УДК 004.4'416
DOI: 10.25559/SITITO.17.202103.593-602

Научная статья

Мультипрофильная статическая бинарная оптимизация приложения на основе трасс исполнения

С. А. Лисицын[1,2]
[1] Российский исследовательский институт Huawei, г. Москва, Российская Федерация
121614, Российская Федерация, г. Москва, ул. Крылатская, д. 17, корп. 2
[2] ФГАОУ ВО «Московский физико-технический институт (национальный исследовательский университет)», г. Долгопрудный, Российская Федерация
141701, Российская Федерация, Московская область, г. Долгопрудный, Институтский переулок, д. 9
lisitsyn.sergey@huawei.com

Аннотация

Профилирование – сбор характеристик об исполнении программ, которые можно использовать во время оптимизаций. Одним из примеров использования является статический бинарный оптимизатор BOLT. Основной прирост производительности в нём достигается за счёт перекомпоновки кода на основе полученного во время исполнения приложения профиля. Однако, реализована данная оптимизация без учета переключения между сценариями приложения, когда происходит смена собираемых характеристик программы во время её исполнения. Это приводит к тому, что при исполнении приложения на сценарии, отличном от профилируемого, прирост производительности будет меньше, либо будет наблюдаться регрессия. Объединение всех возможных профилей программы в один не решает все возможные проблемы. В данной работе описывается алгоритм мультипрофильного анализа кода, учитывающий смену сценариев работы программы. Данный алгоритм основан на обработке трасс исполнения приложения, собираемых с помощью динамической бинарной инструментации. На основе этого анализа можно проводить оптимизацию по размещению кода в зависимости от его принадлежности определенному сценарию, что будет приводить к повышению локальности кода. Также на основе мультипрофильного анализа можно проводить оптимизацию, производя дублирование кода, попадающего одновременно в несколько сценариев.

Ключевые слова: бинарная оптимизация, симуляция, моделирование, динамическая бинарная инструментация, профилирование, RISC, BOLT, ARM

Автор заявляет об отсутствии конфликта интересов.

Для цитирования: Лисицын, С. А. Мультипрофильная статическая бинарная оптимизация приложения на основе трасс исполнения / С. А. Лисицын. – DOI 10.25559/SITITO.17.202103.593-602 // Современные информационные технологии и ИТ-образование. – 2021. – Т. 17, № 3. – С. 593-602.

© Лисицын С. А., 2021

Контент доступен под лицензией Creative Commons Attribution 4.0 License.
The content is available under Creative Commons Attribution 4.0 License.

Original article

Multi-Profile Static Binary Optimization of the Application Based on Execution Traces

S. A. Lisitsyn[a,b]
[a] Huawei Russian Research Institute, Moscow, Russian Federation
17 Krylatskaya St., building 2, Moscow 121614, Russian Federation
[b] Moscow Institute of Physics and Technology (National Research University), Dolgoprudny, Russian Federation
9 Institutskiy per., Dolgoprudny 141701, Moscow Region, Russian Federation
lisitsyn.sergey@huawei.com

Abstract
Profiling is the collection of characteristics about the execution of programs that can be used during optimizations. One example of usage is the BOLT static binary optimizer. The main performance gain in it is achieved by recomposing the code based on the profile obtained during application execution. However, this optimization is implemented without considering switching between application scenarios when there is a change in the collected characteristics of the program during its execution. This leads to the fact that when the application is executed on a scenario other than the profiled one, the performance gain will be less, or regression will be observed. Combining all possible program profiles into one does not solve all possible problems. This paper describes an algorithm for multi-profile code analysis, considering the change of scenarios of the program. This algorithm is based on processing application execution traces collected using dynamic binary instrumentation. Based on this analysis, it is possible to optimize the placement of code depending on its belonging to a certain scenario, which will lead to an increase in the locality of the code. Also, based on multi-profile analysis, it is possible to optimize the duplication of code that falls into several scenarios simultaneously.

Keywords: Binary optimization, simulation, dynamic binary instrumentation, profiling, RISC, BOLT, ARM

The author declares no conflict of interest.

For citation: Lisitsyn S.A. Multi-Profile Static Binary Optimization of the Application Based on Execution Traces. *Sovremennye informacionnye tehnologii i IT-obrazovanie = Modern Information Technologies and IT-Education.* 2021; 17(3):593-602. DOI: https://doi.org/10.25559/SITITO.17.202103.593-602

1. Введение

В настоящее время крупномасштабные вычисления концентрируются в центрах обработки данных, поэтому задачи оптимизации запускаемых на них программ является актуальной. Размер кода приложений для данных центров значительно превышают размер кэша инструкций, что делает задачу компоновки кода важной оптимизацией для повышения их производительности [1], [2].
Компоновка первый раз происходит после компиляции исходного кода, когда компилятор не знает распределения горячего и холодного кода. Предварительно проинструментировав программу, можно собрать профиль исполнения и подать его компилятору для учета при компоновке. Альтернативный вариант – собрать информацию исполнения без инструментации полученного бинарного файла и произвести его оптимизирующую двоичную трансляцию с перекомпоновкой горячего и холодного кода [3].
Примером такого транслятора является BOLT (Binary Optimization and Layout Tool) [4]. Основной его оптимизацией является перекомпоновка линейных участков кода на основе информации исполнения, а именно, статистиках использования кода и условных переходов. За счёт этой информации повышается локализация кода [5]. Однако BOLT рассматривает профиль как статичную картину исполнения без временной характеристики. Учёт профильной информации, разбитой на промежутки времени, может позволить выделить основные сценарии работы приложения и дополнительно повысить локальность кода.
В первой части статьи описываются проблемы профильных оптимизаций, с которыми сталкиваются профилировщики. Далее проводится описание предложенного метода выделения мультипрофильности на основе трассы исполнения приложения. По результатам работы данного алгоритма получилось убрать регрессии, полученные на оптимизации с одним профилем.

