J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin // Microtome Publishing. 2015. Vol. 17. P. 1–7.

5. Ingersoll G. Introducing apache mahout. Scalable, commercial friendly machine learning for building intelligent applications. URL: http://www.ibm.com/developerworks/library/j-mahout/j-mahout-pdf.pdf (18.10.2016).

6. Mell P., Grance T. The NIST Definition of Cloud Computing. Recommendations of the National Institute of Standards and Technology / Computer Security Division Information Technology Laboratory National Institute of Standards and Technology. Gaithersburg, 2011.

7. Gorlatch S. Extracting and implementing list homomorphisms in parallel program development // Science of Computer Programming. 2015. Vol. 33, iss. 1. P. 1–27.

8. Fog computing and its role in the internet of things / F. Bonomi, R. Milito, J. Zhu, S. Addepalli // Proc. of MCC, Helsinki, Finland, 17 Aug. 2012. P. 13–16.

9. Hewitt C., Bishop P., Steiger R. A universal modular ACTOR formalism for artificial intelligence // 3rd Intern. joint conf. on Artificial intelligence, Stanford, California, 1973. P. 235–245.

10. Kholod I., Petuhov I., Kapustin N. Creation of data mining cloud service on the actor model // NEW2AN/ruSMART.LNCS. 2015. Vol. 9247. P. 585–598.

11. Kholod I., Petuhov I. Creation of Data Mining Algorithms as Functional Expression for Parallel and Distributed Execution. In: Malyshkin, V. (eds.) // Parallel Computing Technologies. LNCS. 2015. Vol. 9251. P. 62–68.

I. I. Holod, I. V. Petuhov
Saint Petersburg Electrotechnical University «LETI»

METHOD FOR ESTIMATION OF DATA ANALYSIS EFFICIENCY IN A DISTRIBUTED ENVIRONMENT

Describes the method of configuring a network of actors to perform data mining in a distributed environment. Source data for the method are the characteristics of the data set, the algorithm analysis and runtime. Results of applying the method are the number of clusters, networks of actors, the number of different types of actors, as well as the placement of actors is determined at runtime.

Data mining, distributed analysis, distributed systems, actors model

УДК 004.622

Е. К. Лепилкина, С. И. Якушкин
Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В. И. Ульянова (Ленина)

Создание инфраструктуры сбора данных о статических характеристиках программ для задачи подбора оптимизационных опций для компиляторов на базе LLVM

Рассматривается вопрос о способе сбора и хранения статических характеристик отдельных базовых блоков программ до и после выполнения различных оптимизаций для компиляторов, разработанных на основе системы LLVM. Также рассматривается возможность дальнейшего доступа к этим данным для анализа эффективности работы компиляторов.

Статические характеристики, компилятор, система LLVM, Elasticsearch, оптимизационные проходы компилятора

Современные компиляторы представляют собой достаточно сложные системы с множеством опций, позволяющих получать высокопроизводительный машинный код. Все существующие компиляторы предоставляют большие возможности по оптимизации программ, однако зачастую подобрать опции компиляции достаточно сложно, так как необходимы данные о компилируемой программе, причем после работы большинства проходов эти сведения могут сильно измениться

[1]. Для анализа работы компилятора и подбора эвристик, на основе которых выполняется оптимизация, необходимо иметь возможность получить эти сведения. При уже существующем большом наборе свойств по конфигурации компиляторов нужно понимать как их правильно использовать, так как до сих пор существуют области, где важны высокая производительность кода и его минимально возможный размер. Примером такой области может служить разработка программ для встраиваемых систем и цифровой обработки сигналов.

Существуют различные работы по автоматическому подбору опций компиляции [2], [3]. И естественно, задача выбора, сбора и хранения характеристик программ являлась первоочередной и достаточно важной, так как хорошо продуманная и реализованная инфраструктура может использоваться долгое время для различных задач анализа и повышения эффективности работы компиляторов и значительно упростить и ускорить решение данных задач.

