

ՀՀ ԳԱԱ ԻՆՖՈՐՄԱՏԻԿԱՅԻ ԵՎ ԱՎՏՈՄԱՏԱՑՄԱՆ ՊՐՈՔԼԵՄՆԵՐԻ ԻՆՍՏԻՏՈՒՏ

Վարդանյան Մամիկոն Աշոտի

ԿՈՄՊԻԼԵՅԱՏՈՐԻ ՄԻ ՔԱՆԻ ՉԱՓՈՐՈՇԻՉՆԵՐՈՎ ԱՐԴՅՈՒՆԱՎԵՏ  
ՕՊՏԻՄԻԶԱՅԻԱՆԵՐԻ ՓՆՏՐՄԱՆ ԱՎՏՈՄԱՏԱՑՎԱԾ ՄԵԹՈՂ՝ ՀԻՄՆՎԱԾ  
ՊԱՐԵՏՈ ԴՈՄԻՆԱՆՏՈՒԹՅԱՆ ՎՐԱ

Ե.13.04 – «Հաշվողական մեքենաների, համալիրների, համակարգերի և ցանցերի մաթեմատիկական և ծրագրային ապահովում» մասնագիտությամբ տեխնիկական գիտությունների թեկնածուի գիտական աստիճանի հայցման ատենախոսության

ՍԵՂՄԱԳԻՐ

Երևան - 2015

---

ИНСТИТУТ ПРОБЛЕМ ИНФОРМАТИКИ И АВТОМАТИЗАЦИИ НАН РА

Варданян Мамикон Ашотович

МЕТОД АВТОМАТИЧЕСКОГО ПОДБОРА ЭФФЕКТИВНЫХ ОПТИМИЗАЦИЙ  
КОМПИЛЯТОРА ПО НЕСКОЛЬКИМ КРИТЕРИЯМ НА ОСНОВЕ ПАРЕТО-  
ДОМИНИРОВАНИЯ

АВТОРЕФЕРАТ

диссертации на соискание ученой степени кандидата технических наук по специальности  
05.13.04 – «Математическое и программное обеспечение математических машин,  
комплексов, систем и сетей»

Ереван - 2015

Ատենախոսության թեման հաստատվել է Ռուսաստանի գիտությունների  
ակադեմիայի Համակարգային ծրագրավորման ինստիտուտում

Գիտական ղեկավար՝ ֆիզ.մաթ.գիտ. դոկտոր Վ.Պ. Բվաննիկով

Պաշտոնական ընդդիմախոսներ՝ ֆիզ.մաթ.գիտ. դոկտոր Ս.Կ.Շուքրյան  
տեխ.գիտ.թեկնածու Մ.Ղ.Գյուրջյան

Առաջատար կազմակերպություն՝ Հայաստանի ազգային պոլիտեխնիկական  
համալսարան

Պաշտպանությունը կայանալու է 2016թ. հունվարի 29-ին, ժ. 15.00-ին ՀՀ ԳԱԱ  
Ինֆորմատիկայի և ավտոմատացման պրոբլեմների ինստիտուտում գործող 037  
«Ինֆորմատիկա և հաշվողական համակարգեր» մասնագիտական խորհրդի  
նիստում հետևյալ հասցեով՝ Երևան, 0014, Պ. Սևակի 1:

Ատենախոսությանը կարելի է ծանոթանալ ՀՀ ԳԱԱ ԻԱՊԻ գրադարանում:  
Սեղմագիրը առաքված է 2015թ. Դեկտեմբերի 28-ին:

Մասնագիտական խորհրդի գիտական  
քարտուղար, ֆիզ.մաթ.գիտ.դոկտոր



Հ. Գ. Սարգսյանյան

---

Тема диссертации утверждена в Институте системного программирования РАН

Научный руководитель: доктор физ.-мат. наук В.П.Иванников

Официальные оппоненты: доктор физ.-мат. наук С.К.Шукурян  
кандидат тех.наук М.Г.Гюрджян

Ведущая организация: Национальный политехнический университет Армении

Защита состоится 29 января 2016г. в 15.00 часов на заседании специализированного  
совета 037 «Информатика и вычислительные системы» Института проблем информатики  
и автоматизации НАН РА по адресу: 0014, г. Ереван, ул. П. Севака 1.

С диссертацией можно ознакомиться в библиотеке ИПИА НАН РА.

Автореферат разослан 28-го декабря 2015г.

Ученый секретарь специализированного  
совета, доктор физ. мат. наук



А. Г. Саруханян

## ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

**Актуальность работы.** Современные компиляторы, как правило, включают сотни различных оптимизаций, которые, работая на разных этапах компиляции, влияют на качество сгенерированного кода. При этом набор оптимизаций (опций компилятора), при которых генерируется оптимальный объектный код зависит как от целевой архитектуры машины, так и от специфики выбранного приложения. Обычно разработчики и пользователи приложений используют параметры компилятора по умолчанию (например -O2 в компиляторах GCC и LLVM), определяющие базовые наборы оптимизаций. Такие наборы определяются разработчиками компиляторов известными наборами тестов (как правило, SPEC<sup>1</sup>) и обеспечивают компиляцию оптимального кода в среднем для данного набора. Обычно, для разных приложений оптимальные наборы опций компилятора могут различаться, в том числе, в зависимости от целевой платформы. При этом, для конкретного приложения можно значительно увеличить производительность только за счет правильного подбора параметров компилятора. Задача усложняется еще тем, что критериями оптимальности могут являться наряду с производительностью, также размер кода, энергопотребление, время компиляции и т.д. Кроме того, нередко требуется оптимизировать одновременно по нескольким критериям (например, по производительности и размеру кода). В некоторых случаях критерии могут не согласовываться между собой, и тогда приходится использовать компромиссные подходы.

Подбор опций компилятора и значений параметров компиляции для конкретного приложения требует глубокого знания инфраструктуры компилятора и особенностей целевой платформы, поэтому для разработчиков и пользователей приложений, не специализирующихся в области компиляторных технологий, актуальны автоматические методы подбора параметров компиляции. Существующие инструменты автоматического подбора оптимизаций компилятора (в том числе с поддержкой многокритериального поиска) не удовлетворяют одновременно таким актуальным требованиям пользователей и разработчиков программного обеспечения как, например, поддержка подбора числовых параметров компиляции, многокритериальный поиск, параллельная кросс-компиляция и запуск на нескольких тестовых устройствах с целевой архитектурой.

Обеспечить улучшение эффективности работы приложения можно также посредством корректировки исходного кода. Корректировка исходного кода библиотеки может повысить уровень применимости ряда межмодульных оптимизаций (например, открытая вставка между модулями) компилятора на этапе компоновки, так как многие библиотеки (особенно более старые) при написании не были рассчитаны на такие возможности компилятора. Некоторые библиотеки содержат некорректные объявления области видимости своих функций, что приводит к тому, что все глобальные функции самой библиотеки автоматически экспортируются в качестве ее интерфейса. Это препятствует

---

<sup>1</sup> J. L. Henning. SPEC CPU2000: Measuring CPU performance in the new millennium. IEEE Computer, 33(7):28–35, July 2000

выполнению межмодульных оптимизаций на этапе компоновки, так как эти функции могут быть вызваны из других библиотек во время выполнения кода. Корректировка соответствующих объявлений в библиотеках позволяет существенно повысить применимость оптимизаций на этапе компоновки и соответственно улучшить производительность и уменьшить размер кода, при этом не нарушается целостность библиотеки.

