УДК 004.05

О ВОПРОСЕ ИЗМЕРЕНИЯ ВКЛАДА ПРОГРАММИСТСКИХ РЕШЕНИЙ В ПРОИЗВОДИТЕЛЬНОСТЬ ПРОГРАММ

Л. В. Городняя[1] [0000-0002-4639-9032], Т. А. Андреева[2] [0000-0002-1124-9499]

[1,2]*Институт систем информатики им. А.П. Ершова СО РАН, 630090 г. Новосибирск, проспект Академика Лаврентьева, 6*

[1,2]*Новосибирский государственный университет, 630090, Новосибирск, ул. Пирогова, 1*

[1]gorod@iis.nsk.su, [2]ata@iis.nsk.su

Аннотация

Статья нацелена на привлечение внимания к вопросам, возникающим в связи с проблемой оценки влияния программируемых решений на продуктивность программирования и производительность программ в процессе обучения программированию и улучшения программных приложений с сохранением их правильности. Проанализированы результаты некоторых экспериментов по этим вопросам. Предложена гипотеза, что функциональные модели могут быть полезны как метрическая шкала, позволяющая отделять особенности используемых языков и систем программирования от характеристик программ и запрограммированных решений. Описаны результаты небольшого демонстрационного эксперимента, направленного на исследование зависимости производительности программ от выбора компилятора и отдельно от представления программируемых решений на определённом языке программирования. Анализ полученных результатов позволяет наметить подход к созданию методики, позволяющей выяснять такие зависимости. При создании методики может быть учтён многолетний опыт оценки учебных и олимпиадных работ по программированию, проявивший ряд не вполне очевидных аспектов проблемы.

Ключевые слова*: измерение качества программ, продуктивность программирования, производительность программ, программистские решения, функциональное программирование*

ВВЕДЕНИЕ

Вопросы оценки влияния программируемых решений на продуктивность программирования и производительность программ в тематике исследований Лаборатории информационных систем ИСИ СО РАН традиционно рассматриваются в связи с разработкой методов обучения программированию, включая создание в последние годы парадигмальной методики анализа и сравнения языков программирования. При рассмотрении этого вопроса принято во внимание, что жаргон практического программирования использует понятие «язык программирования» как «компилятор входного языка типовой системы программирования, функционирующей на базе определённой конфигурации оборудования». При непосредственных измерениях производительности программ вместо оценки программируемых решений измеряется производительность комплекса из машины, компилятора и программы. Такие измерения слабо отражают вклад принятых программистом решений в производительность полученной программы, что при обучении программированию препятствует мотивации улучшать работающие решения, а также затрудняет сравнение с подобными решениями в других программах.

В данной статье рассмотрены результаты довольно известного эксперимента по сравнению производительности языков Lisp, C/C++ и Java [1], проходившего на базе бенчмарка «Какой язык быстрее всех?» [2] и заслужившего внимание программистов-практиков и студентов[1]. Кроме того, для демонстрационных экспериментов были привлечены механизмы платформы JDoodle [3], нацеленной на поддержку обучения и самообучения практическому программированию с помощью доступных онлайн компиляторов для 76 языков программирования. Часть наблюдений в этом направлении фактически происходит с учётом опыта оценки решений учебных и олимпиадных задач по программированию. В статье также рассмотрена возможность применения функциональных моделнй в качестве метрической шкалы, позволяющей отделять особенности аппаратуры, используемых

[1] Некоторые студенты включаются в соревнование по созданию более быстрых решений эталонных задач.

языков и систем программирования от характеристик программ и запрограммированных решений.

Бум языкотворчества в области проблемно-ориентированных языков программирования (ЯП), знаменующий переход практики программирования от накопления опыта на уровне эффективных библиотечных модулей к накоплению удобных подъязыков, показал важность измерения различий в определениях новых ЯП, улучшаемых версий программ и выборе инструментов для практических работ. В разных источниках упоминаются от 20-ти до 70-ти парадигм программирования, список которых видоизменяется и расширяется в зависимости от актуальности тех или иных затруднительных проблем программирования, а также моды на особенности популярных ИТ. Стремительно растущее число ЯП требует наличия методики, позволяющей оценивать пользу от их появления, обосновывать целесообразность трудозатрат на их изучение и улучшение практиками. Известно, что перенос программируемых решений на уровень аппаратуры повышает производительность приложений примерно в 150 раз, а смена компилятора может повысить её в 50 раз и более. Остаётся не вполне ясным, точнее, трудно измеримым, влияние собственно программируемых решений на производительность программ.

Изложение начинается с анализа результатов игры «Какой язык быстрее всех?» [2] на примере языков C++, Go, Clisp, Java, Haskell, Pascal, Python и сравнения Lisp, C/C++ и Java. Далее описан многолетний опыт оценки учебных и олимпиадных работ по программированию, проявивший ряд не вполне очевидных аспектов проблемы. Затем приведены результаты небольшого компьютерного эксперимента по измерению производительности этих ЯП и языка Closure на платформе JDoodle на примерах программ решения задачи вычисления чисел Фибоначчи, опубликованных на сайте «Энциклопедия языков программирования» [4].

