ТЕХНІЧНІ НАУКИ

Акімов С.В.
студент;
Науковий керівник: **Святний В.А.**
доктор технічних наук, професор,
Донецький національний технічний університет

ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ ІНТЕРПРЕТАТОРІВ МОВИ ПРОГРАМУВАННЯ PYTHON З ТОЧКИ ЗОРУ ПРОДУКТИВНОСТІ

Python – це мова програмування високого рівня, яка дозволяє програмісту перекладати ідеї в робочий код без прикладання великої кількості зусиль. Як і в багатьох мовах програмування, які мають високий рівень абстракції, одним з мінусів Python є відносно великий час виконання [1, с. 22]. З роками були створені альтернативні інтерпретатори у спробах зберегти простий синтаксис Python при одночасному збільшенні його продуктивності. У цій роботі розглянуто особливості реалізації трьох з них, а саме CPython, Cython і PyPy, з точки зору продуктивності щодо часу виконання.

Python – це динамічно типізована інтерпретована мова програмування, тобто немає необхідності оголошувати тип змінних в коді і код не потрібно компілювати заздалегідь. Замість цього, коли запускається Python програма, Python має інтерпретатор, який компілює код Python в байт-код, який потім виконує віртуальна машина Python (PVM). PVM просто перебирає інструкції в байт-коді, виконуючи їх одну за іншою, до виходу з програми. Під час виконання PVM визначає типи змінних і переводить інструкції байт-коду в машинні інструкції, які розуміє ЦПП [2, с. 36]. І переклад в байт-код, і виконання, яке здійснює PVM, включені в інтерпретатор Python. Таким чином структура часу виконання Python може бути зведена до двох завдань: вихідний код Python компілюється в байт-код, який потім виконується PVM.

Перевага цього методу полягає в тому, що код Python не залежить від платформи, тільки інтерпретатор повинен знати специфіку різних платформ. Переклад на байт-код перед виконанням також є кроком оптимізації, оскільки байт-код може виконуватися значно швидше, ніж початковий вихідний код [3, с. 67]. Недолік полягає в тому, що інтерпретація байт-коду і визначення типів змінних під час виконання впливає на продуктивність в порівнянні з роботою заздалегідь скомпільованого коду [3, с. 68].

CPython є інтерпретатором вихідного коду і оригінальною та стандартної реалізацією Python, реалізованою в портативному ANSI C [3, с. 70]. CPython надає API рівня C [4]. Це дозволяє розробникам створювати модулі розширення або вбудовувати Python в інші додатки. Модуль розширення Python являє

собою скомпільовану динамічну бібліотеку C, яка може виконуватися інтерпретатором Python. Коли PVM виконує код в модулі розширення, він більше не інтерпретує байт-код. Замість цього він безпосередньо запускає скомпільований код в модулі розширення. Це усуває накладні витрати інтерпретатора [2, с. 42].

Cython є одночасно мовою програмування і оптимізуючим статичним компілятором [5]. Мова програмування є надмножиною мови Python, яка забезпечує легкий доступ до функцій C/C++, а також до типів C. У зв'язку з цим, код Python за деякими винятками вже є допустимим кодом Cython. Cython додає кілька ключових слів, наприклад, *cdef* для додавання статичної типізації. Компілятор Cython трансформує Cython-код в ефективний C/C++ код, який може бути скомпільовано в модуль розширення Python або автономний виконуваний файл. Cython використовує API Python/C і тому залежить від CPython.

Мета Cython – оптимізація компілятора Python [6]. Розробники, що використовують Cython, повинні мати можливість надати компілятору Cython код Python. Потім компілятор створює еквівалентний C-код, який працює настільки швидко, наскільки це можливо. Якщо необхідно щоб код працював ще швидше, Cython дозволяє оголошувати статичні типи C та Python для змінних [7]. Додаючи статичну типізацію до змінних Cython, інтерпретатор може виконати код без необхідності визначати типи цих змінних. За словами Роберта Бредшоу, це може призвести до стократного прискорення [6].

