UNIWERSYTET GDAŃSKI
WYDZIAŁ MATEMATYKI, FIZYKI I INFORMATYKI

Krzysztof Wiśniewski
numer albumu: 274276

Kierunek studiów: Bioinformatyka
Specjalność: Ogólna

Optymalizacja oprogramowania w języku Python do analizy stanów kwantowych.

Praca licencjacka
wykonana
pod kierunkiem
dr hab. Marcin Wieśniak, prof. UG

Gdańsk 2023

Spis treści

1 Wstęp **2**
1.1 Dlaczego Python? 2
1.2 Cel pracy 3
1.3 Pochodzenie programu 4
1.4 Dostępne narzędzia 4
1.4.1 Kompilacja AOT[1] 4
1.4.2 Kompilacja JIT 5
1.5 Selekcja narzędzi 6

Odwołania **7**

[1]AOT compilation (Ahead Of Time compilation) - kompilacja przed czasem wykonywania programu.

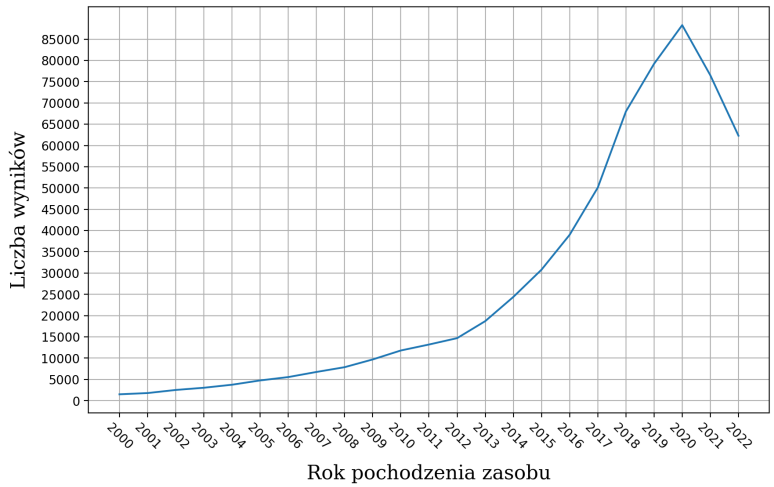
Streszczenie

W tej pracy przeprowadzam analizę efektywności metod optymalizacji, która koncentruje się na minimalizacji czasu wykonania, oprogramowania napisanego w języku Python[25][4], skupiającego się na arytmetyce macierzowej, na przypadku programu CSSFinder służącego do analizy stanów kwantowych pod kątem detekcji splątania kwantowego. Pośród rozważanych metod obecna będzie standardowa implementacja w języku Python z wykorzystaniem biblioteki NumPy[10][14], wersja wzbogacona o kompilację JIT przy pomocy biblioteki Numba[7][9], wersja skompilowana do kodu maszynowego przy pomocy biblioteki Cython[3][19] i kompilatora GCC[26] oraz implementacja w języku Rust[33], również skompilowana do kodu maszynowego.

1 Wstęp

1.1 Dlaczego Python?

Język Python zachęca użytkowników prostotą składni, łatwością tworzenia kodu, dynamicznym systemem typów, automatycznym zarządzanie pamięcią, mnogością dostępnych bibliotek otwartoźródłowych, oraz rozbudowaną społecznością programistów. Na przestrzeni ostatnich 20 lat język stworzony przez Guido van Rossum zanotował intensywny wzrost popularności. Pokazują to liczne zestawienia, w tym zestawienie najczęściej wykorzystywanych języków programowanie na GitHub'ie[15], w którym Python w roku 2022 zajął 2 miejsce, czy też zestawienie TIOBE Index[36], uznające ten język za obecnie najbardziej rozpowszechniony pośród doświadczonych programistów (Maj 2023).



Rysunek 1: Ilość wyników zwróconych przez wyszukiwarkę Google Scholar dla zapytania 'python language' z podziałem na rok wydania.

Niestety, interpretowany kod, napisany w Pythonie, pomimo licznych zalet, posiada również dotkliwą wadę - pod względem wydajności znacząco odstaje od kompilowanych języków programowania (C[30], C++[31], Rust[32]). Natomiast, dzięki nakładowi pracy wielu zespołów programistów, obecnie istnieją metody pozwalające na obejście tej niedogodności.

