На правах рукописи

Четверина Ольга Александровна

Повышение качества компиляции кода в режиме по умолчанию

Специальность 05.13.11 —
«математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей »

Автореферат
диссертации на соискание учёной степени
кандидата физико-математических наук

Москва — 2019

Работа выполнена в АО «МЦСТ».

Научный руководитель:	**Нейман-заде Мурад Искендер оглы**, кандидат физико-математических наук
Официальные оппоненты:	**Галатенко Владимир Антонович,** доктор физико-математических наук, заведующий сектором Федерального государственного учреждения «Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук»
	Намиот Дмитрий Евгеньевич, кандидат физико-математических наук, Московский государственный университет им. М.В. Ломоносова, старший научный сотрудник
Ведущая организация:	Федеральное государственное учреждение «Федеральный исследовательский центр «Информатика и управление» Российской Академии Наук»

Защита диссертации состоится 16 мая 2019 года в 15:00 на заседании диссертационного совета Д 002.087.01 при Федеральном государственном бюджетном учреждении науки «Институт системного программирования им. В.П. Иванникова Российской академии наук» по адресу: 109004, Москва, ул. Александра Солженицына, 25.

С диссертацией можно ознакомиться в библиотеке и на сайте Федерального государственного бюджетного учреждения науки «Институт системного программирования им. В.П. Иванникова Российской академии наук».

Автореферат разослан _____ 2019 г.

Ученый секретарь
диссертационного совета Д 002.087.01
кандидат физ.-мат. наук Зеленов С.В.

Общая характеристика работы

Актуальность темы исследования

Производительность современных вычислительных машин в значительной мере зависит от качества планирования исполняемого кода оптимизирующими компиляторами в соответствии с особенностями архитектуры.

С целью обеспечения оптимального планирования, в процессе компиляции к коду применяется последовательность преобразований, которые формируют более эффективный семантически эквивалентный код. Для части преобразований качественный результат их применения можно определить однозначно по промежуточному представлению кода. Однако в большинстве случаев оптимальный способ применения преобразований либо зависит от недоступных на этапе компиляции характеристик исполнения кода, таких как профиль исполнения или особенности обрабатываемых данных, либо не может быть точно вычислен за приемлемое время. Для более точного применения преобразований используются различные техники, включающие в себя использование полученного на тренировочных данных профиля исполнения кода (PGO), обеспечение возможности одновременного анализа кода всей программы (IPA) или явная настройка оптимизаций регулирующими опциями. Максимальная производительность кода, которую удается при этом получить, будем называть пиковой производительностью. Разница в производительности кода, собранного в режиме по умолчанию и при пиковой сборке, особенно значительна для архитектур со статическим планированием кода, поскольку в динамическом случае работу по уточнению профиля и анализу данных берет на себя архитектура. Еще большее отличие возникает при компиляции под архитектуры с высокой параллельностью на уровне команд (EPIC).

Однако указанные техники достижения пиковой производительности не всегда доступны для использования. И построение профиля исполнения, и явная настройка оптимизаций, как ручная, так и автоматическая, требуют тренировочного исполнения приложения. При этом качественно подобрать входные данные и получить достоверный полный профиль исполнения удается только для сравнительно небольших приложений, а использование неточного профиля может приводить к ощутимому отрицательному эффекту. Так, на

машине Эльбрус с EPIC архитектурой и профессионально сформированном для тестирования производительности пакете spec2000 CFP автором было выявлено среднее замедление приложений на 20% при компиляции без оптимизаций неиспользуемых в тренировочном запуске процедур.

Вторая проблема достижения высокой производительности заключается в значительных затратах времени на осуществление компиляции. Даже в процессе проведения компиляции в режиме по умолчанию компилятором Эльбрус осуществляется более 300 этапов компиляции, и затрачивается в среднем в 20 раз больше времени, чем в случае неоптимизирующей сборки. Для ряда задач такой расход времени становится неприемлемо большим при предварительной компиляции и недопустимым для Just-in-time компиляции, при которой производительность явным образом зависит от скорости оптимизирующей компиляции. Техники же многократной компиляции с использованием различных опций оптимизации и сравнением времен исполнения с выбором наименьшего, или итерационные решатели, реализованные для процессоров ARM, серверов Sun и для других вычислительных машин, практически не применяются промышленно. То есть, помимо проблемы недоступности ряда техник, повышающих качество оптимизации кода, пользователи сталкиваются и с проблемой ограниченного времени, доступного для анализа и компиляции кода.

Необходимость повышения скорости исполнения кода в режиме по умолчанию с одновременным учетом времени компиляции определяет актуальность диссертационной работы.

Цель исследования

Целью представленного исследования является поиск и обоснование методов повышения качества компиляции кода в режиме по умолчанию, доступному к применению для большинства приложений. В соответствии с этой целью были поставлены следующие **задачи**:

- доработать имеющиеся и разработать альтернативные оптимизационные преобразования, позволяющие приблизить базовую оптимизацию к пиковой;
- разработать наборы оптимизационных последовательностей с настройками;

– разработать метод качественного учета времени исполнения, времени компиляции и других характеристик;
– разработать механизм поиска лучшего набора оптимизаций по промежуточному представлению.

Научная новизна

Научной новизной обладают следующие результаты работы.

– Построен единый функционал для попроцедурной многокритериальной оценки качества компиляции.
– Теоремы о соответствии точек на границе Парето минимумам функционала и о представлении функций, сохраняющих порядок при растяжениях.
– Утверждение о представлении функционала качества компиляции для одновременного учета времени исполнения и времени компиляции.
– Формулировка задачи минимизирующей классификации с заданной функцией потери. Алгоритм ее решения и метод перехода к аддитивной функции потери для отдельных минимумов. Теоремы о сходимости.
– Методы оптимизации циклов посредством раскрутки части их путей и теоремы, позволяющие оценить их эффективность. Метод выбора адресов данных для подкачки нерегулярных чтений структур. Алгоритм конвейеризации с забросом чтений и ограничением планируемого времени исполнения цикла.