Cython має можливість автоматично виводити типи змінних. В якості запобіжного заходу це за замовчуванням виконується тільки тоді, коли типізація змінних не може змінити семантику коду. Використовуючи директиву компілятора *infer_types*, користувач може доручити Cython бути більш вільним в автоматичній типізації.

Прискорення, що досягається Cython, залежить від того, для чого код призначено. Згідно до питань по Cython, що часто задаються, Cython дуже хороший в підвищенні продуктивності для керуючих структур *if-elif-else*, циклів *for*, а також звичайних вбудованих типів, таких як списки, словники і рядки [8]. За словами Курта Сміта, Cython також працює дуже добре, в порівнянні з CPython, для математичних операцій і викликів функцій [2, с. 88]. Як уже згадувалося раніше, додавання статичних оголошень типів запобігає інтерпретатор від необхідності шукати тип змінної. Коли, наприклад, два об'єкти, назвемо їх *a* і *b*, складаються разом в Python, інтерпретатор повинен спочатку визначити їх тип і викликати його метод *__add__* з параметрами *a* і *b*. У методі *__add__* параметри необхідно розпакувати для вилучення базового типу C. Якщо *a* і *b* будуть, наприклад, типу *float*, то їх базовий тип C буде *double*. В Python числа, рядки і кортежі є незмінними. Це означає, що результат необхідно зберегти в новому об'єкті типу *float*. Вся ця процедура виконується під час виконання і вимагає багато накладних витрат. Коли використовується статична типізація, цей процес не потрібен, так як типи *a* і *b* вже відомі. Додавання компілюється в одну команду машинного коду, яка потім викликається під час виконання.

В Python функції є об'єктами першого класу. Це означає, що вони ведуть себе так само, як об'єкти, що містять стан, і мають більш розвинену поведінку,

ніж функції C [2, с. 27]. Покажчики функцій в C додають деяку додаткову функціональність, але функції Python можна створювати як під час імпорту, так і динамічно під час виконання, створювати анонімно з використанням ключового слова *lambda*, визначати всередині інших функцій, викликати за допомогою ключових аргументів і визначати зі значеннями параметрів за замовчуванням, що неможливо в C. Недоліком цього є те, що це пов'язано з витратами на продуктивність. Виклик функцій Python на кілька порядків повільніше, ніж виклик функцій C [2, с. 52]. Cython допускає типізацію функцій, які здатні виконуватися швидше, ніж функції Python, оскільки вони скомпільовані в високо оптимізований C-код, і здатний безпосередньо викликати функції в Python/C API. Як і для змінних, накладні витрати інтерпретатора ігноруються. Фактично, виклик статично типізованої функції в Cython так само ефективний, як виклик чистої функції C [2, с. 58].

PyPy – це реалізація мови програмування Python, написана на RPython, підмножині мови Python, зі своїм власним інтерпретатором [9] [10]. PyPy спроектований для більш високої продуктивності, ніж CPython, використовуючи трасувальний Just-in-Time (JIT)-компілятор (компіляція «на льоту»). JIT, який використовується в PyPy, є мета-трасувальним JIT-компілятором. Він не кодує ніякі мовні семантики і не профілює виконання програми. Замість цього він профілює роботу інтерпретатора, що виконує програму [11]. Команда PyPy вибрала реалізацію JIT таким чином, щоб зробити його незалежним від мов програмування. Відстежуючи виконання інтерпретатора замість виконання програми, JIT вибирає семантику мови з інтерпретатора.

Під час трасування інтерпретатора, JIT-компілятор намагається знайти ділянки коду, які виконуються кілька разів, так звані «гарячі точки». Вони визначаються шляхом підрахунку кількості запусків певних частин коду. Коли гаряча точка знайдена, PyPy перемикається в спеціальний режим, який називається режимом трасування, де всі операції наступного виконання записуються в так звану трасу. Траса являє собою список операцій і їх операндів, а також їх результати з одного прогону гарячої точки, наприклад, одного циклу *for*. Траса компілюється в оптимізований код асемблера, який працює дуже швидко і готовий до використання при наступному запуску гарячої точки. PyPy використовує кілька методів оптимізації в своєму компіляторі; серед інших константна згортка, вилучення загального підвиразу, вбудовування функцій і інваріантне переміщення коду [11].

