DOI 10.37882/2223–2966.2021.05–2.07

ОБЗОР ОСНОВНЫХ ИНСТРУМЕНТОВ ОПТИМИЗАЦИИ В ПРОГРАММИРОВАНИИ

OVERVIEW OF THE MAIN OPTIMIZATION TOOLS IN PROGRAMMING

A. Bazarova

Summary. The article provides an overview of the main optimization tools in programming. In the research process, special attention is paid to the latest optimization methods, which include interprocedural analysis, direct analysis of global and static variables, profile optimization, embedding, and complex branch optimization. We also studied the advantages and disadvantages of the PVS-Studio analyzer and the Gurobi optimizer. Particular emphasis is placed on the tools for analyzing the source code, their capabilities and limitations are indicated, as well as selection criteria. In addition, an analysis was carried out of tools that are among the top three and allow high-quality analysis and verification of Java code.

Keywords: optimization, tools, methods, analysis, Java, security.

Базарова Анна Максимовна
Старший преподаватель, Ухтинский государственный технический университет
anna_sh94@inbox.ru

Аннотация. В статье проведен обзор основных инструментов оптимизации в программировании. В процессе исследования отдельное внимание уделено новейшим методам оптимизации, которые включают в себя межпроцедурный анализ, прямой анализ глобальных и статических переменных, оптимизацию по профилю, встраивание, комплексную оптимизация ветвей. Особый акцент сделан на инструментах анализа исходного кода, обозначены их возможности и ограничения, а также критерии отбора. Кроме того, проведен анализ инструментов, которые входят в тройку лидеров и позволяют на качественно новом уровне проводить анализ и проверку кода для Java.

Ключевые слова: оптимизация, инструменты, методы, анализ, Java, безопасность.

В последние годы появилось новое поколение передовых встраиваемых продуктов. Эти продукты, начиная от лазерных принтеров до сетевых маршрутизаторов и видеоигр, все чаще требуют огромных уровней вычислительной мощности, которые могут быть достигнуты только за счет использования высокоскоростных микропроцессоров RISC и CISC [1]. Несмотря на то, что разработчики теперь могут выбирать из широкого спектра недорогих, высокопроизводительных процессоров, требования рынка к лучшим, более быстрым продуктам заставляют искать высокую производительность различными способами. Однако, поскольку все разработчики имеют одинаковый доступ к новейшим высокоскоростным аппаратным компонентам, сложно добиться преимущества только с их помощью. В результате конкурентные преимущества продукта разработчики все чаще ищут и достигают за счет усовершенствования и развития программного обеспечения.

Смещение акцента на более высокую производительность программного обеспечения резко повысило важность инструментов разработки программного обеспечения. С повсеместным использованием языков высокого уровня, таких как JAVA, C# для разработки программного обеспечения, технология и методы оптимизации начинают играть более важную роль, чем когда-либо, в оказании помощи разработчикам на пути достижения целей проектирования.

Оптимизация может происходить на разных уровнях, в зависимости от того, насколько она близка к машинному коду. В веб-разработке есть возможность выполнять только оптимизацию более высокого уровня. Также код может быть оптимизирован и на архитектурном уровне с помощью интеллектуальных шаблонов проектирования, на уровне исходного кода оптимизация может быть достигнута благодаря применению передовых методов кодирования и соответствующих инструментов. Кроме того, можно улучшить производительность приложения, внедрив руководства по стилю кодирования в рабочий процесс [2].

Таким образом, актуальность обозначенных проблем и вопросов обуславливает выбор темы данной статьи.

Возможности и перспективы развития методов и инструментов оптимизации в ИТ-сфере и, в частности, в разработке программного кода входят в круг научных интересов Dixie M Hisley, Teixeira, Thiago SFX; Gropp, William; Гнездиловой Н.А., Щучка Т.А., Акимовой Е.Н.

Однако повышенные требования в производительности, быстродействию и эффективности разрабатывае-

мых приложений предопределяют необходимость регулярной актуализации имеющихся наработок и решений.

Цель исследования заключается в проведении обзора основных инструментов оптимизации в программировании.