На данный момент существует фреймворк Collective Knowledge, разработанный Г. Фурсиным и А. Лохмотовым [3]. Он предназначен для сбора и совместного доступа к различным данным, в том числе и характеристикам программ и платформам при компиляции различными средствами. Однако, так как Collective Knowledge является универсальным инструментом, он достаточно сложен в настройке под конкретную задачу, не имеет возможности получения данных на отдельных этапах компиляции, требует подробного описания компилируемой программы в специальном формате и в основном ориентирован на сбор данных о платформе. Но получить состояние характеристик при выполнении некоторого прохода компилятора с помощью данной системы, к сожалению, не удастся. Для этого нужны знание и доступ к внутреннему устройству компилятора, т. е. необходимо разработать систему, ориентированную на определенный класс компиляторов с единой архитектурой.

В основном все современные промышленные компиляторы разрабатываются на основе системы LLVM [4], реализующей виртуальную машину с RISC-подобными инструкциями. На основе LLVM реализуют свои проекты такие компании, как Google, Apple, Adobe, ARM и т. д. При этом на данный момент нет возможности получить характеристики программы стандартными средствами LLVM. Единая инфраструктура сбора данных, ориентированная именно на компиляторы на базе LLVM, позволит, во-первых, получить подробную информацию, которая будет полезна при анализе для настройки конфигурации компилятора и не только, во-вторых, инфраструктура будет применима для анализа и решения задач на большом количестве промышленных компиляторов, а не для некоего конкретного компилятора.

Задача сбора статических характеристик в LLVM. Программный код, предназначенный для компиляции, можно описать в виде множества базовых элементов IR-кода, являющихся промежуточным представлением в LLVM: модулей, функций, циклов и базовых блоков, т. е. в виде множества $\{M, F, L, B\}$, где M – множество модулей, входящих в программу; F – множество функций; L – множество циклов; B – множество базовых блоков.

Каждый из элементов, входящих в любое из вышеперечисленных множеств, можно характеризовать набором статических характеристик S_M, S_F, S_L, S_B соответственно. Каждый компилятор, разработанный на основе LLVM, имеет определенное множество оптимизирующих проходов $P = \{P_M, P_F, P_L, P_B\}$. Основная задача, рассмотренная в статье, заключается в сборе, хранении и соотнесении полученного при различных оптимизирующих опциях набора статических характеристик $\{S_M, S_F, S_L, S_B\}$ в любой момент времени компиляции – до или после выполнения любого из проходов $p \in P$.

Характеристики подбирались вручную посредством анализа кода основных оптимизационных проходов, чтобы понять, на основе каких параметров производится оптимизация. Примером статических характеристик для циклов могут служить: количество недублируемых инструкций, количество непрямых ветвей, количество выходов из цикла, количество базовых блоков в цикле и т. д. На самом деле, набор характеристик может быть уникальным для каждой конкретной задачи, поэтому была предложена базовая иерархия классов, добавленная в LLVM.

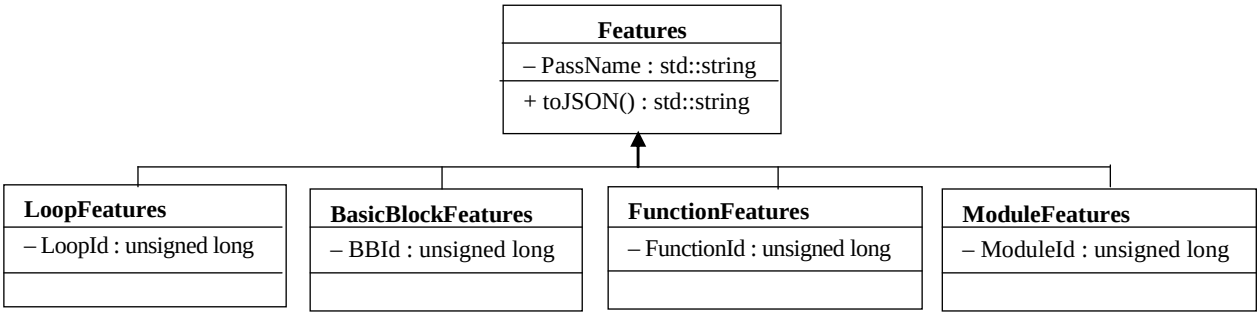


Рис. 1

Представленная на рис. 1 иерархия классов позволяет добавлять новые подвиды характеристик для каждого типа элементов промежуточного представления, разделяя их на логические блоки характеристик, имеющих нечто общее. Ниже в иерархии располагаются специализированные классы, описывающие отдельные характеристики.