1. «ГОРАЗДО ЛЕГЧЕ ЧТО-ТО ИЗМЕРИТЬ, ЧЕМ ПОНЯТЬ, ЧТО ИМЕННО ВЫ ИЗМЕРЯЕТЕ»[2]

Организаторы эксперимента «Какой язык быстрее всех?» [2] сразу предупреждают: «Будем реалистами: большинство людей, как правило, ничуть не озабочено производительностью программ»[3]. Сложилось не вполне благоприятное положение дел, при котором заказчики улучшения ПО понимают работу по усовершенствованию интерфейсов, согласны оценивать её продуктивность, в то время как польза работ по оптимизации процессов обработки данных не столь очевидна.

Тем не менее, проект развивался с 2002 по 2021 годы, и в отчёте о результатах его работы представлены данные об испытаниях заметного числа языков (C++, C, Rust, Fortran, Julia, C#, Ada, Chapel, Haskell, Go, Pascal, F#, Ocaml, Java, Swift, Java, Script, Lisp, Dart, Racket, PHP, Erlang, Lua, Python, Smalltalk, Ruby, Perl) и сформулированы выводы о присущих им парадигмах программирования. Измеряли скорость, память и объём программ. Чтобы узнать, какой ЯП (точнее, его компилятор) самый быстрый, организаторы эксперимента проводили измерения в стиле спортивного чемпионата, использовав эталонные задачи с заранее определёнными методами решения. Идея состоит в том, чтобы собрать коллекцию эталонных задач и написать программы их решения с помощью одних и тех же методов на разных языках. Обнаружение сходства с эталонами означает существование неявной общей модели, позволяющей анализировать программы на соответствие ожиданиям пользователя, т. е. на решение проблемы применимости (usability) [5].

Большинство компиляторов выполняет оптимизирующие преобразования программ, сохраняющие их функциональную эквивалентность. В зависимости от состава таких преобразований, полученные в результате компиляции варианты кода программы могут различаться по производительности. Организаторы эксперимента столкнулись с рядом проблем, решить которые удалось лишь частично.

[2]Изречение приписывается Уильяму Джону Салливану (род. 1976) – исполнительному директору Фонда свободного программного обеспечения и активному участнику проекта GNU [14].
[3] «It's important to be realistic: most people don't care about program performance most of the time» [2].

Первая проблема: нет критериев отбора эталонных задач. Поэтому проект «The Computer Language Benchmarks Game» [2] – это не более чем хорошая опорная точка для экспериментальной подготовки и дальнейшей проверки подходов к выбору эталонных задач, решения которых могут стать ориентиром при выборе или оценке компиляторов и создании метрической системы для измерения производительности программ.

Вторая проблема: как сравнивать по скорости слишком разные ЯП, реализованные на разных компьютерах? Пока требования к решению эталонных задач ограничивали лишь метод на уровне функциональной эквивалентности, а не средства или конструкции на уровне семантической эквивалентности.

Третья проблема: установление эквивалентности программ. Функционально эквивалентные программы могут не обладать семантической эквивалентностью, а потому их производительности могут резко различаться.

Четвёртая проблема: не все важные особенности программ вообще допускают непосредственное измерение, такие как эргономичность, надёжность, безопасность. Трудно измерить различия в подходах к дисциплине памяти, параллельному программированию, регулярным выражениям, арифметике с произвольной точностью, технике реализации и кодирования структур данных, хотя они являются неотъемлемой частью практического программирования.

Учитывая этот большой комплекс проблем, организаторы эксперимента выбрали нечто среднее между хаосом и жёсткостью, чтобы можно было независимо создавать программы на разных ЯП, а не просто механически конвертировать с одного ЯП на другой. Они отдавали себе отчёт в том, что пока нет научных оснований для такого сравнения, и не выясняли, насколько измерение производительности программ зависит от производительности компиляторов и парадигм программирования, поддержанных разными ЯП.

Организаторы эксперимента рассудили, что лучший выбор эталонов для измерения производительности – это реальные приложения[4]. Поэтому в список эта-

[4] Представленные в отчёте программы задачи (n-body, spectral-norm, mandelbrot, pidigits, fasta, k-nucleotide, reverse-complement, binary-trees, simple) не являются «реальными приложениями».

лонных задач не были включены ядра, представляющие собой небольшие ключевые части реальных приложений, игрушечные программы из начальных заданий по программированию и синтетические тесты, представляющие собой небольшие «поддельные» программы. Действительно, дистанция от учебных и игрушечных программ до реальных приложений велика, но она может быть постепенно систематически преодолена, а изучение массового опыта оценки учебных и синтетических программ способно дать научные основания методике измерений производительности программ. Образовательная система должна выполнять функцию оценки уровня знаний и умений обучающихся, и это является достаточной причиной для заинтересованности в создании методики оценки результатов выполнения учебных заданий по программированию.

Таким образом, для эксперимента по измерению скорости языков была создана небольшая коллекция задач, к решению которых были предъявлены чёткие требования по методу их решения и объёму обрабатываемых данных, гарантирующие функциональную эквивалентность программ решения одной и той же задачи [2]. Для большинства задач было выполнено несколько вариантов решений, показавших различную производительность.

