УДК 004.4:004.051:004.31:004.9

Залежність швидкодії програм від інструментальних засобів розробки та синтаксичних конструкцій

Юрій Антонов
доцент, канд. фіз.-мат. наук
ORCID: 0000-0001-9285-2988
iu.antonov@donnu.edu.ua
y.s.antonov@gmail.com

Донецький національний університет імені Василя Стуса

Ключові слова:
компілятор;
інтерпретатор;
швидкодія;
оптимізація роботи програми.

Запропоновано реалізацію автоматизованої інформаційної системи, що дає змогу досліджувати та аналізувати вплив інструментальних засобів розробки та синтаксичних конструкцій на швидкодію роботи програм. У системі застосовано архітектуру трирівневих баз даних. Спеціалізований додаток LST Client використано як клієнт, LST Web Service як сервер додатків, а MySQL як реляційна СУБД. Взаємодія додатка LST Client з LST Web Service відбувається за допомогою Web API з використанням Application Key. Application Key являє собою унікальний ключ, який створюється для кожного пристрою, що бере участь у дослідженні. Для тих випадків, коли у досліджені має взяти участь пристрій, розташований поза межами intranet-мережі, підключення здійснюється аналогічним способом, але через спеціально налаштований SSH-тунель. Для кожного пристрою створюється відповідний користувач з авторизацією за особистим RSA-ключем максимальної довжини. Дослідження проведено для мов програмування C, C++, Fortran, Java, C#, JavaScript, PHP, Python з використанням компіляторів та інтерпретаторів, що належать до x64 архітектури під керуванням Windows 10 для освітніх установ. Результати експериментів засвідчили, що на швидкодію програми впливає версія компілятора чи інтерпретатора, та набір синтаксичних конструкцій, що використовується, наприклад, ++i та i++. Лише компілятори мов програмування C, C++ та Fortran змогли оптимізувати код та не виконувати зайві цикли. Розроблена система дає змогу максимально автоматизувати процес тестування швидкодії програмного коду. Запропонована система може бути використана для оцінювання нових версій мов програмування та програмного забезпечення. Вона також може бути задіяна під час викладання дисциплін, пов’язаних із програмуванням, для звернення уваги студентів на слабкі та сильні місця мов програмування та інструментальних засобів розробки.

DOI: 10.31558/2786-9482.2024.1.3

Вступ

ІТ-галузі притаманні стрімкі трансформації, внаслідок яких стек технологій та інструментів дуже швидко змінюється. Деякі технології або мови програмування з'являються та займають ринок, а потім зникають або продовжують своє існування багато років. Може здаватися, що питання швидкодії (оптимізації) програм вже не актуальне, оскільки можна скористатись хмарними технологіями і отримати більше потужностей для розв'язання тих чи інших задач, однак ці потужності надаються не безкоштовно, і за всі використані ресурси треба буде платити. Оптимізація програм за швидкодією може бути корисною у випадках спортивного програмування, розробки складних високонавантажених систем, що займаються розрахунками, або операційних систем реального часу.

Постановка задачі та аналіз публікацій

Під час створення програмного забезпечення розробникам необхідно не лише власноруч реалізовувати програмний код та Unit-тести, а й використовувати різноманітні компілятори / інтерпретатори, алгоритми, бібліотеки, фреймворки, патерни програмування. І тут одразу виникає питання, як на працездатність програми вплинуть нові версії бібліотеки (фреймворку) та версія компілятора чи інтерпретатора, використання тієї чи іншої синтаксичної конструкції; використання тих чи інших опцій компілятора чи інтерпретатора тощо. До того ж розробникам, архітекторам та DevOps-ам потрібно розуміння, чи варто переходити на нову версію програмного продукту, наприклад, з .NET 5 на .NET 6 чи на .NET8, та з яких саме причин – нові можливості розробки, заощадження деяких системних ресурсів тощо.

Зміни безпосередньо у синтаксисі мови програмування є доволі очевидними і дуже рідко залишаються без уваги, так само, як і поява нової мови програмування. Водночас більш глибокі зміни в інструментальних засобах розробки (компіляторах, інтерпретаторів, стандартних бібліотеках) або принципи їх роботи можуть залишатись без уваги.

Сьогодні проблемам програмування присвячено велику кількість робіт. Так, у роботі [1] проводиться порівняння швидкодії арифметичних операцій над великими цілими числами для мов програмування C++ із бібліотекою GMP та Python зі вбудованою підтримкою такої можливості. Під час експериментальних досліджень автори розглядали такі операції над великими цілими числами: множення двох чисел; піднесення до степеня за модулем; знаходження залишку за модулем; знаходження найбільшого спільного дільника. Авторами було встановлено, що реалізація арифметичних алгоритмів над цілими числами довільної точності на C++/GMP є в середньому у 4 рази швидшою за Python-реалізацію, але програмування на C++ є складнішим. Недоліками роботи можна вважати наявність виключно псевдокоду для кожного з алгоритмів, без детальної реалізації на мовах C++ та Python.