Также, соответственно, для однозначного определения каждого отдельного элемента были добавлены идентификаторы в соответствующие классы элементов, представляющие собой результат хеш-функции от неизменяемых свойств данных элементов. Это было сделано из-за необходимости иметь возможность точно идентифицировать аналогичный базовый блок, цикл и т. д., несмотря на то, что при других опциях компиляции, а следовательно, при других оптимизационных проходах, будет получено другое промежуточное представление для данных базовых элементов и другой машинный код. Это является важным условием для возможности дальнейшей обработки и анализа получаемых данных.

По сути, описанные выше классы и их наследники являются лишь структурами для хранения и структурированного представления в json-формате. Для непосредственного сбора характеристик и их вывода была разработана дополнительная функциональность, построенная по аналогии с существующим функционалом вывода промежуточного IR-кода. Для этого были добавлены дополнительные опции в LLVM: -print-features-before, -print-features-after, -print-features-before-all, -print-features-after-all, которые позволяют, соответственно, собирать и выводить характеристики до/после определенного прохода или до/после всех проходов. Это обеспечивает гибкость по сбору характеристик для любой прикладной задачи по анализу данных. Для этого были реализованы специализированные классы проходов, которые частично сами, частично на основе других уже имеющихся анализирующих проходов собирают все необходимые характеристики нужного типа в зависимости от типа основного прохода. Данные проходы добавляются в стандартный менеджер проходов LLVM в случае передачи описанных ранее флагов. Проходы для сбора характеристик добавляются в менеджер специфичным способом, так как они не могут быть добавлены заранее, как все основные проходы, и при этом они могут зависеть от анализирующих проходов, что отличает их от простых проходов для вывода IR-кода. Также был добавлен флаг для вывода данной информации в файл с целью последующего сохранения.

Система для сбора характеристик на основе набора тестовых программ. Для решения любых задач необходим достаточный набор исходных данных, которые к тому же периодически должны обновляться при модернизации и изменении компиляторов. В связи с этим необходима система, которая бы позволяла собирать статические характеристики при различных опциях компилятора на наборе тестовых программ за один запуск. В составе системы LLVM существует отдельный проект TestSuite [5] с набором тестовых программ, предназначенный для тестирования и замеров производительности компиляторов. Данный тестовый набор легко расширяем и поэтому подходит для сбора данных о статических характеристиках программ. Для автоматического запуска и сбора данных сразу для многих возможных конфигураций была разработана система, позволяющая не только запускать, но и автоматически сохранять результаты работы в базе данных для последующей работы с ними. Основными входными данными для нее является основной набор опций компилятора (для clang примерами могут служить оптимизирующие флаги –O2, –O3, –Os и т. д.) и набор дополнительных опций, влияние которых на генерируемый код и необходимо

