УДК 681.324

П.П. Кравченко, В.В. Хашковский

О ВОЗМОЖНОСТИ ПРИМЕНЕНИЯ СИСТЕМНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ LLVM ДЛЯ НАУЧНЫХ И ПРИКЛАДНЫХ ИССЛЕДОВАНИЙ В ОБЛАСТИ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

Рассмотрена проблема организации современных научных исследовании в области математического и программного обеспечения ЭВМ, связанная с разработкой вспомогательного программного кода, в том числе и с использованием генераторов компиляторов, а также подход к сборке пакетов программ на основе традиционного средства – GCC и современные средства компиляции программ на основе инфраструктуры LLVM, включая промежуточное представление программного кода с помощью этих средств. Приведены примеры использования LLVM для решения некоторых задач системного уровня и обозначены возможности применения LLVM для задач в области информационной безопасности.

Системное программное обеспечение; LLVM; трансформации; промежуточное представление, информационная безопасность; компиляция.

P.P. Kravchenko, V.V. Khashkovsky

ABOUT APPLICATION SYSTEM SOFTWARE LLVM IN SCIENTIFIC AND APPLIED RESEARCH IN INFORMATION SECURITY

The paper discusses the problem of the organization of modern scientific research in the field of mathematical and computer software related to the development of the auxiliary code, including the use of generators and compilers, as well as the approach to build software packages based on the traditional tools – GCC and modern means of compiling a program on the basis Infrastructure LLVM, including the intermediate representation of code with these tools. Examples of the use LLVM to solve some problems in system-level and indicated the possibility of using LLVM for problems in information security.

System software; LLVM, transformation; the intermediate representation; information security; compile.

Значительная часть усилий современных исследователей нередко тратится на подготовку вспомогательного периферийного кода, который часто практически не имеет отношения к сути решаемой научной или прикладной задачи. Для многих специальностей послевузовского профессионального образования (аспирантуры) поддержка исследований в предметной области в определенной мере обеспечивается средствами моделирования (достаточно распространены различные пакеты математического, имитационного, твердотельного, 3D-моделирования, всевозможные САПРы и проч.). При этом положение дел с разработкой вспомогательного программного кода до сих пор остается напряженным для тех специальностей, область исследований которых непосредственно связана с моделями, методами и алгоритмами проектирования и анализа программ и программных систем, их эквивалентных преобразований, верификации и тестирования, языками программирования и системами программирования, семантикой программ, моделями, методами, алгоритмами, языками и программными инструментами для организации взаимодействия программ и программных систем. Это обусловлено, в первую очередь, тем, что в данном случае инструмент является и объектом, и средством, и методом исследования. Для решения научных и прикладных задач используется аппарат теории автоматов, формальных языков, компиляторов и т.д. Высокая трудоемкость этих задач, привела в свое время к созданию целого класса системного программного обеспечения: так называемых генераторов компиляторов (например, *yacc*, *bison* – как синтаксические анализаторы и *flex* – лексический анализа-

тор). Однако такого рода программные средства хотя и могут быть эффективно применены для разработки языков, тем не менее имеют серьезные ограничения при решении практических задач. Причем эти ограничения связаны не с функциональными возможностями генераторов, а с технологией разработки программного обеспечения. В данном случае это означает, что, например, для добавления некоторых функциональных возможностей в уже существующий язык программирования требуется реализовать весь синтаксический набор, уже реализованный в существующих компиляторах и соответствующий стандартам языка, и только после этого заниматься добавлением новых языковых конструкций, включая модификацию синтаксического, лексического анализаторов и кодогенератора. При этом, чтобы результат разработки был практически востребован, необходимо реализовать совместимость с уже существующими компиляторами, что само по себе является трудоемкой задачей.

Таким образом, практическая ценность результатов исследований в области математического и программного обеспечения вычислительных машин может быть достигнута только при максимально полном следовании существующим стандартам и использовании существующих средств системного программного обеспечения.