Решение перечисленных задач проблем связано с существенными трудозатратами и требует высокой квалификации, и необходимо разработать соответствующие средства автоматизации для решения указанных проблем. В Институте системного программирования РАН проводятся исследования и разрабатывается набор инструментов TACT (Toolkit for Automatic Compiler Tuning), обеспечивающий максимальное использование возможностей компилятора для оптимизации конкретного приложения под выбранную целевую архитектуру. Настоящая работа является составной частью этих исследований.

**Целью диссертационной работы** является разработка метода автоматического выбора опций компилятора и значений его параметров, обеспечивающих оптимизацию целевой программы в соответствии с выбранным критерием (включая возможность многокритериальной оптимизации) для заданного приложения и целевой платформы. Кроме того, необходимо разработать метод автоматической проверки и корректировки исходного кода библиотек, обеспечивающий выполнение оптимизаций во время компоновки.

Разработанные методы должны быть реализованы в рамках инфраструктуры набора инструментов TACT.

**Методы исследования**, используемые в диссертационной работе, включают методы поиска, основанные на генетических алгоритмах, методы статистической обработки данных и методы разработки программного обеспечения.

**Научная новизна.** В диссертационной работе были получены следующие результаты:

- Разработан метод автоматического подбора оптимизаций компилятора для конкретного приложения и целевой архитектуры. Данный метод поддерживает также многокритериальную оптимизацию на основе Парето-доминирования.
- Исследовано и оценено влияние конфигураций генетического алгоритма на процесс эволюционного поиска во время оптимизации.
- Разработан метод автоматической проверки и корректировки исходного кода библиотек программ, обеспечивающий улучшение эффективности выполнения оптимизаций на этапе компоновки.

**Практическая ценность.** Разработанные методы реализованы в рамках инструментальной инфраструктуры TACT, учитывая особенности компиляторов GCC и LLVM и характеристики применяемости концепций генетического алгоритма. Автоматическое улучшение качества применяемости компилятора на выбранные приложения позволяет разработчикам и пользователям приложений (при наличии

исходного кода) получить прирост производительности и/или уменьшить размер исполняемого кода до нескольких десятков процентов за приемлемое время, не требуя от них знания инфраструктуры компилятора и целевой платформы.

Программная система разрабатывалась в рамках научно-исследовательского проекта корпорации Samsung и с ее согласия находится в открытом доступе (Open-source).

#### **Положения, выносимые на защиту:**

- Метод автоматического подбора опций компилятора и значений его параметров, обеспечивающий генерацию эффективного в соответствии с заданным критерием оптимальности целевого кода для конкретного приложения на заданной платформе. Данный метод поддерживает также многокритериальную оптимизацию на основе Парето-доминирования.
- Метод автоматической проверки и корректировки исходного кода библиотек, обеспечивающий эффективное выполнение оптимизаций на этапе компоновки
- Инструментальное средство на основе предложенных методов с использованием генетического алгоритма, позволяющее отражать влияние разных значений конфигураций инструмента на качество результирующего кода.

**Апробация и публикации работы.** Основные результаты и положения диссертационной работы обсуждались и докладывались на семинарах Института системного программирования РАН (2012-2015 гг.), научно-исследовательских семинарах Вычислительного центра РАН (Москва, Россия, 2014) и МФТИ (ГУ) (Москва, Россия, 2014), представлялись на международной конференции по вычислительным наукам и информационным технологиям “Computer Sciences and Information Technologies” (CSIT) 2013 в г. Ереване, Армения и на международной научно-практической конференции «International Conference on Computational Science» (ICCS) 2013 в г. Барселона, Испания.

Результаты работы отражены в пяти публикациях, список которых приведен в конце автореферата.

**Структура и объем работы.** Диссертация состоит из введения, четырех глав, заключения и списка использованной литературы (90 наименований). Основной текст изложен на 102-х страницах.

## **ОСНОВНОЕ СОДЕРЖАНИЕ РАБОТЫ**

**Во введении** обоснована актуальность работы и практическая значимость темы диссертационной работы, кратко изложено состояние предметной области, сформулированы цели и основные задачи исследования, выделены научные результаты,

отличающиеся новизной, положения, выносимые на защиту, и приведена практическая ценность полученных результатов. Дается краткий обзор работы.

**В главе 1** производится обзор работ, имеющих отношение к теме диссертации. В начале главы приводятся термины, понятия, которые будут использованы в тексте диссертации. Далее рассматриваются широко используемые компиляторные инфраструктуры с открытым исходным кодом (в первую очередь GCC и LLVM) и особенности выполнения ими оптимизаций. Рассматриваются подходы, позволяющие улучшить качество сгенерированного кода. Первым подходом является ручной анализ - для выбранных приложений собирается профиль выполнения, определяются часто вызываемые функции, и вручную выполняется анализ полученного кода. Это довольно трудоемкий процесс, т.к. необходимо не только выделить те ассемблерные команды, которые больше всего влияют на производительность, но и определить, какие именно оптимизации компилятора или их параметры привели к появлению этих команд. Однако, ручной анализ помогает выявить общие закономерности, которые могут помочь в дальнейших оптимизациях других приложений. Очень часто требуется получить хорошо сгенерированный код, не вникая в инфраструктуру компилятора. Для этого широко используется подход “автоматического поиска эффективных оптимизаций (примеры приводятся ниже), чему и будет посвящена основная часть диссертации. Далее анализируются существующие методы, позволяющие автоматически получить эффективные оптимизации для конкретного приложения на целевой платформе.

Существует некоторое количество систем, позволяющих автоматически подобрать эффективные опции компилятора. Они часто основаны на принципах эволюционного поиска, но есть также методы, основанные на машинном обучении, других приближенных алгоритмах поиска и т.д. Рассмотрены и проанализированы некоторые доступные инструментальные средства.

- MILEPOST<sup>2</sup> (Machine Learning for Embedded Programs optimization) – исследовательский компилятор с применением методов машинного обучения. Он реализован как надстройка над исходным компилятором GCC, расширяя его функциональность посредством интерфейса ICI (Interactive Compilation Interface). Данный инструмент имеет множество уникальных свойств (поддержка динамических особенностей программы, полученных во время профилирования и т.д.), однако также немало недостатков (обучение проводится по случайным опциям, отсутствует многокритериальный поиск и т.д). Разработка инструмента MILEPOST не завершена и на данный момент не осуществляется.
- PEAK<sup>3</sup> - инструментальное средство, разработанное в Purdue University и позволяющее посредством комбинации нескольких приближенных алгоритмов

---

<sup>2</sup>G. Fursin, C. Miranda, O. Temam, M. Namolaru, E. Yom-Tov, A. Zaks, B. Mendelson, E. Bonilla, J. Thomson, H. Leather, C. Williams, M. O’Boyle, P. Barnard, E. Ashton, E. Courtois, and F. Bodin, “MILEPOST GCC: machine learning based research compiler,” in Proceedings of the GCC Developers’ Summit, Jul 2008

<sup>3</sup>Z. Pan, R. Eigenmann “PEAK—A Fast and Effective Performance Tuning System via Compiler Optimization Orchestration” in Code Generation and Optimization, 2006. CGO 2006

поиска за приемлемое время подобрать оптимальный набор опций компилятора. В описании инструментального средства говорится о преимуществах по отношению к рассмотренным автором инструментам, однако в нем отсутствует описание поддержки подбора числовых параметров компиляции, многокритериального поиска и т.д, которые очень важны в рамках данной задачи. При этом инструмент не находится в открытом доступе.

