2. *Nuriev N.K., Starygina S.D.* Didakticheskaya inzheneriya: razrabotka reglamenta pedagogicheskogo testirovaniya // Obrazovatel'nye tekhnologii i obshchestvo. 2017. № 20 (4).
3. *Pechenyj E.A., Starygina S.D.* Didakticheskaya inzheneriya: model' postroeniya optimal'nogo raspisaniya dlya potochnogo testirovaniya // Obrazovatel'nye tekhnologii i obshchestvo. 2017. № 4. S. 430–442.
4. *Starygina S.D., Nuriev N.K., Pechenyj E.A.* Didakticheskaya inzheneriya: otsenka slozhnosti i prodolzhitel'nosti testa dostizhenij // Science of Europe. 2017. Vol 2, no. 14. P. 17–21.
5. *Khuziakhmetova A.R., Nuriev N.K., Khuziakhmetova R.N.* Otsenka prodolzhitel'nosti testirovaniya v zavisimosti ot slozhnosti testa // Sovremennye problemy nauki i obrazovaniya. 2019. № 3.

DOI: 10.25586/RNU.V9187.20.03.P.163 УДК 004.051

А.Ф. Зайцев

СРАВНИТЕЛЬНЫЙ АНАЛИЗ ПРОИЗВОДИТЕЛЬНОСТИ ИНТЕРПРЕТАТОРОВ CPYTHON И SBCL

Показана методика и результаты сравнительного анализа производительности интерпретаторов CPython и SBCL. Методика предполагает использование метода имитационного моделирования. Путем многократного запуска и имитации вычислений, реализуемых одним и тем же алгоритмом, представленным в виде двух программ для языков программирования Lisp и Python, производится измерение количества затраченного времени в процессе интерпретации. Полученные результаты измерений можно будет сравнить между собой и выяснить, какой из интерпретаторов более эффективен, для дальнейшего выбора при решении конкретных задач.
Ключевые слова: анализ, сравнение, производительность, интерпретатор, трансляция, программирование, CPython, SBCL.

A.F. Zaytsev

COMPARATIVE ANALYSIS OF PERFORMANCE CPYTHON AND SBCL INTERPRETERS

This article shows the approach and benchmarking results of CPython and SBCL interpreters. The approach involves the use method of imitation simulation. By repeatedly running and imitating computations implemented according to the same algorithm, presented in the form of two programs for the programming languages Lisp and Python, the amount of time spent on the interpretation process is measured. The obtained measurement results can be compared with each other and, thus, find out which of the interpreters is more effective for further choice when solving specific problems.
Keywords: analysis, comparison, performance, interpreter, translation, programming, CPython, SBCL.

Введение

В данной работе предложена методика проведения сравнительного анализа производительности интерпретаторов двух разных языков программирования – Lisp и Python.

Язык Lisp был создан Джоном Маккарти в 1958 г. [4; 6]. За все время существования данного языка для него было разработано огромное количество интерпретаторов. Самым современным и производительным на сегодняшний день является SBCL (Steel Bank Common Lisp) – бесплатный интерпретатор, разработанный в соответствии со стандартом диалекта Common Lisp [1; 3]. Его разработка была начата в декабре 1999 г., а последняя версия выпущена в марте 2020 г. Исходные коды интерпретатора доступны для загрузки с официального сайта [9]. В документации он указывается как компилятор из-за дополнительной возможности трансляции не в промежуточное представление, а непосредственно в машинный код, налету (jit-компиляция). Эта возможность задействована по умолчанию. Однако есть возможность использовать режим интерпретатора.

В настоящее время наиболее популярным языком программирования является Python [7; 8]. Он широко используется в научном сообществе. Его дизайн ориентирует на написание хорошо читаемого кода, а синтаксис позволяет описывать алгоритмы за меньшее число строк, чем это возможно на C++ или Java. Синтаксис языка очень прост и краток, а стандартная библиотека включает в себя большой объем полезных функций и удобные структуры данных. Язык интерпретируемый и кроссплатформенный. Все это очень важно при разработке больших и сложных систем. Для языка Python тоже существует множество интерпретаторов. CPython является стандартным официальным интерпретатором языка. Его исходный код доступен для изучения и загрузки на сайте github [10].