Деякі більш глибокі результати порівняння ефективності мов програмування C, C++, Java, Perl, Python, Rexx, Tcl виконано у роботі [2]. Досліджено лише одну алгоритмічну проблему, але над її розв'язанням працювало 74 різні програмісти. Кожен програміст реалізовував код самостійно. Частина суб'єктів була студентами магістратури комп'ютерних наук. Автори аналізували такі характеристики: час роботи програми; обсяг пам'яті, що

споживає програма; розмір початкового коду; щільність коментарів; структура програми; надійність; обсяг зусиль для написання програми. Результати дослідження показали, що для розв'язання проблеми за допомогою скриптових мов програмування (Perl, Python, Rexx, Tcl) потрібно вдвічі менше часу, ніж для мов, що компілюються (C, C++, Java). Щодо часу роботи та споживання пам'яті скриптові мови програмування виявились кращими за Java, та не набагато гіршими ніж, C або C++.

У роботі [3] автори порівнюють швидкодію роботи Java-програми на мікрокомп'ютері Raspberry Pi та класичному настільному комп'ютері. Для проведення дослідження автори створили 2 програми, а саме програму для швидкого алгоритму пошуку простих чисел до заданого цілого числа N (решето Аткіна) та програму для роботи з числами з комою, що плаває. Кількість запусків програми підібрано так, щоб стандартне відхилення не перевищувало 5% від середнього часу роботи програми. У роботі не наведено абсолютних значень швидкості, а лише відносні – швидкість роботи програми на стандартному персональному комп'ютері прийнято за одиницю. В результаті встановлено, що використання Oracle JDK дає змогу виконувати програми у 10–20 разів швидше, порівняно з OpenJDK.

У роботі [4] розглядається проблема обрання мови програмування для навчання або розробки програмного забезпечення за критерієм швидкодії програм. Численні дослідження проводились для таких мов програмування: C, C++, Fortran, Java, C#, PHP, Python, JavaScript. До недоліків роботи можна віднести наявність програмного коду лише для мови програмування C та відсутність опису платформи, на якій проводились дослідження.

У роботі [5] за допомогою мови програмування Java створено вебдодаток та протестована його швидкодія для різних типів серверів у хмарних сервісах AWS EC2, з подальшим визначенням оптимальних варіантів для розгортання цього вебдодатка. Оптимальним, на думу авторів, є таке рішення, яке дає максимальне співвідношення продуктивності та стабільності до ціни утримання такої системи. Для запуску тестів та візуалізації отриманих результатів використовувався інструмент Gatling.

Дослідження енергоефективності як нової метрики для оцінювання ефективності програмного забезпечення здійснено у роботах [6, 7]. Зокрема, в [6] створено рейтинг енергоефективності 27 мов програмування на основі аналізу впливу швидкості виконання програми та використання оперативної пам'яті на рівень спожитої енергії. Лідерами рейтингу є C, Pascal та Ada.

Отже, можна побачити, що дослідження впливу інструментальних засобів розробки та синтаксичних конструкцій на швидкодію програм проведено недостатньою мірою. Також не розкрита повністю проблема використання інформаційних систем для оцінки швидкодії різноманітних програм. ***Метою роботи*** є виявлення впливу інструментальних засобів розробки та синтаксичних конструкцій на швидкодію роботи програм для різних операційних систем, пристроїв, інтерпретаторів та компіляторів за допомогою автоматизованої інформаційної системи.

Інформаційна система для експериментальних досліджень швидкодії

Інформаційна система має відповідати таким функціональним вимогам:

- функціонування системи та запуск тестів має відбуватися для різних мов програмування, компіляторів та інтерпретаторів;
- запуск тестів має здійснюватися на різних пристроях зі збереженням інформації про операційну систему, процесор та обсяг оперативної пам'яті;
- система має давати можливість задавати різні параметри для компіляторів (інтерпретаторів) та середовища виконання;
- для чистоти експерименту буде вимірюватись час роботи виключно якогось алгоритму, а не програми загалом;
- запуск кожної програми відбуватиметься задану кількість разів поспіль з обов'язковим збереженням результатів у базі даних, і для подальшого аналізу враховуватиметься лише найменше значення;
- файли з програмним кодом зберігаються на рівні файлової системи, а не бази даних;
- звіти за тестом генеруються з урахуванням пристрою, на якому він виконувався.

Для реалізації системи візьмемо за основу архітектуру трирівневих баз даних. Клієнтом буде виступати спеціалізований додаток LST Client, на другому рівні буде працювати LST Web Service поверх вебсервера Apache, а на третьому рівні буде розташована реляційна СУБД MySQL (рис. 1).

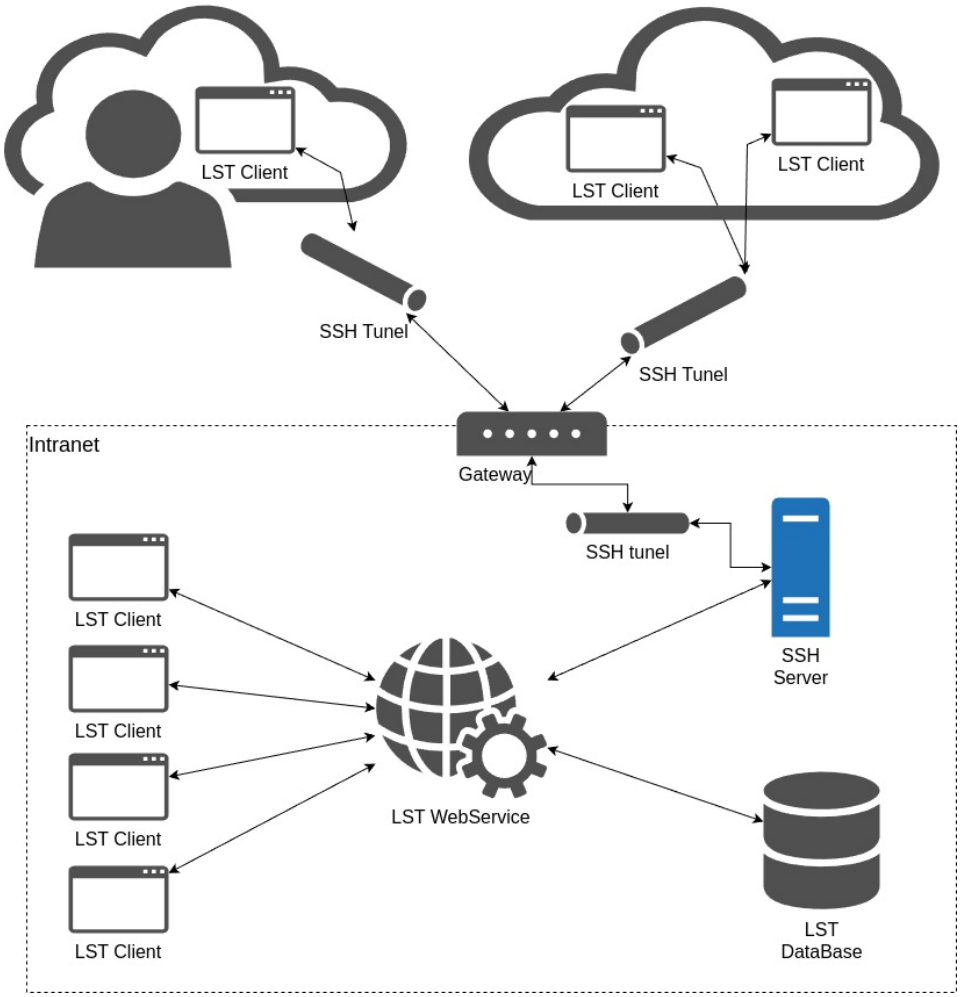

Рисунок 1. Концептуальна схема взаємодії компонентів системи LST

Взаємодія додатка LST Client з LST Web Service відбувається за допомогою Web API з використанням Application Key. Application Key являє собою унікальний ключ, який створюється для кожного пристрою, що бере участь у дослідженні. Для тих випадків, коли у досліджені має взяти участь пристрій, розташований поза межами intranet-мережі, підключення здійснюється аналогічним способом, але через спеціально налаштований SSH-тунель. Для кожного пристрою створюється відповідний користувач з авторизацією за особистим RSA-ключем максимальної довжини. Реляційна база даних інформаційної системи, за винятком компонентів авторизації та безпеки, наведена на рис. 2.

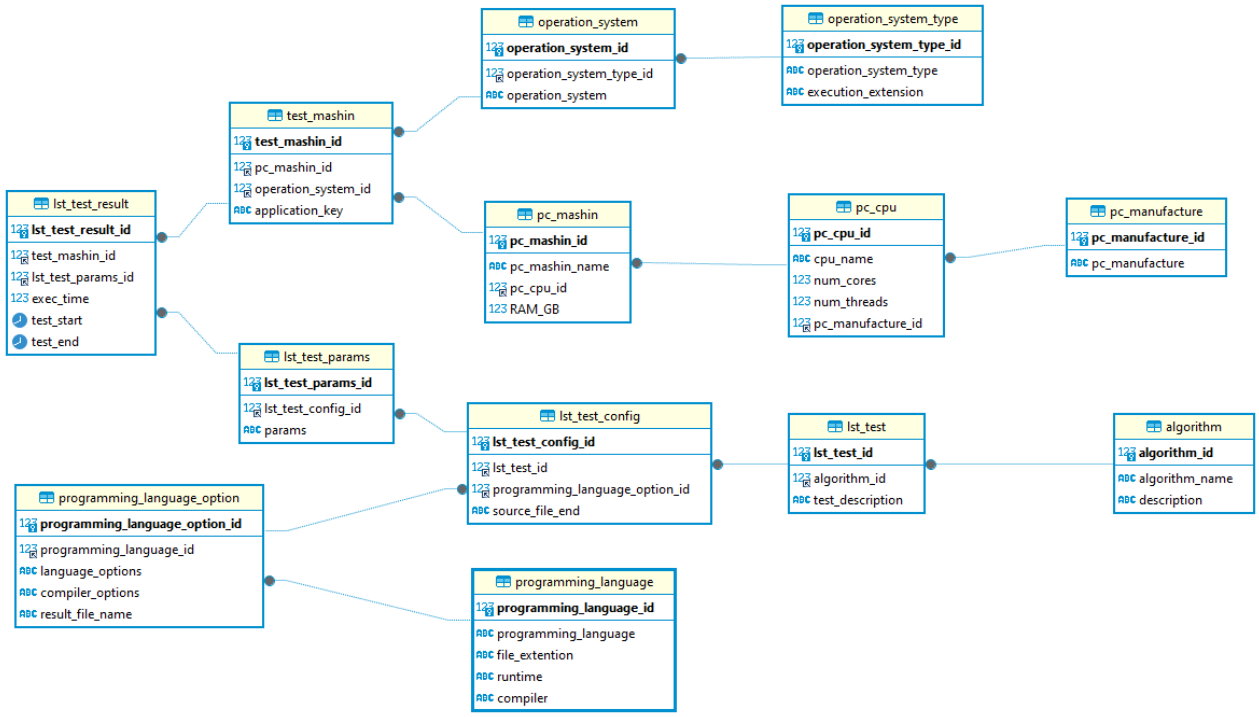

Рисунок 2. Фрагмент бази даних LST

Для експериментального дослідження впливу інструментальних засобів розробки та синтаксичних конструкцій на швидкодію роботи програм пропонується такий тестовий алгоритм:

```
for i = 1…M
   for j = 1…M
      for k = 1…M
         for t = 1…M
            s=i+j+k+t;
```

На перший погляд, мета такого коду – обчислення суми та її накопичення у циклі. Однак, якщо подивитись більш детально на останній рядок, ми побачимо, що накопичення не відбувається, і в результаті у змінній `s` буде зберігатись число, що дорівнює $4M$. Незважаючи на простоту запропонованого алгоритму, він дає змогу розкрити оптимізаційні можливості компіляторів та інтерпретаторів і зрозуміти, наскільки глибоко та якісно відбувається цей процес. За його допомогою можна і відслідковувати, чи відбуваються зміни з виходом нових версій компіляторів або інтерпретаторів.

Експерименти проведемо для компіляторів та інтерпретаторів мов програмування з табл. 1. Усі обчислення проведемо для хостів з табл. 2 для компіляторів / інтерпретаторів, що належать до x64 архітектури.

Таблиця 1. Специфікація інструментальних засобів

Мова	Інструмент	Псевдонім
C	gcc	C
C++	g++	C++
Fortran	gfortran	Fortran
Java	javac	Java
C#	csc	C# .NET
C#	dotnet core3	C# Core
JavaScript	Node JS	JavaScript
PHP	php5	PHP 5
PHP	php7	PHP 7
Python	pypy	Python pypy
Python	python3	Python 3

Таблиця 2. Параметри хостів

Machine ID	CPU	RAM	OS
4	Core i3 6100	8	Windows 10 для освітніх установ x64
6	Core i5 10500	32	Windows 10 для освітніх установ x64

Тестовий алгоритм реалізуємо мовою програмування C так:

```
for(int i = 1; i <= M; i++)
   for(int j = 1; j <= M; j++)
      for(int k = 1; k <= M; k++)
         for(int t = 1; t <= M; t++)
            s=i+j+k+t;
```

Для аналізу роботи компілятора створимо 4 різні програми – без оптимізації та з використанням трьох стандартних опцій -O, -O2 та -O3. Додатково створимо ще одну версію коду, в якій замість `i++` використовувався варіант `++i`. У табл. 3 наведено результати досліджень для різних значень M, з яких видно, що неоптимізований код у 5–6 разів повільніший, ніж код, створений за допомогою опції -O. Для опцій -O2 та -O3 час виконання програм дорівнює нулю, оскільки компілятор зміг визначити, що змінна `s` змінюється у циклі, але остаточне значення обчислюється на останній ітерації циклу. Компілятор обчислює це значення ($4M$) не вставляючи цикли в код програми. Для мови програмування C++ код алгоритму буде виглядати так само, як і для мови C, а результати та закономірності майже ідентичні результатам для мови C (табл. 4). Тривалість виміряно у секундах.

Таблиця 3. Тривалість виконання тестових програм на мові C

Тест	Machine ID	*M*					
		50	100	150	200	250	300
C(i++)	4	0	0.156	0.828	2.594	6.344	13.095
C(++i)	4	0	0.156	0.828	2.594	6.329	13.033
C(-O; i++)	4	0	0.031	0.156	0.484	1.156	2.36
C(-O; ++i)	4	0	0.031	0.156	0.484	1.141	2.36
C(-O2, -O3; i++)	4	0	0	0	0	0	0
C(i++)	6	0.009	0.142	0.702	2.199	5.335	10.998
C(++i)	6	0.01	0.145	0.708	2.199	5.324	11.017
C(-O; i++)	6	0.002	0.028	0.137	0.408	0.975	1.994
C(-O; ++i)	6	0.002	0.028	0.132	0.408	0.975	1.995
C(-O2, -O3; i++)	6	0	0	0	0	0	0

Таблиця 4. Тривалість виконання тестових програм на мові C++

Тест	Machine ID	*M*					
		50	100	150	200	250	300
C++(i++)	4	0	0.156	0.828	2.61	6.344	13.142
C++(++i)	4	0	0.156	0.828	2.594	6.329	13.095
C++(-O; i++)	4	0	0.031	0.156	0.484	1.172	2.391
C++(-O; ++i)	4	0	0.031	0.156	0.484	1.172	2.407
C++(-O2,-O3; i++)	4	0	0	0	0	0	0
C++(i++)	6	0.009	0.142	0.702	2.199	5.325	10.998
C++(++i)	6	0.009	0.142	0.703	2.199	5.316	11
C++(-O; i++)	6	0.002	0.028	0.136	0.416	0.991	2.023
C++(-O; ++i)	6	0.002	0.029	0.137	0.416	0.992	2.02
C++(-O2,-O3; i++)	6	0	0	0	0	0	0

Код тестової програми на мові Fortran виглядає так:

```
do i=1,M
   do j=1,M
      do k=1,M
         do l=1,M
            s=i+j+k+l
         end do
      end do
   end do
end do
```

Опції компіляції -O2 та -O3 для Fortran забезпечують нульовий час виконання програми. Неоптимізований код виявився повільніший у 6.5–7.2 разів, ніж код, створений за допомогою опції -O (табл. 5).

Таблиця 5. Тривалість виконання тестових програм на мові Fortran

Тест	Machine ID	*M*					
		50	100	150	200	250	300
Fortran	4	0	0.203	1.063	3.406	8.375	17.391
Fortran(-O)	4	0	0.031	0.153	0.484	1.172	2.406
Fortran(-O2, -O3)	4	0	0	0	0	0	0
Fortran	6	0	0.172	0.891	2.844	7.031	14.625
Fortran(-O)	6	0	0.017	0.125	0.406	0.984	2.031
Fortran(-O2, -O3)	6	0	0	0	0	0	0

Для мов Java та C# програмний код для реалізації алгоритму буде виглядати так само, як і для мови C. З даних, наведених у табл. 6 та 7, видно, що під час використання `i++` та `++i` різниця між часом виконання програми майже не помітна; вона знаходиться у межах похибки. Позначка Aggres означає, що програма Java запускалася з параметром `-XX: +AggressiveOpts`. Використання оптимізації для мови C# дає змогу збільшити швидкість роботи програми вдвічі за компілятора csc.exe та у 6 разів за інструмента dotnet run. За збільшення кількості ітерацій результати для Java, оптимізованого C# .NET та C# Core стають майже однаковими.

Таблиця 6. Тривалість виконання тестових програм на мові Java

Тест	Machine ID	*M*					
		50	100	150	200	250	300
Java(i++)	4	0.005	0.046	0.172	0.5	1.187	2.406
Java(++i)	4	0.005	0.031	0.172	0.5	1.172	2.391
Java(Aggres; i++)	4	0.006	0.047	0.171	0.5	1.187	2.406
Java(Aggres; ++i)	4	0.005	0.031	0.172	0.5	1.172	2.391
Java(i++)	6	0.004	0.033	0.144	0.428	1.009	2.042
Java(++i)	6	0.004	0.033	0.144	0.427	1.006	2.062
Java(Aggres; i++)	6	0.004	0.032	0.145	0.43	1.012	2.052
Java(Aggres; ++i)	6	0.005	0.032	0.144	0.428	1.01	2.069

Таблиця 7. Тривалість виконання тестових програм на мові C#

Тест	Machine ID	*M*					
		50	100	150	200	250	300
C#.NET(i++)	4	0.062	0.122	0.369	1.022	2.383	4.833
C#.NET(++i)	4	0.061	0.122	0.368	1.021	2.379	4.82
C#.NET(/o; i++)	4	0.06	0.092	0.22	0.555	1.24	2.469
C#.NET(/o; ++i)	4	0.059	0.091	0.221	0.553	1.236	2.464
C#Core(++i, i++)	4	0.004	0.038	0.168	0.515	1.22	2.49
C#.NET(i++)	6	0.006	0.057	0.267	0.819	1.962	4.022
C#.NET(++i)	6	0.006	0.057	0.267	0.819	1.957	4.022
C#.NET(/o; i++)	6	0.004	0.032	0.143	0.426	1.008	2.049
C#.NET(/o; ++i)	6	0.004	0.032	0.142	0.426	1.008	2.049
C#.NET8(i++)	6	0.012	0.16	0.774	2.407	5.83	12.02
C#.NET8(opt i++)	6	0.01	0.037	0.146	0.425	1.002	2.034
C#.NET6(i++)	6	0.012	0.158	0.77	2.401	5.833	12.011
C#.NET6(opt i++)	6	0.004	0.032	0.14	0.422	0.999	2.034
C#.NET3.1(i++)	6	0.013	0.159	0.773	2.405	5.837	12.029
C#.NET3.1(opt i++)	6	0.004	0.032	0.141	0.423	1	2.034

Для мови програмування JavaScript програмний код алгоритму буде виглядати так само, як і для мови C. З табл. 8 видно, що операція `++i` виконується швидше, ніж `i++`, на 1.2–4.8%, залежно від кількості ітерацій для хоста №4 зі старою версією Node.JS, тоді як для хоста №6 з Node.JS 22 ця різниця є непомітною.

Таблиця 8. Тривалість виконання тестових програм на мові JavaScript

Тест	Machine ID	*M*					
		50	100	150	200	250	300
JavaScript(i++)	4	0.027	0.398	1.991	6.258	15.141	31.047
JavaScript(++i)	4	0.026	0.389	1.925	6	14.589	29.987
JavaScript(i++)	6	0.008	0.059	0.264	0.802	1.936	3.924
JavaScript(++i)	6	0.007	0.0589	0.264	0.798	1.925	3.887

Для мови програмування PHP програмний код алгоритму, що розглядається, буде таким самим, як і для мови програмування C. Використання у мові програмування PHP конструкції `++i` робить програму на 6–10% швидшою, порівняно з варіантом, де використовується конструкція `i++` (табл. 9). Для хоста №4 інтерпретатор PHP 7 на 28% швидший за інтерпретатор PHP 5 як для `++i`, так і для `i++`. Для хоста №6 за конструкції `i++` маємо, що PHP 7 на 16.5–18% швидший за інтерпретатор PHP 8, та на 84–85% швидший

за PHP 5. Для синтаксичної конструкції `++i` PHP 7 на 31% швидший за інтерпретатор PHP 8 та на 91% швидший за PHP 5 (рис. 4).

Таблиця 9. Тривалість виконання тестових програм на мові PHP

Тест	Machine ID	M					
		50	100	150	200	250	300
PHP5(i++)	4	0.241	3.818	19.206	60.75	147.946	306.009
PHP5(++i)	4	0.219	3.494	17.487	55.205	134.789	279.996
PHP7(i++)	4	0.186	2.953	14.95	47.2113	115.179	238.54
PHP7(++i)	4	0.17	2.716	13.637	43.02	105.211	218.85
PHP5(i++)	6	0.227	3.555	17.979	56.485	137.732	284.984
PHP5(++i)	6	0.212	3.311	16.736	52.611	128.59	266.465
PHP7(i++)	6	0.122	1.92	9.689	30.593	74.543	154.614
PHP7(++i)	6	0.112	1.736	8.737	27.632	67.462	139.636
PHP8(i++)	6	0.146	2.268	11.287	36.09	88.067	182.346
PHP8(++i)	6	0.146	2.269	11.418	36.076	88.188	182.612

