Министерство образования и науки Российской Федерации
Московский физико-технический институт
(национальный исследовательский университет)

Физтех-школа радиотехники и компьютерных технологий
Кафедра микропроцессорных технологий в телекоммуникационных сетях и вычислительных системах

Выпускная квалификационная работа бакалавра по направлению 03.03.01
«Прикладные математика и физика»

Исследование и разработка системы анализа кода для повышения производительности программ на языке программирования высокого уровня

Студентка Б01-008 группы
Лирисман К. С.

Научный руководитель
Гаврин Е. А.

Долгопрудный
2024

Содержание

Аннотация

В настоящий момент активно развивается статически типизированный управляемый язык программирования, являющийся расширенной и более быстрой версией языка TypeScript (далее TS). Основная идея разработки спецификации и компилятора этого языка программирования — сделать его максимально похожим на TS для упрощения перехода будущих разработчиков между TS и выбранным для исследования языком, а также для ускорения переписывания существующих на TS приложений. Таким образом, между целевым языком и TS формируется общая часть, - корректная с точки зрения TS. Оставшуюся часть называют не поддерживаемой TS.

В данной работе предлагается выделить из целевого ЯП подмножество, наиболее выгодное с точки зрения производительности, в том числе за счёт отсечения удобного для разработчиков, но медленного функционала. Таким образом, пока сам язык неминуемо разивается в сторону общности с TS‘ом ради легкости перехода, предложенная система помогает держать фокус на производительности.

Анализ происходит в момент компиляции исходного кода и предлагает включение желаемых проверок группами или по отдельности путем добавления флагов компиляции. Реализовано 8 типов проверки, опирающихся на спецификацию выбранного языка программирования высокого уровня и реализацию его компилятора. К общим с TS проверкам относятся: неявная упаковка и распаковка, ускорение проверок равенства, запрет инструкций верхнего уровня, исключая классы и функции, предложение установки модификатора для класса или метода как финального и другие. К несовместимым с TS относятся: исползование корутин вместо асинхронных функций, предложение использовать модификатор, ограничивающий наследование классов и методов в случае, если на момент проверок у них нет потомков. Предлагается использование двух режимов работы проверяющей системы — в состоянии предупреждений, а именно, предложений, не обязывающих разработчика к исправлением замечаний и не влияющих на результат работы программы, и в состоянии, приводящему к ошибке при ненулевом количестве предложений, ожидающих от пользователя последующих исправлений, и считающегося за ошибку компиляции. Предложенная система формулирует рекомендации для повышения производительности. Однако существуют сценарии использования, когда разработчикам необходимо отключение этих проверок. С помощью произведенной системы регулирования режимов проверок пользователи целевого языка могут активировать или деактивировать любую опцию по одной или включать группами. Также можно

снять с проверки определенные строки или целые частей кода - всё это реализуемо с помощью предложенного аналога системы точечного отключения проверок Clang Tidy непосредственно в исходном коде программы в виде многострочных или однострочных комментариев. Таким образом, данная система значительно повышает скорость работы и время запуска приложения на выбранном статически типизированном языке высокого уровня, позволяет разработчикам простыми методами улучшить их код и потенциально избежать некоторых ошибок.

Глава 1

Введение

Языки программирования разрабатываются с основной целью облегчить использование компьютеров большим количеством людей без необходимости подробного знания их внутренней структуры. Языки соответствуют типу приложений, которые будут программироваться с использованием этого языка. Идеальным языком был бы тот, который точно выражает спецификацию решаемой проблемы и преобразует ее в серию инструкций для компьютера. Достичь этого идеала невозможно, поскольку часто отсутствует четкая спецификация проблемы, а разработка алгоритма на основе спецификаций требует предметных знаний и опыта. Существует большое количество языков, более тысячи, каждый из которых предназначен для своего класса приложений. Все современные языки программирования созданы машинно-независимыми. Другими словами, структура языка программирования не будет зависеть от внутренней структуры конкретного компьютера. Необходимо иметь возможность выполнить программу, написанную на этом языке программирования, на любом компьютере, независимо от того, кто его изготовил и какой модели. Такие языки известны как машинно-независимые языки программирования высокого уровня.

Область языков программирования динамична и даже несколько хаотична. По мере появления на рынке более сложных аппаратных систем появляются новые компьютерные приложения. Эти приложения порождают новые языки для решения таких приложений. Другой тенденцией является постоянное увеличение сложности приложений по мере того, как аппаратное обеспечение становится более сложным и дешевым. Увеличение размера программ требует новых методов решения проблемы сложности, сохраняя при этом низкую стоимость разработки программ и обеспечивая их корректность.

В современном программировании особенно значимы языки программирования высокого уровня, которые позволяют разработчикам сфокусироваться на логике приложений, а не на деталях управления памятью или другими низкоуровневыми задачами. Однако, на таких языках возникают проблемы с производительностью, особенно в контексте больших и сложных проектов. Люди вынуждены создавать новые языки программирования высокого уровня, так как:

1. Новые языки могут быть спроектированы с целью оптимизации производитель-

ности и улучшения эффективности разработки программного обеспечения. Это может включать в себя более эффективное использование ресурсов компьютера, упрощение синтаксиса для повышения читаемости кода или введение новых функций и конструкций для облегчения работы разработчиков.

2. Новые языки могут быть созданы для решения конкретных проблем или задач, которые не могут быть эффективно решены с использованием существующих языков. Например, некоторые языки могут быть специально разработаны для работы с распределенными системами, большими данными или машинным обучением.