1.2 Cel pracy

Praca ta ma na celu weryfikację efektywności wybranych metod poprawy czasu wykonania oprogramowania CSSFinder, zaimplementowanego w języku Python. W dalszej jej części opiszę specyfikę poszczególnych metod optymalizacji, w jaki sposób zmieniają wydajność programu oraz spróbuję wskazać prawdopodobne powody dla których niektóre z uzyskanych wyników konsekwentnie odstają od oczekiwań które można mieć wobec wykorzystanych narzędzi.

Do przeprowadzenia takich analiz konieczne było wielokrotne ponowne implementowanie algorytmu. Funkcjonalny kod opisywany w tej pracy dostępny jest w repozytoriach Gita[27] w serwisie GitHub [40][41][43]. W skutek prac projektowych utworzona została również grupa publicznych pakietów, które można pobrać z serwisu PyPI:

- `cssfinder`[39]
- `cssfinder_backend_numpy`[42]
- `cssfinder_backend_rust`[44]

Zainstalowanie ich jest możliwe przy pomocy menadżera pakietów języka Python[24], np. `pip`[20]. Pakiety są kompatybilne z implementacją CPython w wersjach 3.8 - 3.10 i były testowane na systemach Windows (10), Linux (Ubuntu 22.04) oraz macOS (12).

1.3 Pochodzenie programu

Program CSSFinder bazuje na algorytmie zaproponowanym przez E. Gilberta[2] pozwalającym na znalezienie odległości pomiędzy punktem, a zbiorem wypukłym. Korzysta z faktu że możliwe jest zastosowanie tego algorytmu do analizy stanów kwantowych pod kątem detekcji splątania kwantowego[11][12]. Algorytm ten wielokrotnie, z sukcesem, był wykorzystany do analizy problemów z dziedziny fizyki kwantowej[16][13].

1.4 Dostępne narzędzia

Poniżej wymienię powszechnie wykorzystywane metody pozwalające na skrócenie czasu wykonania kodu napisanego w języku Python. Do tych metod zaliczam też przepisanie kodu na język niższego poziomu. Przez niektóre osoby może być to uznane za nadużycie, poddam tą kwestię pod dyskusję w dalszej części tekstu.

1.4.1 Kompilacja AOT[2]

Pierwsza i obecnie najpopularniejsza implementacja języka Python została utworzona z wykorzystaniem języka C. Posiada ona możliwość importowania bibliotek współdzielonych napisanych w językach niższego poziomu (C, C++, Rust, oraz inne). Efekt ten można uzyskać na kilka sposobów:

1. `Moduł ctypes`[22] - posiada on API które pozwala opisać interfejs funkcji obcej (tj. takiej która została napisana w języku niższego poziomu i skompilowana do kodu maszynowego) i wywołać tak opisaną funkcję. Metoda ta nie cieszy się popularnością pośród bibliotek otwartoźródłowych.

2. `Moduły rozszerzeń`[23] - interpreter posiada interfejs w języku C oraz wsparcie dla dobrze opisanych konwencji które pozwalają odpowiednio spreparować bibliotekę współdzieloną (linux - so, windows - dll), tak żeby interpreter Pythona był w stanie automatycznie wykryć dostępne

[2]AOT compilation (Ahead Of Time compilation) - kompilacja przed czasem wykonywania programu.

w bibliotece symbole. W tym przypadku warto dodać, że pomimo, że oficjalna dokumentacja wspomina tylko o językach C i C++, w rzeczywistości powstały biblioteki które pozwalają wykorzystać wiele innych języków programowania, takich jak Rust przy pomocy Py03[21] lub GO z użyciem biblioteki gopy[17].

3. `Cython`[19][3] - jest to zarówno nadzbiór języka Python, który rozszerza jego składnię o możliwość statycznego typowania, jak i pakiet dostępny na PyPI, zawierający transpilator, potrafiący przetłumaczyć dedykowany język na C/C++, a następnie, wykorzystując osobno zainstalowany kompilator, skompilować do kodu maszynowego. Cython nie wymaga aby w kompilowanym kodzie znajdowały się dodatkowe adnotacje, wzwiązku z czym możliwe jest skompilowanie czystego, nietypowanego, kodu Pythona do kodu maszynowego. Pozwala to na pozbycie się obciążenia ze strony procesu interpretacji oraz skorzystać z optymalizacji, oferowanych przez współczesne kompilatory. Nie usuwa to jednak obciążenia ze strony dynamicznego systemu typów, czyniąc kompilację bez adnotacji mało efektywną.