2. Проблема оптимизации по профилю

Профилирование программы – это сбор характеристик работы программы. В случае работы оптимизатора BOLT необходимы значения счетчика команд и последние взятые переходы с информацией от предсказателя переходов. Во время сбора характеристик программа исполняется по определенному сценарию: определенные входные данные, параметры окружения, контекст и т.д. Полученная профильная информация будет показывать характеристики этого выбранного сценария.
После оптимизации приложения с данным профилем, запуск по данному сценарию будет показывать прирост производительности. Но при проверке производительности с другими входными параметрами на данном приложении такого же прироста производительности получить не удастся, вероятнее будет происходить регрессия.

2.1 Пример приложения с множественными сценариями

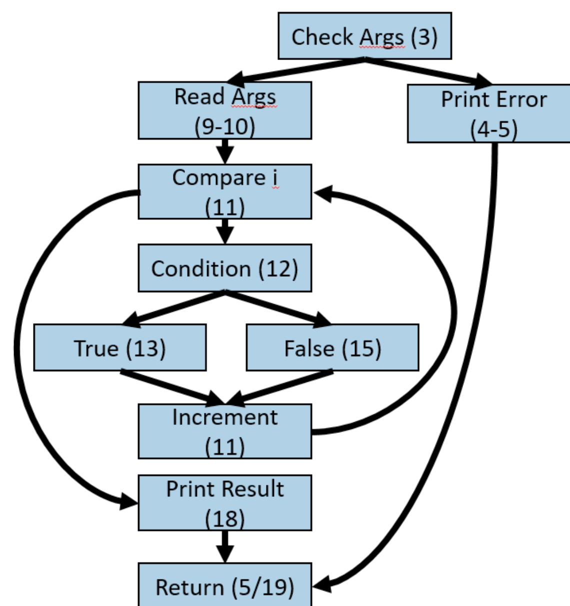
Рассмотрим пример приложения, который имеет два сценария исполнения.

```
1.  int main(int argc, char** argv)
2.  {
3.    if (argc != 3) {
4.      printf("Need two arguments\n");
5.      return 1;
6.    }
7.    double res = 0.0;
8.    uint32_t TRUE = 0, FALSE = 0;
9.    uint32_t arg = atoi(argv[1]);
10.   uint32_t itNumber = atoi(argv[2]);
11.   for (uint32_t i = 1; i < itNumber; i++) {
12.     if (condition(i, arg)) {
13.       res += trueFunc(i, &TRUE);
14.     } else {
15.       res += falseFunc(i, &FALSE);
16.     }
17.   }
18.   printf("[T:%d | F:%d] %lf\n", TRUE, FALSE, res);
19.   return 0;
20. }
```

Р и с. 1. Пример оптимизируемого кода

F i g. 1. Optimized code example

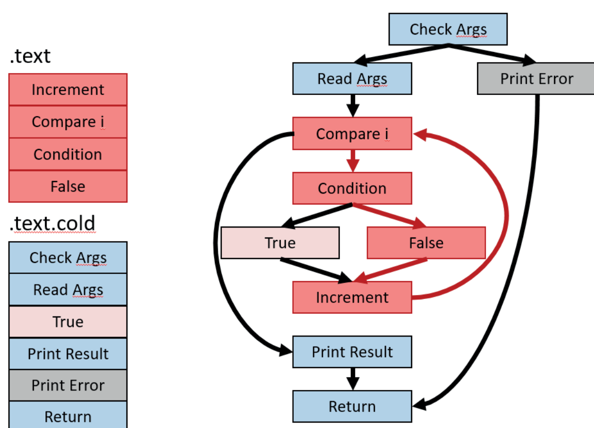
Его граф потока управления будет состоять из десяти линейных участков и выглядеть следующим образом:



Р и с. 2. Граф потока управления

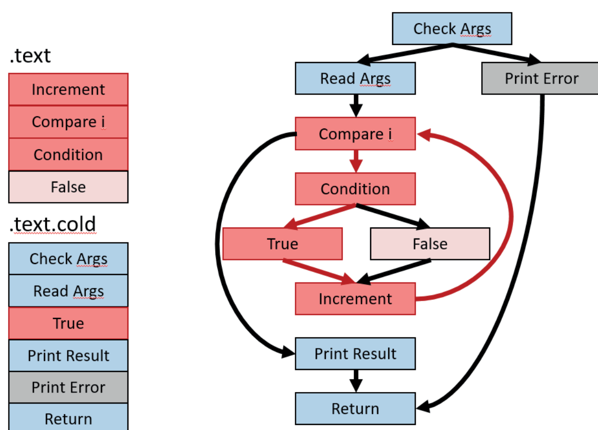
F i g. 2. Control flow graph

После сбора профилировочной информации и прохождений оптимизаций BOLT расположит часто использованные линейные участки в секции кода .text, остальные будут размещены в секции кода .text.cold.