Организаторы эксперимента в своём отчёте признали, что за почти 20 лет проекта у них не было времени проверять исходный код созданных программ-решений, чтобы убедиться, что разные реализации программы сопоставимы, что это действительно одна и та же программа, написанная на разных ЯП. Напротив, сравнение программ друг с другом происходило в таком духе, будто разные ЯП были созданы для одной и той же цели, что отнюдь не так.

Организаторы сравнения ЯП также признали, что им пока не удалось научиться определять относительную производительность ЯП, был лишь накоплен опыт, позволяющий понять нерешённые пока проблемы и получить простое представление об общей производительности каждого ЯП (точнее, его компилятора) [2, 6].

Опубликованные результаты показывают, что разброс скоростей редко превосходит 1 к 10. Python в этом «чемпионате» часто оказывается на последних местах. Это подтверждает исходное предположение о слабом интересе практиков к производительности программ и даже систем программирования. Практикам

важнее продуктивность программирования, что подтверждается и ходом истории технологии программирования. Не исключено, что Python был бы чемпионом в соревновании на скорость получения работоспособной программы, технологичность процесса разработки программ, продуктивность программирования, – что говорит о том, что производительность компиляторов нужно понимать шире, чем качество кода.

Таблица 1. Результаты измерения скорости программ на конкретных ЯП для разных задач

Задача	ЯП в порядке показанных результатов	Пропорция и диапазон
Fannkuch-redux	**C++** | **Go** | **Clisp** | **Haskell** | **Java** | **Pascal** | **Python** 03.26 | 8.31 | 9.38 | 10.33 | 10.48 | 10.56 | 5 min	1/3 3.26 – 10.56 **/5 min Python**
n-body	**C++** | **Pascal** | **Go** | **Haskell** | **Java** | **Clisp** | **Python** 2.17 | 6.28 | 6.37 | 6.43 | 6.77 | 7.70 | 9.00	1:4 2.17 – 9.00
spectral-norm	**C++** | **Go** | **Clisp** | **Pascal** | **Haskell** | **Java** | **Python** 0.72 | 1.43 | 1.44 | 1.44 | 1.47 | 1.55 | 112.97	1/2 0.72 –1.55 **/112.97 Python**
mandelbrot	**C++** | **Haskell** | **Go** | **Pascal** | **Java** | **Clisp** | **Python** 03.26 | 8.31 | 9.38 | 10.33 | 10.48 | 10.56 | 5 min	1/5 0.84 – 4.17 / **177.35 Python**
pidigits	**C++** | **Pascal** | **Java** | **Go** | **Python** | **Haskell** | **Clisp** 0.59 | 0.73 | 0.79 | 0.86 | 1.16 | 1.44 | 3.17	1/6 0.59 – 3.17
fasta	**C++** | **Haskell** | **Java** | **Go** | **Clisp** | **Pascal** | **Python** 0.77 | 0.86 | 1.20 | 1.27 | 4.38 | 5.58 | 36.90	1/6 0.77 – 5.58 **/ 36.90 Python**

Задача	ЯП в порядке показанных результатов						Пропорция и диапазон
k-nucleotide	**C++**	**Java**	**Go**	**Haskell**	**Clisp**	**Python**	1/8 1.96 – 15.12 **/46.31 Python**
	1.96	4.83	7.46	11.69	15.12	46.31	
reverse-complement	**C++**	**Go**	**Java**	**Haskell**	**Pascal**	**Clisp**	**Python**	1/10 0.64 – 6.63
	0.64	1.34	1.57	3.16	3.69	5.69	6.63	
binary-trees	**C++**	**Pascal**	**Java**	**Haskell**	**Clisp**	**Go**	**Python**	1/13 0.94 – 12.48 **/44.70 Python**
	0.94	2.04	2.51	4.42	5.44	12.48	44.70	

В этом плане показательны результаты другого проекта, посвященного сравнению языков Lisp и Java, выполненного четырнадцатью программистами-добровольцами, написавшими 16 программ (12 на Common Lisp и 4 на Scheme)[5], производительность которых сравнили с C/C++ и Java с точностью до минимизации рисков при выполнении проектов. Результаты эксперимента показали, что Clisp или Scheme уступают по скорости работы C/C++. При сравнении производительности Clisp с C/C++ и Java было констатировано, что Clisp обладает более быстрым циклом отладки программ, чем C/C++ или Java, что означает более высокую продуктивность программирования [1].

Экспериментаторы отметили, что программисты с меньшим опытом программирования на Clisp или Scheme нередко программируют быстрее и лучше, чем более опытные программисты на C/C++ или Java, и пришли к выводу, что программирование на языке Lisp имеет тенденцию повышать квалификацию программистов, делать их хорошими программистами на любом ЯП. Не исключено, что язык Lisp даёт программистам больше времени на продумывание и улучшение программ.

[5] В источнике нет сведений о характере задач.

Традиционный вопрос, почему столь замечательный язык как Lisp не обладает адекватной популярностью, пока не получил простого ответа. Одна из гипотез: во всем виноват миф о том, что Lisp слишком большой и медленный[6], а также другие предрассудки, исторически сложившиеся в пионерскую эпоху программирования. В противовес этому мифу авторы сравнения языков Lisp и Java приводят результаты своего эксперимента, отмечая характеристики продуктивности программирования и квалификации программистов.