Цель исследования

Целью исследования является анализ производительности двух интерпретаторов – SBCL и CPython – при обработке алгоритмов, использующих в частности такие структуры данных, как списки и деревья.

Для достижения цели необходимо выполнить следующие задачи:

- разработать методику проведения сравнительного анализа производительности выбранных интерпретаторов;
- провести вычислительный эксперимент, согласно предложенной методике;
- выполнить сравнительный анализ результатов, полученных в ходе эксперимента.

Важно также отметить, что цель данного исследования не предполагает разработку какого-то универсального «бенчмарка». Предложенный подход является лишь частным случаем бенчмаркинга. Необходимость данного подхода заключается в обосновании выбора того или иного интерпретатора при разработке программного обеспечения в дальнейших исследованиях автора.

Материал и методы исследования

В качестве материалов исследования используются электронно-вычислительные машины и инструментальные средства программного обеспечения – языки программирования (Lisp, Python) и их интерпретаторы (SBCL, CPython).

Исследование предполагает использование таких научных методов, как анализ, синтез, формализация, имитация, моделирование, алгоритмизация, тестирование, оценивание, сравнение. Методика исследования предполагает использование метода имитационного

(логико-математического) моделирования. Путем многократного запуска и имитации вычислений, реализуемых одним и тем же алгоритмом, представленным в виде двух программ для языков программирования Lisp и Python, производится измерение количества затраченного времени в процессе интерпретации. Полученные результаты измерений можно будет сравнить между собой и выяснить, какой из интерпретаторов более эффективен.

Основная часть

Для проведения вычислительного эксперимента согласно предложенной методике необходимо реализовать некий имитационный алгоритм в виде двух программ на языках программирования Lisp и Python. В качестве реализации был взят и модифицирован алгоритм обработки бинарных деревьев из проекта The Computer Language Benchmarks Game [2; 5]. Данный алгоритм рекурсивно создает множество двоичных деревьев с указанной высотой, выполняет их обход, добавляет новые узлы, считает их количество, а после удаляет, освобождая занятую память. При этом всегда существует хотя бы одно двоичное «долгоживущее» дерево, пока другие деревья создаются и освобождаются. Данного алгоритма вполне достаточно для проверки производительности интерпретаторов, так как в его реализации используются такие структуры данных, как списки и деревья. Модификация алгоритмов включает в себя код, необходимый для измерения производительности. В частности для алгоритма на языке Python были задействованы функции из модуля timeit.

На рисунке 1 показаны результаты анализа производительности интерпретатора CPython, полученные путем имитации вычислений при интерпретации описанного выше алгоритма.

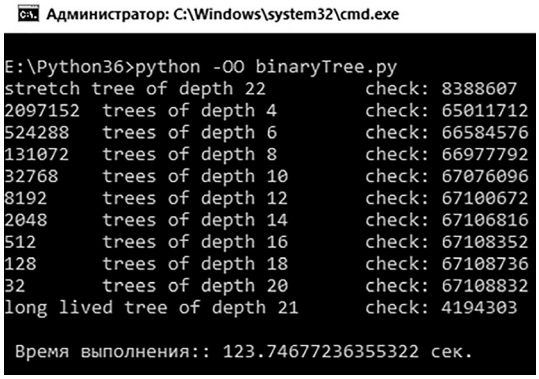

Рис. 1. Результаты анализа производительности интерпретатора CPython

В процессе имитации сначала было создано 2097152 бинарных дерева с высотой, равной четырем, и был выполнен обход 8388607 узлов. Затем значение высоты увеличивалось, и вычисления повторялись заново. Результаты показывают, что общее время интерпретации алгоритма составило 123,746 сек.

Интерпретатор SBCL в отличие от CPython может работать в двух режимах – простой интерпретации и jit-компиляции. По умолчанию активен режим jit-компиляции. Данный

режим подразумевает использование одноименного метода, который был изобретен для увеличения производительности интерпретаторов. Основная идея такого метода заключается в дополнительной трансляции байт-кода в машинный код и его выполнении непосредственно во время работы программы.

На рисунке 2 темно-серым цветом отмечена фаза jit-компиляции.