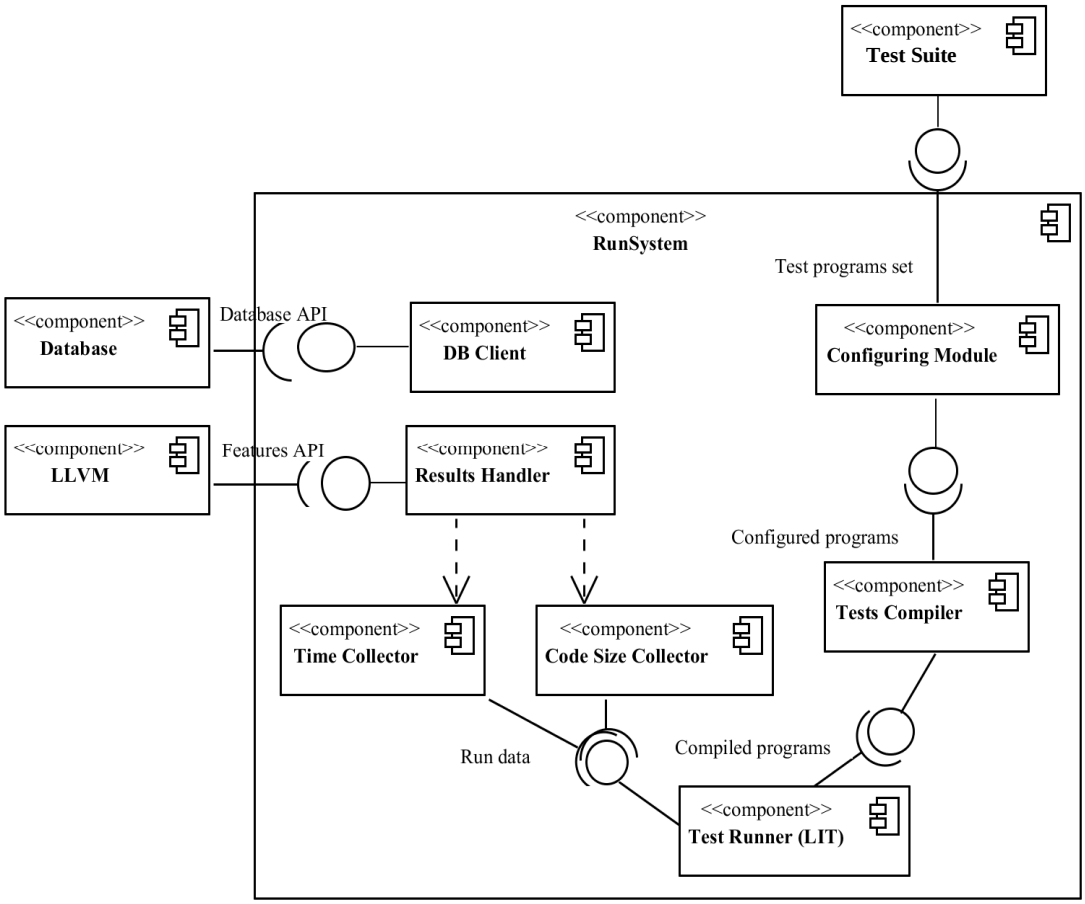


Рис. 2

отследить в решаемой задаче анализа. Архитектура системы представлена в виде диаграммы компонентов (UML 2.0) на рис. 2.

Система самостоятельно конфигурирует указанный пользователем компилятор для вывода статических характеристик в файлы с именами, рассчитанными на основе хеш-функции от неизменных свойств основных элементов, что позволяет в дальнейшем найти соответствие данных для одних и тех же программ и их частей при сохранении. Результаты работы конфигурации и запуска находятся в отдельной директории, указанной пользователем. Исполняемые файлы программ, полученные в результате компиляции, находятся в директориях, настроенных системой специальным образом для возможности дальнейшего разбора и сохранения. Далее производится запуск полученных бинарных файлов. Система проектировалась с учетом возможности разработки и подключения различных модулей на данном этапе. Это связано с тем, что запуск и, соответственно, замеры времени выполнения могут быть получены с помощью различных средств, как входящих в состав LLVM, так и сторонних, с различным уровнем детализации, который непосредственно зависит от поставленной задачи по анализу данных. По умолчанию в системе существует 2 модуля, позволяющих измерить время выполнения всей программы с помощью средства lit и получить время по отдельным функциям с помощью различных профилировщиков gprof, oprof и т. д. Также в системе предусмотрен модуль для измерения размера кода, так как данный показатель тоже является важным во множестве существующих задачах анализа.

Все измеренные показатели и собранные статические характеристики хранятся в файлах в директориях. Поэтому непосредственно после запуска система сохраняет все данные из файлов в базе данных. Статистические характеристики являются достаточно плохо структурированными данными за счет их разнообразия и возможной неполноты, так как различные проходы ориентируются на разные характеристики, и в некоторых задачах нет смысла собирать и обрабатывать полный набор существующих статических характеристик. К тому же объем получаемых данных при запуске даже на существующем стандартном наборе тестовых программ достаточно велик, если учитывать объемы данных для реальных задач