4. `mypyc`[37] - podobnie do Cythona, jest to transpilator, natomiast zamiast korzystać z dedykowanego języka , bazuje na dodanych w Pythonie 3.5[6], pierwotnie opisanych przez PEP 484[29] i PEP 483[28], adnotacjach typów. Jest on rozwijany obok projektu mypy - pakietu do statycznej analizy typów dla języka Python, również opartej na adnotacjach typów[34].

Ponieważ w każdym z wymienionych przypadków, kod niższego poziomu jest kompilowany przed dostarczeniem do użytkownika, pozwala to na wykorzystanie zaawansowanych możliwości automatycznej optymalizacji dostarczanych przez współczesne kompilatory, na przykład LLVM, które jest sercem implementacji `clang` (język C++) oraz `rustc` (język Rust).

1.4.2 Kompilacja JIT

Niektóre języki, w calu maksymalizacji wydajności sięgają po kompilację w czasie działania programu, zazwyczaj skrótowo określaną JIT[3]. Ta występuje w różnych odmianach, natomiast najszerzej wykorzystywane są środowiska uruchomieniowe które rozpoczyna pracę w trybie interpretera i śledzą zachowanie programu, po czym na podstawie charakterystyki wykonywania kodu wybierają które jego fragmenty przekształcić w kod maszynowy.

[3]JIT compilation (Just In Time compilation) - kompilacja wykonywana w trakcie działania aplikacji[18].

Na takiej zasadzie działa silnik języka JavaScript - V8[5]. Ze względu na dynamiczne typowanie tego językach, kompilacja bez śledzenia byłaby nieefektywna. Co ciekawe tą samą drogą podążyły też języki takie jak Java czy C#, które ze względu na wykorzystywane w nich statyczne typowanie, mogłyby być teoretycznie pre-kompilowane[4], czy to przed dostarczeniem do użytkownika, czy też po uruchomieniu aplikacji. Kompilacja wspierana profilowaniem w czasie pracy (PGO - Profile Guided Optimization), jest jednak znacznie oszczędniejsza, ponieważ unika kompilacji rzadko wykorzystywanych fragmentów kodu.

W momencie pisania tej pracy istnieją dwa szeroko dostępne i aktywnie utrzymywane narzędzia oferujące kompilację JIT dla języka Python. Jednym z nich jest pełna alternatywna implementacja języka Python - PyPy[38]. Wykonywana przez nią kompilacja JIT działa on na podobnej zasadzie do uprzednio wymienionych - śledzi cały kod który wykonuje i automatycznie decyduje które fragmenty skompilować do kodu maszynowego[1]. Drugą jest biblioteka Numba[7][9]. Ona, w przeciwieństwie do PyPy, wymaga aby fragmenty kodu, które mają być skompilowane, miały postać funkcji oznaczonych dedykowanymi dekoratorami[5].

1.5 Selekcja narzędzi

Weryfikacja każdej z tych metod byłaby czasochłonna oraz często nieuzasadniona - wiele z tych narzędzi nie było tworzonych z myślą o przypadku który rozważamy lub wykazuje się mniejszą wydajnością od rozwiązań równoważnych. Ostatecznie analizie poddałem następujące warianty implementacji programu CSSFinder:

- w języku Python, bazującą na bibliotece NumPy,
- w języku Python, bazującą na bibliotece NumPy z kompilacją JIT z użyciem pakietu Numba,
- w języku Python, bazującą na bibliotece NumPy z kompilacją AOT z wykorzystaniem pakiety Cython i kompilatora GCC,
- w języku Rust, bazującą na bibliotece ndarray z kompilacją AOT z wykorzystaniem rustc (LLVM)

[4]W gwoli ścisłości, oba te języki najpierw kompilowane są do formy pośredniej, kodu bajtowego, jeszcze przed trafieniem do użytkownika, a dopiero później, po uruchomieniu aplikacji, do kodu maszynowego[8][35].