- OpenTuner<sup>4</sup>- расширяемая инструментальная среда (framework), разработанная в MIT и позволяющая пользователю с помощью добавления механизма и конфигурации поиска подобрать значения по выбранным критериям. В этой среде реализована возможность подбора эффективных оптимизаций компилятора. Несмотря на то, что инструмент дает хорошие результаты на некоторых простых тестах (таких как умножение матриц, и т.д.), в нем пока не реализованы некоторые полезные свойства (такие, как многокритериальный поиск, поддержка параллельной кросс-компиляции и запуска на целевой платформе, и т.д.).
- Среди доступных инструментов на базе генетического алгоритма одним из наиболее успешных является ACOVEA<sup>5</sup>. Инструмент позволяет использовать пользовательские скрипты для контролирования процесса эволюции, поэтому он может улучшать производительность, размер кода или любой другой указанный пользователем критерий. Несмотря на то, что инструмент является одним из самых развитых, доступных в данный момент, в нем отсутствуют некоторые важные возможности (многокритериальный поиск, отсутствие возможности параллельной компиляции и выполнения на нескольких устройствах и т.д.).
- COLE (Compiler Optimization Level Exploration)<sup>6</sup> - исследовательский инструмент для автоматического поиска уровней оптимизации, оптимальных по Парето, на основе многоцелевого эволюционного поиска. Ожидается, что он будет доступен с открытым исходным кодом. Он ещё не готов до конца, но автор утверждает, что COLE способен выполнять настройку компилятора с учетом сочетания параметров. COLE является одним из немногих инструментов, позволяющих подобрать эффективные опции компилятора по двум критериям, однако не поддерживает некоторые полезные свойства (подбор параметров компиляции, поддержка кросс-платформенности, и т.д.).

Большинство известных инструментов автоматического подбора опций компилятора работают только с одним критерием оптимальности (оптимизируется либо только производительность, либо только размер исполняемого файла, либо только время компиляции и т.д.) Желательно иметь возможность поиска подходящего набора опций компилятора сразу по нескольким критериям. Обычно в случае многокритериальной оптимизации используются компромиссы, позволяющие объединить несколько

---

<sup>4</sup>Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly "OpenTuner: An Extensible Framework for Program Autotuning" International Conference on Parallel Architectures and Compilation Techniques. Edmonton, Canada. August, 2014. Slides. Bibtex.

<sup>5</sup>ACOVEA home page. [www.coyoteulch.com/products/acovea/index.html](http://www.coyoteulch.com/products/acovea/index.html)

<sup>6</sup> Kenneth Hoste, Lieven Eeckhout. COLE: Compiler Optimization Level Exploration. [http://boegel.kejo.be/ELIS/pub/hoste08cole\\_CGO08\\_paper.pdf](http://boegel.kejo.be/ELIS/pub/hoste08cole_CGO08_paper.pdf)

критериев в один. Один из основных подходов такого объединения основан на принципе Парето-доминирования<sup>7</sup>. В Швейцарском федеральном институте технологии (Цюрих) разработан алгоритм многокритериального поиска данных SPEA2, который является улучшенной версией известного алгоритма SPEA (Strength Pareto Evolutionary Algorithm) и основан на принципе Парето-доминирования. В публикации, описывающей алгоритм, проводится сравнительный анализ с другими известными алгоритмами многокритериального поиска (такими как, NSGA-II и PESA), где алгоритм SPEA2 обеспечивает лучшую устойчивость и скорость сходимости при выпуклой границе Парето. Этот алгоритм и был взят за основу для разработки метода подбора опций и значений параметров компиляции, описываемого в следующей главе.

**В главе 2** приводится описание метода автоматического подбора опций и значений параметров компиляции (как для одной целевой функции, так и по нескольким критериям) для определенного приложения под конкретную архитектуру.

Каждый из современных многоплатформенных компиляторов GCC и LLVM имеет более 200 опций и более 70 числовых параметров оптимизации. Для облегчения задания опций и значений параметров компиляции разработчиками компиляторов предложены базовые наборы оптимизаций (-O1, -O2, -O3, -Os), которые обеспечивают хорошую производительность для широко используемых наборов тестов (обычно, для набора тестов SPEC). Но несмотря на то, что базовые наборы удовлетворяют требованиям широкого круга пользователей, для конкретных приложений и архитектур они могут не обеспечить желаемый результат, и возникает необходимость подбора опций. Кроме того, необходимо учитывать, что базовые наборы оптимизаций были разработаны в основном для самой используемой архитектуры x86, но существуют не менее востребованные архитектуры (например, такие как ARM), для которых базовые наборы могут быть менее приемлемы. В LLVM в зависимости от языка могут использоваться разные фронтенды, такие как Clang для языков C/C++ и при подборе эффективных опций важно учитывать, как опции фронтенда (Clang), так и опции бекэнда (LLVM).

В некоторых случаях, как показывает практика, можно значительно повысить эффективность сгенерированного кода, отключив оптимизации базового набора (это обычно бывает связано со специфическими особенностями исходного кода приложения). Кроме того, многие оптимизации, плохо взаимодействуют друг с другом и неправильные их комбинации могут значительно ухудшить конечные результаты. Численные параметры применимости оптимизаций могут иметь достаточно большой диапазон значений, что значительно затрудняет процесс выбора оптимального значения.

Задача по подбору опций компилятора для конкретного приложения и архитектуры является крайне трудоемкой, и найти оптимальный набор оптимизаций для компиляторов GCC и LLVM не представляется возможным в полиномиальное время, и для ее решения необходимо использовать приближенные методы. В данной работе в качестве таких

---

<sup>7</sup> E. Zitzler, L. Thiele. Multi-objective algorithms: a comparative case study and the strength Pareto approach. IEEE Trans Evol Comput 1999; 3(4): 257–71.

методов используются методы на базе генетического алгоритма, которые также применяются в рассмотренных инструментальных средствах ACOVEA и COLE.

Как отметили выше, в качестве базового генетического алгоритма поиска данных был выбран SPEA2. В представленной работе данный алгоритм был доработан для удовлетворения требованиям как многокритериального поиска, так и поиска по одной целевой функции. Кроме того, к этому алгоритму добавлена возможность поддержки нескольких популяций параллельно, что позволяет повысить эффективность метода.

В алгоритме SPEA2 используется концепция элитаризма, при которой наиболее приспособленные особи предыдущих поколений сохраняются в архиве, повышая использование ранее найденных решений. Размер архива в SPEA2 не изменяется в процессе работы алгоритма.