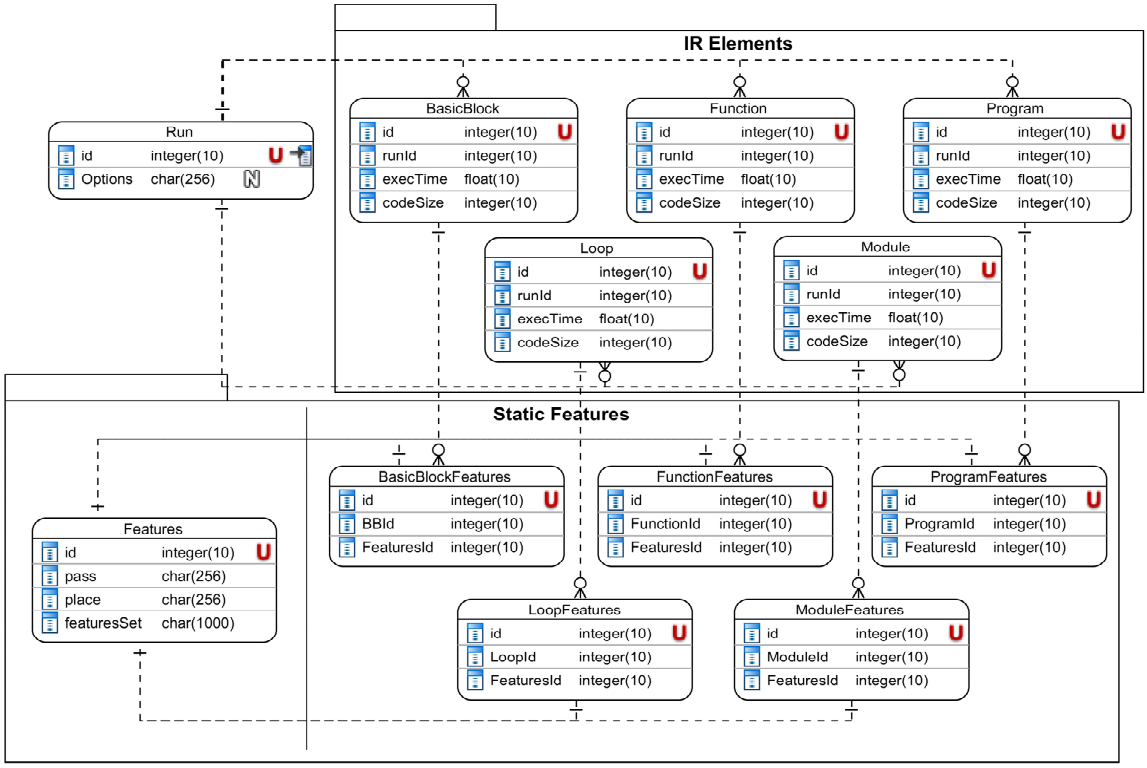


Рис. 3

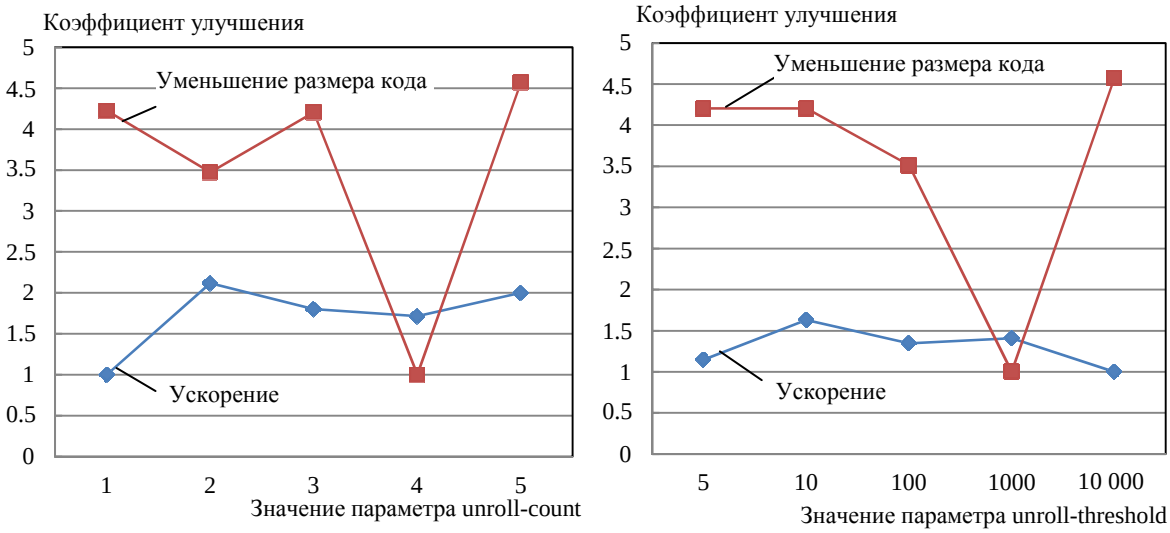
компаний. Как следствие, реляционные базы данных не подходят для этой задачи. При последующей обработке и анализе полученных данных доступ к ним должен осуществляться быстро, при этом должна существовать возможность составлять сложные запросы для сравнения полученных данных при разных конфигурациях компилятора. Исходя из данных требований, для хранения была выбрана система Elasticsearch [6], которая позволяет хранить текстовые файлы и производить быстрый полнотекстовый поиск по ним, за счет чего она может быть использована в данном случае в качестве NoSQL-базы данных. Все данные хранятся в json-файлах, которые разработанная система формирует на основе полученных при запуске данных и сохраняет в Elasticsearch.