[5]Obecnie dostępny jest też dekorator pozwalający na kompilację klas, niestety jest on raczej niestabilny i nie radzi sobie w wielu sytuacjach.

Odwołania

[1] Carl Friedrich Bolz i in. „Tracing the meta-level: PyPy's tracing JIT compiler". W: *Proceedings of the 4th workshop on the Implementation, Compilation, Optimization of Object-Oriented Languages and Programming Systems*. 2009, s. 18–25.

[2] Patrick Lindemann. „The gilbert-johnson-keerthi distance algorithm". W: *Algorithms in Media Informatics* (2009).

[3] Stefan Behnel i in. „Cython: The best of both worlds". W: *Computing in Science & Engineering* 13.2 (2010), s. 31–39.

[4] Mark Lutz. *Learning python: Powerful object-oriented programming*. Ó'Reilly Media, Inc.", 2013.

[5] Gem Dot, Alejandro Martínez i Antonio González. „Analysis and optimization of engines for dynamically typed languages". W: *2015 27th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE. 2015, s. 41–48.

[6] Yury Selivanov Elvis Pranskevichus. *What's New In Python 3.5*. 2015. URL: `https://docs.python.org/3/whatsnew/3.5.html` (term. wiz. 14. 05. 2023).

[7] Siu Kwan Lam, Antoine Pitrou i Stanley Seibert. „Numba". W: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. ACM, list. 2015. DOI: `10.1145/2833157.2833162`. URL: `https://doi.org/10.1145/2833157.2833162`.

[8] Microsoft. *NET fundamentals documentation - Managed Execution Process*. 2015. URL: `https://learn.microsoft.com/en-us/dotnet/standard/managed-execution-process` (term. wiz. 14. 05. 2023).

[9] Inc. Anaconda i in. *Numba documentation*. 2020. URL: `https://numba.readthedocs.io/en/stable/user/index.html` (term. wiz. 12. 05. 2023).

[10] Charles R. Harris i in. „Array programming with NumPy". W: *Nature* 585.7825 (wrz. 2020), s. 357–362. DOI: `10.1038/s41586-020-2649-2`. URL: `https://doi.org/10.1038/s41586-020-2649-2`.

[11] Palash Pandya, Omer Sakarya i Marcin Wieśniak. „Hilbert-Schmidt distance and entanglement witnessing". W: *Physical Review A* 102.1 (2020), s. 012409.

[12] Marcin Wieśniak i in. „Distance between bound entangled states from unextendible product bases and separable states". W: *Quantum Reports* 2.1 (2020), s. 49–56.

[13] Mirko Consiglio, Tony JG Apollaro i Marcin Wieśniak. „Variational approach to the quantum separability problem". W: *Physical Review A* 106.6 (2022), s. 062413.

[14] NumPy Developers. *NumPy documentation*. 2022. URL: https://numpy.org/doc/stable/ (term. wiz. 12. 05. 2023).

[15] Inc. GitHub. *The top programming languages*. 2022. URL: https://octoverse.github.com/2022/top-programming-languages (term. wiz. 14. 05. 2023).

[16] Marcin Wieśniak. „Two-Qutrit entanglement: 56-years old algorithm challenges machine learning". W: *arXiv preprint arXiv:2211.03213* (2022).

[17] The go-python Authors. *go-python/gopy: gopy generates a CPython extension module from a go package*. 2023. URL: https://github.com/go-python/gopy (term. wiz. 14. 05. 2023).

[18] Wikipedia contributors. *Wikipedia - Just-in-time compilation*. 2023. URL: https://en.wikipedia.org/wiki/Just-in-time_compilation (term. wiz. 15. 05. 2023).

[19] *Cython C-Extensions for Python*. 2023. URL: https://cython.org/ (term. wiz. 14. 05. 2023).

[20] The pip developers. *pip · PyPI*. 2023. URL: https://pip.pypa.io/en/stable/ (term. wiz. 14. 05. 2023).

[21] The PyO3 developers. *PyO3 user guide*. 2023. URL: https://pyo3.rs/v0.18.3/ (term. wiz. 14. 05. 2023).

[22] Python Software Foundation. *ctypes — A foreign function library for Python*. 2023. URL: https://docs.python.org/3/library/ctypes.html (term. wiz. 14. 05. 2023).