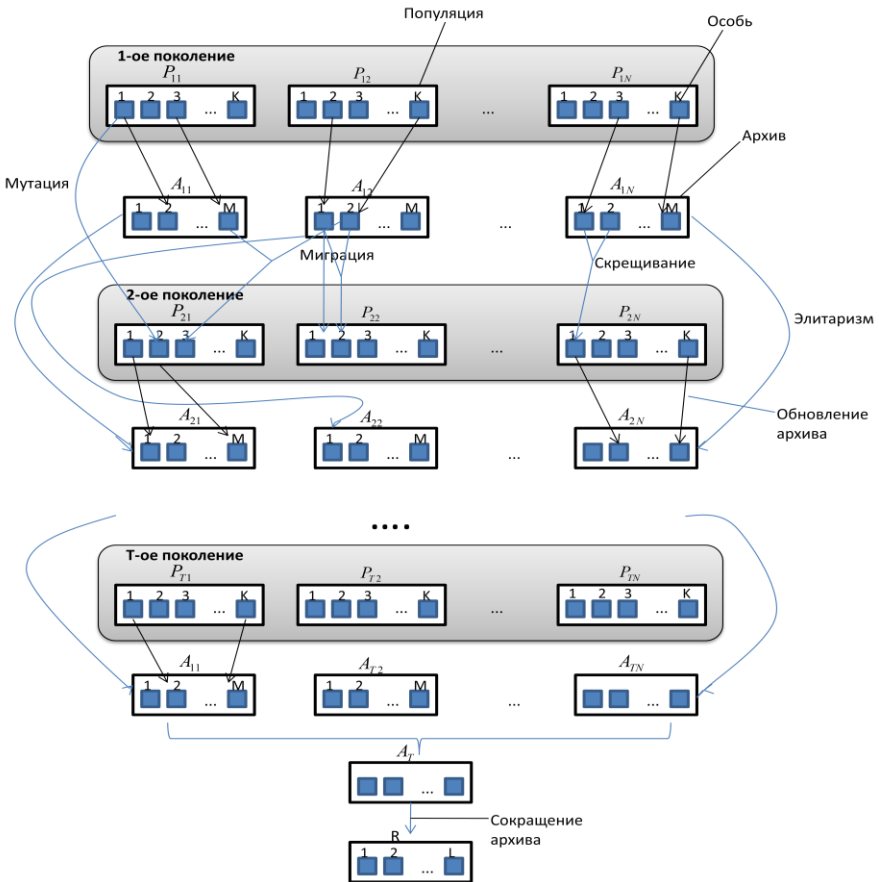


Рис.1. Схематическая демонстрация метода подбора опций и параметров компилятора.

В данной работе используются некоторые термины, заимствованные из генетических алгоритмов, в том числе из SPEA2, такие как поколение, популяция, особь, скрещивание, мутация, ген и т.д. В контексте предложенного метода подбора опций компилятора гены представляют собой булевы значения применяемых опций (-f/-fno, оптимизация включается или нет) и целочисленные значения числовых параметров компиляции (--paams), особи – наборы опций и параметров компиляции заданной неизменной длины, популяция - множество особей определенной размерности.

В разработанном в рамках диссертационной работы методе проведены ряд изменений и модификаций в примененном алгоритме эволюционного поиска SPEA2. Приведем основные особенности и отличия предложенного в методе алгоритма от базового SPEA2.

- Разработан подход, который позволяет использовать несколько параллельных популяций одновременно и применить оператор миграции между ними.
- В отличии от базового алгоритма, данный метод поддерживает поиск как по одному критерию так и по нескольким.
- Разработан механизм объединения архивов популяций для каждого поколения в один общий архив.
- При селекции выбор родительских особей проводится как среди особей архива предыдущего поколения, так и среди особей текущей популяции (в SPEA2 при селекции используются только особи архива).
- Использован метод кластеризации точек на границе Парето при многокритериальном поиске.

Опишем разработанный в данной работе метод автоматического подбора оптимизаций компилятора для конкретного приложения и целевой архитектуры, который схематически продемонстрирован на рисунке 1.

*Вход:  $N$  (число популяций),  $K$  (количество особей в одной популяции),  $M$  (размер архива),  $T$  (максимальное число поколений),  $S$  (стратегия эволюционного поиска),  $L$  (выбранное число решений)*

*Выход:  $R$  (множество предлагаемых решений)*

- 1. Инициализация первого поколения.** Создать произвольные популяции  $P_{11}, P_{12}, \dots, P_{1N}$ . Для каждой популяции  $P_{1i}$ , где  $i = 1 \dots N$  сгенерировать  $K$  наборов опций и параметров компилятора и создать пустой архив  $A_{1i}$ . Присвоить  $t = 1$ .
- 2. Вычисление приспособленности.** Вычислить значение приспособленности для всех особей текущих популяций  $P_{t1}, P_{t2}, \dots, P_{tN}$ . В случае стратегии многокритериального поиска применить принцип Парето силы, предложенный в SPEA2, а при поиске по одному критерию значение приспособленности соответствующего вычисленному результату (производительность, размер кода и т.д.) на целевой платформе.
- 3. Обновление архивов.** При  $t > 1$ , используя текущие популяции  $P_{ti}$  и архивы предыдущего поколения  $A_{t-1i}$ , где  $i = 1 \dots N$  обновить архивы  $A_{ti}$  для текущего поколения. В случае многокритериального подбора включить все недоминированные особи из  $P_{ti} \cup A_{t-1i}$  в архив  $A_{ti}$ . Если размер  $A_{ti}$  меньше  $M$ , включить доминируемые особи с наибольшим значением приспособленности до полного заполнения. Если размер архива больше  $M$  использовать оператор исключения, предложенный в SPEA2. При поиске по

одному критерию включить в  $A_{t+1i}$   $M$  наиболее приспособленных особей из  $P_{ti} \cup A_{t-1i}$ .

4. **Объединение архивов и уменьшение числа текущих решений.** Объединить все текущие архивы  $A_{ti}$ , где  $i = 1 \dots N$  в один архив  $\bar{A}_t$ . При многокритериальном поиске использовать метод кластеризации для сокращения числа скоплений точек на границе Парето (соответствующей архиву  $\bar{A}_t$ ) до  $L$  центров и включить сокращенное множество в  $\bar{R}_t$ . В случае поиска по одному критерию включить в  $\bar{R}_t$   $L$  наиболее приспособленных особей из архива  $\bar{A}_t$ .
5. **Завершение:** Если  $t \geq T$  (число поколений) или достигнуты ограничения по другим критериям, завершить работу алгоритма, присвоить  $R = \bar{R}_t$  и на выходе получить множество  $R$ .
6. **Создание популяций нового поколения.** Для каждой популяции  $P_{ti}$  и архива  $A_{ti}$ , где  $i = 1 \dots N$  с помощью бинарной турнирной селекции<sup>8</sup> выбрать приспособленные особи и применяя над ними операторы мутации и скрещивания, а также используя миграцию из особей архивов других популяций получить новую популяцию  $P_{t+1i}$ . Присвоить  $t = t+1$ , перейти на шаг 2.

Представим анализ некоторых конфигураций генетического алгоритма, полученных в результате проводимых тестов.

При инициализации особей популяций в первом поколении, применяются вероятностные методы произвольного выбора его генов. Значения булевых опций компилятора (-fortion) обычно выбираются случайным образом. Для числовых параметров (-params), имеющих большие диапазоны значений, подбираются случайные значения с использованием равномерного или нормального распределения.

При равномерном распределении, вероятность выбора значения параметра одинакова на всем диапазоне значений. При нормальном распределении, в зависимости от значения средне квадратичного отклонение  $\sigma$  (которое подбирается пользователем), вероятность разброса значений параметра в диапазоне значений значительно отличается. При этом, пользователь может руководствоваться следующими данными, полученными во время тестирования системы: при  $\sigma=1$  около 30% параметров при инициализации получают значение, равное значению по умолчанию, 53% с отклонением до 10% от значения по умолчанию, 74% с отклонением до 20% от значения по умолчанию и 97% с отклонением до 50% от значения по умолчанию, а при  $\sigma=2$  указанные значения равны, соответственно, 24%, 40%, 55% и 85%. Результаты тестирования показали, что нормальное распределение обеспечивает лучшую сходимость за счет уменьшения разброса значений.

При скрещивании особей (crossover), особый интерес представляет выбор пары приспособленных родительских особей. Этим в ГА занимается оператор селекции. В разработанном методе, при селекции, в отличие от SPEA2, выбираются особи как из

---

<sup>8</sup>U. K. Chakraborty, K. Deb and M. Chakraborty "An analysis of selection algorithms: A Markov chain approach", *Evolutionary Computation*, vol. 4, no. 2, pp.133-167 1996

текущего архива, так и из популяции предыдущего поколения, что делает метод менее консервативным и обеспечивает большую возможность выбора разнообразных генов.