Теоретическая и практическая значимость

Теоретическую ценность представляет собой обоснование постановки задачи машинного обучения в виде требования минимизации функционала качества, зависящего от назначения классов объектам всей выборки, а также предложенные алгоритмы решения такой задачи и доказанные утверждения об их сходимости. Интерес представляют и доказанные утверждения о строении многокритериального функционала качества, удовлетворяющего свойству сохранения порядка при растяжениях.

Практическую значимость представляют предлагаемые оптимизационные преобразования, доступные для применения в режиме компиляции по умолчанию; разработанный набор оптимизирующих

последовательностей; механизм построения классификационного решения на основе статистической информации о компиляции тренировочного пакета задач. Эффективность описанных механизмов и методов оценивалась путем замера времени исполнения и компиляции на вычислительном комплексе с микропроцессором «Эльбрус» с VLIW архитектурой. Замеры производились на задачах пакетов, разработанных для замера производительности процессоров: SPEC CPU95, SPEC CPU2000.

Методология и методы исследования

Методы исследования заимствованы из областей системного программирования, технологии компиляции и различных областей математики:

- методы анализа потоков данных и управления программы для выявления способов повышения параллельности на уровне операций;
- методы анализа работы памяти вычислительных машин при исполнении приложений;
- методы математической статистики;
- методы математического анализа;
- методы машинного обучения;
- методы теории графов;
- методы математической логики;
- методы теории алгоритмов.

Положения, выносимые на защиту

В процессе проведения диссертационного исследования были получены следующие результаты, выносимые на защиту.

- Разработаны методы частичной раскрутки коротких путей и рекуррентностей циклов, позволяющие лучше спланировать сложные циклы с разновесными ветвлениями. Доказаны утверждения об оценке их эффективности. Предложены неагрессивные методы оптимизации работы с памятью.
- Предложена числовая многокритериальная оценка качества компиляции, учитывающая попроцедурный характер оптимизации. Доказаны утверждения о ее представлении и показано ее преимущество по сравнению с вероятностной оценкой качества.
- Сформулирована задача минимизирующей классификации, соответствующая обучению с построенной оценкой качества

компиляции. Разработаны алгоритмы ее решения и доказаны утверждения об их сходимости. Построенная в соответствии с разработанными методами классификация попроцедурного назначения оптимизационной последовательности внедрена в промышленный компилятор для архитектуры Эльбрус.

Апробация

Результаты, полученные в работе, были доложены на научных конференциях и семинарах:

- 56-ой научной конференции МФТИ, Москва, МФТИ, 2013;
- Национальном Суперкомпьютерном Форуме НСКФ-2013, 2013;
- I-ой Всероссийской научно-технической конференции «Расплетинские чтения», Москва, 2014;
- Spring/Summer Young Researchers' Colloquium on Software Engineering – SYRCoSE, Самара, 2015;
- II-ой Международной конференции «Инжиниринг & Телекоммуникации — En&T», Москва/Долгопрудный, 2015;
- Open Conference on Compiler Technologies, Москва, РАН, 2015;
- Семинаре кафедры МаТИС механико-математического факультета МГУ им. М.В. Ломоносова, 2018.

Публикации

По теме диссертации опубликовано 10 печатных работ [1]-[10] и получено свидетельство о государственной регистрации программы для ЭВМ [11]. Работы [4], [5], [6], [8], [9], [10] опубликованы в изданиях из списка ВАК на русском и иностранном языке. Статья [10] опубликована в журнале, который также входит в список Scopus. В работах [3], [4] и публикации [11] личный вклад автора заключается в выборе числовой оценки качества компиляции, создании наборов оптимизационных последовательностей и в разработке и реализации методов машинного обучения для выбора последовательности из набора. В совместной работе [5] вклад автора заключается в анализе программ и выбору для них пиковых наборов опций компиляции, а также в разработке новых методов оптимизации кода и предложениях по коррекции ранее реализованных. В совместной работе [8] автором предложен метод последовательного выбора пар для инлайн-подстановок на основе изменения значения функции, зависящей от предсказанных характеристик компиляции.

Личный вклад автора

Все представленные в диссертации результаты получены лично автором.

Структура и объем работы Работа состоит из введения, четырех глав и заключения. Основной текст диссертации (без приложений и списка литературы) занимает 115 страниц, общий объем – 125 страницы с 18 рисунками и 6 таблицами. Список литературы содержит 73 наименования.

Содержание работы

Во **введении** обосновывается актуальность темы работы, сформулированы цели и задачи, описывается теоретическая и практическая значимость результатов.

В **первой главе** рассматриваются различные использующиеся современными компиляторами (gcc, lcc) способы настройки оптимизации кода приложений и поставлена задача попроцедурной направленной оптимизации.

Во-первых, описаны основные режимы компиляции: уровни оптимизации, режим межмодульной оптимизации, двухфазный режим со сбором профиля исполнения и установка общих правил для кода программы и ее исполнения. Во-вторых, пояснена сложность настройки качественного применения последовательности преобразований. Большинство преобразований обладают свойством двойственности с точки зрения использования ресурсов, таких как планируемое количество тактов исполнения, требуемое количество регистров, нагрузка на кэш-память и других. Совместное влияние преобразований в последовательности на использование ресурсов делает задачу наиболее эффективной компиляции нерешаемой точно за разумное время. Вдобавок оптимальный набор преобразований может зависеть от входных данных, то есть от недоступной на этапе компиляции информации. По указанным причинам для настройки работы компилятора используются эвристические методы, а для более качественной оптимизации конкретного приложения может быть использована дополнительная настройка набора преобразований.

Самыми известными методами автоматической настройки последовательности преобразований являются итеративные решатели.

Описанные в научной литературе итеративные решатели для архитектур Intel Itanium, ARM, Sparc и других, заключаются в многократной компиляции кода с различными наборами настроек и сравнении результатов либо по временам исполнения на вычислительной машине, либо по оценочным временам исполнения, вычисляемым по спланированному коду. Итеративные методы позволяют существенно повысить производительность кода, в том числе, поскольку допускают возможность разной настройки компиляции отдельных процедур, входящих в состав приложений. Однако все решатели такого типа требуют значительного времени компиляции и тренировочного исполнения приложений на представительных данных, в результате чего они практически не применимы для больших приложений.