Итак, как упоминалось ранее, более высокая производительность достигается за счет перехода на более быстрые процессоры, особенно архитектуры RISC. Процессоры RISC обеспечивают большую производительность благодаря своей способности очень быстро выполнять относительно простые инструкции. Поскольку пропускная способность процессора зависит от постоянного поступления исполнительной инструкции, производительность может сильно ухудшаться, если ему приходится часто ждать, пока инструкции или данные будут получены из внешней памяти или записаны во внешнюю память. Роль компилятора в минимизации таких нежелательных событий имеет решающее значение для оптимальной производительности [3].

По мере того, как новейшие процессоры CISC становятся все более RISC-подобными, а рынок требует более быстрого и плотного кода, появился новый класс методов оптимизации компилятора. Во многих случаях эти новые оптимизации включают сложные методы анализа программ, которые значительно расширили возможности применения хорошо зарекомендовавших себя оптимизаций, таких как распространение констант или копий. Фактически, эти новые методы анализа позволяют компилятору анализировать исходный код C или C ++ высокого уровня, чтобы более разумно комбинировать оптимизацию для достижения максимального эффекта.

Новейшие инструменты и методы оптимизации включают в себя:

1. Межпроцедурный анализ
2. Прямой анализ глобальных и статических переменных
3. Оптимизация по профилю
4. Встраивание
5. Комплексная оптимизация ветвей

Рассмотрим кратко эти методы.

Межпроцедурный анализ

Как известно, производительность может серьезно снизиться, если процессору придется слишком часто взаимодействовать с основной памятью. Самый эффективный способ уменьшить доступ к памяти — минимизировать операции загрузки и сохранения, сохраняя как можно больше переменных в регистрах. Одним из особенно эффективных методов устранения ненужных загрузок и сохранений является межпроцедурный анализ, или IPA. IPA работает, проверяя фактический исходный код функции всякий раз, когда она вызывается. Это позволяет точно определить, будет ли функция обращаться к статическим переменным, находящимся в данный момент в регистрах.

Прямой анализ глобальных и статических переменных

Программы, написанные для 8- и 16-разрядных архитектур, часто широко используют глобальные и статические переменные, а не локальные. Это было сделано из соображений производительности, поскольку такие архитектуры обычно предоставляют очень ограниченное количество регистров. К сожалению, когда такой код выполняется на современных архитектурах с большим количеством регистров, он может существенно снизить производительность, заставляя процессор обращаться к памяти каждый раз, когда он считывает или обновляет значение переменной. Анализ динамических переменных позволяет компилятору определить, какие переменные следует поместить в регистры. Хотя этот анализ часто используется компиляторами с локальными переменными, необходимо принимать во внимание, что его выполнение для статических и глобальных переменных намного сложнее.

Оптимизация по профилю

Суть данного метода сводится к тому, что компилятор генерирует версию приложения, содержащую минимальные инструменты, которые записывают информацию о поведении программы в файл или буфер памяти, с последующей загрузкой на главный компьютер. Затем эта информация возвращается компилятору при следующей компиляции приложения. Данные профилирования позволяют принимать более разумные решения по оптимизации, основанные на реальных данных времени выполнения, а не на предположениях. Например, распределение регистров базируется на фактическом количестве раз использования переменной, и предложения if-then-else, которые меняются местами, если первая часть выполняется чаще. Компилятор также может использовать данные профилировщика для более разумного комбинирования оптимизаций. Хорошим примером является то, что при некоторых оптимизациях, таких как развертывание или встраивание цикла, скорость зависит от размера. Не имеет смысла применять развертывание цикла к редко выполняемым циклам, если минимальный размер кода также является важной целью. Однако, не подлежит сомнению тот факт, что эти оптимизации следует применять в отношении как можно более часто выполняемых участков кода.

Встраивание

Встраивание функций повышает производительность, заменяя вызов функции телом самой функции. Это устраняет накладные расходы на переход к подпрограмме и возвращение из нее.

Комплексная оптимизация ветвей

Поскольку повторное использование кода становится критическим фактором для более быстрой доставки продуктов на рынок, очень важно писать хорошо структурированный код. Этот инструмент оптимизации позволяет переписывать код, содержащий переходы и переходы в условные переходы, где можно сравнивать результат.

В процессе исследования программных продуктов, которые дают возможность оптимизировать разрабатываемое приложение, особое внимание, по мнению автора, необходимо уделить инструментам анализа исходного кода (SAST), которые предназначены для анализа непосредственно самого первоначального кода или его скомпилированных версий с целью помочь найти недостатки безопасности.

