DOI: 10.37943/AITU.2020.46.64.010

I. Lazarev
Master of Computing technology and software
altait2006.94@gmail.com, orcid.org/0000-0002-4854-4600
Karaganda Technical University, Kazakhstan

I. Tomilova
PhD, Associate Professor, Department of Information-Computing Systems
tomilova_kstu@mail.ru, orcid.org/0000-0001-8782-5627
Karaganda Technical University, Kazakhstan

THE RATIONAL USE OF COMPUTING POWER IN MODERN TRENDS IN THE DEVELOPMENT OF ELECTRONICS

Abstract: This article considers methods and ways of automating software optimization using additional programs and analyzing their algorithms and the principle of operation. Since against the background of the rapid growth of computing power, the question arises about these resources' rational use. With the development of technologies, software development is faced with the problem of degradation of the quality of created applications and the lack of search for new optimization methods. Initially, software development faced an insurmountable wall in the form of a brutal resource limit, which encouraged developers to look for new strategies and tricks to implement the conceived project. Currently, due to breakneck pace of development of computing power, developers have lost this wall as an incentive in search of new methods of implementation and optimization. The developers faced a new problem - the rational use of computing power. For many next-generation applications, I/O constraints limit the performance level that can be achieved. A large and important class of resource-intensive applications is irregular, contains complex data-dependent execution behavior, and dynamic, with resource requirements that change over time. Because the interactions between the application and system software vary between applications and at runtime, analysts are looking to optimize performance need runtime libraries and analysis tools to uncover application I/O behavior. A Developer can use both manual optimization methods and optimizing additional software or optimizing compilers. Optimizing other software and compilers will help reduce the time spent on the optimization stage.

Key words: Development, software, optimization, resources.

Scientific Journal of Astana IT University
ISSN (P): 2707-9031
ISSN (E): 2707-904X

Лазарев И.О.
Магистрант, специальность «Вычислительные технологии и программное обеспечение»
altait2006.94@gmail.com, orcid.org/0000-0002-4854-4600
Карагандинский технический университет, Казахстан

Томилова Н.И.
к.т.н., доцент кафедры информационных-вычислительных систем
tomilova_kstu@mail.ru, orcid.org/0000-0001-8782-5627
Карагандинский технический университет, Казахстан

РАЦИОНАЛЬНОЕ ИСПОЛЬЗОВАНИЕ ВЫЧИСЛИТЕЛЬНЫХ МОЩНОСТЕЙ В СОВРЕМЕННЫХ ТЕНДЕНЦИЯХ РАЗВИТИЯ ЭЛЕКТРОНИКИ

Аннотация: В рамках данной статьи рассматриваются методы и способы автоматизации оптимизации программного обеспечения при помощи их программ. Проанализируем алгоритмы и принцип действия. Так как на фоне быстрого роста вычислительных мощностей возникает вопрос о рациональном использовании данных ресурсов. С развитием технологий разработка ПО встречается с проблемой деградации качества создаваемых приложений и отсутствия поиска новых методов оптимизации. Первоначально разработка программного обеспечения столкнулась с непреодолимой стеной в виде жестокого лимита ресурсов, что стимулировало разработчиков искать новые методы и ухищрения для реализации задуманного проекта. В данный момент, из-за колоссально быстрых темпов развития вычислительных мощностей, разработчики потеряли данную стену как стимул в поисках новых методов реализации и оптимизации. Разработчики столкнулись с новой проблемой – рациональное использование вычислительных мощностей. Для многих приложений следующего поколения ограничения, налагаемые вводом-выводом, ограничивают уровень достижимой производительности. Большой и важный класс ресурсоемких приложений является нерегулярным, содержит сложное, зависящее от данных поведение выполнения, и динамический, с меняющимися потребностями в ресурсах, которые со временем меняются. Поскольку взаимодействия между приложением и системным программным обеспечением меняются между приложениями и во время выполнения одного приложения, аналитикам, стремящимся оптимизировать производительность, требуются библиотеки времени выполнения и инструменты анализа, которые могут выявить поведение ввода-вывода приложения. Для достижения поставленной цели можно использовать как ручные методы оптимизации, так и оптимизирующее стороннее ПО или оптимизирующие компиляторы. Оптимизирующее стороннее ПО и компиляторы помогут сократить время, затрачиваемое на этап оптимизации.