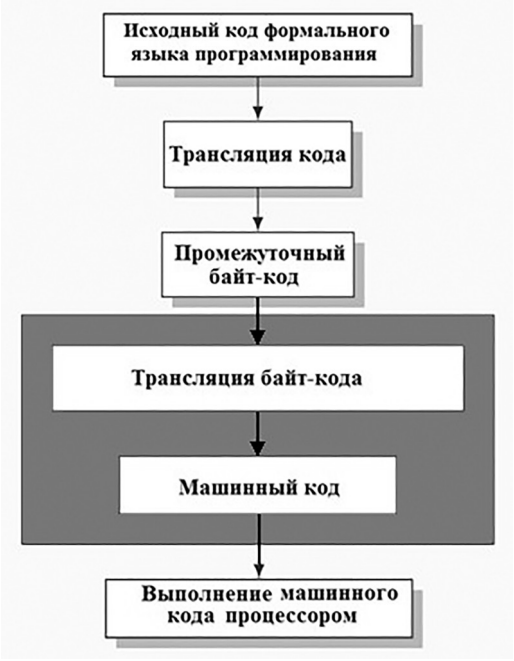

Рис. 2. Фаза jit-компиляции в процессе работы интерпретатора

Таким образом, достигается более высокая скорость вычислений по сравнению с интерпретируемым напрямую байт-кодом.

Так как интерпретатор CPython не имеет возможности jit-компиляции, измерение производительности интерпретатора SBCL будет производиться в режиме простой интерпретации. Чтобы переключиться в режим простой интерпретации в SBCL, необходимо перед кодом любого алгоритма в самом начале добавить команду (setfsb-ext:*evaluator-mode* :interpret).

Для анализа производительности в интерпретаторе SBCL имеется встроенная функция time. Данная функция может принимать в качестве аргумента другую функцию или выражение, а по окончании его выполнения отображать информацию о затраченном времени и количеству используемой памяти.

На рисунке 3 показаны результаты работы функции time, отображающие информацию о производительности при интерпретации описанного выше алгоритма, реализованного на языке Lisp.

Из результата видно, что время работы алгоритма составило 756,246 сек. При этом количество созданных деревьев и другие значения абсолютно совпадают с представленными на рисунке 1.

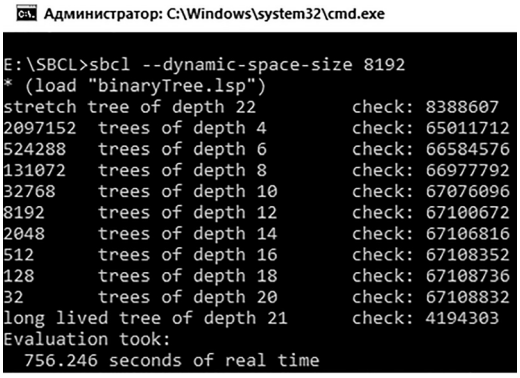

Рис. 3. Результаты анализа производительности интерпретатора SBCL

Результаты исследования и их обсуждение

Основные результаты сравнительного анализа производительности интерпретаторов CPython и SBCL приведены в таблице.

Результаты сравнительного анализа производительности интерпретаторов

Язык / интерпретатор	**Производительность (время интерпретации), с**
Lisp / SBCL 2.0.0	756,246
Python / CPython 3.6	123,746

Из результатов сравнительного анализа следует, что интерпретатор CPython более производительный, чем интерпретатор SBCL.

Важно понимать, что результаты времени интерпретации, полученные с разных компьютеров, могут значительно отличаться, так как зависят не только от используемых настроек интерпретаторов, но и от архитектурной и операционной платформы.

Представленные результаты были получены при использовании процессора IntelCore i5-3470 (3.20 GHz) и операционной системы Windows 10 Pro (версия 10.586, 64bit). Версии обоих интерпретаторов также 64-разрядные.

Чтобы избежать большого различия получаемых результатов, предлагается использовать единый интернет-ресурс по запуску и выполнению алгоритмов, например wandbox.org [11], или другой, более подходящий. При этом все вычисления будут производиться на едином сервере с одинаковыми настройками интерпретаторов, единой архитектурной и операционной платформой.

Также стоит отметить, что интерпретатор SBCL в режиме jit-компиляции оказался бы намного быстрее CPython. Но такое сравнение было бы не совсем корректным ввиду отсутствия данного режима в интерпретаторе CPython. Для проведения подобного сравнительного анализа необходимо произвести поиск и замену интерпретатора CPython на аналогичный с поддержкой режима jit-компиляции, например PyPy.