3. Некоторые языки могут быть разработаны для удовлетворения специфических потребностей определенных областей или индустрий. Например, языки для разработки игр, веб-приложений, научных вычислений или встроенных систем имеют свои особенности.

4. Разработка новых языков программирования является частью процесса постоянной эволюции и инновации в области информационных технологий. Новые языки могут предложить новые идеи, концепции и подходы, которые могут привнести свежий взгляд на программирование и стимулировать развитие отрасли в целом.

В этой работе мы обращаемся новому к языку программирования, который активно развивается как быстрая альтернатива уже широко используемому TypeScript. Основная цель разработки этого языка заключается в создании расширенной и более производительной версии TypeScript. При этом важно сохранить совместимость с TypeScript для облегчения перехода существующих проектов и обучения новых разработчиков.

Современная динамичная среда программирования требует от разработчиков не только функциональности, но и высокой производительности своих приложений. В этом контексте особенно актуальными становятся инструменты, способные автоматизированно анализировать и оптимизировать исходный код программ. В этом контексте, важную роль играют инструменты, способные анализировать исходный код программ, для повышения производительности программ на языке программирования высокого уровня

Следует уточнить, что подразумевается под повышением производительности в данной работе. В целом существует неколько параметров программы, которые определяют качество произвоительности. Это потребление энергии во время работы программы, время исполнения программы, то есть ее performance, или, например, итоговый размер бинарного файла для исполнения. Эти и другие параметры вместе двигать к идеальному состоянию невзможно, так как он тесно связаны, и улучшение одной характеристики зачастую неминуемо ведет к ушудшению зависимой. Размер итогово файла после компиляции является статическим параметром, то есть зависящим только от компилятора, использованной архитектуры и примененных оптимизаций.

Производительность и потребление энергии работающего приложения являются динамическими параметрами программы. Это означает, что они значительно зависят от

устройства, на которых им предстоит исполняться. Конечно, это делает замеры значительно более сложными и распределенными, но можно опираться на сравнение формата "до"и "после"в среднем в рамках выбранных устройств и оценить полученную динамику в среднем между разными девайсами уже после получения их локальных результатов.

Скорость выполнения программы и энергопотребление - это два важных, но различных аспекта, которые могут быть оптимизированы по-разному. Некоторые изменения в языке программирования или в реализации компилятора могут привести к увеличению скорости выполнения программы за счет более эффективного использования ресурсов процессора или памяти. Однако, эти же изменения могут привести к увеличению потребления энергии, например, за счет увеличения числа операций или увеличения нагрузки на процессор. Примеры таких изменений могут включать в себя:

1. Увеличение использования параллелизма: параллельные алгоритмы могут увеличить скорость выполнения программы, но при этом могут потреблять больше энергии из-за увеличенной нагрузки на процессор.

2. Изменение алгоритма или структуры данных: некоторые изменения в алгоритмах или структурах данных могут сделать программу более эффективной с точки зрения скорости выполнения, но могут также потреблять больше энергии из-за увеличенного числа операций или использования памяти.

3. Оптимизации компилятора: оптимизации компилятора могут улучшить скорость выполнения программы, но в некоторых случаях могут привести к увеличению потребления энергии из-за более сложных оптимизационных процессов.

Однако в данной работе не предлагается использование иных методов ускорения, отличных от связанных со спецификацией и реализацией его компилятора. Поэтому вышеупомянутые нерелевантны, а ипользование более высокоуровневых абстракций предполагается заранее считать вредным для быстродействия, поэтому будет предлагаться поиск менее дорогостоящих аналогов с точки зрения скорости исполнения.

Таким образом, в данном случае уместно считать, что меньшее время исполнения программы влечет к меньшему энергопотреблению. Далее под повышением производительности будет пониматься скорость выполнения программы в рамках выбранных устройств и опций компилятора.

Глава 2

Постановка задачи

Цели работы:

1. Разработка системы анализа и предупреждений исходного кода выбранного языка программирования высокого уровня для повышения его производительности и ускорения запуска написанных на нем приложений
2. Реализация системы-аналога Clang Tidy для выборочного отключения выбранных проверок точечно, непосредственно в исходном коде.

Задачи работы:

1. Изучение существующих решений
2. Разработка системы анализа кода на этапе компиляции
3. Анализ текущей спецификация целевого языка программирования на предмет потенциально медленных языковых конструкций и функционала, сбор данных для дальнейшего тестирования.
4. Предоставить вариант исправления, ускорящий работу приложения, корректный с точки зрения выбранного языка программироания, для каждой языковой единицы среди предложенных
5. Протестировать каждое предложение: замерить скорость работы приложения до и после предложенных исправлений
6. Реализовать соответствующую поверку в системе анализа после подтверждения положительных результатов тестирования
7. Разработать систему точечного и группового отключения выбранных разработчиком проверок, поддержать наиболее популярные сценарии использования, опираясь на данные, полученные при изучении существующих решений
8. Поддержать возможность анализа в системе многофайловой сборки

Цели работы разумно считать достигнутыми при выявлении и реализации не менее пяти предложений, ускоряющих работу приложения в среднем не менее, чем на 5 %, а также разработке системы отключения проверок, успешной поддержке проектной сборки и прохождении тестирования, составленного из некоторых потенциальных сценариев применения предложенных решений.

Глава 3

Обзор существующих решений

и снова текст.

Глава 4

Теоретическая часть

так тоже неплохо.

Глава 5

Практическая часть

практикуем.

Глава 6

Заключение

заключаем

Литература

Будет добавлена.