Чтобы исключить большую концентрацию точек (особей) на границе Парето, применен метод кластеризации точек на плоскости, который основан на известном алгоритме кластеризации k-средних<sup>9</sup> (k-means) и позволяет выбрать небольшое число (заранее заданное) самых значимых решений. Поскольку при многокритериальном (производительность/размер кода) подборе опций компилятора крайние точки на границе Парето представляют особый интерес (лучшее производительность и лучший размер кода), алгоритм k-means был модифицирован. В результате корректировки алгоритма, крайние точки Парето границы всегда включаются в итоговый “кластеризованный” результат в качестве центра кластеров, независимо от имеющих место скоплений.

**В главе 3** приводится метод, позволяющий автоматически улучшить эффективность выполнения оптимизаций во время стадии компоновки. Данный метод был реализован в рамках разрабатываемого в ИСП РАН инструментального средства ТАСТ.

В компиляторной инфраструктуре GCC при компиляции с использованием опции `-fplt` в объектный файл дополнительно записывается некое промежуточное представление исходного кода (GIMPLE). Эта дополнительная информация используется на стадии компоновки. В итоге имеет место тот же эффект, как если бы весь исходный код находился в одном файле.

Промежуточное представление на стадии компиляции записывается в специальную ELF-секцию объектного файла (посредством библиотеки Libelf). В некоторых случаях в исходном коде приложения в связи с некорректными объявлениями области видимости функций могут возникнуть определенные проблемы во время выполнения оптимизаций на этапе компоновки. В объектных файлах формата ELF внутренние функции в динамических библиотеках (shared object) имеют скрытую видимость (hidden visibility). Это позволяет внутренним функциям избежать вызовы через таблицу компоновки процедур (PLT), что в свою очередь позволяют компилятору лучше оптимизировать их. Но иногда функции могут экспортироваться, но одновременно использоваться внутри библиотеки. Такие функции не могут иметь “скрытую видимость”, в связи с тем, что экспортируются, но одновременно желательно избежать вызовы через PLT. В компиляторе GCC все функции по умолчанию не имеют “скрытую видимость” (экспортируются) и вызовы через PLT могут препятствовать многим межмодульным оптимизациям внутри библиотеки. Чтобы избежать этого, разработчики или пользователи приложений должны сами явно указывать те функции, которые будут экспортированы. При компиляции с GCC в заголовочные файлы можно поставить атрибут `__attribute__((visibility("default")))` в объявления тех функций, которые должны быть глобальными. Используя опцию `-fvisibility=hidden` компилятор GCC сделает все функции со скрытой видимостью, кроме тех которые были явно объявлены этим

---

<sup>9</sup>Bradley, P. S., Bennett, K. P., & Demiriz, A. (2000). Constrained k-means clustering (Technical Report MSR-TR-2000-65). Microsoft Research, Redmond, WA.

атрибутом. В GCC сейчас также поддерживается директива `#pragma` с объявлением области “видимости” функций для участка своего действия:

Существуют также другие ограничения для выполнения оптимизаций на этапе компоновки. На версиях `libtool` (<2.4) опция `-flto` не передается компоновщику (`linker`) и соответственно `libtool` нужно обновить.

Решение этих проблем вручную является трудоемким для больших приложений и для решения данной задачи был разработан метод, на основе чего реализован инструмент, позволяющий автоматически решить данную задачу. В дальнейшем данный инструмент был интегрирован в разрабатываемое в ИСП РАН набор инструментов TACT.

Опишем метод автоматической проверки и корректировки исходного кода библиотек программ, обеспечивающий улучшение эффективности выполнения оптимизаций на этапе компоновки.

На входе инструмент получает исходный код программы, после чего выполняются следующие этапы. На первом этапе исходная программа компилируется с базовым уровнем оптимизаций `-O2` и запускается на целевой машине. Одновременно проверяется версия `libtool` и при версии <2.4 завершается работа инструмента с ошибкой. Далее исходная программа компилируется с опциями `-O2 -flto` (включаются оптимизации на этапе компоновки) и запускается на целевой машине. С помощью специального скрипта вычислить те заголовочные (`header`) файлы, которые содержат объявления экспортируемых (внешних) функций. В данных заголовочных файлах добавляется явная область видимости, с помощью директивы `#pragma`:

```
#pragma GCC visibility push(default)

// участок кода

#pragma GCC visibility pop
```

На последнем этапе компилируется модифицированная исходная программа с опциями “`-O2 -fvisibility=hidden -flto`” и выходе получаем исполняемый файл (обычно динамическую библиотеку формата `.so`), содержащий более эффективный машинный код. В результате работы автоматического метода на этапе компоновки применимость оптимизаций значительно улучшается, и в зависимости от приложения, получается рост производительности и уменьшение размера бинарного кода до нескольких десятков процентов.

**В главе 4** описана реализация инструмента, разработанного в Институте системного программирования РАН на основе предложенных методов в рамках набора инструментов TACT (Toolkit for Automatic Compiler Tuning).

Программно-аппаратная система TACT состоит из управляющего модуля, выполняемом на компьютере `x86`, который используется для кросс-компиляции настраиваемых приложений, и нескольких тестовых плат (например, архитектуры `ARM`), доступных с управляющего компьютера по протоколу `SSH` и связанных таким образом,

что все устройства имеют один общий каталог через сетевую файловую систему NFS. Связь между управляющим компьютером и вычислительными модулями системы осуществляется модулем управления очередью заданий (Менеджером задач), который позволяет оптимально использовать вычислительные мощности доступных устройств. На рис. 2 продемонстрирована основная часть структуры набора инструментов TACT.

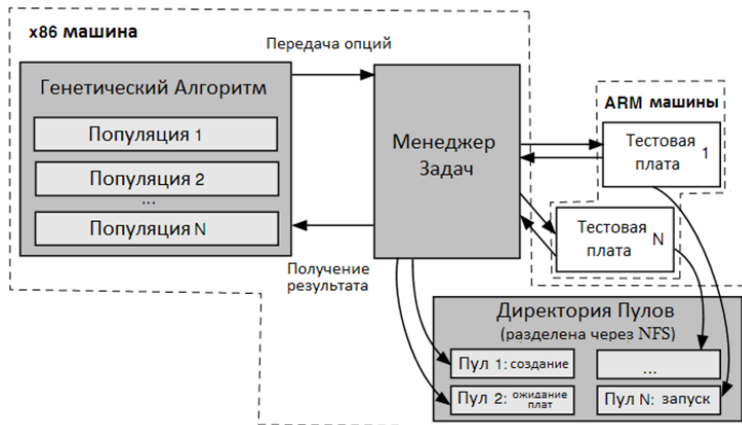


Рис.2: Общая схема работы инструмента TACT.

В рамках проекта TACT было решено интегрировать разработанный метод “автоматического поиска эффективных оптимизаций компилятора” на основе генетического алгоритма, с помощью которого модуль поиска генерирует различные наборы опций компилятора. После кросс-компиляции, используя данные опции на управляющей машине, исполняемый файл отправляется на выполнение на тестовые устройства, используя менеджер задач. Полученная на тестовых устройствах оценка производительности для заданного набора затем используется модулем эволюции для улучшения параметров компиляции в следующем поколении.

TACT использует конфигурационный файл формата XML, где задаются такие параметры, используемые во время работы инструмента, как количество поколений и популяций ГА, размер одной популяции (количество особей), размер архива текущих лучших результатов. Также конфигурационный файл позволяет пользователю выбрать стратегию работы эволюционного алгоритма (по производительности, размеру кода или по двум критериям), количественные коэффициенты мутации и скрещивания в пределах популяции, коэффициент мутации в пределах одной особи и некоторые другие параметры.