Некоторые инструменты на сегодняшний день начинают перемещаться в IDE. Для тех типов проблем, которые могут быть выявлены непосредственно на этапе разработки самого программного обеспечения, эти инструменты являются очень действенными, поскольку позволяют обеспечивает непосредственную обратную связь с разработчиком, в результате чего улучшения могут быть внесены в код во время его разработки [6]. Такая непосредственная обратная связь является очень полезной, особенно в сравнении с выявлением уязвимостей на более позднем этапе цикла разработки.

Сильными сторонами и преимуществами этих инструментов являются следующие.

1. Хорошо масштабируются — могут быть запущены на большом количестве программного обеспечения, а также могут запускаться многократно.
2. Полезны для обнаружения проблем и недостатков, которые такие инструменты могут автоматически найти с высокой степенью уверенности, например, переполнение буфера, ошибки внедрения SQL и т.д.
3. Детализированные выводы для разработчиков — выделяются точные исходные файлы, номера строк и даже части строк, которые затронуты.

Недостатки

1. Многие типы уязвимостей достаточно трудно обнаружить в автоматическом режиме, например, проблемы аутентификации, проблемы контроля доступа, небезопасное использование криптографии и т.д. Существующий на сегодняшний день аппаратный уровень разработки позволяет таким инструментам автоматически находить относительно небольшой процент недостатков приложений. Однако инструменты этого типа постоянно развиваются и улучшаются.
2. Большое количество ложных срабатываний.
3. Часто не удается найти проблемы с конфигурацией, так как они не представлены в коде.
4. Трудно «доказать», что выявленная проблема безопасности является реальной уязвимостью.
5. Многие из этих инструментов испытывают трудности с анализом кода, который невозможно скомпилировать. Аналитики часто не могут скомпилировать код, потому что у них нет нужных библиотек, всех инструкций по компиляции, всего кода и т.д.

Принимая во внимание достоинства и недостатки в процессе выбора инструментов анализа исходного кода целесообразно использовать следующие критерии.

1. Поддерживание используемого языка программирования.
2. Перечень уязвимостей, которые конкретный инструмент может обнаружить.
3. Точность получаемых оценок, наличие статистики по ложноположительным и ложноотрицательным оценкам. Также необходимо выяснить есть ли у инструмента оценка OWASP Benchmark.
4. Понимание используемых библиотек / фреймворков.
5. Необходимость наличия полного набора исходников.
6. Возможность работать с двоичными файлами вместо исходного кода.
7. Интеграция в IDE разработчика.
8. Сложность в настройке, использовании.
9. Способность работать непрерывно и автоматически.
10. Стоимость лицензии на инструмент (некоторые из них продаются на пользователя, на организацию, на приложение, на каждую строку анализируемого кода, также следует помнить, что лицензии на консультации часто отличаются от лицензий для конечных пользователей.)

В таблице 1 представлен обзор некоторых инструментов анализа исходного кода.

И в завершении исследования рассмотрим инструменты, которые входят в тройку лидеров и позволяют проводить широкий анализ и проверку кода для Java.

Таблица 1. Краткое описание некоторых специализированных программ, предназначенных для оптимизации программного кода