[23] Python Software Foundation. *Extending Python with C or C++*. 2023. URL: https://docs.python.org/3/extending/extending.html (term. wiz. 14. 05. 2023).

[24] Python Software Foundation. *Packaging PEPs | peps.python.org*. 2023. URL: https://peps.python.org/topic/packaging/ (term. wiz. 14. 05. 2023).

[25] Python Software Foundation. *Welcome to Python.org*. 2023. URL: https://www.python.org/ (term. wiz. 14. 05. 2023).

[26] Inc. Free Software Foundation. *GCC, the GNU Compiler Collection*. 2023. URL: https://gcc.gnu.org/ (term. wiz. 14. 05. 2023).

[27] Inc. Free Software Foundation. *Git*. 2023. URL: https://gcc.gnu.org/ (term. wiz. 14. 05. 2023).

[28] Ivan Levkivskyi Guido van Rossum. *PEP 483 - The Theory of Type Hints*. 2023. URL: https://peps.python.org/pep-0483/ (term. wiz. 14. 05. 2023).

[29] Łukasz Langa Guido van Rossum Jukka Lehtosalo. *PEP 484 - Type Hints*. 2023. URL: https://peps.python.org/pep-0484/ (term. wiz. 14. 05. 2023).

[30] hanabi1224. *Programming Language and compiler Benchmarks - C VS Python benchmarks*. 2023. URL: https://programming-language-benchmarks.vercel.app/c-vs-python (term. wiz. 14. 05. 2023).

[31] hanabi1224. *Programming Language and compiler Benchmarks - C++ VS Python benchmarks*. 2023. URL: https://programming-language-benchmarks.vercel.app/cpp-vs-python (term. wiz. 14. 05. 2023).

[32] hanabi1224. *Programming Language and compiler Benchmarks - Rust VS Python benchmarks*. 2023. URL: https://programming-language-benchmarks.vercel.app/rust-vs-python (term. wiz. 14. 05. 2023).

[33] Steve Klabnik i Carol Nichols. *The Rust programming language*. No Starch Press, 2023.

[34] Jukka Lehtosalo i mypy contributors. *mypy 1.2.0 documentation*. 2023. URL: https://mypy.readthedocs.io/en/stable/ (term. wiz. 14. 05. 2023).

[35] Tim Lindholm; Frank Yellin; Gilad Bracha; Alex Buckley; Daniel Smith. *The Java Virtual Machine Specification*. 2023. URL: https://docs.oracle.com/javase/specs/jvms/se20/html/index.html (term. wiz. 14. 05. 2023).

[36] TIOBE Software. *TIOBE Index for May 2023*. 2023. URL: https://www.tiobe.com/tiobe-index/ (term. wiz. 14. 05. 2023).

[37] mypyc team. *mypyc 1.2.0 documentation*. 2023. URL: https://mypyc.readthedocs.io/en/stable/ (term. wiz. 14. 05. 2023).

[38] The PyPy Team. *PyPy Home Page*. 2023. URL: https://www.pypy.org/ (term. wiz. 14. 05. 2023).

[39] Krzysztof Wiśniewski. *CSSFinder (Core, PyPI)*. 2023. URL: https://pypi.org/project/cssfinder/ (term. wiz. 14. 05. 2023).

[40] Krzysztof Wiśniewski. *CSSFinder (Core)*. 2023. URL: https://github.com/Argmaster/CSSFinder (term. wiz. 12. 05. 2023).

[41] Krzysztof Wiśniewski. *CSSFinder Numpy Backend*. 2023. URL: https://github.com/Argmaster/cssfinder_backend_numpy (term. wiz. 12. 05. 2023).

[42] Krzysztof Wiśniewski. *CSSFinder Numpy Backend (PyPI)*. 2023. URL: https://pypi.org/project/cssfinder-backend-rust/ (term. wiz. 12. 05. 2023).

[43] Krzysztof Wiśniewski. *CSSFinder Rust Backend*. 2023. URL: https://github.com/Argmaster/cssfinder_backend_rust (term. wiz. 12. 05. 2023).

[44] Krzysztof Wiśniewski. *CSSFinder Rust Backend (PyPI)*. 2023. URL: https://pypi.org/project/cssfinder-backend-numpy/ (term. wiz. 12. 05. 2023).