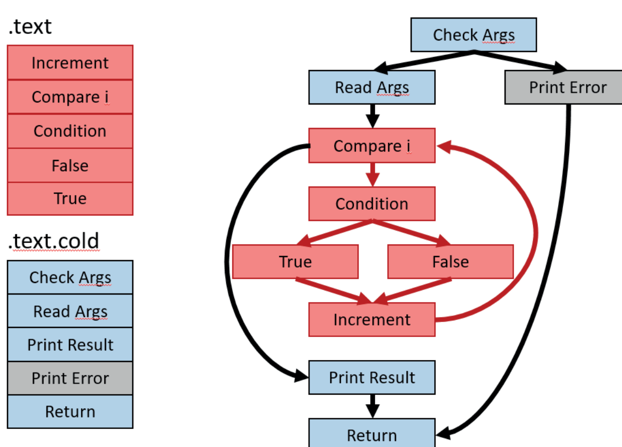
Р и с. 3. Граф потока управления с профилем
F i g. 3. Control flow graph with profile

Для собранного профиля приложение стало более производительным за счёт расположения всего горячего в одном месте. Однако, если исполнение пойдет по другому пути, то производительность понизится из-за перехода в горячий линейный участок «True», расположенный в другой секции:



Р и с. 4. Граф потока управления с измененным профилем
F i g. 4. Control flow graph with modified profile

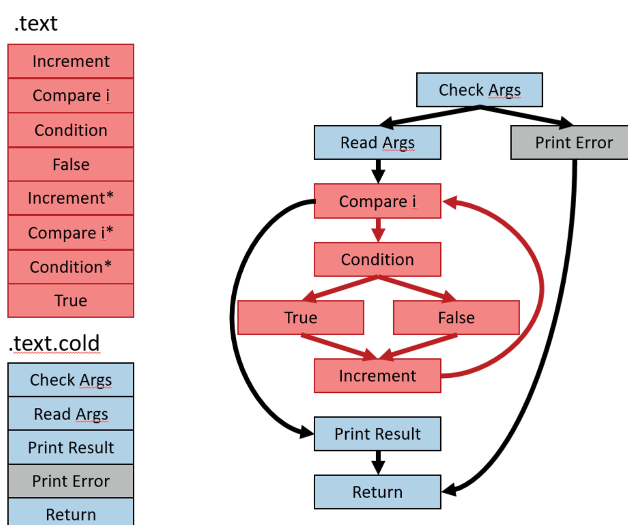
Для решения данной проблемы необходимо покрыть все возможные сценарии работы нашего приложения. В итоге, получив запуски с горячими линейными участками как «True», так и «False», суммарная профильная информация позволит BOLT расположить обе ветки исполнения в оптимизированной секции вместе:



Р и с. 5. Граф потока управления с суммарным профилем
F i g. 5. Control flow graph with summary profile

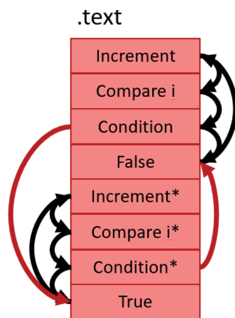
Теперь выделенными в горячую секцию оказываются два сценария. Если при запуске используется только один сценарий, то второй будет занимать место в оптимизированной секции. И учитывая расположение в новой секции линейных участков, предпочтительнее использование «False» ветки, так как она лежит непосредственно после участка «Condition». В случае с веткой «True» будет происходить захват линейного участка «False».

Для того, чтобы это не происходило, необходимо произвести копирование горячих линейных участков, относящихся к различным сценариям. В данном примере это линейные участки «Increment» «Compare i» «Condition». Будут созданы их копии «Increment*» «Compare i*» «Condition*», относящиеся к сценарию «True».



Р и с. 6. Оптимизированная секция с дублированием кода
F i g. 6. Optimized section with code duplication

После всех преобразований полученная оптимизированная секция будет содержать в себе два не пересекающихся сценария. Рассмотрим, как будет происходить выбор исполняемого сценария. После запуска приложения, проверки и считывания аргументов исполнение переходит в линейный участок «Compare i» произвольного сценария, для определенности «False». После чего в линейном участке «Condition» будет происходить выбор нужной ветки исполнения. Это приведет к выбору сценария, по которому пойдёт дальнейшее исполнение.

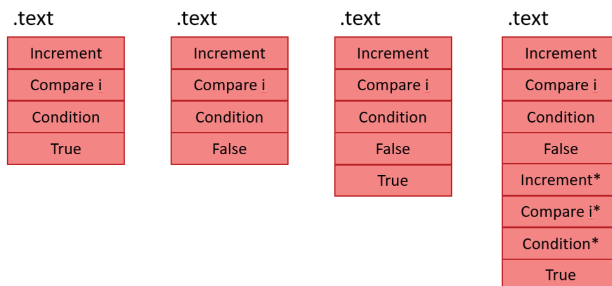


Р и с. 7. Поток управления по секции с дублированием кода

F i g. 7. Section control flow with code duplication

На рисунке красными стрелками отображены переходы между сценариями. Из-за дублирования кода размер секции стал больше, но таким образом получилось изолировать сценарии друг от друга. Однако, если будут частые переключения между ними, то в итоге, количество постоянно исполняемого кода будет включать себя и оригинальные линейные участки, и их дубликаты. Самый негативный случай – перемежение сценариев «True» и «False». В этом случае последовательность исполнения будет следующая: «Condition» => «True» => «Increment*» => «Compare i*» => «Condition*» => «False» => «Increment» => «Compare i» => «Condition» => «True» => ...

В итоге получается 4 варианта реализации оптимизированной секции:



Р и с. 8. Варианты оптимизированных секций

F i g. 8. Optimized Section Options

Первые два варианта подходят только в том случае, если есть уверенность в отсутствии запусков сценариев, не помещенных в секцию. При наличии исполнении обоих сценариев и частом переключении между ними необходимо использовать третий вариант секции. И если переключения между сценариями редкие, то необходимо использовать 4 вариант, вариант оптимизированной секции с дублированием общего кода.

На практике можно реализовать несколько вариантов в бинарном файле и поставить специальные счетчики, которые будут перенаправлять между вариантами оптимизированных секций, но данный вариант в статье рассматриваться не будет. Приведенный пример содержит в себе мало исполняемого кода, поэтому дублирование кода не сильно увеличит размер секции. Но если рассмотреть примеры реальных приложений, пересечение между сценариями могут покрывать больше половины секции кода. В этом случае анализ сценариев исполнения и доказательство необходимости дублирования код становится важной задачей для повышения производительности приложения.