В целом результаты вышеописанных экспериментов показывают, что проблема измерения производительности программ далека от конструктивного решения. Организаторы этих экспериментов приняли ряд важных решений и констатировали ряд непреодолённых трудностей.

1. Они признали, что нужны эталонные задачи, но не дали пояснений относительно эталонности в условиях непрерывного развития ЯП.
2. Они не дали рекомендаций по сопоставлению реальных приложений с предложенными в качестве эталонных, довольно искусственных, задач.
3. Осталась невыясненной взаимосвязь между продуктивностью программирования и достижением производительности программ. В этом отношении системы обучения программированию и проведения конкурсов по программированию обычно имеют правдоподобные оценки трудоёмкости выполнения учебных и олимпиадных задач.
4. Ещё одна сторона оценки продуктивности и производительности связана с их зависимостью от уровня квалификации, способностей, образования и опыта программистов. Обычно любая экономика приходит к пониманию оптимального соотношения цена/качество, допускающего вычисления оптимума методами типа мини-макс. Если качество – это производительность программ, то цена – это продуктивность разработки или улучшения программ. Поэтому измерение производительности имеет смысл при умении измерять и/или прогнозировать продуктивность.

Можно обратить внимание также и на то, что прогресс технологий программирования обусловлен ростом технологичности, приоритетами которой являются

[6] Параллельно с этим существует также мнение, что Lisp легкий и элегантный.

повышение продуктивности программирования и расширение сферы применения программ, что, в свою очередь, чревато потерями в производительности программ. Нередко происходит и снижение требований к квалификации программистов [7, 8].

2. «ДАЙТЕ МНЕ ТОЧКУ ОПОРЫ…»[7]

Эти известные слова Архимеда в практическом программировании понимают как «Дайте мне доступ к компилятору с самой маленькой работающей программой, остальное я пойму сам в непосредственном эксперименте». Работающий компилятор для практика является единственно достоверным документом о реализованном языке программирования. Документация – это не более чем справочный материал, нуждающийся в проверке на соответствие тому, что фактически поддерживает компилятор.

Именно такое понимание процесса изучения ЯП лежит в основе платформы JDoodle [3], содержащей онлайн-компиляторы, калькулятор и встроенный редактор, позволяющие осваивать программирование восходящим методом снизу вверх на базе любых из 76-ти ЯП и двух баз данных. Платформа JDoodle не гарантирует точности или надёжности предложенных материалов, что не мешает использовать её в экспериментах и учебном процессе. Остаются за бортом вопросы решения новых или слабо исследованных задач, характерных для современного программирования в условиях совершенствования аппаратуры и расширения круга пользователей, не обязанных обладать познаниями в сфере теоретического и практического программирования.

[7] Изречение «Дайте мне точку опоры – и я переверну Землю!» приписывается древнегреческому ученому и инженеру Архимеду Сиракузскому (287 до н.э. – 212 до н.э.) [15].

3. «А У НАС В КВАРТИРЕ ГАЗ!»[8]

Автоматизированное оценивание эффективности программ во многом напоминает автоматическое тестирование олимпиадных задач по программированию. Но если первое ещё находится на этапе становления, то второе уже прошло большой путь развития, и его богатый накопленный опыт может быть использован при разработке теоретических основ измерения вкладов различных аспектов (человеческих, программных и аппаратных) в общую производительность программ.

Олимпиады по программированию появились практически одновременно с введением предмета «Основы информатики и вычислительной техники» в программу среднего и среднего специального образования в 80-х годах прошлого столетия.

На начальном этапе своего становления школьные олимпиады всех уровней проходили с обязательным включением в них теоретического тура, причем по причине повсеместного дефицита вычислительной техники на олимпиадах нижних уровней – школьном и даже районном – практического тура могло вообще не быть. Безмашинные теоретические туры проходили следующим образом: участники олимпиады писали тексты программ или алгоритмы решения на бумаге, затем члены жюри читали эти тексты и, выступая в роли синтаксических анализаторов и виртуальных машин, имитировали выполнение программ у себя в голове. Решения задач практических туров уже тогда проверялись по принципу «чёрного ящика»[9]. Члены жюри вручную запускали каждую программу с заранее подготовленными входными данными и фиксировали результаты выполнения.

По мере развития вычислительной техники, компьютерных сетей, операционных систем и роста оснащённости персональными компьютерами школ и ссузов проверка заданий на олимпиадах по программированию школьного уровня постепенно превратилась в тестирование: сбор решений, их компиляция

[8] Строка «А у нас в квартире Газ! А у вас?» из известного стихотворения Сергея Михалкова (1913–2009) [16] «Дело было вечером» часто используется для описания противопоставлений и сравнений.

[9] Способ тестирования, когда заключение о внутренних свойствах исследуемого объекта делается лишь на основании его откликов на различные раздражители.

и выполнение, обработка ошибок, подсчёт затраченного времени, а также первичный анализ полученных результатов были автоматизированы.