Имя/ссылка	Владелец	Лицензия	Языки	Особенности
.NET Security Guard		Открытый исходный код или бесплатно	.NET, C\#, VB.net	
42Crunch		Платный доступ	REST API	Платформа безопасности, которая включает аудит безопасности (SAST), динамическое сканирование соответствия, защиту во время выполнения и мониторинг
APIsecurity.io Security Audit		Открытый исходный код или бесплатно	OpenAPI / Swagger	Онлайн-инструмент для статического анализа безопасности файлов OpenAPI / Swagger
Agnitio		Открытый исходный код или бесплатно	ASP, ASP.NET, C\#, Java, Javascript, Perl, PHP, Python, Ruby, VB.NET, XML	
AppScan Source	HCL Software	Платный доступ		Решение для статического тестирования безопасности приложений, которое помогает выявлять уязвимости на ранних этапах жизненного цикла разработки, понимать их происхождение и потенциальное воздействие, а также устранять проблему
AppScan on Cloud	HCL Software	Платный доступ		Набор для тестирования безопасности облачных приложений, для выполнения SAST, DAST, IAST и SCA в веб-приложениях и мобильных приложениях.
Application Inspector	Positive Technologies	Платный доступ	Java, C \ #, PHP, JavaScript, Objective C, VB.Net, PL / SQL, T-SQL и др.	Объединяет SAST, DAST, IAST, SCA, анализ конфигурации и другие технологии, в т.ч. уникальная абстрактная интерпретация; имеет возможность генерировать тестовые запросы (эксплойты) для проверки обнаруженных уязвимостей во время анализа SAST
Bandit		Открытый исходный код или бесплатно	Python	Bandit — это комплексный сканер уязвимостей исходного кода для Python
Beyond Security beSOURCE	Beyond Security	Платный доступ		Статическое тестирование безопасности приложений (SAST) раньше проводилось отдельно от проверок качества кода, что приводило к ограниченному влиянию и ценности. beSOURCE заботится о качестве безопасности кода приложений и, таким образом, интегрирует SecOps в DevOps
BlueClosure BC Detect	BlueClosure	Платный доступ	Клиентский JavaScript	
Brakeman		Открытый исходный код или бесплатно	Ruby on Rails.	Brakeman — это сканер уязвимостей с открытым исходным кодом, специально разработанный для приложений Ruby on Rails.
CAST AIP		Платный доступ	Поддерживает более 30 языков	Выполняет статический и архитектурный анализ для выявления множества типов проблем безопасности.
Checkmarx Static Code Analysis		Открытый исходный код или бесплатно	Apex, ASP.NET, C\#, C++, Go, Groovy, HTML5, Java, JavaScript, JSP,.NET, Objective-C, Perl, PHP, PL/SQL, Python, Ruby, Scala, Swift, TypeScript, VB.NET, Visual Basic 6	

Таблица 1 (продолжение). Краткое описание некоторых специализированных программ, предназначенных для оптимизации программного кода

Имя/ссылка	Владелец	Лицензия	Языки	Особенности
Codacy		Платный доступ	Python, Ruby, Scala, Java, JavaScript и другие.	Предлагает шаблоны безопасности для разных языков. Интегрируется с такими инструментами, как Brakeman, Bandit, FindBugs и другими (бесплатно для проектов с открытым исходным кодом)
CodeScan Cloud		Платный доступ	Apex, Visualforce и Lightning.	Инструмент для обеспечения качества кода SaaS, ориентированный на Salesforce, использующий горячие точки безопасности SonarQube OWASP для обеспечения видимости безопасности на проприетарных языках Apex, Visualforce и Lightning.
CodeSec		Открытый исходный код или бесплатно	C, C++, C\#, Java, JavaScript, PHP, Kotlin, Lua, Scala, TypeScript, Android	
CodeSonar		Платный доступ	C, C ++, Java и C#	Сопоставляет 10 основных уязвимостей OWASP
CodeSonar		Открытый исходный код или бесплатно	C, C++, Java	
Contrast Assess		Платный доступ		Contrast обеспечивает безопасность кода, фактически не выполняя статический анализ. Contrast проводит интерактивное тестирование безопасности приложений (IAST), сопоставляя код времени выполнения и анализ данных. Он предоставляет результаты на уровне кода, фактически не полагаясь на статический анализ
Coverity		Открытый исходный код или бесплатно	Android, C#, C, C++, Java, JavaScript, Node.js, Objective-C, PHP, Python, Ruby, Scala, Swift, VB.NET	
Dawnscanner		Открытый исходный код или бесплатно	Ruby	Dawnscanner — это анализатор исходного кода безопасности с открытым исходным кодом для Ruby, поддерживающий основные фреймворки MVC, такие как Ruby on Rails, Padrino и Sinatra. Он также работает с веб-приложениями, написанными не на Ruby
Deep Dive		Открытый исходный код или бесплатно	Java	Инструмент анализа байтового кода для обнаружения уязвимостей в развертываниях Java (EAR, WAR, JAR)
DeepSource	DeepSource Corp.	Платный доступ	Python, Go, Ruby и JavaScript.	DeepSource помогает автоматически находить и исправлять проблемы в коде во время его проверки, такие как риски ошибок, анти-шаблоны, проблемы с производительностью и недостатки безопасности.
DerScanner	DerScanner Ltd.	Платный доступ	Более 30 языков	Способен выявлять уязвимости и бэкдоры (недокументированные функции) путем анализа исходного кода или исполняемых файлов, не требуя отладочной информации
ECG	VoidSec	Коммерческий		Инструмент статического анализа исходного кода SaaS TCL, способный обнаруживать реальные и сложные уязвимости безопасности в исходном коде TCL / ADP. Обнаруженные уязвимости будут сопоставлены с 10 основными уязвимостями OWASP

Visual Assist — это хорошо известное расширение производительности для Visual Studio и Unreal Engine 4. Оно написано разработчиками на C ++ для разработчиков на C ++ и имеет множество функций для повышения производительности и отладки кода.