Инструмент TACT был протестирован на множестве приложений с открытым исходным кодом (таких как x264, C-Ray, libevas, и т.д.) на платформах ARM и x86. Инструмент тестировался как для компиляторной инфраструктуры GCC, так и для

LLVM, причем он применялся как для поиска наборов оптимизаций по производительности, так и для производительности и размера бинарного кода (граница Парето). На таблице 1 приведены результаты работы инструмента автоматического подбора эффективных опций и параметров компиляции на платформе ARM (процессор ARM Cortex-A9). В качестве компиляторов были выбраны GCC (gcc-4.9.2) и LLVM (llvm-2.4). Как видим для всех выбранных тестовых приложений производительность полученная в результате работы инструмента значительно превосходит результат с базовым набором оптимизаций -O2.

Таблица 1. Результаты работы инструмента TACT на некоторых тестах

Название теста	GCC (-O2), сек.	GCC (tuned)	Ускорение, %	LLVM (-O2)	LLVM (tuned)	Ускорение, %
x264	7	5.32	24	6,5	5,26	14,1
libevas	14.2	9.9	30	15	13.7	13
C-Ray	5.07	4.7	6	4,92	4,84	1,6

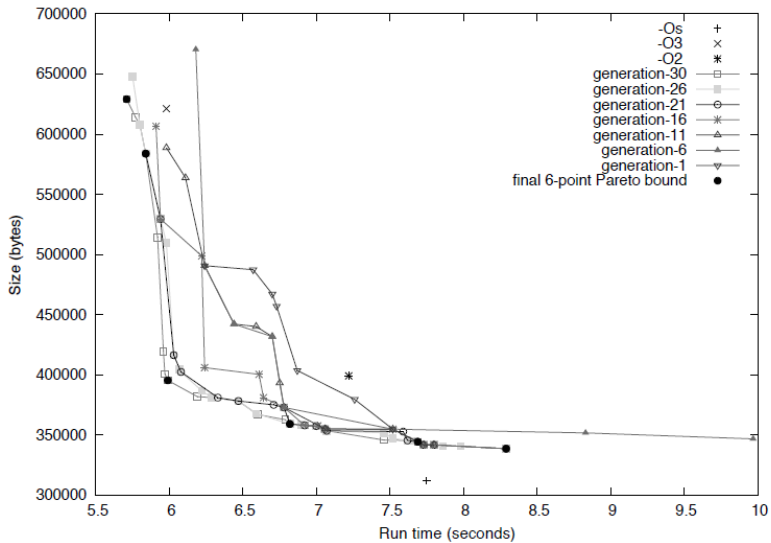


Рис.3: Улучшение графика Парето с 1-е по 30-е поколение для оптимизации приложения x264 по производительности и размеру кода соответственно.

Инструмент TACT был также протестирован для поиска эффективных оптимизаций по двум критериям (производительность и размер бинарного кода). Результаты получены в виде границ Парето. Рассмотрим результаты работы инструмента на примере приложения x264. На Рис. 3 приводиться пример работы метода автоматического

подбора оптимизаций по двум критериям (производительность и размер бинарного кода) на примере приложения x264, также результаты при базовых уровнях оптимизаций компилятора (-Os, -O3, -O2). В работе использовались компилятор GCC (gcc-4.9.2) и архитектура ARMv7. В зависимости от требований пользователя можно выбрать результаты как оптимальные по производительности (левую верхнюю точку) так и по размеру кода (правую нижнюю точку), можно также выбрать набор оптимизаций, при котором такой же размер бинарного кода обеспечивает ускорение производительности на 12% по сравнению с -O2.

Был проведен сравнительный анализ между результатами разработанного нами инструмента TACT и некоторых развитых инструментов (рассмотренных в первой главе) автоматического подбора опций компилятора. Поскольку некоторые из этих инструментальных средств не имеют открытого исходного кода, было решено использовать результаты, взятые из статей описывающих их работу.

Для сравнения результатов OpenTuner и TACT, были взяты тестовые примеры (raytracer, matrix\_multiply, tga\_ga), на которых авторы инструмента проводили эксперименты. Был использован такой же компилятор gcc-4.9.0, и та же целевая архитектура x86\_64. Результаты приведены на таблице 2. На всех рассмотренных тестовых примерах инструмент TACT показывает лучший результат.

В описании инструмента PEAK проводятся эксперименты на тестах из набора SPEC CPU2000 INT, используя компилятор gcc-4.3.3 и архитектуру x86. На таблице 3, приводятся сравнительные результаты инструментов TACT и PEAK на некоторых тестах из набора SPEC CPU2000 INT (с базой данных Train) для компилятора GCC и архитектуры x86.

*Таблица 2. Результаты ускорения производительности с инструментами TACT и OpenTuner по сравнению с базовым набором –O2.*

Название теста	Ускорение opentuner	Ускорение TACT
Raytracer	~15%	~26%
matrix_multiply	~2.8 раза	~4,3 раза
tga_ga	~19%	~34%

*Таблица 3. Результаты ускорения производительности с инструментами TACT и PEAK по сравнению с базовым набором –O2.*

Название теста	Ускорение произв. PEAK	Ускорение произв. TACT
Crafty	7.5%	9.9%
bzip2	2%	1.8%
Eon	9%	21%
Gap	3%	7%
Geomean	4.2%	10.2%

Количество запусков для получения существенного улучшения сгенерированного кода, зависит как от самого приложения и выбранного компилятора, так и от выбранных оптимизаций компилятора и конфигураций настройки (количество поколений и популяций, размера конфигураций в одной популяции, коэффициента мутации и т.д.).

Графики на рис. 4 показывают изменения максимального и среднего улучшения производительности для каждого поколения. На вертикальной оси продемонстрирована текущая производительность по сравнению с базовым уровнем оптимизации (-0,2), учитывая, что "1" соответствует максимальной производительности на последнем поколении. В этих графиках не учтены лучшие наборы оптимизаций, которые были найдены на предыдущих поколениях и хранились в архивах, а также те наборы, на которых произошли ошибки компиляции или запуска.

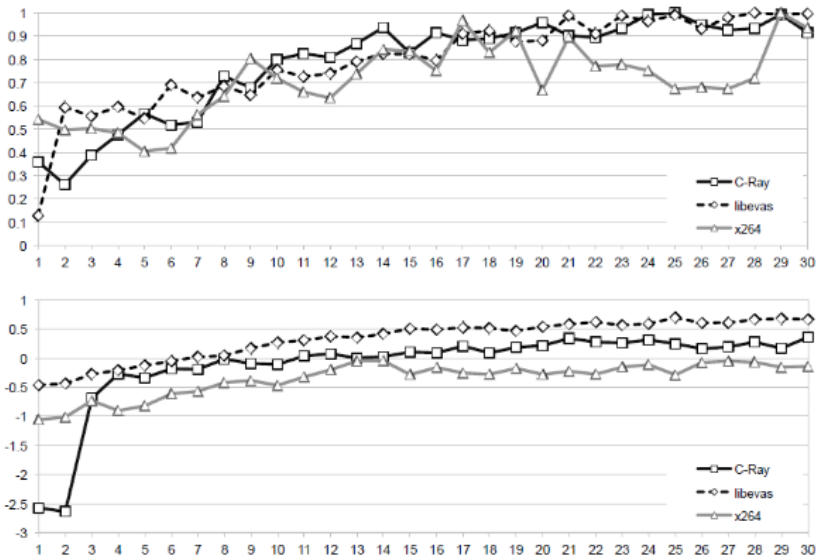


Рис. 4: Лучший (верхнее) и средний (нижнее) результаты улучшения производительности для каждого поколения, 0 соответствует -0,2, 1 - максимальной производительности, достигнутой инструментом

В рамках набора инструментов TACT был также реализован инструмент на основе метода автоматической проверки и корректировки исходного кода библиотеки, для улучшения работы оптимизаций на этапе компоновки. Инструмент был протестирован на множестве тестовых приложений, таких как SQLite, LibEvas, CLucene, Lame, и т.д.