В качестве альтернативного метода настройки компиляции интересной представляется подход машинного обучения с целью выбора оптимальных настроек компиляции по вычислимым характеристикам компилируемого кода. Подобный подход был реализован в Milepost GCC для настройки компиляции приложений. В этом решении оценивается вероятность того, что приложение с некоторыми значениями характеристик соответствует некоторому оптимальному набору настроек.

Подход настройки приложений целиком интересен для небольших вычислительных задач с однотипным составом с точки зрения процедур. Однако для больших приложений интересна также возможность попроцедурной настройки, поскольку составляющие их процедуры уже могут существенно отличаться по частоте исполнения, наличию вычислительных циклов, чтению памяти, размерам кода и так далее.

Определение. Здесь и далее под *линейкой* $l \in L$ будет подразумеваться последовательность оптимизаций и аналитических этапов вместе с настройками. В частности, две линейки могут отличаться только настройками оптимизаций.

В представленной работе предлагается машинное обучение с целью попроцедурного выбора линейки в процессе компиляции.

Схема предлагаемого метода состоит в следующем (Рис. 1):

1. Выбирается набор линеек и в дальнейшем используется как составная часть механизма.

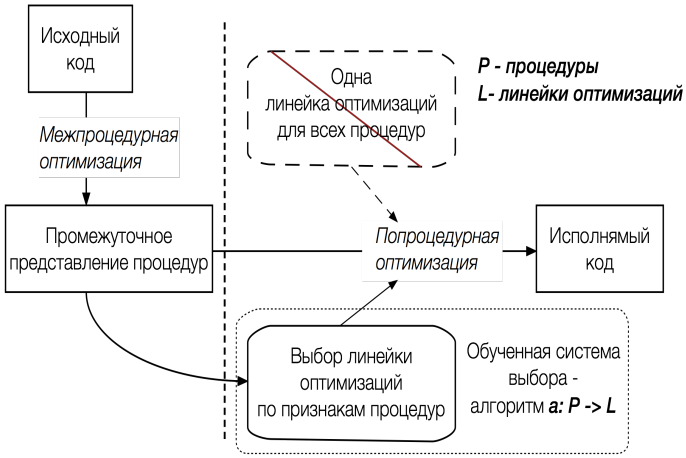

Рис. 1 — Схема направленной оптимизации процедур.

2. В процессе компиляции после проведения межпроцедурных преобразований вычисляются характеристики каждой процедуры.
3. По вычисленным характеристикам с помощью заранее разработанного алгоритма каждой процедуре назначается своя линейка из набора.
4. Дальнейшая, то есть попроцедурная компиляция, осуществляется в соответствии с назначенными линейками.

Вопрос разработки набора опций или оптимизационных последовательной уже в значительной мере исследован в научной литературе для решения задачи построения итерационных решателей. Однако проведенный анализ показал неэффективность или неприменимость в однофазном режиме компиляции ряда оптимизаций, значимых для получения пиковой производительности. В связи с этим была поставлена задача поиска альтернативных методов оптимизации, применимых в однофазном случае. Решению этой задачи посвящена Глава 2. С использованием разработанных оптимизаций были построены 4 линейки, позволившие получить основное ускорение тренировочного пакета, доступное без использования профиля и дополнительных разрешающих опций.

Вторая задача заключается в машинном обучении компилятора с целью выбора по признакам процедуры линейки из набора.

При попытке использовать вероятностный подход для задачи попроцедурного выбора оптимизационной линейки, аналогичный использованному в Milepost для приложений, автором была выявлена проблема, которая заключается несоответствии вероятностной ошибки реальному ожиданию от обучения компилятора. Так, обучение компиляции сетями Байеса с целью повышения производительности и менее приоритетному ускорению компиляции (то есть выбору быстрой компиляции при одинаковом времени исполнения) позволило с вероятностью 95% предсказать лучшую линейку, но при этом привело к среднему замедлению приложений на 21%. Анализ указанного замедления показал следующее:

1) ошибка выбора линейки для процедуры зависит от того, какая именно не оптимальная линейка была выбрана;

2) ошибка выбора линейки для одной и тоже же процедуры зависит от содержащего ее приложения.

Указанные особенности связаны с тем, что в действительности при обучении компилятора требуется повысить *не вероятность выбора оптимальных настроек, а среднюю производительность пакета приложений* с учетом среднего времени компиляции или среднего значения других характеристик.

В Главе 3 подробно рассмотрен вопрос построения числовой характеристики, позволяющей оценить качество обучения компилятора. В Главе 4 сформулирована и решена задача машинного обучения, соответствующая минимизации построенного функционала качества компиляции.

Во **второй главе** рассматривается вопрос поиска и настройки методов оптимизации, применимых в режиме по умолчанию. Автором выявлены основные существующие высокоэффективные методы оптимизации, которые по разным причинам не работают или малоэффективны в случае отсутствия информации о профиле исполнения задач, либо применяются только при передаче компилятору соответствующего флага, и предложены альтернативные методы оптимизации и настройки для использования по умолчанию. Важные отличия работы оптимизаций при отсутствии профильной информации выявились на задачах с большим числом ветвлений, значительная часть которых редко исполняется.

1) Различным образом осуществляется подстановка процедур.

2) Не удается повысить качество конвейеризации циклов с несбалансированными альтернативами классическими методами.

3) Возникают сложности с планированием одновременного исполнения операций соседних веток.

4) Возникает большее число блокировок конвейера при обращениях к памяти.

В дополнение к методам раскрутки цикла (unroll) и конвейеризации (softpipe) для увеличения параллельности вычислений в циклах с неравномерными по счетчикам числа исполнения участками в компиляторах используется преобразование гнездования (nesting). Оно преобразует цикл в гнездо из внутреннего горячего цикла и внешнего цикла с холодными участками, позволяя осуществить конвейеризацию только горячего участка. Однако, оно применимо только к циклам с большой разницей значений счетчиков числа исполнения участков, а в отсутствии информации о профиле вообще не имеет подходящих контекстов. В то же время, применение гнездования дает наибольший эффект в том случае, когда выделяемый горячий участок достаточно короткий по отношению ко всему циклу, то есть когда в цикле есть *несбалансированные ветвления* с точки зрения длин критических путей или числа операций.