В области научной и академической деятельности для организации научных исследований традиционно одно из центральных мест занимают операционные окружения, основанные на применении свободного программного обеспечения (СПО) и unix-подобных систем. В последние годы все больше внимания уделяется GNU Linux. Основным (но далеко не единственным) средством разработки, в том числе и самой операционной системы, является группа компиляторов GCC (*GNU Compiler Collection*; первоначально, когда в 1987 г. Р. Столлман выпустил компилятор языка C, он назывался *GNU C Compiler*). Эта группа (коллекция) компиляторов включает в себя языки C, C++, Objective-C, Fortran, Java, Ada и Go.

В работе компилятора достаточно четко выделяется несколько стадий [1], между которыми передаются результаты работы подсистем компилятора, как представлено на рис. 1.

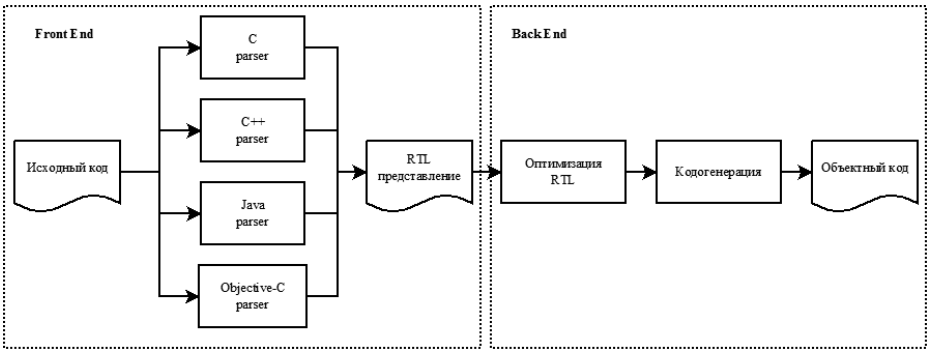

Рис. 1. Схема работы GCC

Первая стадия (*Front End*) выполняет синтаксический анализ и преобразует исходный текст программы на языке программирования к промежуточному представлению RTL (*Register Transfer Language*, язык межрегистровых пересылок). На дальнейших стадиях (*Back End*) происходит дополнительная обработка RTL-представления, например оптимизация, и конечная кодогенерация для целевой платформы. Однако RTL-представления имеют существенный недостаток, связанный с тем, что оптимизация программного кода в RTL-форму сильно зависит от

целевой машинной архитектуры. Поэтому современные версии GCC уже используют для промежуточного представления программного кода форму SSA-деревьев (*Static Single Assignment*), которые хотя и далеки от исходного языка, однако сохраняют достаточно высокий уровень представления программного кода, что позволяет организовать улучшенный анализ и оптимизацию программ.

Несмотря на то, что такая схема набора компиляторов проверена временем и достаточно эффективно используется, в ней все еще мало места для того, чтобы внедриться в цепочку трансляции и реализовать собственные алгоритмы обработки/анализа/оптимизации программного кода перед кодогенерацией.

Другим подходом к построению компилятора и его окружения с 2000 г. также является набор программных средств LLVM (*Low Level Virtual Machine*), который представляет собой систему анализа, трансформации и оптимизации программ [2, 3]. Работа LLVM основана на промежуточном представлении кода (*IR, intermediate representation*), над которым можно производить трансформации во время компиляции, компоновки и даже во время выполнения. Это представление (по сути типизированный трёхадресный код в SSA-форме) является основой для статической и динамической (*JIT, just-in-time*) кодогенерации; поддерживаются платформы x86, x86-64, ARM, PowerPC, SPARC, MIPS, IA-64, Alpha. Система LLVM имеет реализации практически для всех существующих операционных систем и в этом смысле может считаться универсальной системой программирования, хотя на сегодняшний день front end ограничен поддержкой C, C++, Ada и Fortran.

Общая схема работы LLVM представлена на рис. 2. Здесь на вход подается исходный код, после работы front end он преобразуется в байт-код, который подается на вход оптимизатору (*opt*, уровни оптимизации: O0-O3).

Рис. 2. Этапы работы LLVM

После оптимизации байт-код подается на вход кодогенератору (llc), а затем ассемблеру и линковщику кода для целевой платформы. В целом окружение LLVM включает такие инструменты, как opt (работа с байт-кодом, проведение машинно-независимой оптимизации, различные виды анализа и профилирование); llc (кодогенератор из байт-кода в ассемблер целевой машины, машинно-зависимая оптимизация); llvm-ld (линковщик байт-кода); lli (интерпретатор байт-кода, а также JIT-компиляция); llvm-dis (преобразование байт-кода в эквивалентное текстовое представление); llvm-as (преобразование текстового представления в байт-код); llvm-ar (упаковщик); llvmc (драйвер LLVM, управляющий работой других подсистем).