Автоматизация процессов проверки наложила на программную реализацию решений ряд ограничений, обусловленных особенностями тестирующих систем: допустимы только файловые ввод и вывод данных, разрешено использование только «одобренных» компиляторов и т. п. Очень важной стала эффективность выбранного алгоритма решения, поскольку работа любой программы теперь обязана укладываться в заданный временной отрезок.

На всех этапах развития олимпиадного движения особое внимание уделялось правилам вынесения итогового вердикта. В соответствии с принципами «чёрного ящика», решение о правильности или неправильности предоставленного на проверку решения принималось на основании результатов выполнения тестов («ручного» или автоматического). При этом проверка полноты и корректности тестовых наборов до сих пор остаётся на совести разработчиков заданий.

В зависимости от результатов прохождения всех тестов в тестовом наборе, возможны два варианта вынесения итогового вердикта: «*Решение верно обработало все тесты – участник получил оговорённое количество баллов; иначе получил 0 баллов»* либо «*Решение верно обработало только некоторые тесты – участник получил не максимальное, но и не нулевое количество баллов»*. Во втором случае каждый тест имеет индивидуальную «стоимость». Объемные тесты или тесты, корректная обработка которых возможна только при помощи алгоритмов повышенной эффективности, получают более высокую «стоимость».

Примером задачи, у которой существуют два функционально, но не семантически эквивалентных решения, является известная задача поиска кратчайшего пути во взвешенном графе. Время работы рекурсивного алгоритма полного перебора всех путей пропорционально 2^N, где N – количество вершин в графе, а для алгоритма Дейкстры этот показатель пропорционален N^2. При не очень больших N эффективность этих двух алгоритмов различается не слишком сильно, однако по мере увеличения N разница становится все более заметной. Чтобы отсеять решения, использующие неэффективный алгоритм, составители олимпиадных зада-

ний обычно вводят в тестовый набор объемные тесты, время выполнения которых заведомо превысит допустимый максимум в случае полного перебора всех путей.

Вообще важность полноты и корректности тестовых наборов, предназначенных для автоматической оценки правильности или эффективности каких-либо программ, невозможно переоценить, поскольку вердикт о правильности или неправильности выбора конкретного алгоритма, эффективности или неэффективности его реализации принимается на основании результатов выполнения отдельных тестов, каждый из которых должен относиться к какому-либо частному случаю, включая особые точки и границы области всех допустимых значений. Если полнота тестового набора не доказана, вообще говоря, нельзя утверждать, что проверенное решение будет правильно работать на всех возможных входных данных. Его корректность была проверена только для некоторого подмножества области допустимых значений.

4. «… А У ВАС?»[10]

Если сравнивать оценку олимпиадных решений и эксперименты, рассмотренные в начале статьи, то можно увидеть и сходства, и различия.

К основным сходствам можно отнести следующие моменты.

Это, во-первых, многоязычие. И на олимпиадах по программированию, и в описанных экспериментах участники не ограничены одним или двумя ЯП, а организаторы должны выработать некие критерии, согласно которым можно оценивать и сравнивать решения, написанные на различных ЯП.

Во-вторых, отсутствие семантического анализа текста решений. Даже на ранних этапах олимпиадного движения, когда члены жюри принимали от участников исходные тексты программ, а компиляция и выполнение тестов производились вручную, собственно тексты программ рассматривались только в исключительных – спорных или непонятных – случаях. А после введения автоматического тестирования решений подобная практика практически исчезла: вердикт о

[10] Окончание строки «А у нас в квартире Газ! А у вас?» из известного стихотворения Сергея Михалкова (1913–2009) [16] «Дело было вечером».

правильности решения принимается только на основании результатов выполнения тестов.

В-третьих, единый для всех участников набор тестов, служащий основным или вовсе единственным критерием для выставления итоговых баллов.

И, в-четвёртых, автоматизированная фиксация времени, затраченного на работу программы.

Различия же обнаруживаются в следующих моментах.

Во-первых, в учёте или игнорировании искажений, вызванных автоматической оптимизацией программ на этапах компиляции. Поскольку результат оптимизации довольно сильно зависит от конкретной реализации алгоритма, выбранного участником, и может исказить результаты измерений (как в ту, так и в другую сторону), то на многих олимпиадах жёстко оговаривается не только набор разрешённых к использованию компиляторов, но и набор допустимых опций компилирования. Опция оптимизации из этого набора исключается по очевидным причинам. В то же время, в описанных экспериментах по измерению не удалось обнаружить признаков подобного подхода. Это приводит к тому, что измеряются не исходные «чистые» программистские решения, а лишь некоторая их версия, «загрязненная» скрытой автоматической оптимизацией.

Во-вторых, отсутствие систематического научного подхода к анализу тестового набора на предмет его полноты и корректности. Чтобы минимизировать вероятность возможных ошибок в тестах, на многих олимпиадах в конце соревнования набор проверочных тестов становится доступным всем участникам, и на основании анализа этого тестового набора даже принимаются апелляции. А в случае обнаружения серьёзной ошибки все решения перепроверяются на исправленном тестовом наборе. Но в системе бенчмарка «Какой язык быстрее всех?» [2] набор тестов, на основании которых делается вывод о эффективности или неэффективности предложенной программы, скрыт, что делает результаты проверки не очень достоверными. Чаще всего «камнем преткновения» в вопросе эффективности выбранного языка программирования, алгоритма и его реализации являются именно редкие частные или граничные случаи. Таким образом, невозможно выстроить достоверную систему сравнения эффективности программ без всестороннего анализа области допустимых входных данных.