Ключевые слова: Разработка, программное обеспечение, оптимизация, ресурсы.

Введение

Эволюция электроники поражает своими темпами развития. Вычислительные мощности современных компьютеров для домашнего использования сравнимы с суперкомпьютерами 20-го века. Смартфоны, которые сейчас есть почти у каждого так же ничем не хуже суперкомпьютеров. В результате резкого скачка в развитии электроники, разработчики

DOI: 10.37943/AITU.2020.46.64.010

встретили самую страшные проблемы развития своей сферы, лень и деградация. Если сравнивать развитие отрасли разработки и отрасль электроники. Разработчики не успевают за бурными темпами развития электроники.

Из-за того, что электроника развивается быстрее отрасли разработки и возникает проблема рационального использования вычислительных мощностей. Пока отсутствует стена в виде физического ограничения предлагаемых вычислительных ресурсов, разработчики не стремятся искать новые способы экономии данных ресурсов. То, что раньше считалось плохим тоном в разработке программного обеспечения (Далее ПО), в текущих реалиях не является проблемой. На рубеже 1980-2000, разработчики искали любые способы сэкономить заветные пару килобайт, байт чтобы их можно было реализовать и добавить новый функционал или сделать программу еще лучше или улучшить её быстродействие. Существует ряд методов и способов оптимизации программного обеспечения, включая дополнительное стороннее ПО для автоматизации процесса оптимизации разрабатываемого приложения.

Проблема

Как говорилось ранее [1], на данный момент, мало компаний, которые будут обращать внимание на пару затраченных мегабайт памяти, а может и гигабайт. Инновационная Windows 95 на момент выхода занимала всего 44 мегабайта памяти, при этом она имела невообразимое количество функционала и вызвала настоящий фурор в области ПО. Что же случилось с современным рынком программного обеспечения? Только в компаниях, разрабатывающих крупное программное обеспечение задумываются об оптимизации и поисках новых методов реализации, более мелкий рынок пускает все на самотек. Зачем изобретать и искать что-то новое, более быстрое и легкое, если сейчас у всех есть суперкомпьютеры? Отсюда и возникает медленное развитие области разработки ПО.

Разработчики должны думать над вопросами реализации, а не потакать лени и использовать то, что уже было создано. «Зачем менять то, что прекрасно работает? Зачем улучшать велосипед, если он и так справляется со своей функцией?». Для достижения рационального использования ресурсов необходимо покинуть зону комфорта разработки. Так или иначе, из велосипеда создали автомобили.

Оптимизация кода – различные методы преобразования кода ради улучшения его характеристик и повышения эффективности. Среди целей оптимизации можно указать уменьшения объема кода, объема используемой программой оперативной памяти, ускорение работы программы, уменьшение количества операций ввода вывода.

Главное из требований, которые обычно предъявляются к методу оптимизации - оптимизированная программа должна иметь тот же результат и побочные эффекты на том же наборе входных данных, что и неоптимизированная программа. Впрочем, это требование может и не играть особой роли, если выигрыш за счет использования оптимизации может быть сочтен более важным, чем последствия от изменения поведения программы.

Предлагаемое решение

Для решения этой проблемы нужно менять ориентирование разработки ПО с фокуса Гигантизма на минимализм.

Необходимо искать и разрабатывать новые методы и функции, которые будут выполнять задачи с меньшими затратами по вычислительным мощностям, использовать все доступные ресурсы максимально эффективно. Самостоятельно ограничивать себя в вычислительных ресурсах и не смотреть на темпы развития электроники.

