Обзор компилятора lcc для микропроцессора Эльбрус.

А.Л. Маркин, АО «МЦСТ»

Данная работа посвящена обзору оптимизирующего компилятора lcc, используемого для работы с микропроцессорами на архитектуре Эльбрус. Архитектура микропроцессоров Эльбрус спроектирована на основе широкого командного слова (VLIW), которая подразумевает извлечение наибольшего параллелизма уровня инструкций на этапе компиляции. Такой подход требует от компилятора широкого набора оптимизаций и анализов, позволяющих автоматически повышать производительность собираемых приложений [1-4]. Помимо вопросов производительности для компилятора также важны вопросы пользовательских характеристик, таких как актуальность поддерживаемых стандартов языков программирования, совместимость с широко применяемыми компиляторами и их расширениями.

В работе [5] даётся обширное описание возможностей компилятора lcc версии 23, в которое включены сведения о внутреннем устройстве компилятора, особенностях платформы Эльбрус и оптимизациях для неё.

В этой работе даётся обзор основных характеристик компилятора lcc-25, выпущенного в 2020 году, а также, там где это оговорено, для версии компилятора lcc-26, который будет выпущен в 2021 году.

Компилятор lcc написан на языке C, актуальная рабочая версия, и представляет из себя более 2,76 млн. строк кода, 0,74 млн. строк кода из которых принадлежат начальной фазе компиляции (frontend) от фирмы edg [11]. Для языков программирования поддержаны стандарты языка C вплоть до C11, для языка C++ присутствует поддержка большей части стандарта C++17, для языка Fortran поддержан стандарт F2008. Компилятор lcc-25 совместим с компилятором gcc-7.3.0, т. е. воспринимает все его опции и поддерживает большую часть расширений. Следующая версия компилятора будет совместима с gcc-9.3.0.

Совместимость с компилятором gcc на уровне опций и расширений позволяет собирать полноценную ОС Эльбрус, основанную на ОС GNU/Linux, включая сборку непосредственно ядра Linux. По умолчанию для сборки пакетов используется линейка оптимизаций -O3, включающая большое количество оптимизаций, что свидетельствует о высокой надёжности оптимизирующего компилятора.

По мере развития компиляторы приобретают новые возможности и инструменты. Одними из таких инструментов являются библиотеки Address Sanitizer и Memory Sanitizer, позволяющие отслеживать такие проблемы программного обеспечения как выход за границу выделенной области памяти, обращения к удалённой области памяти и утечки памяти. В компиляторе lcc присутствует поддержка этих двух инструментов, выполненная через портирование соответствующих библиотек из компилятора llvm [6].

Помимо опции включения широкой линейки оптимизаций -O3, компилятор lcc имеет дополнительную важную с точки зрения производительности опцию -ffast. Данная опция ужесточает спецификации языка программирования, т. к. требует от программы работы с выравненными массивами, включает агрессивные анализы и оптимизации. Среди них можно отметить анализ пересечения объектов в памяти strict-aliasing [7] и создание условий для включения аппаратно поддерживаемой программной конвейеризации циклов [8].

Компилятор lcc поддерживает возможности межмодульной оптимизации по опции -fwhole, называемой режим компиляции «вся программа». Межмодульная оптимизация представляет все модули программы в виде одного большого модуля, что увеличивает область видимости всех межпроцедурных оптимизаций. Режим «вся программа» позволяет производить более эффективную подстановку функций [9], подстановку неявных вызовов, в том числе подстановку виртуальных вызовов C++ [10], анализ указателей. Ближайшим аналогом режима «вся программа» в компиляторах gcc и llvm является режим оптимизации времени компоновки, вызываемый по опции -flto. Применение межмодульного режима оптимизации позволяет в среднем ускорить производительность компилируемых

приложений до 20%.

Для качественной компиляции необходимо понимать, какие участки кода исполнялись больше, а какие меньше. Для этого существует понятие профилирующей компиляции. Это двупроходная сборка, на первом проходе которой выполняется сборка с опцией -fprofile-generate, после чего программа запускается на обучающем наборе данных. Во время этого обучающего запуска программа выполняет сбор информации о переходах по ветвлениям, вызовах функций и исполнении циклов. На втором проходе производится сборка с опцией -fprofile-use, во время которой компилятор использует полученную информацию для определения оптимального набора и области применения оптимизаций. Использование этой технологии позволяет ускорить исполнение программ в среднем более чем на 17%. Такая схема бывает сложна для пользователя, более того она требует подбора тестовых данных таким образом чтобы программа исполнялась на тех же маршрутах что и при пользовательских данных.

Помимо вопросов удобства пользователя, важным фактором является вопрос производительности вычислительной системы, а задачей оптимизирующего компилятора является создание максимально производительного кода. Одним из принятых способов тестирования производительности компилятора является пакет spec cpu [12], который содержит в себе большой набор тестов с различными видами вычислений.

Доработки оптимизирующего компилятора с каждой новой версией обеспечивают ускорение собираемого программного обеспечения. Например, по сравнению с lcc-23, компилятор lcc-24 на пакете spec cpu2006 в базовом режиме в среднем обеспечивает ускорение на 3.9%, а на пакете spec cpu2017 на 8%. А компилятор lcc-25 по сравнению с lcc-24 обеспечивает среднее ускорение на spec cpu2006 в базовом режиме на 4.6%, а на пакете spec cpu2017 на 7.7%.

Одной из актуальных проблем на данный момент является поддержка новых языков программирования, например языка rust. Для решения этой проблемы создаётся транслятор из промежуточного представления llvm в промежуточное представление lcc. Этот транслятор позволит использовать начальные фазы компиляторов, применяющих llvm в качестве конечной фазы компилятора (backend). Отказ от использования llvm в качестве конечной фазы компиляции для платформы Эльбрус связан с его плохой адаптированностью для архитектур с широким командным словом и необходимостью выполнения большого объёма доработок для получения приемлемой производительности [5]. На текущий момент транслятор уже позволяет lcc использовать в качестве начальной фазы clang, принимающий программы на C и C++, а также принимать на вход программы на языке rust и собирать большую часть библиотек этого языка.

Разработка оптимизирующего компилятора является важной частью работы по созданию микропроцессора. Со временем требования к компилятору растут: требуется поддержка новых стандартов и языков, требуется способность оптимизировать новые типовые программные конструкции и требуется поддержка новых аппаратных возможностей. Компилятор lcc с каждой версией стремится улучшить как удобство использования, так и производительность создаваемого кода. Увеличение производительности достигается как за счёт улучшения самих оптимизаций, так и за счёт использования новых возможностей аппаратных платформ. Среди таких возможностей можно перечислить использование новых векторных инструкций или улучшение поддержки конвейеризации циклов для процессоров начиная с Эльбрус-8СВ. К перспективным исследованиям можно отнести использование возможностей аппаратного профилирования процессоров Эльбрус-16С для динамической оптимизации исполняемого кода.

Литература

1. Ким А.К., Фельдман В.М. Вопросы создания суперЭВМ на основе аппаратной

платформы Эльбрус // Приборы 2009. №1 С. 36-46.

2. Babayan B.A. Main principles of E2k architecture // Free Software Magazine 2002 Vol. 1. №2.

3. Кузьминский М. Отечественные микропроцессоры: Elbrus E2k // Открытые системы 1999. № 05-06.

4. Ким А.К., Перекатов В.И., Ермаков С.Г. Микропроцессоры и вычислительные комплексы семейства «Эльбрус». - СПБ.: Питер, 2013. 272 с.

5. Нейман-заде М.И., Волконский В.Ю. Среды программирования и оптимизирующие компиляторы для компьютеров с микропроцессорами архитектуры «Эльбрус» // Приборы, 2018

6. Русяев Р. М., Нейман-заде М. И., Ермолицкий А. В., Волконский В. Ю. Программно-аппаратные средства выявления ошибок обращения к памяти для архитектуры «Эльбрус» // Вопросы радиоэлектроники. 2017. No 3. С. 33–38.

7. Markin A., Ermolitsky A. Simple Type-Based Alias Analysis for a VLIW Processor. // Tools and Methods of Program Analysis. TMPA 2017. Communications in Computer and Information Science, vol 779. Springer

8. Дроздов А.Ю., Степаненков А.М. Технология оптимизации циклов для арзитектуры с аппаратной поддержкой конвейеризации // Информационные технологии и вычислительные системы. 2004. №3 С. 52-62

9. Ермолицкий А.В, Нейман-заде М.И., Четверина О.А., Маркин А.Л., Волконский В.Ю. Агрессивная инлайн-подстановка функций для vliw-архитектур // Труды института системного программирования РАН. 2015. №6. С. 189-198

10. Маркин А.Л., Ермолицкий А.В. Inline-подстановка вызовов по указателю // 59-я научная конференция МФТИ, 2016

11. https://www.edg.com/

12. https://www.spec.org/