Как видим, автоматическое тестирование решений задач по программированию и измерение эффективности реализаций различных алгоритмов на различных языках программирования действительно имеют много общего. Хочется, однако, чтобы измерение эффективности в своём дальнейшем развитии учло и «обезвредило» те особенности автоматического тестирования, которые способны исказить результаты измерений. Важно также учесть, что в процессе олимпиадной деятельности организаторы конкурсов и преподаватели научились интуитивной оценке сложности и качества программирования не только типовых, но и новых задач, ранее неизвестных участникам. Отражает ли интегральная оценка продуктивность работы участников и производительность сделанных ими программ? Что из опыта оценки олимпиадных работ может получить развитие как методика оценки учебного и производственного программирования или что обладает сходством, а что резко отличается?

Подготовка и проведение лабораторных работ и олимпиад по программированию включают в себя ряд уже зарекомендовавших себя средств и методов комплектации и проверки заданий и отдельно оценивания и тестирования программ решения задач [9–12]. Уже достигнуто понимание ряда особенностей выбора и требования к постановкам задач, известны типичные категории задач, упорядоченные по сложности.

Первые эксперименты можно продолжить на числовых функциях, легко масштабируемых одним параметром: факториал, числа Фибоначчи, простые числа, степенные ряды и т. п. Можно задействовать вещественные и мнимые числа в форме решения квадратных уравнений, извлечения корней, дробных формул с риском деления на ноль. Можно использовать символьную обработку, перестановки или разрастающиеся цепочки.

Шкала сложности программируемых решений может различать

- на первом уровне: элементарные операции, простые переменные, ветвления, доступ к соседнему элементу последовательности, строки, векторы, списки и стеки;
- на втором уровне: функции и процедуры, иерархию областей действия, разные виды ветвлений, структуры данных;

- на третьем уровне: ограничения, диагностики, ловушки, контроль типов данных, рекурсию и циклы;
- на четвёртом уровне: отображения, преобразования, фильтры, свёртки, генераторы и конструкторы, макросы;
- на пятом уровне: многопоточность и имитацию обменов данными между потоками без сложных взаимодействий.

Важно, что связывание производительности программ с программируемыми решениями при таком подходе допускает сопоставление со спектром используемых средств языка или эталонными задачами. Разброс производительности программ, кроме того, обусловлен продуктивностью программирования, т. е. квалификацией программистов, владеющих не только методами решения таких задач, но и определёнными способностями, средствами, техникой программирования. Всё это дополняется умением самостоятельно решать проблемы в пространстве, выходящем за пределы языка и системы программирования, что более подробно описано в работах [6, 13], представляющих парадигмальную методику анализа и сравнения языков программирования, а также неформализуемые особенности процесса разработки программ.

5. «МОЖЕТЕ ИЗМЕРИТЬ – ЗНАЧИТ, ЗНАЕТЕ»[11]

Результаты одного эксперимента по измерению производительности программ на базе платформы JDoodle [3], выполненного на примере вычисления чисел Фибоначчи, показывают возможность измерения вклада программируемых решений. Этот алгоритм интересен тем, что даёт быстрое снижение производительности. Целью эксперимента является «зондирование почвы» — показ возможности создать методику выделения вклада программируемых решений из непосредственно измеримых характеристик комплекса, включающего в себя программу, компилятор и аппаратуру. Полностью и с подробным анализом всех полученных результатов этот эксперимент описан в работе [6], доступной в интернет. Здесь мы приведем только основные выводы.

[11] Изречение «Если вы можете измерить то, о чём говорите, и выразить это в цифрах, – значит, вы что-то об этом предмете знаете» приписывается британскому ученому Уильяму Томсону, лорду Кельвину (1824–1907) [17].

Семантическая эквивалентность измеренных программ позволяет относить разницу в производительности на счёт разницы в компиляторах и аппаратуре, когда запрограммированные решения одинаковы. Варианты программируемых решений одного и того же метода могут использовать разные средства языков программирования и потому быть семантически не эквивалентными.

Разница в показателях Java и Clojure, вероятно, объясняется различием в границах счёта. Это и не удивительно, ведь ЯП создаются для разных целей:

- Программа на языке Java допускает наибольший диапазон данных.
- Язык C++ работает настолько быстро, что обработку малых объёмов трудно измерить.
- Язык Clisp поддерживает заботу о малых заданиях, даже давая им приоритет.
- Язык Clojure – лидер на больших объёмах и активный «пожиратель памяти».
- Компилятор языка Haskell во многом компенсирует недостатки языка.
- Go и Java близки по ряду показателей.
- И конечно, Pascal показывает, что для успешного решения учебных задач многого не надо.

Разница в производительности таких вариантов 1 к 1000 уже объясняется программируемыми решениями. Разброс показателей намного превышает разницу между прогонами на разных компиляторах.

Подобную разницу можно видеть и в результатах «игры» в самый быстрый компилятор, но, как правило, с заметно меньшим разбросом. Беглый взгляд на самую быструю и самую медленную программы, написанные на языке Clisp одним и тем же программистом, показывает, что заметная разница в скорости, скорее всего, объясняется использованием в быстрой программе макросов и многопоточности. Разница между вариантами функционально эквивалентных программ, не обладающих семантической эквивалентностью, на одном и том же ЯП для приведённых примеров существенно выше, чем разница между семантически эквивалентными программами на разных ЯП. Впрочем, пока рассмотрено слишком мало примеров и ЯП для уверенных выводов.

Первые кандидаты в эталонные языки – это языки функционального программирования, обладающие высокой моделирующей силой. На этапе экспериментов достаточно выделять из программ ключевые фрагменты, представляющие программируемые решения, приводить их к нормализованной форме на эталонном языке для измерения.

К проведению эксперимента были привлечены слушатели спецкурса «Парадигмы программирования», ежегодно читаемого в НГУ на факультетах ММФ и ФИТ.

ЗАКЛЮЧЕНИЕ

В статье описаны результаты двух подходов к оценке производительности программ и удобный бенчмарк для постановки обучения программированию, нацеленного на формирование профессионального умения оценивать и измерять вклад программируемых решений в производительность программ. Изложение опыта оценивания решений олимпиадных задач по программированию показывает перспективу формирования научно обоснованной методики измерения производительности программ. Приведены результаты небольшого эксперимента, дающего основания для функционального подхода к разработке методики измерения качества программ.

Основные выводы и гипотезы:

1. Исследование и разработку метрик производительности программ необходимо проводить одновременно с созданием методик прогнозирования продуктивности разработки программ.
2. Задачи можно измерять в терминах задач, а программы – в терминах программ, используя подобие.
3. Первые эксперименты можно выполнить на материале учебных и олимпиадных задач, характеристики которых хорошо известны.

Усложняющим является требование понятности метрики как для программистов, так и для пользователей и менеджеров, но без этого метрика не может получить признание. В дальнейшем предстоит экспериментальное исследование предложенной методики на более широком наборе задач, языков программирования и особенностей их разработки и применения [13].

СПИСОК ЛИТЕРАТУРЫ

1. *Patil S.* Lisp as an Alternative to Java.
URL: https://psachin.gitlab.io/lisp-java-notes.html

2. *Gouy I.* The Computer Language Benchmarks Game “Which programming language is fastest?”
URL: https://benchmarksgame-team.pages.debian.net/benchmarksgame/index.html

3. Платформа JDoodle. URL: https://jdoodle.com

4. Энциклопедия языков программирования. URL: http://progopedia.ru

5. *Купер А.* Психбольница в руках пациентов. Алан Купер об интерфейсах. СПб.: Питер, 2018. 384 с.

6. *Андреева Т.А., Городняя Л.В.* Функциональный подход к измерению вклада программируемых решений в производительность программ: препринт. Новосибирск: ИСИ СО РАН, 2022. 62 с.
URL: https://www.iis.nsk.su/files/preprints/preprint_187.pdf

7. *Weinberg G.M.* The Psychology of Computer Programming. Silver Anniversary Edition, 2011. 288 p.

8. *Липаев В.В.* Человеческие факторы в программной инженерии. Рекомендации и требования к профессиональной квалификации специалистов. М.: СИНТЕГ, 2009. 348 с.

9. *Андреева Т.А.* Возможность автоматизации процесса генерирования тестовых наборов // Universum: технические науки. №8(41). М., Изд. «МЦНО», 2017. С. 5–7.

10. *Andreyeva T.A.* Automation of correctness checking in education // A.P. Ershov Informatics Conference / Educational Informatics Workshop proceedings. July 2–3, 2019. Novosibirsk, 2019. P. 6–15.

11. *Андреева Т.А.* Сборник задач для предолимпиадной подготовки по программированию. Новосибирск: Изд-во НГУ, 2009. 226 с.

12. *Андреева Т.А.* Программирование на языке Pascal. М.: ИНТУИТ, 2016. 277 с.

13. *Городняя Л.В.* Функциональное программирование. Парадигма, модели, методы. Новосибирск: Изд-во СО РАН, 2022. 482 с.

14. *Салливан Уильям Джон*: персональная страница в Википедии. URL: https://ru.wikipedia.org/wiki/Салливан_Уильям_Джон

15. *Архимед Сиракузский:* персональная страница в Википедии. URL: https://ru.wikipedia.org/wiki/Архимед

16. *Михалков Сергей Владимирович*: персональная страница в Википедии. URL: https://ru.wikipedia.org/wiki/Михалков_Сергей_Владимирович

17. *Томсон Уильям* (лорд Кельвин): персональная страница в Википедии. URL: https://ru.wikipedia.org/wiki/Томсон_Уильям_(лорд_Кельвин)