На основе способов описанных Никлаусом Виртом [2], в качестве алгоритма разработки можно использовать трехуровневую систему разработки. Первоначально реализовать за-

Scientific Journal of Astana IT University
ISSN (P): 2707-9031
ISSN (E): 2707-904X

думки без жестоких рамок и ограничителей. После того как программа была реализована, провести оптимизацию и рефакторинг программного кода для уменьшения затрачиваемых вычислительных ресурсов. Последним этапом будет являться рекурсия, программное обеспечение нужно будет «загнать» в максимально тесные рамки для поиска новых методов реализации с меньшими затратами используемых ресурсов, после провести второй этап повторно для достижения максимального эффекта. Данная система удобна тем, что программа уже закончена, всегда есть готовая версия для предоставления её конечному пользователю, если времени уже не осталось. Так же удобна она тем, что время разработки будет использовано с максимальной пользой для конечного продукта.

Рассматривать эффективность оптимизации нужно на все аспекты ПО. Даже самые малые части, которые дадут прирост всего жалкие 2% каждые, в совокупности могут выдать картину, в которой будет сэкономлено до 50% ресурсов. Не стоит недооценивать возможности оптимизации даже самых малых блоков, оптимизация которых «Не имеет смысла».

Оптимизация стоит на трех «китах»: естественность, производительность, затраченное время. Давайте разберемся подробнее, что они означают.

Естественность – код ПО должен быть аккуратным, модульным и легко читабельным. Каждый модуль должен естественно встраиваться в программу. Код должен легко поддаваться редактированию, интегрированию или удалению отдельных функций или возможности без необходимости вносить серьезные изменения в другие части программы.

Производительность – в результате оптимизации вы должны получить прирост производительности программного продукта. Как правило, удачно оптимизированная программа увеличивает быстродействие минимум на 20-30% в сравнение с исходным вариантом.

Время – оптимизация и последующая отладка должны занимать небольшой период времени. Оптимальными считаются сроки, не превышающие 10 – 15 % времени, затраченного на написание самого программного продукта. Иначе это будет нерентабельно.

Самый простой и быстрый способ – анализ программного кода дополнительным программным обеспечением. Данное дополнительное программное обеспечение проведет анализ кода ПО и выдаст информацию о возможностях оптимизации некоторых участков, проверит программу на стрессоустойчивость и выдаст набор рекомендаций для разработанного ПО. Данный вариант удобен еще и тем, что он никак не меняет исходное приложение и выдает только рекомендации по её улучшению. Рассмотрим ниже

Существует вариант и более радикального метода автоматизации данного процесса. Дополнительное программное обеспечение помимо анализа из первого пункта, самостоятельно изменяет исходный код для достижения лучших результатов, проводит повторную проверку кода на стрессоустойчивость и анализ на возможность повторной оптимизации. Данный метод имеет как положительную сторону, так и отрицательную. Положительным является рекурсивная работа приложения, увеличивает глубину и эффективность оптимизации разрабатываемого приложения. Отрицательным аспектом является непосредственное вмешательство в структуры приложения, если данный метод не будет создавать дополнительные версии ПО, будет отсутствовать резервное копирование финальной версии, это может привести к повреждению разрабатываемого приложения, что повлечет за собой увеличение времени на разработки и увеличит затраты.

Данные возможности автоматизации могут сильно упростить этап оптимизации программного кода, улучшить разрабатываемое приложение, увеличить охватываемый сегмент пользователей путем понижения затрачиваемых ресурсов в процессе эксплуатации.

Оптимизация программного кода компиляторами и сторонним по

Основываясь на работе Чилингаровой С.А. [3], оптимизирующий компилятор это подвид компилятора, который минимизирует или максимизирует атрибуты исполняемой про-

DOI: 10.37943/AITU.2020.46.64.010

граммы. Общие требования – минимизировать время выполнения программы, требования к выделяемой памяти и энергопотреблению (последние два популярны для портативных компьютеров).