ER-диаграмма [7] структуры используемой базы данных представлена на рис. 3.

Данная схема легла в основу генерируемых json-файлов разработанной системы, так как внутри Elasticsearch схема строится автоматически на основе передаваемых файлов. Опции запуска представляют собой объединение основных и дополнительных опций, описанных ранее. Также была добавлена специальная сущность для запуска, чтобы впоследствии можно было отличить один запуск от другого, так как между ними мог быть пересобран компилятор, внесены новые проходы и т. д. Статические характеристики были разнесены по типам для упрощения дальнейших запросов к данным. При этом все значения характеристик будут храниться внутри json-массива сущности Features, так как основной задачей при проектировании данной структуры являлось сохранение гибкости системы и поддержки разнообразных статических характеристик.

Сохраненные данные могут быть представлены в виде графиков для возможности не только решения задачи автоматизированной настройки эвристик, но и ручного подбора опций компиляции. Так, например, на рис. 4 представлены графики влияния двух внутренних опций (опции unroll-count – рис. 4, *а* и опции unroll-threshold – рис. 4, *б*) прохода для раскрутки циклов на ускорение и уменьшение размера кода. Как видно из графиков, даже отдельные опции для конкретной программы не всегда очевидны и подобрать значение для получения наиболее эффективного машинного кода бывает непросто.

Таким образом, подбор значения для каждой опции, дающей наилучшее возможное сочетание времени выполнения и размера кода, – непростая задача, так как зависимости нелинейны, имеют локальные минимумы и максимумы и уникальны как для каждой отдельной программы, так и для каждой опции компиляции. В связи с этим разработанная



а *б*

Рис. 4

система является полезным и нужным инструментом при решении задач анализа и подбора оптимизационных опций для компиляторов на базе LLVM.

В результате разработки была получена единая инфраструктура для сбора статических характеристик программ для компиляторов, разработанных на базе системы LLVM. Описанная система собирает более 50 статических характеристик программного кода, которые не доступны ни в одном другом аналоге. К тому же разработанная система не требует дополнительного аннотирования программ, о которых собираются сведения. Поэтому данная инфраструктура достаточно проста в использовании, собирает действительно важные и актуальные сведения, так как нацелена на компиляторы, созданные по одному принципу. При этом обеспечивается гибкость настройки для различных типов задач. К тому же система позволяет легко расширять не только набор собираемых характеристик, но и собирать только нужные в каждом конкретном случае данные за счет выбора проходов.

Данная инфраструктура может значительно упростить задачи по анализу результатов компиляции при использовании различных опций, настроек компилятора, принятии решений об улучшениях проходов и т. д. Таким образом, система может быть использована при решении различных задач по анализу результатов работы компиляторов, что является важной частью при разработке действительно эффективных и отвечающим потребностям пользователей средств. В случае необходимости система может быть расширена за счет использования различных модулей на всех этапах по сбору, сохранению данных и доступу к ним.