Заключение

В процессе исследования была предложена и продемонстрирована методика сравнительного анализа производительности интерпретаторов CPython и SBCL.

Данную методику можно применить не только для сравнительного анализа производительности различных интерпретаторов (при реализации идентичного алгоритма), но и при проведении анализа больших информационных систем и программных комплексов. Например, подобным образом можно измерить эффективность работы внутренних алгоритмов систем, имеющих структурные отличия в реализации, но решающих одну и ту же задачу, с целью их оптимизации или наилучшего выбора.

Подводя итог, можно сделать вывод, что проведение сравнительного анализа производительности интерпретаторов действительно является важным шагом в процессе разработки больших информационных систем. От выбора наиболее подходящего интерпретатора и его настроек практически на пятьдесят процентов зависит то, как эффективно будет работать любая информационная система. Кроме того, стоит уделять внимание анализу эффективности алгоритмов, производя их сравнение, модификацию и оптимизацию с учетом современных тенденций.

Литература

1. *Грэм П.* ANSI Common Lisp. СПб.: Символ-Плюс, 2012. 448 с.
2. Binary-Trees Python 3 #3 Program. Description // The Computer Language Benchmarks Game. URL: https://benchmarksgame-team.pages.debian.net/benchmarksgame/program/binarytrees-python3-3.html (date of the application: 22.06.2020).
3. *Graham P.* ANSI Common Lisp. N. Y.: Prentice Hall, 1996. 432 p.
4. *Knott Gary D*. Interpreting LISP: Programming and Data Structures. Berkeley: Apress, 2017. 144 p.
5. Lisp Binary-Trees Lisp SBCL #3 Program. Description // The Computer Language Benchmarks Game. URL: https://benchmarksgame-team.pages.debian.net/benchmarksgame/program/binarytrees-sbcl-3.html (date of the application: 22.06.2020).
6. *McCarthy J.* LISP 1.5 Programming Manual. Cambridge: The MIT Press, 1963. 106 p.
7. *Rossum van G.* Python Tutorial Release 3.8.1. [S. l.]: Python Software Foundation, 2020. 141 p.
8. *Rossum van G.* Python Tutorial. Technical Report CS-R9526. [S. l.]: Centrum Wiskundeen Informatica (CWI), 1995. 65 p.
9. Steel Bank Common Lisp. URL: http://www.sbcl.org/platform-table.html (date of the application: 22.06.2020).
10. The Python Programming Language. URL: https://github.com/python/cpython (date of the application: 22.06.2020).
11. Wandbox. URL: https://wandbox.org (date of the application: 22.06.2020).

Literatura

1. *Grem P.* ANSI Common Lisp. SPb.: Simvol-Plyus, 2012. 448 s.
2. Binary-Trees Python 3 #3 Program. Description // The Computer Language Benchmarks Game. URL: https://benchmarksgame-team.pages.debian.net/benchmarksgame/program/binarytrees-python3-3.html (date of the application: 22.06.2020).

3. *Graham P.* ANSI Common Lisp. N. Y.: Prentice Hall, 1996. 432 p.
4. *Knott Gary D.* Interpreting LISP: Programming and Data Structures. Berkeley: Apress, 2017. 144 p.
5. Lisp Binary-Trees Lisp SBCL #3 Program. Description // The Computer Language Benchmarks Game. URL: https://benchmarksgame-team.pages.debian.net/benchmarksgame/program/binarytrees-sbcl-3.html (date of the application: 22.06.2020).
6. *McCarthy J.* LISP 1.5 Programming Manual. Cambridge: The MIT Press, 1963. 106 p.
7. *Rossum van G.* Python Tutorial Release 3.8.1. [S. l.]: Python Software Foundation, 2020. 141 p.
8. *Rossum van G.* Python Tutorial. Technical Report CS-R9526. [S. l.]: Centrum Wiskundeen Informatica (CWI), 1995. 65 p.
9. Steel Bank Common Lisp. URL: http://www.sbcl.org/platform-table.html (date of the application: 22.06.2020).
10. The Python Programming Language. URL: https://github.com/python/cpython (date of the application: 22.06.2020).
11. Wandbox. URL: https://wandbox.org (date of the application: 22.06.2020).