Оптимизация компилятора обычно реализуется с использованием последовательности преобразований, алгоритмов [4], которые берут программу и преобразуют ее для создания семантически эквивалентной программы вывода, которая использует меньше ресурсов и / или выполняется быстрее. Было доказано, что некоторые проблемы оптимизации кода являются не полными или даже неразрешимыми. На практике такие факторы, как готовность программиста ждать, пока компилятор завершит свою задачу, устанавливают верхние пределы для оптимизаций, которые может обеспечить разработчик компилятора. (Оптимизация, как правило, требует очень много ресурсов процессора и памяти.) В прошлом ограничения памяти компьютера также были основным фактором, ограничивающим возможности оптимизации. Из-за этих факторов оптимизация редко дает «оптимальный» результат в любом смысле, и на самом деле «оптимизация» может снизить производительность в некоторых случаях; скорее это эвристические методы улучшения использования ресурсов в типичных программах [5].

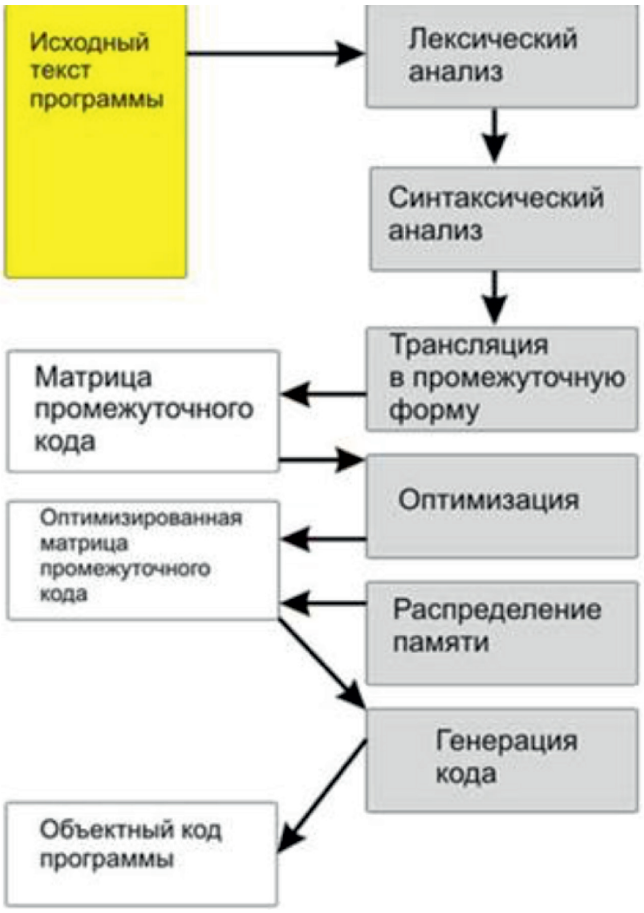

Рис.1. Алгоритм оптимизирующего компилятора.

В начале истории компиляторов оптимизация компилятора была не так хороша, как рукописные. По мере совершенствования технологий компиляции хорошие компиляторы обычно генерируют код, достаточно хороший, чтобы, как правило, больше не требуют больших усилий для программирования оптимизированного вручную кода на языке ассемблера (за исключением некоторых особых случаев).

Scientific Journal of Astana IT University
ISSN (P): 2707-9031
ISSN (E): 2707-904X

Для архитектур ЦП RISC, а тем более для оборудования VLIW, оптимизация компилятора является ключом для получения эффективного кода, поскольку наборы команд RISC настолько компактны, что человеку трудно вручную планировать или комбинировать небольшие инструкции для получения эффективных результатов. Действительно, эти архитектуры были разработаны, чтобы полагаться на разработчиков компиляторов для адекватной производительности.