3. Выделение многопрофильности

Для проведения анализа приложения необходимы трассы исполнения, собранные с помощью динамической бинарной инструментации [6], [7], покрывающие большинство сценариев использования приложения. В конечном итоге необходимо получить соотношение кода с выявленными сценариями. Для оперирования с первой величиной будет использоваться понятие линейного участка кода, или базовый блок. Для действий, связанные со сценариями, будет использоваться понятие фазы из статьи [8]. В итоге, в ходе анализа многопрофильности необходимо построить соответствие между линейными участками и выявленными фазами.

3.1 Интервалы инструкций

Разобьём трассу исполнения на одинаковые интервалы по количеству исполненных инструкций. Это необходимо для понимания общей ситуации исполнения в рамках одного отрезка времени – интервала инструкций. Необходимо найти похожие интервалы и объединить их в фазы.

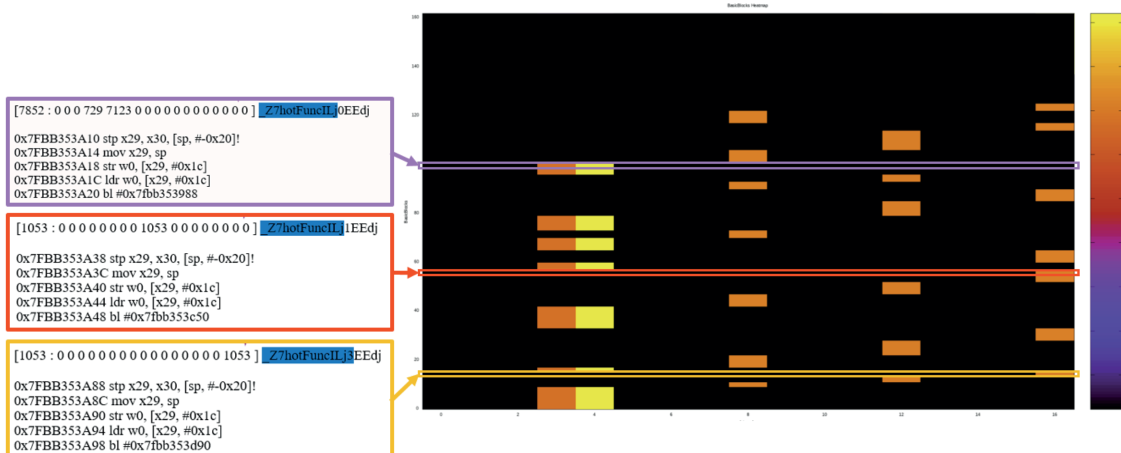
Для большей наглядности есть возможность отобразить граф исполнения в тепловую карту. По оси X – интервалы согласно времени исполнения, по оси Y – номер линейного участка кода.

3.2 Фаза исполнения

На каждом интервале исполнялся определенный набор линейных участков кода, который будет характеризовать интервал инструкций. Если составить n-мерное пространство, где n – количество линейных участков кода, а координаты – количество исполненных соответствующих линейных участков на выбранном интервале, то каждому интервалу в данном пространстве будет соответствовать точка.

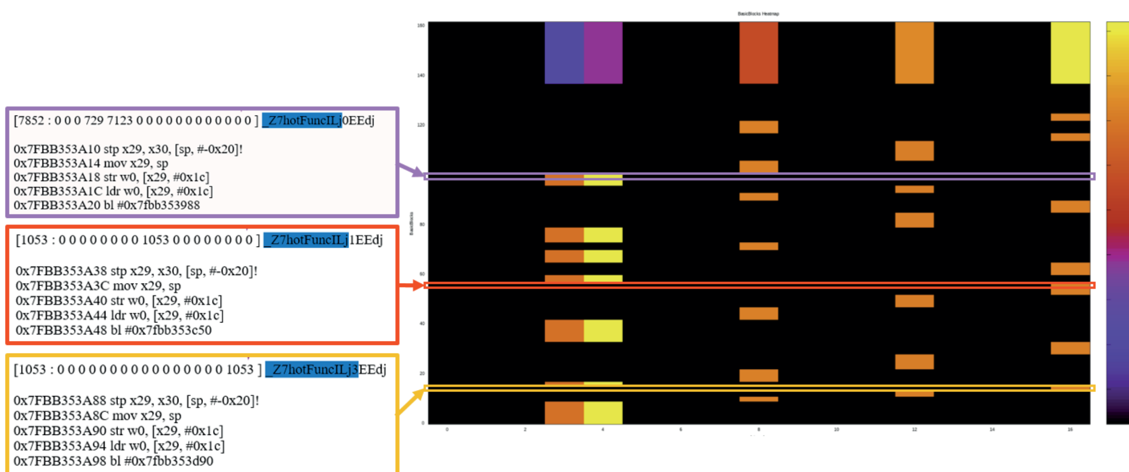
Необходимо произвести кластеризацию интервалов, каждый полученный кластер интервалов инструкций будет означать определенную фазу. Кластеризация будет проводиться с помощью итеративного алгоритма объединения фаз [9]-[18].

В начале алгоритма каждый интервал считается отдельной фазой. Вычисляются расстояния между текущими фазами и выбирается наименьшее из полученных, выбранная пара ближайших фаз объединяются. После чего итерация повторяется до тех пор, пока не останется необходимое количество фаз. Данный алгоритм является по сути жадной кластеризацией, параллельная версия которой была реализована в рамках проделанного исследования.



Р и с. 9. Построение тепловой карты линейных участков

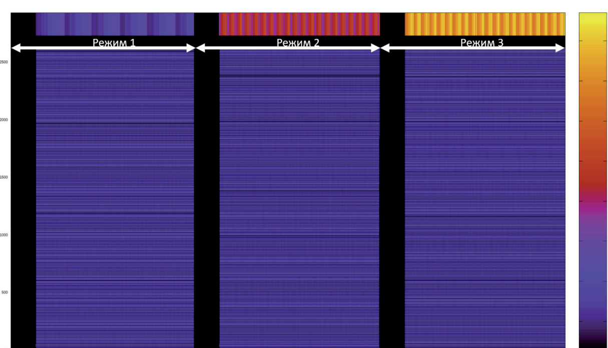
F i g. 9. Construction of a heat map of linear sections



Р и с. 10. Добавление фаз исполнения в тепловую карту (верхняя строка тепловой карты)

F i g. 10. Adding execution phases to the heatmap (top row of the heatmap)

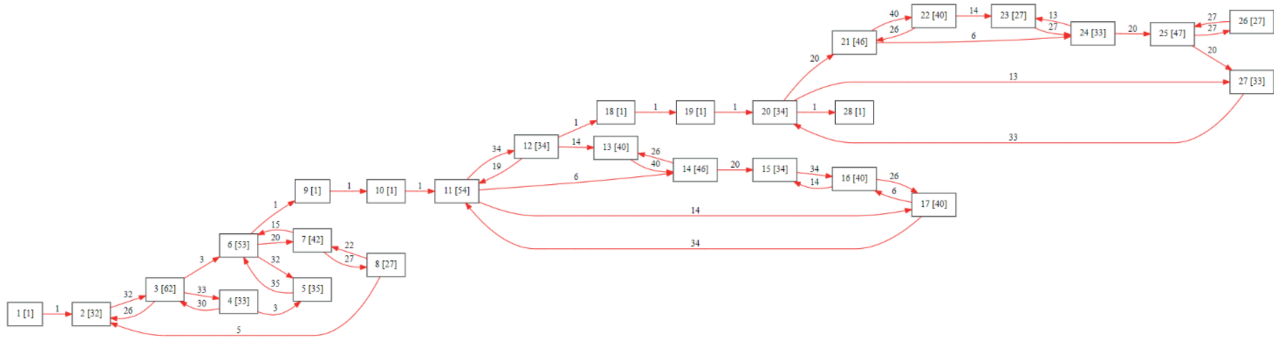
После выделения фаз начинается стадия их анализа.



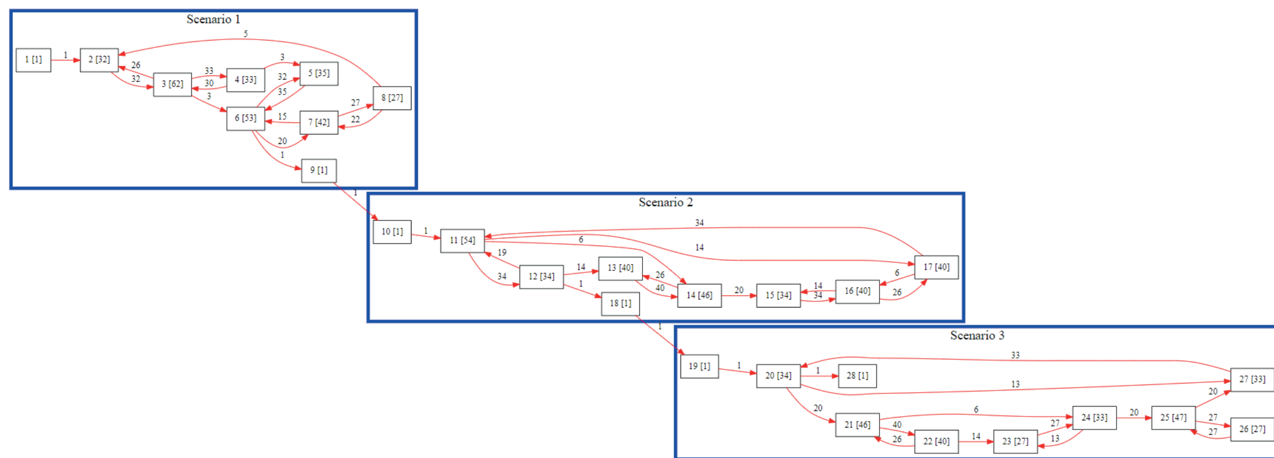
Р и с. 11. Тепловая карта приложения, запущенного в трех разных режимах

F i g. 11. Heat map of an application running in three different modes

По переключениям между фазами строится граф исполнения. Он является аналогом графа потока управления, но на более высоком уровне, показывая переключения не между линейными участками, а между макросостояниями исполнения программы.



Р и с. 12. Граф потока управления по фазам
F i g. 12. Control flow graph by phases

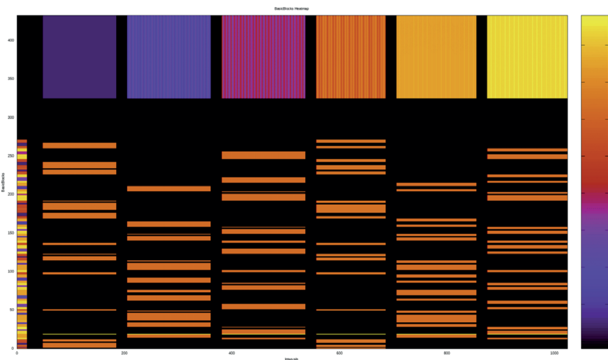


Р и с. 13. Соответствие графа потока управления по фазам со сценариями
F i g. 13. Correspondence of the control flow graph by phases to scenarios

Получив граф фаз состояний можно проанализировать сценарии приложения и выделить характерные особенности, используемые в последствии в оптимизациях.
Если произвести сравнение фаз исполнения со сценариями исполнения приложения, то заметна корреляция, когда на каждый запуск приходятся свои собственные фазы, без пересечения.
Однако, это синтетический пример, поэтому пересечения между сценариями исключены, что позволяет провести оптимизацию перекомпоновки кода без применения дублирования. В реальных приложениях пересечения могут встречаться и для повышения локальности кода необходимо будет дублировать часть кода.

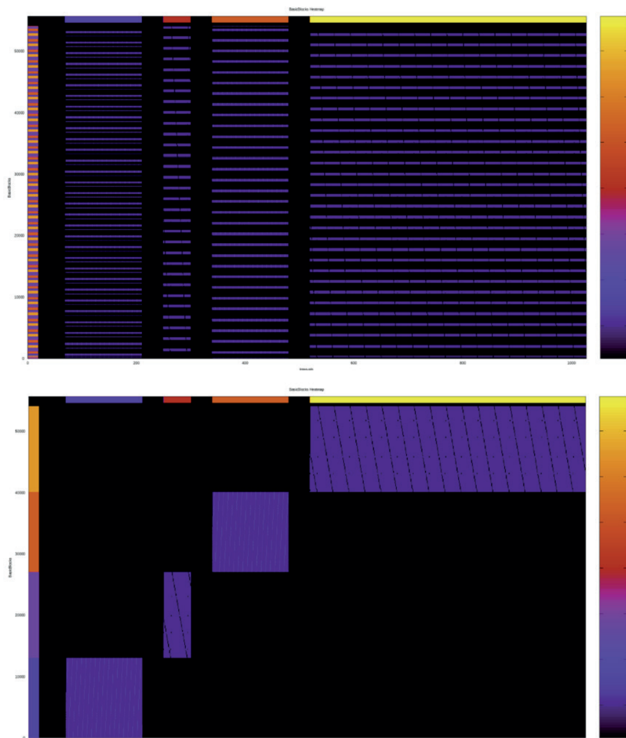
3.3. Мультипрофиль

Для завершения анализа приложения необходимо построить соответствие между фазами и линейными участками приложения. Для этого каждый линейный участок анализируется отдельно, выбирая интервал с его наибольшим количеством исполнений. Линейному участку кода будет соответствовать фаза, которая включает себя этот интервал.
В итоге каждому линейному участку кода соответствует фаза, обозначаема на тепловой карте в столбце слева.



Р и с. 14. Пример построения соответствия линейных участков фазам
F i g. 14. An example of building a correspondence between linear sections and phases

Получив соответствие линейных участков кода фазам, расположение в оптимизированной секции кода будет производиться, размещая весь код в одной фазе. Для этого производится модификация профильной информации для бинарного оптимизатора BOLT, прибавляя константу смещения между фазами. Таким образом бинарные функции одной фазы вероятнее окажутся на соседних адресах в оптимизированном бинарном файле [19]-[24].
Если рассмотреть синтетический тест с разделенными сценариями использованию, то после анализа и оптимизации код расположится в оптимизированной секции таким образом, что регионы исполнения сценариев не будут пересекаться.



Р и с. 15. Пример тепловых карт приложения до (сверху) и после (снизу) оптимизации
F i g. 15. An example of application heatmaps before (top) and after (bottom) optimization

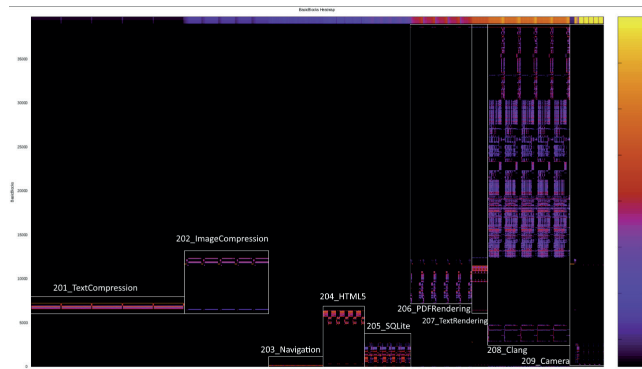
Без проведения мультипрофильного анализа бинарный оптимизатор BOLT имеет профильную информацию, по которой можно получить подобный результат, но на более сложных примерах восстановление глобального потока управления становится не подъемной для BOLTа задачей.

3.4 Дублирование кода

Выстраивая соответствие между линейными участками кода и фазами, выбиралась фаза, к которой принадлежит интервал инструкций с наибольшим количеством исполнений данного линейного участка. Однако, может сложиться ситуация, когда линейный участок одинаково часто исполнялся на интервалах инструкций, относящихся к разным фазам. В этом случае необходимо запоминать такие линейные участки для последующего анализа необходимости дублирования данного кода [25].
Используя эвристически подобранные соотношения выбираются те линейные участки, которые относятся одновременно к нескольким фазам, после чего эти участки помечаются для дублирования бинарным оптимизатором BOLT.
На данный момент идёт реализация подобной оптимизации, поэтому последующие результаты приведены только для примеров с модифицированным на основе мультипрофильного анализа профиля.

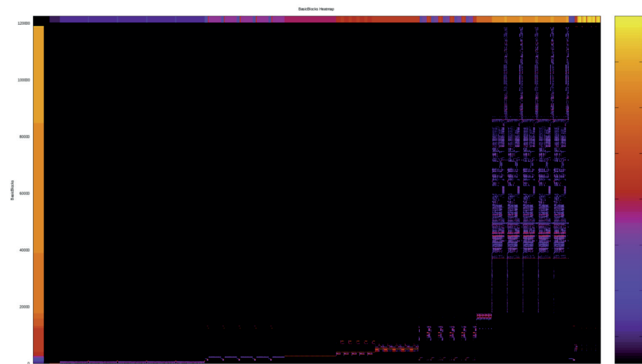
4. Результаты тестирования

Для проверки анализа был использован набор тестов GeekBench. Были записаны трассы его исполнения на различных задачах. На тепловой карте выделены запуски различных задач, представляющие в данном случае сценарии запуска приложения. По верхней строке тепловой карты видно корректное выделение фаз на каждую отдельный тест из набора.



Р и с. 16. Тепловая карта набора тестов Geekbench с выделенными тестами
F i g. 16. Geekbench test suite heatmap with tests highlighted

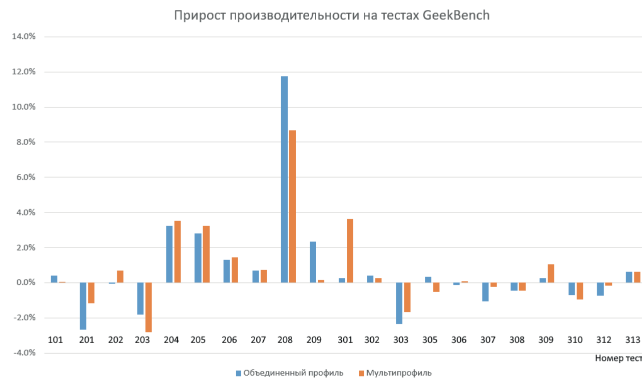
После проведенного анализа и модификации профильной информации производится оптимизация с помощью бинарного оптимизатора BOLT. Отсортированная по фазам тепловая карта выглядит следующим образом:



Р и с. 17. Тепловая карта набора тестов Geekbench после оптимизации
F i g. 17. Geekbench test suite heatmap after optimization

По результатам запуска удалось получить прирост производительности до 4% на некоторых отдельных тестах из набора, но прирост на всех тестах не превышает 1%. Это связано

с отсутствие оптимизации дублирования кода для фазовых пересечений.



Р и с. 18. Сравнение прироста производительности
F i g. 18. Performance Gain Comparison

5. Заключение

В результате проделанной работы разработаны алгоритмы мультипрофильного анализа и был написан анализатор приложений на основе трасс исполнения. Разработана визуализация тепловой карты линейных участков. Анализ с оптимизацией проверены на синтетических тестах и на тестовом наборе Geekbench.

В качестве дальнейшего шага исследования рассматривается реализация оптимизации, дублирующей код для повышения его локальности. Также не рассмотрены используемые платформозависимые оптимизации, которые возможно добавить для ARM архитектуры.

References

[1] Ottoni G., Maher B. Optimizing function placement for large-scale data-center applications. *2017 IEEE/ACM International Symposium on Code Generation and Optimization (CGO).* IEEE Press, Austin, TX, USA; 2017. p. 233-244. (In Eng.) DOI: https://doi.org/10.1109/CGO.2017.7863743

[2] Lebras Y., Charif-Rubial A.S., Jalby W. Combining static and dynamic analysis to guide PGO for HPC applications: a case study on real-world applications. *2019 International Conference on High Performance Computing & Simulation (HPCS).* IEEE Press, Dublin, Ireland; 2019. p. 513-520. (In Eng.) DOI: https://doi.org/10.1109/HPCS48598.2019.9188161

[3] Chen D., Li D.X., Moseley T. AutoFDO: automatic feedback-directed optimization for warehouse-scale applications. *Proceedings of the 2016 International Symposium on Code Generation and Optimization (CGO '16).* Association for Computing Machinery, New York, NY, USA; 2016. p. 12-23. (In Eng.) DOI: https://doi.org/10.1145/2854038.2854044

[4] Panchenko M., Auler R., Nell B., Ottoni G. BOLT: A Practical Binary Optimizer for Data Centers and beyond. *2019 IEEE/ACM International Symposium on Code Generation and Optimization (CGO).* IEEE Press, Washington, DC, USA; 2019. p. 2-14. (In Eng.) DOI: https://doi.org/10.1109/CGO.2019.8661201

[5] Newell A., Pupyrev S. Improved Basic Block Reordering. *IEEE Transactions on Computers.* 2020; 69(12):1784-1794. (In Eng.) DOI: https://doi.org/10.1109/TC.2020.2982888

[6] Li J., Ma X., Zhu C. Dynamic Binary Translation and Optimization. *Journal of Computer Research & Development.* 2007. 44(1):161. (In Eng.) DOI: https://doi.org/10.1360/crad20070123

[7] Nethercote N., Seward J. Valgrind: a framework for heavyweight dynamic binary instrumentation. *ACM SIGPLAN Notices.* 2007; 42(6):89-100. (In Eng.) DOI: https://doi.org/10.1145/1273442.1250746

[8] Vandeputte F., Eeckhout L., de Bosschere K. Exploiting program phase behavior for energy reduction on multi-configuration processors. *Journal of Systems Architecture.* 2007; 53(8):489-500. (In Eng.) DOI: https://doi.org/10.1016/j.sysarc.2006.11.004

[9] Blem E., Menon J., Sankaralingam K. Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures. *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA).* IEEE Press, Shenzhen, China; 2013. p. 1-12. (In Eng.) DOI: https://doi.org/10.1109/HPCA.2013.6522302

[10] Khan T.A., Sriraman A., Devietti J., Pokam G., Litz H., Kasikci B. I-SPY: Context-Driven Conditional Instruction Prefetching with Coalescing. *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO).* IEEE Press, Athens, Greece; 2020. p. 146-159. (In Eng.) DOI: https://doi.org/10.1109/MICRO50266.2020.00024

[11] Lavaee R., Criswell J., Ding C. Codestitcher: inter-procedural basic block layout optimization. *Proceedings of the 28th International Conference on Compiler Construction (CC 2019).* Association for Computing Machinery, New York, NY, USA; 2019. p. 65-75. (In Eng.) DOI: https://doi.org/10.1145/3302516.3307358

[12] Ottoni G., Liu B. HHVM Jump-Start: Boosting Both Warmup and Steady-State Performance at Scale. *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO).* IEEE Press, Seoul, Korea (South); 2021. p. 340-350. (In Eng.) DOI: https://doi.org/10.1109/CGO51591.2021.9370314

[13] Sari A., Butun I. A Highly Scalable Instruction Scheduler Design based on CPU Stall Elimination. *2021 Zooming Innovation in Consumer Technologies Conference (ZINC).* IEEE Press, Novi Sad, Serbia; 2021. p. 105-110. (In Eng.) DOI: https://doi.org/10.1109/ZINC52049.2021.9499298

[14] Neves N., Tomás P., Roma N. Compiler-Assisted Data Streaming for Regular Code Structures. *IEEE Transactions on Computers.* 2021; 70(3):483-494. (In Eng.) DOI: https://doi.org/10.1109/TC.2020.2990302

[15] Ying V.A., Jeffrey M.C., Sanchez D. T4: Compiling Sequential Code for Effective Speculative Parallelization in Hardware. *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA).* IEEE Press, Valencia, Spain; 2020. p. 159-172. (In Eng.) DOI: https://doi.org/10.1109/ISCA45697.2020.00024

[16] Gadioli D., et al. SOCRATES – A seamless online compiler and system runtime autotuning framework for ener-

gy-aware applications. *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE).* IEEE Press, Dresden, Germany; 2018. p. 1143-1146. (In Eng.) DOI: https://doi.org/10.23919/DATE.2018.8342183

[17] Lin Y. Control-Flow Integrity Enforcement with Dynamic Code Optimization. *Novel Techniques in Recovering, Embedding, and Enforcing Policies for Control-Flow Integrity. Information Security and Cryptography.* Springer, Cham; 2021. p. 77-94. (In Eng.) DOI: https://doi.org/10.1007/978-3-030-73141-0_5

[18] Ottoni G. 2018. HHVM JIT: a profile-guided, region-based compiler for PHP and Hack. *ACM SIGPLAN Notices.* 2018; 53(4):151-165. (In Eng.) DOI: https://doi.org/10.1145/3296979.3192374

[19] Valiante E., Hernandez M., Barzegar A., Katzgraber H.G. Computational overhead of locality reduction in binary optimization problems. *Computer Physics Communications.* 2021; 269:108102. (In Eng.) DOI: https://doi.org/10.1016/j.cpc.2021.108102

[20] Hong D.-Y., Wu J.-J., Liu Y.-P., Fu S.-Y., Hsu W.-C. Processor-Tracing Guided Region Formation in Dynamic Binary Translation. *ACM Transactions on Architecture and Code Optimization.* 2018; 15(4):52. (In Eng.) DOI: https://doi.org/10.1145/3281664

[21] Li G., Liu L., Feng X. Accelerating GPU Computing at Runtime with Binary Optimization. *2019 IEEE/ACM International Symposium on Code Generation and Optimization (CGO).* IEEE Press, Washington, DC, USA; 2019. p. 276-277. (In Eng.) DOI: https://doi.org/10.1109/CGO.2019.8661168

[22] Zhou R., Jones T.M. Janus: Statically-Driven and Profile-Guided Automatic Dynamic Binary Parallelisation. *2019 IEEE/ACM International Symposium on Code Generation and Optimization (CGO).* IEEE Press, Washington, DC, USA; 2019. p. 15-25. (In Eng.) DOI: https://doi.org/10.1109/CGO.2019.8661196

[23] Fu S. -Y., Lin C. -M., Hong D. -Y., Liu Y. -P., Wu J. -J., Hsu W. -C. Work-in-Progress: Exploiting SIMD Capability in an ARMv7-to-ARMv8 Dynamic Binary Translator. *2018 International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES).* IEEE Press, Turin, Italy; 2018. p. 1-3. (In Eng.) DOI: https://doi.org/10.1109/CASES.2018.8516794

[24] Arif M., Zhou R., Ho H. -M., Jones T. M. Cinnamon: A Domain-Specific Language for Binary Profiling and Monitoring. *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE Press, Seoul, Korea (South); 2021. p. 103-114. (In Eng.) DOI: https://doi.org/10.1109/CGO51591.2021.9370313

[25] Desmond M., Exton C. An evaluation of the inline source code exploration technique. *Proceedings of the 21st Annual Workshop on Psychology of Programming Interest Group (PPIG 2009).* University of Limerick, Ireland; 2009. 16 p. Available at: https://www.ppig.org/papers/2009-ppig-21st-desmond (accessed 17.06.2021). (In Eng.)

Поступила 17.06.2021; одобрена после рецензирования 12.08.2021; принята к публикации 27.08.2021.
Submitted 17.06.2021; approved after reviewing 12.08.2021; accepted for publication 27.08.2021.

Об авторе:

Лисицын Сергей Алексеевич, ведущий инженер, Российский исследовательский институт Huawei (121614, Российская Федерация, г. Москва, ул. Крылатская, д. 17, корп. 2); аспирант, ФГАОУ ВО «Московский физико-технический институт (национальный исследовательский университет)» (141701, Российская Федерация, Московская область, г. Долгопрудный, Институтский переулок, д. 9), **ORCID: http://orcid.org/0000-0002-8904-0601,** lisitsyn.sergey@huawei.com

Автор прочитал и одобрил окончательный вариант рукописи.

About the author:

Sergey A. Lisitsyn, Lead Engineer, Huawei Russian Research Institute (17 Krylatskaya St., building 2, Moscow 121614, Russian Federation); Postgraduate Student, Moscow Institute of Physics and Technology (National Research University) (9 Institutskiy per., Dolgoprudny 141701, Moscow Region, Russian Federation), **ORCID: http://orcid.org/0000-0002-8904-0601,** lisitsyn.sergey@huawei.com

The author has read and approved the final manuscript.