ABOUT MEASURING OF THE CONTRIBUTION OF SOFTWARE DECISIONS TO PROGRAM PERFORMANCE

L. V. Gorodnyaya[1, 2][0000-0002-4639-9032], T. A. Andreyeva[2] [0000-0002-1124-9499]

[1, 2] *A.P. Ershov Institute of Informatics Systems SB RAS, 6, Acad. Lavrentjev pr., Novosibirsk 630090, Russia*

[1, 2]*Novosibirsk State University, 1, st. Pirogova, Novosibirsk, 630090, Russia*

[1]gorod@iis.nsk.su, [2]ata@iis.nsk.su

Abstract

The article draws attention to the problem of measuring the effect that programming solutions have on the programming productivity and performance, in educational programming and the correctness-saving program improvements. The results of some experiments concerning these questions are discussed. The hypothesis that functional models can provide a metric scale capable of separating features of programming languages and systems from features of programs and programming solutions is proposed. The results of a preliminary demonstrative experiment in studying the dependence of the program productivity upon the opted compiler and, on the other hand, upon the representation of the programming solution in the opted programming languages are described. Analysis of these results leads to a method that can reveal such dependencies. The long experience in sifting educational and contest programs revealed some unnoticed aspects of this problem.

Keywords: *program quality measurement, programming productivity, program performance, programming decisions, functional programming*

REFERENCES

1. *Patil S.* Lisp as an Alternative to Java.
URL: https://psachin.gitlab.io/lisp-java-notes.html

2. *Gouy I.* The Computer Language Benchmarks Game “Which programming language is fastest?”
URL: https://benchmarksgame- team.pages.debian.net/benchmarksgame/index.html

3. JDoodle. URL: https://jdoodle.com (access for registered users only)

4. Entsiklopedia jazykov programmirovaniya. URL: http://progopedia.ru

5. *Cooper A*. The Inmates are Running the Asylum, 2004. 288 p.

6. *Andreyeva T.A., Gorodnyaya L.V.* Funktsionalnyj podhod k izmereniju vklada programmiruemyh reshenij v proizvoditelnost programm : preprint. Novosibirsk, 2022. 62 p. URL: https://www.iis.nsk.su/files/preprints/preprint_187.pdf

7. *Weinberg G.M.* The Psychology of Computer Programming. Silver Anniversary Edition, 2011. 288 p.

8. *Lipaev V.V.* Chelovecheskije faktory v programmnoj inzhenerii. Rekomendatsii i trebovanija k professionalnoj kvalifikatsii spetsialistov. M.: SINTEG, 2009. 348 p.

9. *Andreyeva T.A.* Vozmozhnost avtomatizatsii processa generirovanija testovyh naborov // Universum: tehnicheskie nauki. №8(41). M., 2017. P. 5–7.

10. *Andreyeva T.A.* Automation of correctness checking in education // A.P. Ershov Informatics Conference / Educational Informatics Workshop proceedings. July 2–3, 2019. Novosibirsk, 2019. P. 6–15.

11. *Andreyeva T.A.* Sbornik zadach dlia predolimpiadnoj podgotovki po programmirovaniju. Novosibirsk: Izd-vo NGU, 2009. 226 p.

12. *Andreyeva T.A.* Programmirovanije na jazyke Pascal. M.: INTUIT,2016. 277 p.

13. *Gorodnyaya L.V.* Funktsionalnoe programmirovanije. Paradigma, modeli, metody. Novosibirsk: Izd-vo SO RAN, 2022. 482 p.

14. *William John Sullivan*: personal page on Wikipedia.
URL: https://en.wikipedia.org/wiki/William_John_Sullivan

15. *Archimedes*: personal page on Wikipedia.

URL: https://en.wikipedia.org/wiki/Archimedes

16. *Sergey_Mikhalkov*: personal page on Wikipedia.
URL: https://en.wikipedia.org/wiki/Sergey_Mikhalkov

17. *Lord Kelvin*: personal page on Wikipedia.
URL: https://en.wikipedia.org/wiki/Lord_Kelvin

СВЕДЕНИЯ ОБ АВТОРАХ

ГОРОДНЯЯ Лидия Васильевна *– кандидат физико-математических наук, старший научный сотрудник Института систем информатики имени акад. Андрея Петровича Ершова СО РАН, доцент Новосибирского государственного университета, специалист в области системного программирования и образовательной информатики.*

Lidia Vasiljevna GORODNYAYA – Ph.D., Senior Researcher at the A.P. Ershov Institute of Informatics Systems, Siberian Branch of the Russian Academy of Sciences, Associate Professor at the Novosibirsk State University, specialist in system programming and educational informatics.

email: gorod@iis.nsk.su

ORCID: 0000-0002-4639-9032

АНДРЕЕВА Татьяна Анатольевна *– младший научный сотрудник Института систем информатики имени акад. Андрея Петровича Ершова СО РАН, старший преподаватель Новосибирского государственного университета, специалист в области образовательной информатики.*

Tatiana Anatolevna ANDREYEVA – Junior Researcher at the A.P. Ershov Institute of Informatics Systems, Siberian Branch of the Russian Academy of Sciences, Senior Lecturer at the Novosibirsk State University, specialist in educational informatics.

email: ata@iis.nsk.su

ORCID: 0000-0002-1124-9499

Материал поступил в редакцию 17 декабря 2023 года