Функции:

1. Быстрая навигация — этот инструмент позволяет легко перемещаться к любому файлу, методу, символу или ссылке в проектах и решениях.
2. Проверка и модернизация кода — возможность проверять код на наличие конкретных проблем с качеством, а также модернизировать старый код.
3. Улучшение кода без написания новой функциональности — позволяет упрощать код, улучшать его читаемость, возможность сделать его расширяемым без изменения внешнего поведения.
4. Исправление кода — Visual Assist позволяет исправляет ошибки по мере их совершения.
5. Индивидуальная поддержка.

Embold — это инструмент проверки кода, который анализирует исходный код по 4 параметрам: проблемы с кодом, проблемы с дизайном, показатели и дублирование. Он выявляет проблемы, влияющие на стабильность, надежность, безопасность и ремонтопригодность.

Функции:

1. Запатентованные антишаблоны показывают структурные проблемы на уровне классов, функций и методов в коде, которые негативно влияют на ремонтопригодность.
2. Функция Embold Score помогает точно определить области риска и расставить приоритеты для наиболее важных исправлений.
3. Интуитивно понятные визуальные элементы, такие как интеллектуальные карты, отображающие размер и качество каждого компонента программного обеспечения.
4. Доступны бесплатные ОС и облачные версии.
5. Интегрируется с Github, Bitbucket, Azure и Git и поддерживает более 10 языков.
6. Доступны бесплатные плагины для IntelliJ IDEA, Visual Studio и Eclipse.

Reshift — это программная платформа на основе SaaS, которая помогает командам разработчиков программного обеспечения быстрее выявлять больше уязвимостей в их коде перед развертыванием в производственной среде. Сокращение затрат и времени на поиск и устранение уязвимостей, определение потенциального риска утечки данных и помощь компаниям-разработчикам программного обеспечения в достижении соответствия и нормативных требований.

Функции:

1. Интегрируется с Github и Bitbucket
2. Обеспечивает безопасность в рабочих процессах группы благодаря потоку запросов и избегает переключения на другие панели управления
3. Интеллектуальная сортировка, которая сокращает количество ложных срабатываний с маркировкой проблем
4. Отслеживает уязвимости для каждой функциональной ветки разработчика
5. Идентификация критических уязвимостей перед слиянием с основной веткой
6. Блокировка сборки при появлении новой уязвимости

Таким образом, на сегодняшний день наработан достаточно широкий спектр методов и инструментов, а также специализированного программного обеспечения позволяющих оптимизировать разработку приложений и написание кодов. Они имеют ряд достоинств и недостатков. Выбор конкретного методы зависит от задач, которые перед собой ставит программист, а также от его временных и материальных возможностей и уровня профессиональной подготовки.

ЛИТЕРАТУРА

1. Implementing effective code reviews: how to build and maintain clean code / Giuliana Carullo. Berkeley, CA: Apress, 2020. 196 p.
2. Li, Genghui Multifactorial optimization via explicit multipopulation evolutionary framework // Information sciences. 2020. Volume 512; pp 1555–1570.
3. Perrin, G. Adaptive calibration of a computer code with time-series output // Reliability engineering & system safety. 2020. Volume 196; pp 176–182.
4. Chong, Nathan Code-level model checking in the software development workflow at Amazon Web Services // Software, practice & experience. 2021. Volume 51: Number 4; pp 772–797.
5. Wang, Song Large-scale intent analysis for identifying large-review-effort code changes // Information and software technology. 2021. Volume 130; pp 78–83.
6. Implementing effective code reviews: how to build and maintain clean code / Giuliana Carullo. Berkeley, CA =: Apress, 2020. 196 p.

Журнал «Современная наука: актуальные проблемы теории и практики»