Благодаря возможности вмешиваться в стандартный процесс компиляции, например, на этапе оптимизации, оперируя при этом IR-представлением программы, имеется возможность выполнять любые трансформации исходной программы, включая не только оптимизацию, как таковую, но и серьезное расширение возможностей языка (за счет обработки специализированных вызовов функций без необходимости при этом изменять синтаксический и лексический анализаторы языка программирования) [4, 5], выполнять профилирование и последующие исследования работы программы во время выполнения на реальных данных [6] и многое другое.

Широкие возможности LLVM именно в части анализа и трансформации программного кода на этапе компиляции (а в некоторых случаях и на этапе исполнения – JIT) могут являться основой для разработки и реализации современных методов и

систем защиты информации, что особенно важно в контексте широкомасштабного внедрения свободного программного обеспечения, в том числе и в органах государственной власти. Применение технологий информационной безопасности, использующих трансформации LLVM, позволит уже на этапе сборки программного продукта на целевой платформе выполнить необходимый анализ программного кода на наличие недокументированных возможностей (так называемых закладок), на этапе исполнения выполнить проверки признаков несанкционированной модификации программного кода, модифицировать существующие программные пакеты широкого назначения для применения в специальных условиях и так далее.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Тим Джонс М.* Знакомимся с GCC 4: на русском языке [Электронный ресурс] // URL: http://www.ibm.com/developerworks/ru/library/l-gcc4/ (дата обращения: 05.02.2012).
2. The LLVM Compiler Infrastructure: на английском языке [Электронный ресурс] // URL: http://llvm.org (дата обращения: 28.01.2012).
3. Википедия: свободная электронная энциклопедия: на русском языке [Электронный ресурс] // URL: http://ru.wikipedia.org/wiki/LLVM (дата обращения: 28.01.2012).
4. *Данилов И.Г.* Организация доступа к распределённой памяти в прототипе распределённой программной транзакционной памяти DSTM_P1, Высокопроизводительные вычислительные системы // Труды молодых ученых ЮФУ и ЮНЦ РАН. – Таганрог: Изд-во ТТИ ЮФУ, 2011. – С. 50-55.
5. *Данилов И.Г.* Современные подходы к созданию многопоточных приложений для многомашинных конфигураций с эмуляцией общей памяти // Известия ЮФУ. Технические науки. – 2012. – № 1 (126). – С. 92-96.
6. *Аветисян А.И., Долгорукова К.Ю., Курмангалеев Ш.Ф.* Динамическое профилирование программы для системы LLVM: на русском языке [Электронный ресурс] // URL: http://www.ispras.ru/ru/proceedings/docs/2011/21/isp_21_2011_71.pdf (дата обращения: 28.01.2012).

Статью рекомендовал к опубликованию д.т.н., профессор Я.Е. Ромм.

Кравченко Павел Павлович – Технологический институт федерального государственного автономного образовательного учреждения высшего профессионального образования "Южный федеральный университет" в г. Таганроге; e-mail: kravch@tsure.ru; 347928, г. Таганрог, пер. Некрасовский 44; тел.: 88634371673; кафедра математического обеспечения и применения ЭВМ; д.т.н.; профессор.

Хашковский Валерий Валерьевич – e-mail: vvx@softengine.ru; кафедра математического обеспечения и применения ЭВМ; к.т.н.; доцент.

Kravchenko Pavel Pavlovich – Taganrog Institute of Technology – Federal State-Owned Autonomy Educational Establishment of Higher Vocational Education “Southern Federal University”; e-mail: kravch@tsure.ru; 44, Nekrasovsky, Taganrog, 347928, Russia; phone: +78634371673; the department of software engineering; dr. of eng. sc.; professor.

Khashkovsky Valery Valerjevich – e-mail: vvx@softengine.ru; the department of software engineering; dr. of eng. sc.; associate professor.