Однако оптимизирующие компиляторы отнюдь не идеальны. Компилятор не может гарантировать, что для всего исходного кода программы будет выведена самая быстрая или наименьшая из возможных эквивалентных скомпилированных программ; такой компилятор принципиально невозможен, потому что это решило бы проблему остановки (предполагая полноту Тьюринга).

Это можно доказать, рассмотрев вызов функции foo(). Эта функция ничего не возвращает и не имеет побочных эффектов (нет ввода-вывода, не изменяет глобальные переменные и «живые» структуры данных и т.д.). Самая быстрая эквивалентная программа будет просто исключать вызов функции. Однако, если функция foo() фактически не возвращает, то программа с вызовом foo() будет отличаться от программы без вызова; оптимизирующий компилятор должен будет определить это, решив проблему остановки [6,7].

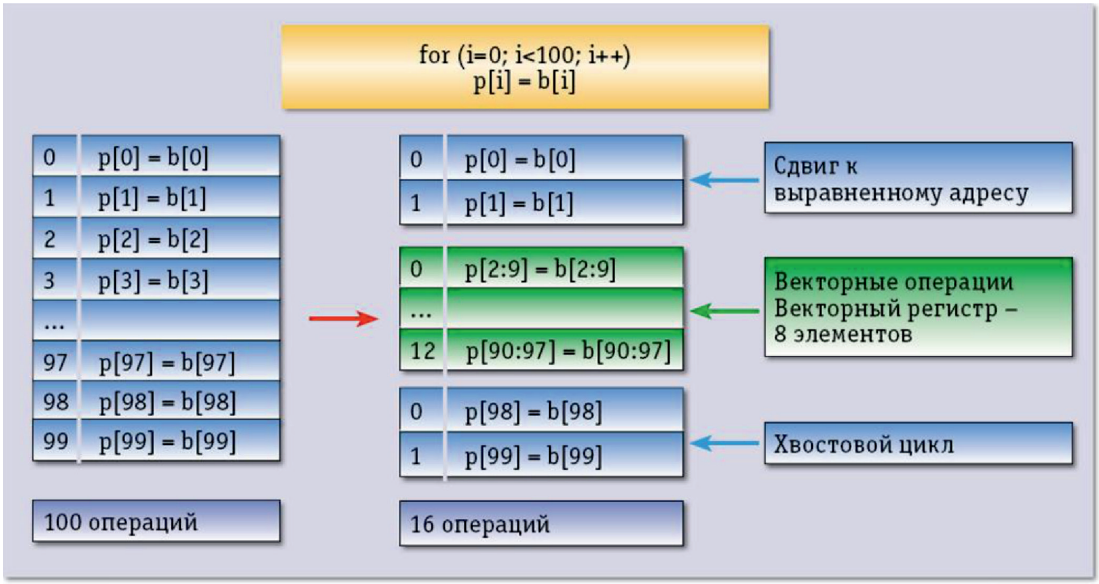

Рис.2. Процесс оптимизации.

Кроме того, существует ряд других, более практичных проблем с оптимизацией технологии компиляции:

Оптимизирующие компиляторы фокусируются на относительно незначительных улучшениях производительности с постоянными коэффициентами и обычно не улучшают алгоритмическую сложность решения. Например, компилятор не изменит реализацию пузырьковой сортировки и использует вместо нее mergesort.

Компиляторы обычно должны поддерживать множество противоречивых целей, таких как стоимость реализации, скорость компиляции и качество сгенерированного кода [8].

Компилятор обычно имеет дело только с частью программы за раз, часто с кодом, содержащимся в одном файле или модуле; В результате он не может учитывать контекстную информацию, которая может быть получена только путем обработки других файлов.

Затраты на оптимизацию компилятора: любая дополнительная работа требует времени; оптимизация всей программы занимает много времени для больших программ.