Список використаних джерел:

1. Gorelick M. High Performance Python: Practical Performant Programming for Humans / Michael Gorelick. – Sebastopol: O'Reilly Media, Inc., 2014. – 340, p.
2. Smith K. Cython. / Kurt Smith. – Sebastopol: O'Reilly Media, Inc., 2014. – 340, p.
3. Лутц М. Изучаем Python, 4-е издание. / Марк Лутц – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 1280 с., ил.
4. Документація Python. Python C API. [Електронний ресурс]. – Режим доступу: https://docs.python.org/2/c-api/intro.html
5. Домашня сторінка Cython. [Електронний ресурс]. – Режим доступу: http://cython.org

6. «Sage Days 29 talk: Robert Bradshaw – Cython.» [Електронний ресурс]. – Режим доступу: https://www.youtube.com/watch?v=osjSS2Rrvm0

7. Документація Cython. Faster code via static typing. [Електронний ресурс]. – Режим доступу: http://docs.cython.org/src/quickstart/cythonize.html

8. Часті питання по Cython. Is Cython faster than CPython? [Електронний ресурс]. – Режим доступу: https://github.com/cython/cython/wiki/FAQ#is-cython-faster-than-cpython

9. PyPy Project. What is PyPy [Електронний ресурс]. – Режим доступу: http://pypy.org/

10. Документація RPython. [Електронний ресурс]. – Режим доступу: https://rpython.readthedocs.org/en/latest/

11. C. F. Bolz, A. Cuni, M. Fijalkowski, M. Leuschel, S. Pedroni, andA. Rigo, «Runtime Feedback in a Meta-tracing JIT for Efficient DynamicLanguages» [Електронний ресурс]. – Режим доступу: http://doi.acm.org/10.1145/2069172.2069181

Андрусенко В.В.

студент,

Харківський національний університет радіоелектроніки

РОЗПІЗНАВАННЯ МОВИ. ВИДІЛЕННЯ ІНФОРМАЦІЙНИХ ОЗНАК МЕТОДОМ МЕЛ-ЧАСТОТНИХ КЕПСТРАЛЬНИХ КОЕФІЦІЄНТІВ

Розпізнавання мови є однією з найпоширеніших задач у галузі машинного навчання на сьогоднішній день. Це, перш за все, зумовлено стрімко зростаючими потребами людства до інтерфейсів взаємодії між людиною та електротехнікою, а також необхідністю впровадження автоматизованих рішень у різноманітних сферах діяльності з метою мінімізації часових та матеріальних витрат. Прикладами використання автоматизованих систем розпізнавання мови у реальному часі можуть бути системи розпізнавання у колл-центрах, які допомагають надавати консультації щодо базових питань та обслуговувати більшу кількість користувачів без необхідності у збільшенні чисельності персоналу, або ж такі сервіси, як Google Voice та Siri, що надають можливість голосового пошуку. Різноманітні рішення до задачі розпізнавання мови розробляються вже протягом багатьох років. У результаті цього була розроблена загальна структура системи автоматичного розпізнавання мови, яка зображена на рисунку 1. Зокрема у межах даної роботи буде більш детально розглянуто таку її складову, як виділення інформаційних ознак [1] та одну з її реалізацій – мел-частотні кепстральні коефіцієнти [2].

Виділення інформаційних ознак з акустичного сигналу є ключовим етапом розпізнавання, адже саме його вихідні дані використовуються для тренування акустичних моделей, наприклад, нейронної мережі або прихованої моделі Маркова, а також для встановлення відповідності між вхідними даними та існуючими у тренувальній базі зразками. Серед найбільш поширених методів виділення ознак виділяють наступні: кодування з лінійним предиктором, перцепційне лінійне передбачення та мел-частотні кепстральні коефіцієнти.