В таблицах 4 и 5 приводятся результаты работы инструмента на некоторых популярных библиотеках, демонстрирующие соответственно улучшение размера полученного бинарного кода и производительности. Второй, третий и четвертые столбцы

демонстрируют соответственно результаты с базовым уровнем -O2, с включением -flto и итоговому результату после работы инструмента.

Таблица 4. Размер бинарного кода в байтах при применении инструмента

Библиотека	-O2	-flto	-flto -fvisibility	улучшение в %
Clucene	733924	615475	<b>567807</b>	<b>22.63%</b>
SQLite	343938	344010	<b>300676</b>	<b>12.57%</b>
Evas	27159	27114	<b>25036</b>	<b>7.81%</b>
Lame	197375	197325	<b>184604</b>	<b>6.47%</b>
x264	438867	438950	<b>427106</b>	<b>2.7%</b>

Таблица 5. Производительность в секундах при применении инструмента

Библиотека	-O2	-flto	-flto -fvisibility	улучшение в %
SQLite	8.1	8.2	6.86	15.3%
Lame	10.17	10.17	10.17	0%
x264	30.14	30.14	30.08	0%

**Заключение.** В работе были изложены разработанные методы поиска эффективных оптимизаций компилятора (как для одной целевой функции, так и по нескольким критериям) для данного приложения и платформы. Также разработан метод автоматической проверки и корректировки исходного кода библиотеки, обеспечивающий улучшения эффективное выполнение оптимизаций на этапе компоновки. Данные методы интегрированы в инструментальную инфраструктуру TACT.

## ОСНОВНЫЕ РЕЗУЛЬТАТЫ ДИССЕРТАЦИОННОЙ РАБОТЫ

- Разработан метод автоматического подбора эффективных оптимизаций компилятора для конкретного приложения на заданной платформе. Данный метод поддерживает также автоматический подбор эффективных наборов оптимизаций на основе Парето-доминирования. Исследованы и оценены влияния конфигураций генетического алгоритма на процесс эволюции [1,2,3,5].
- Разработан метод автоматической проверки и корректировки исходного кода библиотек программ, обеспечивающий улучшение эффективности выполнения оптимизаций на этапе компоновки [4].
- Разработанные методы были реализованы в рамках инструментальной инфраструктуры TACT учитывая особенности компиляторов GCC и LLVM и применимости концепций генетического алгоритма [1,3,4,5].

## СПИСОК РАБОТ ОПУБЛИКОВАННЫХ ПО ТЕМЕ ДИССЕРТАЦИИ

1. Мамикон Варданын. Метод автоматического подбора наборов эффективных оптимизаций компилятора по нескольким критериям на основе Парето-доминирования. Вестник Инженерной академии Армении (ВИАА). 2015, том 12, №2.
2. Роман Жуйков, Дмитрий Плотников, Мамикон Варданын. Автоматическая настройка оптимизационных преобразований компилятора GCC для платформы ARM. Труды Института системного программирования РАН. 2012, том 22, с. 49-66.
3. Dmitry Plotnikov, Dmitry Melnik, Mamikon Vardanyan, Ruben Buchatskiy. Automatic Tuning of Compiler Optimizations and Analysis of their Impact. Procedia Computer Science, ICCS 2013, том 18, с. 1312–1321.
4. Дмитрий Мельник, Шамиль Курмангалеев, Арутюн Аветисян, Андрей Белеванцев, Дмитрий Плотников, Мамикон Варданын. Оптимизация приложений для заданных статических компиляторов и целевых архитектур: методы и инструменты. Труды Института системного программирования РАН. 2012, том 26, с. 343-356.
5. Dmitry Plotnikov, Dmitry Melnik, Mamikon Vardanyan, Ruben Buchatskiy, Roman Zhuykov. An Automatic Tool for Tuning Compiler Optimizations. 2013 Computer Science and Information Technologies (CSIT), с. 1-7.

## Resume

Mamikon Vardanyan

### A METHOD FOR AUTOMATED COMPILER TUNING USING MULTI-OBJECTIVE SEARCH BASED ON PARETO-DOMINANCE

Modern compilers typically include hundreds of different optimizations that are working on different stages of compilation thus affecting the quality of the generated code. Wherein the set of optimizations (compiler options), during which the most optimal object code is generated, depends on both the target machine architecture, and the specifics of the selected application. Typically, developers and users of applications use default compiler options (such as -O2 in the GCC and LLVM), defining the basic set of optimizations. Such sets are determined by the developers of compilers for popular test suites (usually, SPEC) and ensure optimal code compilation on average for the given test suite. It is well known that for various apps the "best" set of optimizations can vary, moreover, depending on the target platform and for a particular application can be enhanced its performance by several dozen percent only by the proper selection of the compiler parameters. The problem is complicated also by the fact that the optimality criteria along with performance, as well can be code size, power consumption, compile time, etc. Besides, often it is required to optimize simultaneously for multiple criteria (e.g. performance and code size). In some cases criteria may not be compatible with each other and then we have to use some trade-offs.

Selection of compiler options and compilation parameter values for the specific application requires a deep knowledge of the infrastructure of the compiler and features of the target platform. For regular developers and users of applications the automated methods for selecting the parameters of compilation are of great interest. These methods are implemented in the existing instruments, but in many instruments, some useful features such as multi-criteria search, support of parallel cross-compile and run on the target platform, etc., are not implemented.

To improve the efficiency of the application can also be done by correcting the source code. For example the correction of the library's source code can improve the applicability of a number of inter-module optimization (such as an inlining between the modules) of the compiler at link time, since many libraries (especially older ones) were not designed for compiler with link-time capabilities. Some libraries contain inappropriate declarations in the scopes of its functions, which leads to the fact that all the global functions of the library are automatically exported as its interface. This prevents the implementation of inter-module optimizations at link time, since these functions can be called from outside the runtime code. Correction of respective declarations in libraries can significantly enhance the applicability of optimizations at link time and thus improve the performance and reduce code size.

The solution of these tasks is a major problem, since it is associated with significant labor costs and requires high skills. It is necessary to develop appropriate instruments of automatization. At the Institute for System Programming of Russian Academy of Sciences is

being developed a set of tools named TACT (Toolkit for Automatic Compiler Tuning), providing maximum utilization of compiler capacities for optimization of a specific application under the chosen target architecture.

The aim of the thesis is to develop a method to automatically select the compiler options and values of its parameters, ensuring optimization of the target program, in accordance with the selected criterion (including the possibility of optimization by multiple criteria) for a given application and target platform. In addition, it is necessary to develop a method to automatically check and correct the source code of libraries that provides execution optimizations at link time. The developed methods are being implemented within the frameworks of the instrumental tools of TACT.

### **Main results of the work**

- A method for automatic search of compiler options and values of its parameters has been developed, providing generation of effective optimal target code for a particular application on a given platform in accordance with a predetermined criterion. This method also supports multi-objective optimizations based on Pareto-dominance [1, 2, 3, 5].
- A method for automatic testing and correction of the source code of libraries has been developed, effectively implementing optimizations at link time [4].
- An instrumental tool has been implemented based on the proposed methods using a genetic algorithm. In addition, it has shown how parameters of the genetic algorithm affect the tuning results achieved with the tool [1, 3, 4, 5].