С целью увеличения параллельности исполнения сложных циклов с несбалансированными ветвлениями в режиме без профиля или с небольшим отличием значения счетчиков участков автором разработан метод *частичной раскрутки путей исполнения* (short path unroll). Под сложными понимаются циклы с внутренними циклами, вызовами или большим количеством узлов. Метод состоит в выделении части путей от головы цикла до обратных дуг (U) в соответствии с некоторым алгоритмом и копировании их в теле цикла один или более раз. Число созданных копий при этом - это фактор частичной раскрутки k. Применение частичной раскрутки обеспечивает в дальнейшем планирование одновременного исполнения нескольких итераций короткого (простого) участка и одной итерации длинного (сложного). Схема частичной раскрутки цикла проиллюстрирована на Рис 2, где $U = \{A,B,C\}$, фактор раскрутки $k = 1$.

Для предложенного преобразования автором сформулированы утверждения, позволяющие оценить его эффективность (доказательства всех сформулированных автором теорем и лемм приведены в опубликованных работах и в основном тексте диссертации).

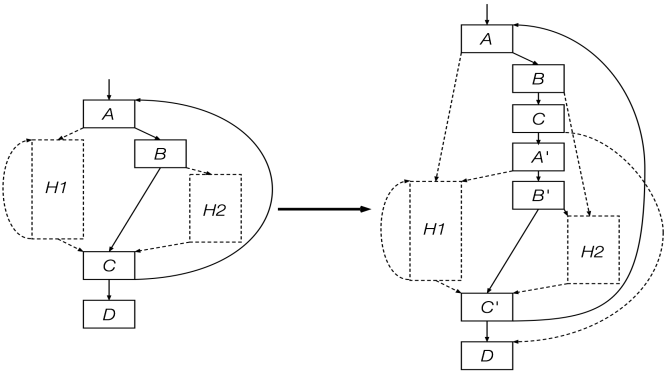

Рис. 2 — Схема применения преобразования раскрутки коротких путей цикла

Теорема. Пусть среднее число итераций исходного цикла равно I_L, полная вероятность пройти от стартового узла до узла с его единственной обратной дугой, не посетив узлы остатка, равна $p(U)$. При этом вероятности всех ветвлений цикла независимы. Тогда среднее число итераций цикла, полученного путем частичной раскрутки с фактором k равно

$$I_{L(U,k)} = \frac{I_L}{1+\sum_{i=1}^{k} d_p(L,U)^i},\ d_p(L,U) = p(U) \cdot \left(1 - \frac{1}{I_L}\right).$$

Теорема. Пусть критический путь выделенного участка $cr(U)$, а критический путь всего цикла cr_L, тогда после частичной раскрутки U с фактором k критический путь нового цикла $\mathrm{cr}'_L = \mathrm{cr}_L(U,k) \leq \mathrm{cr}_L + k * cr(U)$.

При отсутствии ресурсных ограничений планируемое время исполнения неконвейеризируемого цикла равно произведению числа итераций и длины критического пути, поэтому доказанные теоремы позволяют оценить изменение его времени исполнения.

Следствие.

Пусть планирование циклов ограничивается длиной критического пути, то есть ресурсное ограничение не достигается. Тогда времена исполнения исходного цикла t_L и цикла после применения раскрутки $t_L(U,k)$ относятся как:

$$\frac{t_L}{t_L(U,k)} = \frac{\mathrm{cr}_L \cdot I_L}{\mathrm{cr}_L(U,k) \cdot I_{L(U,k)}} = \frac{\left(1+\sum_{i=1}^{k}\left(p(U)\cdot\left(1-\frac{1}{I_L}\right)\right)^i\right)\cdot \mathrm{cr}_L}{\mathrm{cr}_L(U,k)} \quad (1),$$

$$\frac{t_L}{t_L(U,k)} \geq \frac{1+\sum_{i=1}^{k}\left(p(U)\cdot\left(1-\frac{1}{I_L}\right)\right)^i}{1+\frac{k\cdot cr(U)}{\mathrm{cr}_L}} \quad (2),$$

при достаточно больших $I_L \geq 2$ можно еще больше упростить вычисления

$$\frac{t_L}{t_L(U,k)} \approx \frac{1+\sum_{i=1}^{k} p(U)^i}{1+\frac{k\cdot cr(U)}{\mathrm{cr}_L}} \quad (3).$$

Вторая часть предложенных в диссертации преобразований и их настроек при компиляции без профиля касается уменьшения количества блокировок по чтениям. Существующие методы уменьшения количества блокировок по чтениям заключаются в планировании кода с забросом чтений на большое расстояние от использования и в построении дополнительных операций, которые заранее подкачают какие-то участки данных в кэш-память, или prefetch. Однако значительная их часть без достаточно точного профиля не эффективна и используется только по опции из-за сопутствующего увеличения планируемого времени исполнения и дополнительной нагрузки на кэш.

Для использования по умолчанию автором предложены неагрессивные методы оптимизации работы с памятью. Во-первых, предложен метод подкачки нерегулярных чтений в неконвейеризуемых циклах, заключающийся в избирательной подкачке данных по вероятным адресам чтений структур. Во-вторых, предложен метод ограничения расстояния заброса чтений при конвейеризации. Описанный алгоритм позволяет осуществить конвейеризацию с забросом чтений и при этом увеличить спланированное время работы цикла не больше, чем в заданное количество (par) раз.

Алгоритм. Регулируемый заброс чтений.

1: Произвести конвейеризацию цикла без увеличения дистанции для чтений, пусть при этом получается число стадий S.