DOI: 10.37943/AITU.2020.46.64.010

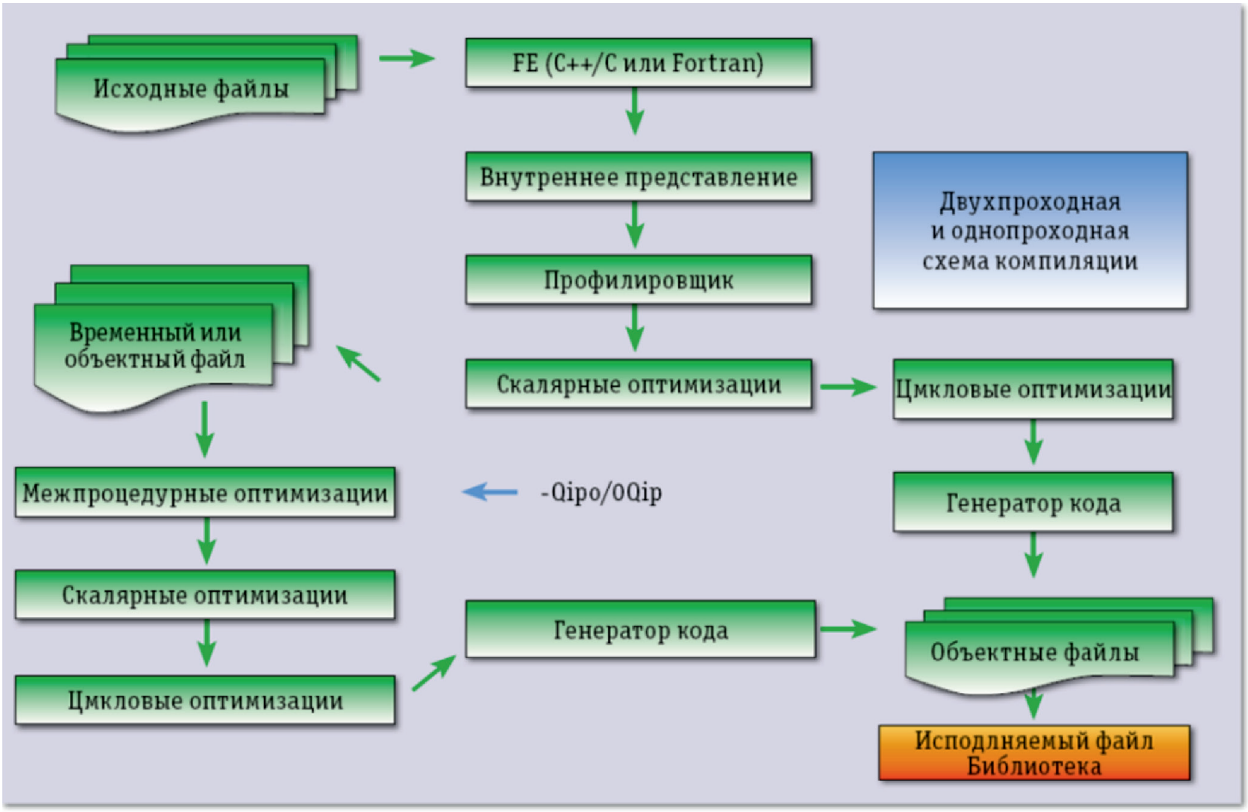

Рис.3. Структура алгоритма оптимизации.

Часто сложное взаимодействие между фазами оптимизации затрудняет поиск оптимальной последовательности для выполнения различных фаз оптимизации.

Кроме того, алгоритмы оптимизации являются сложными и, особенно при использовании для компиляции больших и сложных языков программирования, они могут содержать ошибки, которые вносят ошибки в сгенерированный код или вызывают внутренние ошибки во время компиляции. Любые ошибки компиляции могут сбивать с толку пользователя, но особенно в этом случае, поскольку может быть неясно, что логика оптимизации ошибочна. В случае внутренних ошибок проблема может быть частично улучшена с помощью «отказоустойчивого» метода программирования, в котором логика оптимизации в компиляторе закодирована так, что ошибка перехватывается, выдается предупреждающее сообщение, и остальная часть компиляции приступает к успешному завершению [9].