## Ամփոփում

Վարդանյան Մամիկոն Աշոտի

ԿՈՄՊԻԼՅԱՏՈՐԻ ՄԻ ՔԱՆԻ ՉԱՓՈՐՈՇԻՉՆԵՐՈՎ ԱՐԴՅՈՒՆԱՎԵՏ  
ՕՊՏԻՄԻԶԱՅԻԱՆԵՐԻ ՓՆՏՐՄԱՆ ԱՎՏՈՄԱՏԱՑՎԱԾ ՄԵԹՈԴ՝ ՇԻՄՎԱԾ  
*ՊԱՐԵՏՈ ԴՈՄԻՆԱՆՏՈՒԹՅԱՆ ՎՐԱ*

Ժամանակակից կոմպիլյատորները սովորաբար պարունակում են հարյուրավոր տարբեր օպտիմիզացիաներ, որոնք աշխատելով կոմպիլյացիայի տարբեր փուլերում, ազդում են գեներացվող մեքենայական կոդի որակի վրա: Ընդ որում՝ օպտիմիզացիաների հավաքածուն (տարբերակը), որի ժամանակ գեներացվում է առավել օպտիմալ օբյեկտային կոդ, կախված է ինչպես մեքենայի նպատակային ճարտարապետությունից, այնպես էլ ընտրված ծրագրային հավաքածուի առանձնահատկություններից: Սովորաբար ծրագրային համակարգեր մշակողներն ու նրանցից օգտվողները կիրառում են կոմպիլյատորի օպտիմիզացիաների բազային հավաքածուները (օրինակ “-O2” GCC և LLVM կոմպիլյատորների համար): Այդ հավաքածուները որոշվում են կոմպիլյատորներ մշակողների կողմից հայտնի թեստային հավաքածուների համար (որպես կանոն SPEC)՝ ապահովելով տվյալ հավաքածուի շրջանակներում միջինում օպտիմալ կոդի գեներացիա:

Ինչպես հայտնի է, ծրագրային տարբեր հավաքածուների համար լավագույն օպտիմիզացիաների հավաքածուները կարող են տարբերվել այդ թվում՝ կախված նպատակային պլատֆորմայից, և կոնկրետ ծրագրային հավաքածուի համար տասնյակ տոկոսներով կարելի է մեծացնել արտադրողականությունը, միայն թե պետք է ճիշտ ընտրված լինեն կոմպիլյատորի պարամետրերը: Խնդրի բարդությունը պայմանավորված է նրանով, որ բացի արտադրողականությունից՝ օպտիմալության չափորոշիչներ կարող են լինել նաև կոդի չափը, էներգաօգտագործումը, կոմպիլյացիայի ժամանակը և այլն: Բացի այդ՝ հաճախ պահանջվում է օպտիմիզացնել՝ միաժամանակ ըստ մի քանի չափորոշիչների (օրինակ՝ արտադրողականության և կոդի չափի): Որոշ դեպքերում չափորոշիչները կարող են լինել անհամատեղելի, և այդ դեպքում ստիպված կիրառվում են համատեղման ընդունելի մոտեցումներ:

Ծրագրային կոնկրետ հավաքածուի համար կոմպիլյատորի տարբերակների և կոմպիլյացիայի պարամետրերի արժեքների ընտրությունը պահանջում է խորը գիտելիքներ կոմպիլյատորի ենթակառուցվածքի և նպատակային պլատֆորմայի առանձնահատկությունների պարագայում: Սովորական մշակողների և ծրագրային հավաքածուներից օգտվողների համար ակտուալ են կոմպիլյացիայի պարամետրերի ընտրության ավտոմատ մեթոդները: Այդպիսի մեթոդներ իրականացված են ծրագրային համապատասխան գործիքներում, բայց դրանց մի մասում բացակայում են որոշ օգտակար հատկություններ, որոնցից են, օրինակ, բազմաչափորոշիչային փնտրումը, զուգահեռ կրոսս-կոմպիլյացիայի իրականացումը նպատակային պլատֆորմի վրա և այլն:

Գրադարանի աշխատանքի արդյունավետությունը հնարավոր է նաև լավացնել ծրագրային կոդի ճշգրտման միջոցով, որը կարող է բարձրացնել միջմոդուլային մի շարք օպտիմիզացիաների կիրառելիությունը (օրինակ՝ inlining մոդուլների միջև), քանի որ շատ գրադարաններ (մանավանդ ավելի հին) չունենին կոմպիլյատորի այդպիսի հնարավորություններ:

Որոշ գրադարաններ պարունակում են իրենց ֆունկցիաների տեսանելիության շրջանակի ոչ պատշաճ հայտարարություններ, որոնք հանգեցնում են նրան, որ գրադարանի բոլոր գլոբալ ֆունկցիաները ավտոմատ արտահանվում են որպես նրա ինտերֆեյս: Դա խոչընդոտում է միջմոդուլային օպտիմիզացիաների կատարմանը ներկցման փուլում, քանի որ այդ ֆունկցիաները կարող են ներմուծված լինել դրսի գրադարանների կոդմիջ կոդի կատարման ժամանակ:

Գրադարաններում համապատասխան հայտարարությունների ճշգրտումը հնարավորություն է տալիս էականորեն բարձրացնելու օպտիմիզացիայի կիրառությունները ներկցման փուլում՝ համապատասխանորեն լավացնելով արտադրողականությունը ու փոքրացնելով կոդի չափը:

Թվարկված խնդիրների լուծումը համարվում է լուրջ խնդիր, քանի որ կապված է էական դժվարությունների հետ և պահանջում է բարձր որակավորում: Անհրաժեշտ է մշակել համապատասխան ավտոմատիզացման միջոցներ: ՌԴ գիտությունների ակադեմիայի Սիստեմային ծրագրավորման ինստիտուտում մշակվում է TACT (Toolkit for Automatic Compiler Tuning) գործիքների հավաքածուն, որն ապահովում է կոմպիլյատորի մաքսիմալ հնարավորությունների օգտագործումը կոնկրետ ծրագրային համախմբի օպտիմիզացիայի համար՝ ըստ ընտրված նպատակային ճարտարապետության:

### **Ատենախոսության հիմնական արդյունքները**

- Կոմպիլյատորի տարբերակների և նրա պարամետրերի արժեքների ավտոմատ ընտրման մեթոդի մշակումը, որը կոնկրետ ծրագրային հավաքածուի համար նշված պլատֆորմայում ապահովում է նպատակային կոդի արդյունավետ գեներացումը՝ համապատասխան տրված օպտիմալության չափորոշիչների: Տվյալ մեթոդը ապահովում է նաև բազմաչափորոշիչ օպտիմիզացիա Պարետո դոմինանտության հիման վրա [1, 2, 3, 5]:
- Գրադարանների սկզբնական կոդի ավտոմատ ստուգման և ճշգրտման մշակված մեթոդը ապահովող օպտիմիզացիայի արդյունավետ իրականացումը ներկցման փուլում [4]:
- Գործիքային միջոցների իրականացումը՝ հիմնված առաջարկված մեթոդների վրա և գենետիկ ալգորիթմի կիրառմամբ: Դիտարկված են գործիքի տարբեր ձևեր՝ ընտրությունների ազդեցությունը արդյունքում ստացվող կոդի որակի վրա պարզելու նպատակով [1, 3, 4, 5]: