УДК 004.4'23

Анализ использования вариантов инструментации, сохраняющей информацию, привязанную к местам вызова функций, для оптимизации программ по их профилю исполнения для языков программирования Си/Си++ на базе проекта LLVM

Холявин Н.А. (Университет ИТМО)
Научный руководитель – M.S. CS, Косов П.В.

Введение. Существует несколько реализаций PGO (profile guided optimization) для языков C/C++, с разными уровнями детализации [1][2]. Закрытая реализация, входящая в состав средств компиляции Visual C++ от компании Microsoft, как известно [3], способна создавать профиль исполнения программы с информацией, привязанной к местам вызова функций, в отличие от стандартного подхода открытых решений (например, проект LLVM или GCC), не различающего или ограниченно различающего контексты исполнения функций. Однако такой вариант PGO бы дал возможность ускорить полученные программы при коммерческом использовании модифицированной версии (например, с добавлением новой целевой платформы) компилятора с открытым исходным кодом.

Основная часть. На момент написания работы, PGO в проекте LLVM имеет следующие варианты [4]:

- *Sample-based* (на основе самплирования) — используются внешние аппаратные счётчики, что приводит к меньшим затраты на замеры [5];
- *Instrumentation-based* (на основе инструментации) — используются специально сгенерированные инструкции для взаимодействия со счётчиками. Имеет более высокую точность, детерминизм и возможность определения адресов косвенных вызовов функций и значений аргументов.

Инструментация же может быть следующих типов для языков Си/Си++:

- *Clang frontent instrumentation* (инструментация на уровне clang, т.е. функции соответствуют таковым в исходном языке программирования [6]). Применяется для анализа покрытия кода;
- *Middle-end* (IR) *instrumentation* (инструментация на уровне IR), то есть измерения проводятся над функциями промежуточного представления программы. Применяется для оптимизаций.

Для доработки и использования в оптимизациях был выбран PGO, основанный на *middle-end* инструментации, из-за своих преимуществ.

По признаку зависимости от контекста исполнения функций в проекте LLVM представлены следующие варианты PGO:

- *CSPGO (Context-Sensitive PGO)* — PGO с *middle-end* инструментацией, где счётчики соответствуют путям исполнения внутри функций, к которым применены некоторые оптимизации, например инлайнинг.

Это позволяет получить информацию более зависимую от контекста исполнения, однако применённые оптимизации используют некоторые эвристики, а не данные профиля, что приводит к необходимости применять ещё один проход PGO;

- *CSSPGO (Context-Sensitive Sample PGO)* — *sample-based* PGO, который вместе с самплами сохраняет текущий стек вызовов функций. Имеет все недостатки/преимущества *sample-based* PGO, но сохраняет контекст исполнения [7];
- Обычный вариант PGO с *middle-end* инструментацией.

Таким образом, сохранение информации о контексте вызова функций для *middle-end* PGO может позволить получить PGO с повышенной точностью и детерминизмом, использованием контекста исполнения, при котором достаточно будет одного цикла компиляция с инструментацией — сбор данных — рекомпиляция с использованием профиля.

Таким образом, предлагается:

1. Реализовать внутреннее хранение и модификацию данных профиля программы во время исполнения, в том числе доработать библиотеку времени исполнения компонента compiler-rt;
2. Модифицировать формат сохранения профиля программы, разработать преобразование данных из внутреннего формата хранения профиля в текстовый и бинарный форматы файлов на жёстком диске и обратно;
3. Разработать процесс преобразования нового формата данных профиля программы для расчёта весов (вероятностей) различных веток инструкций условного перехода и сохранения их в метаданные промежуточного представления кода программы;
4. Внедрить использование модифицированных данных PGO в оптимизацию частичного инлайнинга функций и оценить прирост производительности относительно базовой реализации PGO.

Выводы. Проведён анализ возможности модификации текущей реализации PGO в проекте LLVM, внедрён модуль PGO в проекте LLVM, оценена возможность использования вариантов инструментации, сохраняющей информацию, привязанную к местам вызова функций на практике.

Список использованных источников:

1. Profile-Guided Optimizations Overview. — Текст : электронный // CITA : [сайт]. — URL: https://www.cita.utoronto.ca/~merz/intel_c10b/main_cls/mergedProjects/optaps_cls/common/optaps_pgo_ovw.htm (дата обращения: 19.02.2023).

2. Profile-Guided Optimization (PGO). — Текст : электронный // Intel® C++ Compiler Classic Developer Guide and Reference : [сайт]. — URL: https://www.intel.com/content/www/us/en/develop/documentation/cpp-compiler-

developer-guide-and-reference/top/optimization-and-programming/profile-guided-optimization-pgo.html (дата обращения: 19.02.2023).

3. Profile-guided optimizations. — Текст : электронный // Microsoft Learn : [сайт]. — URL: https://learn.microsoft.com/en-us/cpp/build/profile-guided-optimizations?view=msvc-170 (дата обращения: 19.02.2023).

4. Clang Compiler User's Manual. — Текст : электронный // Clang 17.0.0git documentation : [сайт]. — URL: https://clang.llvm.org/docs/UsersManual.html#profile-guided-optimization (дата обращения: 19.02.2023).

5. SamplePGO - The Power of Profile Guided Optimizations without the Usability Burden — Текст : электронный // IEEEXplore : [сайт]. — URL: https://ieeexplore.ieee.org/document/7069298 (дата обращения: 03.09.2023).

6. Profile-guided optimization in Clang: Dealing with modified sources — Текст : электронный // Red Hat Developer : [сайт]. — URL: https://developers.redhat.com/blog/2020/07/06/profile-guided-optimization-in-clang-dealing-with-modified-sources# (дата обращения: 03.09.2023).

7. [llvm-dev] [RFC] Context-sensitive Sample PGO with Pseudo-Instrumentation — Текст : электронный // Google Groups : [сайт]. — URL: https://groups.google.com/g/llvm-dev/c/1p1rdYbL93s (дата обращения: 03.09.2023).