Работа по улучшению технологии оптимизации продолжается. Одним из подходов является использование так называемых пост проходных оптимизаторов (некоторые коммерческие версии которых относятся к программному обеспечению мэйнфреймов конца 1970-х годов). Эти инструменты получают исполняемый вывод «оптимизирующим» компилятором и оптимизируют его еще больше. Пост проходные оптимизаторы обычно работают на языке ассемблера или на уровне машинного кода (в отличие от компиляторов, которые оптимизируют промежуточные представления программ). Производительность пост проходных компиляторов ограничена тем фактом, что большая часть информации, доступной в исходном коде, не всегда доступна для них.

Поскольку производительность процессора продолжает расти быстрыми темпами, а пропускная способность памяти увеличивается медленнее, оптимизации, которые снижают требования к пропускной способности памяти (даже за счет того, что процессор выполняет относительно больше инструкций), станут более полезными. Примеры этого, уже упомянутые выше, включают оптимизацию гнезд петли и повторную материализацию.

Scientific Journal of Astana IT University
ISSN (P): 2707-9031
ISSN (E): 2707-904X

Рис.4. Архитектура оптимизирующего компилятора.

Сравнительный анализ

В качестве оптимизирующих компиляторов рассмотрим компилятор предоставляемый компанией Intel. Как оптимизирующее программное обеспечение рассмотрим Scala Optimizer и Proguard.

Для проведения сравнительного анализа оптимизирующего компилятора и стороннего оптимизирующего ПО воспользуемся единой для всех формулой расчетов для тестирования. Формула включает в себя 10 шагов суммирования, и входные значения будут статичны для всех тестируемых языков программирования и средств оптимизации.

На основе полученных данных о быстродействии программного обеспечения сравним эффективность данных методов [10].

$$\sum_{i=1}^{10} \frac{3x^2 + 11y * 4z^2}{11x + 2y} \tag{1}$$

где:

x, y, z – переменные, которые задаются 10 раз в диапазоне от 1 до 200.

i – номер шага.

DOI: 10.37943/AITU.2020.46.64.010

Таблица 1. Входные значения для формулы

Шаг	x	y	z
1	112	13	24
2	53	78	41
3	76	193	15
4	11	43	64
5	68	31	27
6	156	47	34
7	13	51	12
8	97	23	112
9	54	49	36
10	31	187	86

При использовании оптимизирующих компиляторов можно достичь следующих показателей:

Таблица 2. Оптимизирующие компиляторы Intel

Язык	Оптимизирующие компиляторы Intel		
	Время выполнения кода до оптимизации	Время выполнения оптимизированного кода	Экономия времени
C++	1.75	1.15	34%
Java	1.01	0.581	43%
PHP	5.33	4.49	16%
Python	2.51	3.21	-27%

Не все языки программирования показывают высокие показатели эффективности, Язык программирования Питон плохо поддерживает данный механизм.

При использовании оптимизирующего ПО результаты тестирования на различных языка разработки следующие:

Таблица 3. Scala Optimizer

Язык	Scala Optimizer		
	Время выполнения кода до оптимизации	Время выполнения оптимизированного кода	Экономия времени
C++	1.75	1.42	19%
Java	1.01	0.79	21%
Visual Basic	2.78	2.77	<1%
Python	2.51	1.8	28%

Показатели эффективности значительно меньше, чем у оптимизирующих компиляторов, однако питон в данном тесте выдал хорошие результаты эффективности в отличие от предыдущего. Для разных языков программирования существуют разное оптимизирующее программное обеспечение, имеющих различные алгоритмы работы.