СПИСОК ЛИТЕРАТУРЫ

1. Koen Langendoen modern compiler design / D. Grune, K. van Reeuwijk, E. Henri, Bal Ceriel, J. H. Jacob // Springer. 2011. P. 821.

2. Fursin G., Miranda C., Temam O. MILEPOST GCC: machine learning based research compiler // GCC Summit individual papers, Ottawa, 2008. P. 1–13.

3. Pekhimenko G., Brown A. D. Efficient Program Compilation through Machine Learning Techniques // Software Automatic Tuning. Springer. 2010. P. 335–351.

4. Lattner C., Adve V. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation // Proc. of the Intern. Symp. on Code Generation and Optimization. IEEE, 2004. P. 81–87.

5. Formalizing the LLVM intermediate representation for verified program transformations / J. Zhao, S. Nagarakatte, Milo M. K. Martin, S. Zdancewic // Proc. POPL'12 Proc. of the 39th annual ACM SIGPLAN-SIGACT symp. on Principles of programming languages, New York, 2012. P. 427–440.

6. Gormley C., Tong Z. Elasticsearch: The Definitive Guide. O'Reilly Media, 2015. 687 p.

7. Буй Д. Б., Сильвейструк Л. Н. Формализация структурных ограничений связей в модели «сущность-связь» // Интеллектуальные системы и компьютерные науки: междунар. конф., М., 23–27 окт. 2006 г. М.: Изд-во Моск. ун-та, 2006. Т. 1, ч. 1. С. 76–79.

E. K. Lepilkina, S. I. Yakushkin
Saint Petersburg Electrotechnical University «LETI»

DEVELOPING INFRASTRUCTURE FOR COLLECTION STATIC FEATURES OF PROGRAMS TO THE TASK OPTIMIZATION OPTIONS SELECTION FOR COMPILERS BASED ON LLVM SYSTEMS

About method of collection and storing of static features of separate basic blocks of programs before and after execution of different optimizations for compilers developed on base system LLVM is discussed. Also possibility of future data access for analysis of compiler efficiency is described.

Static features, compiler, system LLVM, Elasticsearch, optimization passes

УДК 519.7+681.51

Т. Л. Качанова, Б. Ф. Фомин
Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В. И. Ульянова (Ленина)

Системные эффекты многофакторных воздействий в открытых системах

Описано решение общей задачи о типологии системных эффектов многофакторных воздействий в открытых системах применительно к многомерным массивам гетерогенных данных для большого количества влияющих и реагирующих величин. Решение получено методами физики открытых систем на основе знания онтологии систем.

Открытые системы, физика открытых систем, онтологическое знание, эффекты многофакторных воздействий, большие данные, системная аналитика

Рассмотрим задачу определения эффектов многофакторных воздействий применительно к многомерному массиву гетерогенных данных. Такой массив должен быть образован единым набором показателей. В нем выделены показатели, рассматриваемые как действующие факторы. Их количество не ограничивается, многообразие их реальной изменчивости проявлено в исходном массиве данных. Все остальные показатели в массиве данных считаются реагирующими на воздействия. Их количество также не ограничивается. Изменчивость реагирующих показателей обусловлена множественными внутрисистемными взаимодействиями, связанными с влиянием действующих факторов.

Задача об эффектах многофакторных воздействий поставлена и решена авторами данной статьи как задача о типологии изменчивости реагирующих показателей в зависимости от изменчивости действующих факторов. Сложность решения такой задачи обусловлена большой размерностью массива данных, гетерогенностью данных, большим количеством влияющих и реагирующих величин, разнообразием способов шкалирования величин, наличием пропущенных значений данных. Преодолеть сложность решения этой задачи позволяет подход, основанный на идеях и методах системологии, реализованных в физике открытых систем (ФОС) [1]–[4].

Научный метод ФОС полагает, что многомерный массив гетерогенных данных трансформируется в исходное эмпирическое описание открытой системы в ее актуальных состояниях с учетом среды. Эмпирическое описание системы представляется таблицей «Объект – свойство», каждая строка которой (объект) представляет одно конкретное актуальное состояние системы, заданное вектором значений показателей (свойств). В этом векторе конкретным значениям действующих показателей отвечают конкретные значения реагирующих показателей. Технологии ФОС извлека-