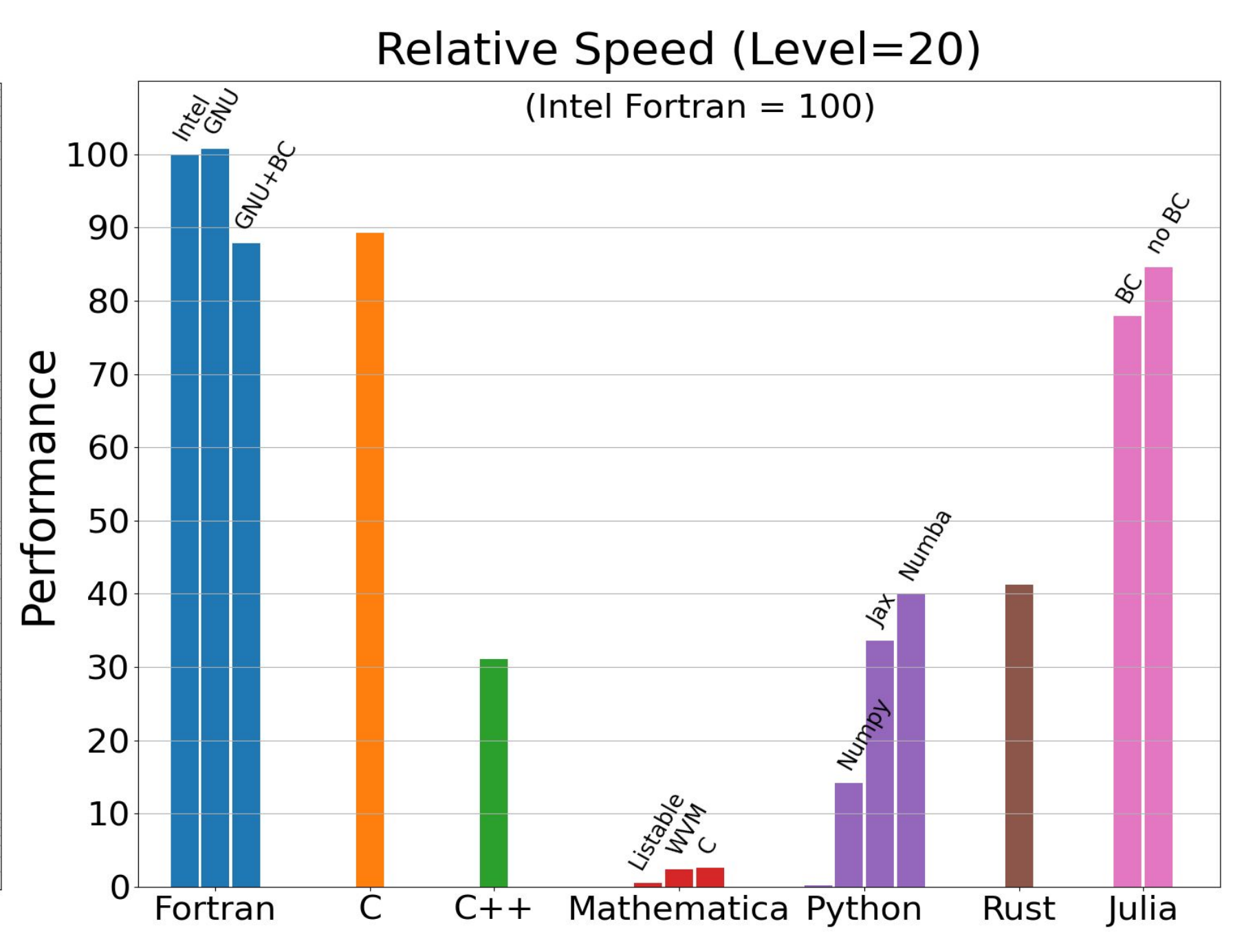
Programming Languages

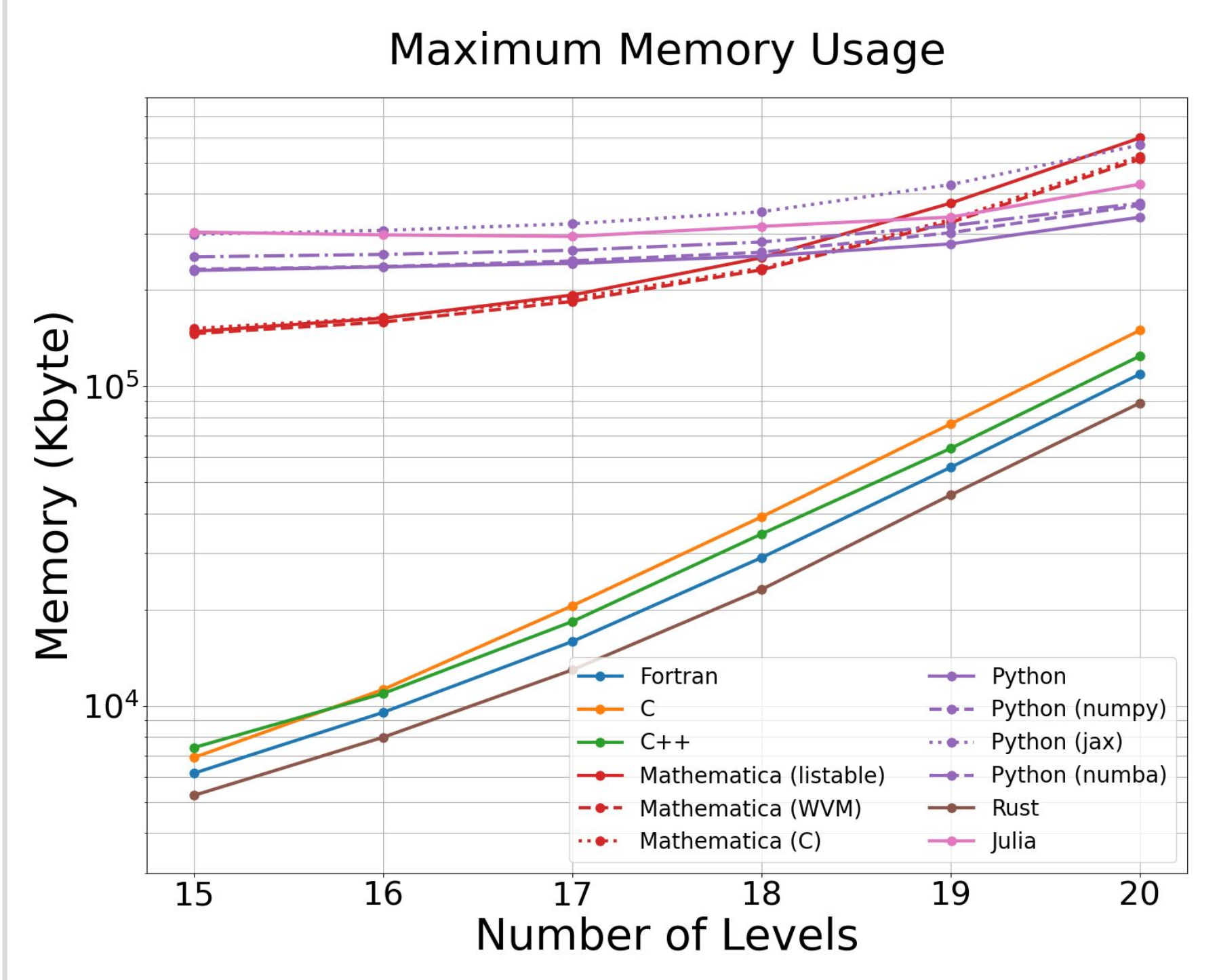
	Fortran	C	C++	Mathematica	Python	Rust	Julia
Appeared	1957	1972	1985	1988	1991	2010	2012
Latest stable version	Fortran2018	C17	C++20	13.0.1	3.10.7	1.64.0	1.8.2
Object-oriented	✔	X	✔	✔	✔	✔	✔
Compiler/Interpreter/JIT	C	C	C	I	I/J	C	J
Interactive mode	X	X	X	✔	✔	X	✔
Array indexing	1-based	0	0	1	0	0	1
Garbage Collection	X	X	X	✔	✔	X	✔
TIOBE Ranking 2022 (2021)	15 (17)	2 (1)	4 (4)	-	1 (2)	26	21 (23)
Stack Overflow Ranking 2022 (Most loved)	38	33	25	-	6	1	5

- https://www.wikipedia.org/
- TIOBE ranking : https://www.tiobe.com/tiobe-index/
- Stack Overflow ranking : https://survey.stackoverflow.co/2022/#most-loved-dreaded-and-wanted-language-love-dread
- JIT: just-in-time compilation

Comparison







- BC: For test cases with BC, we turn on the runtime bound check on arrays that diminishes performance but improve memory safety.
- For the languages which adopts just-in-time (JIT) compilation like Julia, Python-Numba and Python-JAX, the wall clock time is measured from the second step of the loop, because the first step of the loop has overheads to compile machine code.
- We do not consider thread parallelism or distributed memory. These show performances on a single thread. However, SIMD (Single Instruction Multiple Data) using AVX-512 instructions has applied by compliers.

- Specification of the computer used here : OpenStack [Intel Xeon Processor (Cascadelake) 2.7 GHz x 8 core / RAM 128GB]
- Fortran/C/C++: gcc 12.2.0, Mathematica 13.1.0, Python 3.10.4 (cpython), Rust: rustc 1.64.0, Julia 1.8.2

Summary / Conclusion

- In terms of computational speed, Fortran and C are the best choices.
- Rust and Julia show comparable calculation speed to those of Fortran and C/C++.
- A very careful optimization is required to achieve sufficient performance with Python.
- Rust shows a noticeable performance when it comes to memory management, which is safe and less demanding.
- For high-level languages like Python, they often provide ways to make code runs fast for a subset of code. NumPy is the fundamental package scientific computing with Python which delivers several orders of magnitude increase compared to the pure Python version. Still, calculation with NumPy creates temporary arrays and extra for-loops. These can be removed using modules that support JIT which translates Python code to optimized machine code at runtime. For the use-case like multigrid, we find Numba is more convenient to implement and gives slightly better performance than JAX.
- These comparison should be considered for reference only. The algorithms are slightly different depending on the language, and the optimization may not be perfect, which depend on the author's experience with each language. The language performance also depends on the problem.
- Modern programming languages have advantages for user convenience. Various applications are expected in the field of numerical astrophysics in the future.
- We share all codes of our project with github whose link is given by QR code:

Acknowledgment
This work was supported by KIRD (Korea Institute of Human Resources Development in Science and Technology) under the Learning Lab Program. The authors acknowledge that the computational work reported in this poster was performed on the KASI Science Cloud platform supported by Korea Astronomy and Space Science Institute.