Scientific Journal of Astana IT University
ISSN (P): 2707-9031
ISSN (E): 2707-904X

Таблица 4. Proguard

Язык	Proguard		
	Время выполнения кода до оптимизации	Время выполнения оптимизированного кода	Экономия времени
C++	1.75	1.26	28%
Java	1.01	0.68	32%
Visual Basic	2.78	2.66	4%

Показатели эффективности рознятся от одной оптимизирующей программы к другой, и разница эффективности колеблется от 5% до 10%.

Базироваться на выборе метода, следует в первую очередь от языка программирования. Разные методы и способы, разное оптимизирующее ПО, по-разному показывают себя в различных средах разработки.

Заключение

Таким образом, существуют различные способы автоматизированной оптимизации ПО. Оптимизировать код ПО, необходимо после этапа его разработки, для этого установить «искусственную стену» нехватки ресурсов для поиска возможных методов и способов оптимизации.

Оптимизирующий компилятор и дополнительное оптимизирующее программное обеспечение позволит существенно сократить затрачиваемое время на оптимизацию разрабатываемого ПО, но способ не является полностью законченным.

Данный способ позволит существенно сократить используемые вычислительные ресурсы ПО на крупных проектах, увеличит их быстродействие и устойчивость к неполадкам в программе. Данный метод не является гарантом качественной оптимизации, так как может так же и нарушить структуру работы ПО, однако помогает значительно ускорить процесс оптимизации.

Для того чтобы рационально использовать вычислительные мощности недостаточно только дополнительного ПО как оптимизирующий компилятор. Необходимо разрабатывать и искать новые методы и способы оптимизации ПО в текущих реалиях, так как количество и разнообразие языков программирования растет, так же на ровне с ними растут и потребности в оптимизации. Методы и алгоритмы оптимизации по-разному работают на языках программирования, показывая большие отличия в эффективности их применения.

Литература

1. Лазарев, И.О. (2020). Рациональное использование вычислительных мощностей в современных тенденциях развития электроники, *Труды Международной научно-практической online конференции «Интеграция науки, образования и производства-основа реализации Плана нации», 1,* 1055.
2. Никлаус Вирт (2010). *Построение компиляторов*. М.: ДМК Пресс, ISBN 978-5-94074-585-3, 0-201-40353-6, 1, 153.
3. Чилингарова, С.А. (2006). Методы оптимизации для динамических (just-in-time) компиляторов. *Компьютерные инструменты в образовании,* (2).
4. Кнут Дональд Эрвин (2019). *Искусство программирования. Основные алгоритмы*. М.: Вильямс, ISBN: 978-5-8459-1984-7, 1, 277.
5. Зубков Сергей Владимирович (2017). *Assembler. Для DOS, Windows и Unix*, Издательство: ДМК-Пресс, ISBN: 978-5-94074-725-3, 1, 288.

6. Бек Кент, Брант Джон, Фаулер Мартин. (2017). Рефакторинг. Улучшение проекта существующего кода. М.: Диалектика, ISBN: 978-5-9909445-1-0, Том 1, стр. 250.
7. Стив Макконнелл (2017). Совершенный код. Мастер-класс. М.: Русская редакция, ISBN: 978-5-7502-0064-1, 1, 249.
8. Кормен Томас, Штайн Клиффорд, Ривест Рональд, Лейзерсон Чарльз (2019). *Алгоритмы. Построение и анализ*. Издательство: Диалектика, ISBN: 978-5-907114-11-1, 1, 140.
9. Ахо, А., Лам, М., Сети, Р., & Ульман, Д. (2008). Компиляторы: принципы, технологии и инструментарий. *М.: Вильямс*. ISBN: 978-5-8459-1932-8, 1, 257.
10. Курт Гантерот (2019). *Оптимизация программ на C++. Проверенные методы повышения производительности*. М.: Диалектика, ISBN: 978-5-907144-58-3, 1, 49.