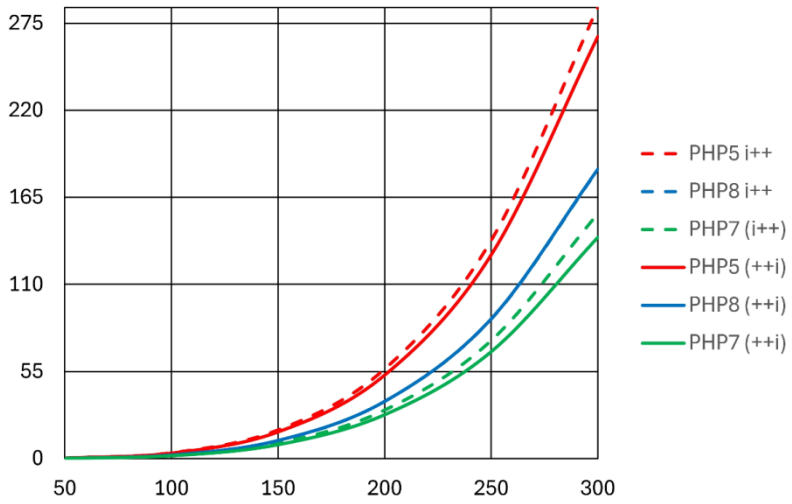

Рисунок 4. Тривалість виконання тестової програми на PHP для хоста №6

Базовий код на Python виглядає так:

```
startTime=time.time()
for i in range(1,M):
for j in range(1,M):
for k in range(1,M):
for t in range(1,M):
S=i+j+k+t
endTime=time.time()
```

Для перевірки гіпотези, що код на Python виконується швидше, якщо він розміщений в окремій функції, створимо ще таку програму:

```
def test(M):
S=0
myr=range(1,M)
for i in myr:
   for j in myr:
      for k in myr:
         for t in myr:
            S=i+j+k+t
return S
startTime=time.time()
S=test(M)
endTime=time.time()
```

Для хоста №4 використання коду у вигляді функції підвищує швидкодію на 56–62% за стандартного інтерпретатора Python, та на 53–73% для інтерпретатора pypy (табл. 10). Для хоста №6 використання функції підвищує швидкодію на 99–119% за стандартного інтерпретатора Python та на 87–93% за інтерпретатора pypy. Використання pypy пришвидшує програми у 42–63 рази.

Таблиця 10. Тривалість виконання тестових програм на мові Python

Тест	Machine ID	*M*					
		50	100	150	200	250	300
Python 3	4	0.688	10.767	57.756	197.302	504.366	1114.116
Python 3(proc)	4	0.464	7.186	35.55	121.888	319.627	713.882
Python pypy	4	0.015	0.265	1.266	3.938	9.485	19.517
Python pypy(proc)	4	0.016	0.141	0.719	2.432	6.198	11.266
Python 3	6	0.488	7.662	42.353	149.586	370.926	818.553
Python 3(proc)	6	0.19	3.131	19.31	70.315	185.049	410.683
Python pypy	6	0.018	0.177	0.85	2.635	6.4038	13.224
Python pypy(proc)	6	0.01	0.097	0.453	1.39	3.333	6.85

Висновки

Проведені дослідження показали, що:

- усі мови програмування та інструменти розробки мають свої тонкощі та особливості, які мають бути досліджені;
- у разі декількох компіляторів / інтерпретаторів їх поведінка, параметри та якість створених програм можуть суттєво відрізнятися;
- з випуском нових версій компіляторів / інтерпретаторів, ситуація може суттєво змінюватися;
- операційна система та апаратно-програмне забезпечення впливають на результати тестів;
- синтаксичні конструкції `++i` та `i++` можуть по-різному оброблюватися компіляторами / інтерпретаторами та впливати на роботу програми;
- лише компілятори мов програмування C, C++ та Fortran змогли оптимізувати код та не

створювати 4 зайві цикли. Для інших інструментальних засобів ситуація залишається незмінною вже протягом 7 років.

Розроблена інформаційна система дає змогу максимально автоматизувати процес тестування швидкодії програмного коду з його подальшим аналізом. Система може бути використана для таких завдань:

- виявлення кращих практик та слабких місць обраного стеку технологій під час розробки програмного забезпечення;
- оцінювання нових версій мов програмування та програмного забезпечення на предмет доцільності переходу на них;
- акцентування уваги студентів на слабких та сильних місцях мов програмування та інструментальних засобів розробки під час викладання дисциплін із програмуванням.

Проведені тести не є вичерпними та не відображають повної картини, але достатньою мірою ілюструють мінливість результатів залежно від способу написання коду, параметрів компілятора / інтерпретатора та середовища виконання. Розглянута у роботі система, після мінімальних змін також може бути задіяна у автоматизованих системах контролю знань з метою перевірки правильності написання програм.