2: Вычислить $S_{\text{new}} = (I-1)\cdot(\text{par}-1) + S\cdot\text{par}$

3: **Если** $S_{new} > S + 1$, **то** :

4: Вычислить дистанцию заброса load-а:

length = $\min(\text{LongLoad}, S_{\text{new}}\cdot \text{II} - (\text{time}_{\text{back}} - \text{time}_{\text{load}}))$

LongLoad - максимальная задержка при промахе

$\text{time}_{\text{back}}$ - время планирования обратной дуги

$\text{time}_{\text{load}}$ - время планирования load
5: Попробовать спланировать цикл с length.
6: **Если** $S > S_{\text{new}}$ или не хватило регистров, **то** *length=length-1* и **повторить** 5.

Экспериментальные результаты применения предложенных методов показали значительное ускорение ряда приложений и небольшое ускорение исполнения в среднем в режиме однофазной компиляции. Так, преобразование частичной раскрутки показало ускорение до 8,4% на нескольких приложениях пакета CINT, предварительная подкачка данных по вероятным адресам позволила получить до 24,6%, а заброс чтений до 95% ускорения. Несмотря на замедление некоторых приложений, было показано среднее ускорение на 1,85% (CINT) и 0,2% (CFP) для частичной раскрутки, 1,5% (CINT) и 0,92% (CFP) для подкачки нерегулярных чтений, -0,2% (CINT) и 6% (CFP) для заброса чтений. Корреляция эффекта оптимизации со свойствами приложений позволяет сделать вывод о возможности машинного обучения компилятора их более точному применению.

В **третьей главе** решается задача построения единого числового критерия, позволяющего оценить попроцедурное качество компиляции с точки зрения сразу нескольких критериев для пакета задач [7],[11].

Пусть L^* - множество всех компиляторов. Компиляция $l^* \in L^*$ определяется применением линеек из множества всевозможных линеек L к процедурам P, то есть $l^* : P \to L$. По результатам компиляции l^* конечного подмножества процедур $P^t \subset P$, $|P^t| = n$, может быть замерено $r * n$ характеристик $f_j(p_i, l^*)$, $j = 1,...r$. Для рассматриваемой задачи f_j могут означать время исполнения процедуры $exe(p,l^*)$, время ее компиляции $comp(p,l^*)$ или размер полученного кода $size(p,l^*)$.

Задача построения *функционала качества компиляции* - обобщить в единый числовой критерий характеристики $f_j(p_i, l^*)$, то есть построить функцию $F(P^t,l^*) = F'(f_1((p_1, l^*)),f_2((p_1, l^*))...f_r((p_n, l^*)) \to \mathbb{R}$. Для определенности будем считать, что лучшему качеству компиляции l^* при фиксированном P^t соответствует меньшее значение F. Поскольку

пользователь видит результаты работы только на всем приложении целиком, можно считать, что для процедур одного приложения $\forall j$ значения характеристики f_j будут связаны суммой.

Чтобы получить единую числовую оценку производительности компиляторов и процессоров на пакете приложений обычно используется среднее геометрическое значение времен исполнения приложений. Для K приложений с процедурами P^t она выглядит следующим образом:

$$F\left(P^t, l^*\right) = \underset{K}{geomean}\left\{\sum_{ki}\mathrm{exe}\left(p_{\mathrm{ki}}, l^*\right)\right\} \quad (4),$$

где $\underset{K}{geomean}\left\{x_k\right\} = \sqrt[K]{\prod_K (x_k)}$, ki - индексы процедур приложения номер k.

Далее автор переходит к задаче оценки результата компиляции по нескольким характеристикам. Для одного приложения и одновременного учета времени компиляции и времени исполнения это означает построение функции вида $F(f_1(P^t,l^*), f_2(P^t,l^*)) = F'\left(\sum_i exe(p_i,l^*), \sum_i comp(p_i,l^*)\right)$. В общем случае одновременно получить минимум времени компиляции и времени исполнения невозможно. В подобных случаях ищутся *Парето-минимумы*, то есть такие $l^*_m \in L^*$, что не существует $l^* \in L^*$, для которых $f_j(P^t,l^*_m) >= f_j(P^t,l^*) \forall j = 1..k$ и $f_j(P^t,l^*_m) > f_j(P^t,l^*)$ для некоторого j. Переход от поиска Парето-минимумов к построению и поиску минимумов функции $F'(f_1,...,f_k)$ со скалярным значением называют скаляризацией.

В работе проиллюстрировано, что в общем случае скаляризация не позволяет вычислить минимум $l^*_{min} = argmin(F\left(P^t, l^*\right))$, $l^* : P \to L$ как $l^*_{min} = argmin(F_1(l^*(p_1))),...(F_n(l^*(p_n)))$. Таблица 1 описывает разницу оценки качества компиляции для приложений и процедур по одному или нескольким критериям, для удобства отклонение функционала от минимума названо ошибкой.

Далее в работе доказаны утверждения, которые позволяют однозначным образом скаляризовать целевую функцию задачи оптимизации процесса компиляции по временам исполнения и временам компиляции процедур.

Во-первых, доказано общее утверждение, касающееся скаляризации.

Таблица 1 — Оценка качества для процедур пакета приложений

Объекты	**1 приложение**	**K приложений**
1 критерий - время исполнения (4)	попроцедурно вычислимые минимумы, ошибка не зависит от других процедур	попроцедурно вычислимые минимумы, ошибка зависит от других процедур
>1 критерия	Парето-минимумы, и поиск минимумов, и ошибка скаляризованной функции зависят от других процедур	

Теорема. Если скаляризация для многокритериальной задачи представляет из себя монотонную строго возрастающую функцию от критериев, то ее минимум достигается на границе Парето-минимумов для этих критериев.

Во-вторых, доказано, что можно однозначно определить вид функционала качества компиляции. Из-за последующей многократности запусков задач и для кросс-компиляции требуется сохранение выбираемых линеек при пропорциональном изменении времен исполнения или компиляции всех процедур пакета приложений. Это означает *сохранение минимума функционала* на любом подмножестве при растяжениях координат. Для одного приложения это означает, что после растяжений $F'(\sum exe(p_i,l^*), \sum comp(p_i,l^*))$ достигнет минимума при том же назначении l^* .

Сохранение минимума на любом подмножестве равносильно сохранению порядка значений функционала при растяжениях:

$\forall x_1 = \langle x_{11},...x_{1k}\rangle \in \mathbb{R}^k_+, x_2 = \langle x_{21},...x_{2k}\rangle \in \mathbb{R}^k_+$ из $F(x_1) \le F(x_2)$

следует, что

$\forall \alpha > 0, \forall j \in \overline{1,k}$: $F(x_{11},...\alpha x_{1j}...,x_{1k}) \le F(x_{21},...\alpha x_{2j}...,x_{2k})$ (5).

В представленной диссертационной работе описано все множество функций от двух переменных, обладающих этим свойством.

Лемма. Пусть $f(x,y)$— монотонно возрастающая (монотонно убывающая) определенная всюду на $\mathbb{R}^2_+$ функция от двух переменных, представимая в виде $f(x,y) = g(x^k \cdot y^j)$, тогда

1) $g(x)$ - монотонно возрастающая (монотонно убывающая) функция от одного параметра,
2) $f(x,y)$ удовлетворяет свойству (5).

Теорема. Пусть $f(x,y)$— строго монотонная определенная всюду на $\mathbb{R}_+^2$ непрерывная функция от двух переменных, удовлетворяющая свойству (5). Тогда она представима в виде:

$f(x,y) = g(x^q \cdot y), q \in R, q > 0.$

Дополнительно доказана лемма минимумах, позволяющая сократить количество функционалов, которые рассматриваются для поиска оптимального по Парето решения.

Лемма. Пусть определенная всюду на $X \subset \mathbb{R}^k$ функция от k переменных представима в виде $F(x_1, ...x_k) = g\left(x_1^{j1} \cdot ... \cdot x_k^{jk}\right)$ для всех $x = \langle x_1, ...x_k\rangle \in X$, где g - строго возрастающая функция. Тогда, если существует $\min\limits_{x \in X}(F(x))$ (что для конечного X всегда верно), то он достигается в точке минимума $\min\limits_{x=\langle x_1,...x_k\rangle \in X}(x_1^{j1} \cdot ... \cdot x_k^{jk})$.

Утверждение. Из доказанных теорем и лемм следует, что минимумы всех скаляризованных целевых функций процесса компиляции могут быть найдены путем минимизации функционалов вида:

$F(l^*(p_1), ...l^*(p_n)) = (\sum_i \mathrm{exe}(l^*(p_i)))^r * (\sum_i \mathrm{comp}(l^*(p_i)))^j$ (6).

В случае нескольких задач, используя (4), получаем:

$F(l^*(p_1), ...l^*(p_n)) =$

$$\underset{K}{geomean}\left\{\left(\sum_{ki}\mathrm{exe}(l^*(p_{\mathrm{ki}}))\right)^r * \left(\sum_{ki}\mathrm{comp}(l^*(p_{\mathrm{ki}}))\right)^j\right\} \quad (7).$$

При минимизации разных функционалов вида (6) могут быть найдены разные точки, то есть разные Парето-минимумы. Однако, при этом некоторые точки границы Парето не могут быть вычислены. В диссертации автором сформулирована и доказана лемма, которая описывает соответствие Парето-минимумов и возможных минимумов функционалов через их возможные линии уровня:

$x^q \cdot y = c, q \in \mathbb{R}, q > 0$ (8).

Суть леммы проиллюстрирована на Рис. 3 .

Пример применения построенного функционала качества показан для значений параметров r=7, j=1 на компиляторе под архитектуру Эльбрус. В точке его минимума было достигнуто ускорение исполнения на 12,36% и уменьшение времени компиляции в 1,9 раз в среднем на пакетах приложений spec2000 benchmark.

Применение построенного интегрального функционала качества оправдано для выбора линейки, для мониторинга качества

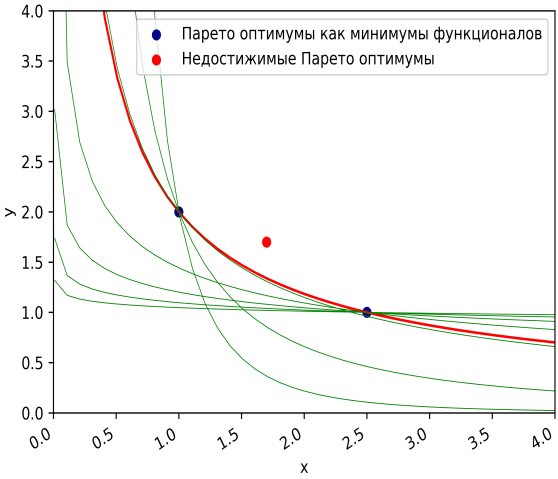

Рис. 3 — Достижимые Парето оптимумы.

разрабатываемого компилятора и для настройки отдельных преобразований. Так, в [8] функционал для одного приложения был использован для оптимизации подстановки процедур.

В **четвертой главе** формулируется и решается задача машинного обучения для минимизации функционала качества.

Пусть, как и выше, P - множество процедур, L^* - множество компиляторов, компиляция $l^* \in L^*$ определяется применением линеек из множества линеек L к процедурам P, то есть $l^* : P \to L$. И пусть задан функционал качества, определенный на каждом конечном конечном подмножестве $P^t \subset P$, $l^* \in L^*$: $F(P^t, l^*) \to \mathbb{R}$. Без ограничения общности можно считать, что L, и как следствие L^*, тоже конечно.

Задача попроцедурного выбора оптимизационной линейки состоит в том, чтобы найти алгоритм

$$a : P \to L \text{ такой, что } F(P,a) = F\left(a\left(p_1\right)...a\left(p_n\right)\right) \to \min_{a \in A}.$$

Поставленная задача наиболее близка к задаче классификации, в которой для множества объектов X определено множество допустимых ответов Y, и есть целевая функция $y^* : X \to Y$, значения которой известны только на конечном подмножестве X^l объектов. При этом требуется найти алгоритм $a : X \to Y$, приближающий y^* на всем X. Для измерения ошибки классификации используется эмпирический

риск, который представляет собой средневзвешенную по объектам ошибку алгоритма:

$$Q(a,X^l) = \frac{1}{n}\sum_{p\in P}\lambda(a,x), \text{ где } n = |X^l|.$$

В диссертационной работе показано, что *задача минимизации функционала качества* существенно отличается от задачи классификации, поскольку: не определены, и в общем случае не определяемы, верные классы для отдельных объектов; для неаддитивного по объектам функционала качества, каковым является функционал качества компиляции, нельзя вычислить ошибку алгоритма для отдельного объекта. Поэтому автор вводит другой класс задач машинного обучения.

Определение. Пусть задано множество объектов X и множество допустимых ответов Y, а $Y^* : X \to Y$ - всевозможные отображения из X в Y. И пусть на любых $X^t \subset X$, $y^* \in Y^*$ определена функция $F(X^t, y^*) \to \mathbb{R}$, значения которой известны на обучающей выборке $\left(X^l, Y^*\right) \subset (X, Y^*)$. Задача *минимизирующей классификации* состоит в поиске алгоритма $a : X \to Y$, минимизирующего F на всем X. Для оценки качества алгоритма используется функционал потери, означающий отклонение функционала от минимума на объектах обучающей выборки:

$$F(a) = F\left(X^l, a(X^l)\right) - \min_{y^*\in Y^*}\left(F(X^l, y^*)\right).$$

Для решения задачи минимизирующей классификации автором предложено два метода.

Первый - классификация по функционалу качества. Предлагаемый метод можно отнести к алгоритмам логической классификации. В этом случае разрабатывается набор правил $\{\phi_k\}$, то есть логических формул или предикатов, позволяющий отделять объекты одного класса от объектов других классов. Говорят, что правило ϕ_k покрывает x, если $\phi_k(x) = 1$. Для минимизирующей классификации вычислить информативность правила для класса невозможно, поэтому автор вводит понятие относительной информативности.

Определение. Пусть все объекты уже отнесены к некоторым классам, тогда *относительная информативность правила* ϕ для класса c вычисляется как изменение значения функционала F

при назначении класса c объектам, которые покрываются правилом ϕ. Для использования относительной информативности необходимо назначить всем объектам некоторый исходный класс, для определенности будем его называть *базовый класс*.

Далее описан алгоритм, заключающийся в поиске на каждом шаге правила с наибольшей относительной информативностью. Успешно найденное правило изменяет класс для покрываемой им области. Предлагаемый алгоритм напоминает взвешенное голосование или бустинг, но под общее разбиение на классы подбирается не вес правила, а само правило.

Алгоритм. Построение последовательности закономерностей при классификации по функционалу качества (КФК):

1: назначить всем объектам базовый класс

2: подсчитать значение функционала F для обучающей выборки

3: **повторять** пока удается найти предикат φ с положительной относительной информативностью (для класса C)

4: добавить φ_k с самой высокой относительной информативностью в последовательность $\varphi_1,...\varphi_{(k-1)}$

5: назначить всем объектам $x : \varphi_k(x) = 1$, класс C

6: вычислить новое значение функционала F

7: $k{+}{+}$

Чтобы определить класс объекта x, надо найти:

$\phi = \underset{j<=k}{max}\{\phi_j : \phi_j(x) = 1\}$.

Для описания предлагаемого алгоритма поиска закономерности вводится понятие *матрицы частичных ошибок*, содержащей отклонения функционала от минимального значения при изменениях координат *одной из точек* достижения минимума.

Пронумеруем $y_1,...y_k \in Y$, $x_1,...x_n \in X$. По обучающей выборке вычислим одну из точек, на которой функционал качества достигает минимума:

$F_{min} = F(b_1,...b_n) = \underset{(c_1,...n)\in Y^n}{min}(F(c_1,...c_n))$.

Тогда элемент (i,j) матрицы размера $n \times k$ вычисляется как:

$$\partial(x_i,y_j) = \log\left(\frac{F(b_1,...,b_{(i-1)},y_j,b_{(i+1)},...b_n)}{F_{\min}}\right).$$

Для поиска предиката с высокой относительной информативностью автором предложен алгоритм двоичного покоординатного поиска области, содержащей объект с высокой частичной ошибкой.

Теорема: Алгоритм КФК с предложенным поиском правил останавливается. Если на каждом шаге получится подобрать правило с положительной относительной информативностью по функционалу, то алгоритм КФК за конечное число шагов приведет к разбиению на классы, соответствующему нулевой потере.

Второй предложенный метод решения задачи минимизирующей классификации состоит в том, чтобы перейти к обычной задаче классификации с аддитивным функционалом ошибки с минимумом в одном из минимумов исходного функционала. В работе осуществляется такой переход с помощью матрицы частичной ошибки и минимума, по которому она построена: положим $y^*(x_i) = b_i$ - известные значения целевой функции, $W = \sum \partial(x_i,a(x_i))$ - аддитивный функционал потери.

Такой переход можно использовать для применения классических методов классификации - нейросетей, логических алгоритмов, метода опорных векторов и так далее. Кроме того, может быть осуществлен временный переход к использованию такой ошибке при попадании к локальные минимумы описанной выше классификации.

Лемма. Если минимум единственный, то точная минимизация суммы частичных ошибок W совпадёт с результатом точной минимизации функционала.

Лемма. Если минимум не аддитивного функционала потери единственный и для всех объектов выборки отличается их признаковое описание, то при применении Алгоритма КФК к частичной ошибке возможно выбрать такие предикаты для классов, что за конечное число шагов будет достигнут минимум .

Замечание: 1) Если минимум функционала не единственный, то минимум частичной ошибки может не быть минимумом функционала. 2) Отдельный шаг классификации в сторону уменьшения частичной ошибки может приводить к увеличению значения функционала потери даже при единственности минимума.

Почти независимым от выбора модели обучения этапом является выбор признаков, описывающих объекты: $(r_1(x),...r_R(x))$. Для рассмотренной задачи такие признаки - это вычислимые характеристики процедур по их промежуточному представлению после

межпроцедурной оптимизации. Коррелирующими с результатами компиляции линейками признаками оказались общее количество операций в процедуре, количество различных операций в процедуре, оценочное время исполнения процедуры, плотность процедуры, количество различных операций с учетом счетчиков числа исполнения, максимальная глубина вложенности циклов и среднее количество операций в узлах управляющего графа.

Экспериментальные результаты применения компиляции с автоматическим выбором оптимизационной линейки с построением 10 областей для классов, что соответствует максимум 140 сравнениям или базовым правилам, показаны на пакетах приложений spec2000. Для задач с целыми вычислениями (пакет CINT) было получено среднее ускорение времени исполнения на 2,9% и времени компиляции на 26,7%. Задачи пакета CFP, т.е. задачи с плавающими вычислениями, в среднем ускорились на 8,2% по исполнению, а среднее время компиляции сократилось на 6,7%. В качестве тестовых приложений, не входящих в обучающую выборку, был использован пакет spec95. На нем было получено ускорение производительности на 35% и компиляции на 94% от ускорения, полученного на обучающем пакете [6].

Также было проведено сравнение минимума и результата машинного обучения. Поскольку значительная часть ускорения времени компиляции в теоретической оптимальной точке оказалась вызвана понижением уровня компиляции неисполняемых процедур, то для сравнения использовался функционал только с временем исполнения. В этом случае при осуществлении 7-ми шагов классификации было достигнуто 88% возможного ускорения приложений в среднем [2].

В **заключении** приводятся основные результаты диссертационной работы и направления дальнейших исследований.

Основные результаты диссертационной работы:

1. Предложены преобразования частичной раскрутки критических путей и длин рекуррентностей цикла, позволяющие лучше спланировать сложные циклы с разновесными ветвлениями. Для обоих методов доказаны утверждения об оценке их эффективности.

2. Предложены неагрессивные методы оптимизации работы с памятью, применимые в режиме по умолчанию: метод увеличения дистанции от чтений с ограничением планируемого времени исполнения цикла и метод построения подкачки структур в циклах.
3. Предложен механизм автоматического выбора оптимизирующей последовательности преобразований для каждой процедуры на раннем этапе компиляции.
4. Предложена числовая многокритериальная оценка качества компиляции, учитывающая попроцедурный характер оптимизации. Показано ее преимущество по сравнению с вероятностной при решении задач машинного обучения компиляторов.
5. Сформулирована и доказана теорема о представлении всех функций двух переменных, обладающих свойством сохранения порядка при растяжениях. Доказано утверждение о представлении функционала качества компиляции, учитывающего время исполнения и время компиляции.
6. Сформулирована задача минимизирующей классификации, соответствующая задаче машинного обучения компилятора с построенным функционалом качества. Предложены методы решения задачи минимизирующей классификации, доказаны утверждения об их сходимости.

Представленные методы использованы при разработке оптимизирующего компилятора с языков C, C++ и Фортран для микропроцессоров «Эльбрус». Суммарное ускорение от совокупности примененных методов в базовом режиме компиляции составило в среднем около 6,1% для задач с целыми вычислениями и около 15% для задач с плавающими вычислениями. При этом время, требующееся для компиляции приложений с целыми вычислениями, сократилось больше, чем на 20%, а с плавающими - больше, чем на 6%.

Список работ автора, опубликованных по теме диссертации

[1] Четверина О.А. "Сравнение и выбор эффективной стратегии компиляции". Труды 56-й научной конференции МФТИ, Радиотехника и кибернетика, 2013. С. 66-68.

[2] Четверина О.А. "Статический выбор эффективной стратегии оптимизации кода.". Национальный Суперкомпьютерный Форум НСКФ-2013, 2013.

[3] Четверина О.А., Нейман-заде М. И., Степанов П. А. "Система направленной оптимизации (СНОП)". I Всероссийская научно-техническая конференция "Расплетинские чтения". 2014. С. 190-191.

[4] Четверина О.А., Степанов П. А., Нейман-заде М. И. "Автоматическая направленная оптимизации процедур (СНОП)". Вопросы радиоэлектроники. 2015. № 3. С. 64-76.

[5] В.Ю. Волконский, А.В.Брегер, А.Ю.Бучнев, А.В.Грабежной, А.В.Ермолицкий, Л.Е.Муханов, М.И.Нейман-заде, П.А.Степанов, О.А.Четверина. "Методы распараллеливания программ в оптимизирующем компиляторе". Вопросы радиоэлектроники. 2012. Т. 4. № 3. С. 63-88.

[6] O.A. Chetverina. “Procedures classification for optimizing strategy assignment”. Proceedings of the Institute for System Programming, Volume 27 (Issue 3), 2015, pp. 87-100.

[7] Четверина О.А. “Сравнение использования различных функционалов в пространстве назначения стратегий оптимизации”. II Международная конференция Инжиниринг & Телекоммуникации — En&T 2015, Тезисы докладов, 2015. С. 201-202.

[8] А. В. Ермолицкий, М. И. Нейман-заде, О. А. Четверина, А. Л. Маркин, В. Ю. Волконский. "Агрессивная инлайн-подстановка функций для VLIW-архитектур". Труды Института системного программирования РАН. 2015. Т. 27. № 6. С. 189-198.

[9] Четверина О.А., "Методы коррекции профильной информации в процессе компиляции". Труды Института системного программирования РАН. 2015. Т. 27. № 6. С. 49-66.

[10] Четверина О. А. "Повышение производительности кода при однофазной компиляции". Программирование, 2016, № 1. С. 51-59. (O. A. Chetverina, Alternatives of profile-guided code optimizations for one-stage compilation, Programming and Computer Software, January 2016, Volume 42, Issue 1, pp 34–40.)

[11] Четверина О.А., Нейман-заде М.И., Степанов П.А., Линчик М.И. "Система выбора и использования индивидуальных последовательностей оптимизаций для компиляции процедур оптимизирующим компилятором под архитектуру Эльбрус". Свидетельство о государственной регистрации программы для ЭВМ № 2018666284 от 13.12.2018.