Література

1. Новокшонов, А. К. (2016). Аналіз ефективності реалізації арифметичних алгоритмів на мовах програмування C++ та Python. *Проблеми програмування*, (2–3), 26–31. https://doi.org/10.15407/pp2016.02-03.026
2. Prechelt, L. (2000). An empirical comparison of C, C++, Java, Perl, Python, Rexx and Tcl. *IEEE Computer*, *33*(10), 23–29.
3. Дідух, О. І., Тищенко, В. В. (2015). Порівняння швидкодії Java на мікрокомп'ютері Raspberry Pi. *Вісник Національного технічного університету України «Київський політехнічний інститут». Серія: Радіотехніка. Радіоапаратобудування*, *60*, 107–113.
4. Антонов, Ю. С., Дзігора, К. Р. (2017). Проблема обрання мови програмування, як інструменту для навчання та розробки. *Матеріали наукової конференції професорсько-викладацького складу, наукових працівників і здобувачів наукового ступеня за підсумками науково-дослідної роботи за період 2015–2016 рр.* (с. 23–25). Донецький національний університет імені Василя Стуса.
5. Сігунов, О., Демків, Л. (2022). Дослідження швидкодії обробки паралельних запитів хмарними сервісами AWS. *Електроніка та інформаційні технології*, *20*, 30–41. http://dx.doi.org/10.30970/eli.20.4
6. Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J. P., & Saraiva, J. (2021). Ranking programming languages by energy efficiency. *Science of Computer Programming*, *205*. https://doi.org/10.1016/j.scico.2021.102609
7. Gordillo, A., Calero, C., Moraga, M. Á. et al. (2024). Programming languages ranking based on energy measurements. *Software Quality Journal*. https://doi.org/10.1007/s11219-024-09690-4

Рукопис отримано – 06/08/2024; прийнято до публікації – 17/08/2024.

Dependence of program speed on development tools and syntactic constructions

Yuriy Antonov

Abstract

This article proposes the implementation of an automated information system that allows you to research and analyze the influence of development tools and syntactic structures on the speed of the programs. A fragment of the database scheme of this system and a conceptual scheme of the interaction of components are given. The requirements for such a system are formulated. During the implementation of the system, the architecture of three-level databases was used. The specialized application LST Client was used as the client, LST WebService and the MySQL relational DBMS were used as the application server. The LST Client application interacts with the LST Web Service using the Web API and the Application Key. The Application Key is a unique key that is created for each device participating in the study. For those cases when the device to participate in the research is located outside the intranet network, the connection is made in a similar way, but through a specially configured SSH tunnel. For each device, a corresponding user is created with authorization based on a personal RSA key of the maximum length. Numerical studies were performed for programming languages C, C++, Fortran, Java, C#, JavaScript, PHP, Python using compilers/interpreters belonging to the x64 architecture running Windows 10 for educational institutions. The conducted research made it possible to establish that the speed of the program is affected by the version of the compiler / interpreter and the set of syntax constructions used, for example ++i and i++. Only C, C++, and Fortran compilers were able to optimize the code and avoid unnecessary loops. The developed system makes it possible to automate the process of testing the speed of the software code as much as possible. This system can be used to evaluate new versions of programming languages and software. The obtained results make it possible to evaluate the expediency of switching to new versions of the software or to focus students' attention on the weak and strong points of a certain programming language and development tools when teaching disciplines related to programming.

Keywords: Compiler; interpreter; program operation speed; optimization of the program.

References

1. Novokshonov, A. K. (2016). Analiz efektyvnosti realizatsii aryfmetychnykh alhorytmiv na movakh prohramuvannia C++ ta Python. *Problemy prohramuvannia*, (2–3). 26–31. https://doi.org/10.15407/pp2016.02-03.026
2. Prechelt, L. (2000). An empirical comparison of C, C++, Java, Perl, Python, Rexx and Tcl. *IEEE Computer*, *33*(10), 23–29.
3. Didukh, O. I., Tyshchenko, V. V. (2015). Porivniannia shvydkodii Java na mikrokompiuteri Raspberry Pi. *Visnyk Natsionalnoho tekhnichnoho universytetu Ukrainy «Kyivskyi politekhnichnyi instytut». Seriia: Radiotekhnika. Radioaparatobuduvannia, 60*, 107–113.
4. Antonov, Yu. S., Dzihora K. R. (2017). Problema obrannia movy prohramuvannia, yak instrumentu dlia navchannia ta rozrobky. *Materialy naukovoi konferentsii profesorsko-vykladatskoho skladu, naukovykh pratsivnykiv i zdobuvachiv naukovoho stupenia za pidsumkamy naukovo-doslidnoi roboty za period 2015–2016 rr.* (s. 23–25). Donetskyi natsionalnyi universytet imeni Vasylia Stusa.
5. Sihunov, O., Demkiv, L. (2022). Doslidzhennia shvydkodii obrobky paralelnykh zapytiv khmarnymy servisamy AWS. *Elektronika ta informatsiini tekhnolohii*. *20*. 30–41. http://dx.doi.org/10.30970/eli.20.4
6. Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J. P., & Saraiva, J. (2021). Ranking programming languages by energy efficiency. *Science of Computer Programming*, *205*. https://doi.org/10.1016/j.scico.2021.102609
7. Gordillo, A., Calero, C., Moraga, M. Á. et al. (2024). Programming languages ranking based on energy measurements. *Software Quality Journal.* https://doi.org/10.1007/s11219-024-09690-4