Математический аппарат распределения регистров на основе вычисления уровня абстракции

К.В. Кононенко

Национальный исследовательский ядерный университет «МИФИ»
Kirill.Kononenko@gmail.com

В данной работе мы описываем математический аппарат для быстрого распределения регистров. Данные алгоритмы используют в своей основе математические аппараты статической верификации и вычисления уровня абстракции. Они внедрены в специализированную библиотеку для компиляции на лету libjit linear scan. Специализацией данной библиотеки является выполнение языка MS-CIL с ориентацией на встраиваемые системы на кристалле. Мы исходим из того, что, в-частности, алгоритм точной расскраски графа имеет NP-сложность, которая не подходит для задач реального времени. Также в средах выполнения, в которых используется компиляция на лету, NP-сложность раскраски графа делает малопригодной ее использование для распределения регистров и генерации машинного кода.

1. Введение

Традиционно считается, что распределение регистров наиболее эффективно решается с помощью алгоритмов раскраски графа, однако этот метод без использования специальных эвристик имеет NP сложность. Для динамической компиляции на лету важны как время, затраченное непосредственно на компиляцию, так и непосредственно время выполнения. Таким образом, необходимо учитывать их сумму. Мы расширили определение распределения регистров до новых границ и допустили, что его задачей является не только формирование их множества, которое получают непосредственной раскраской графа, но также минимизация времени компиляции и оптимизация свойств сгенерированного машинного кода. Глобальное распределение регистров выполняется до момента генерации машинного кода. Локальное распределение регистрв производится непосредственно во время его генерации. В нашей работе мы оптимизировали глобальное распределение регистров на основе информации, полученной в результате анализа пространства жизни переменных в промежуточном представлении программы. Разработка наших алгоритмов распределения регистров состоит из трёх независимых шагов. Во-первых, доказательство универсальности и точности для различных множеств программ создаваемого алгоритма. Во-вторых, создание алгоритма вычисления множества A, которое содержит информацию об отображении каждой переменной на множество регистров и множество локальных переменных. В-третьих, создание алгоритма, который использует информацию из множества A для минимизации числа доступов к памяти во время выполнения программы в сгенерированном машинном коде.

Мы неформально определили, что распределение регистров для компиляторов на лету является задачей минимизации числа доступов к памяти, что предполагает решение заданного уравнения, при котором все операции в машинном коде выполнялись бы на регистрах процессора. Такое определение следует из того, что мы ориентировались на такой тест производительности как pnetmark. Несмотря на это, домен оптимизации распределения регистров можно варьировать в таких направлениях, как безопасность содержимого машинных регистров или размер конечного объектного кода.

2. Вычисление уровня абстракции

В созданной нами математической модели имеется возможность вычислить результат выполнения программы на конкретном аппаратном обеспечении в любой

момент времени до ее выполнения, если все варианты ее поведения являются детерминированными и конечными. Для этого применяется математическая модель поведения реального аппаратного обеспечения. С другой стороны, любая программа может иметь бесконечное число сторонних эффектов от непосредственной однозначно детерминированной области доступа программы, такие как чтение и запись в распределённую память. Данные стороние эффекты могут быть бесконечными и даже невычислимыми. Хороший генератор машинного кода поддерживает различные возможные механизмы его генерации и оптимизации распределения регистров. Таким образом, для выбора верного решения необходимо своевременно проанализировать данные бесконечные варианты. Это может быть выполнено с помощью доменно ориентированной оптимизации и анализа поведения программы. Мы используем для данных вычислений линейный анализ и математический аппарат вероятностных пространств на базе аксиоматики Колмогорова. Мы определяем значение, которое называем уровнем абстракции, как вероятность заблаговременно произвести верное решение о свойствах программы до времени непосредственного выполнения программы на реальном аппаратном обеспечении на основе анализа ограниченного множества за линейное время вычислений. Таким образом, мы формализуем неформальный тезис, что более абстрактный байт-код виртуальной машины может быть выполнен более эффективно, чем менее абстрактный байт-код. Ограниченные ресурсы рекомендованы для распределения в направлении решений с наибольшим уровнем абстракции с целью, чтобы получить наибольшее значение математического ожидания выбранного пути выполнения программы, который будет пройден во время непосредственного выполнения программы. Другими словами, мы рассматриваем машинный код, сгенерированный динамическим компилятором, как конечное упорядоченное множество случайных величин. Каждая из данных случайных величин имеет своё распределение. Несмотря на это, данные распределения могут быть невычислимыми или даже быть бесконечно делимыми.

Таким образом, может быть проведена аналогия с марковскими сетями с конечным числом линейно упорядоченных узлов и применен аппарат описания свойств сигнала, аналогичный вейвлет преобразованиям, аппарату характеристких функций, аналогичных функциям Колмогорова и Леви-Хинчина, для вычисления свойств случайных величин. Данные узлы являются представлением поведения программы. Мы считаем, что в случае неопределённости, недетерминированности и множества возможностей с равными весами, оптимизация и ресурсы должны быть направлены в направлении решения с максимальным ожидаемым значением веса корректного решения заблаговременно до времени выполнения программы в заданном домене оптимизации.

3. Абстрактная интерпретация

Мы применяем абстрактную интерпретацию для предсказания свойств и поведения программы до момента ее выполнения. Частая ошибка, по нашему мнению, заключается в том, что считается, что абстрактная интерпретация имеет экспоненциальное время выполнения. Однако время выполнения абстрактной интерпретации может быть полиномиальным и являться экспоненциальным только в самых выраженных случаях, которые достаточно редки для того, чтобы их не рассматривать в реальной жизни.

Мы определим, что задачей распределения регистров при заданном множестве свойств программы является, в первую очередь, корректность ее выполнения, и, во вторую, минимизация числа обращений к локальной стековой памяти.

Мы применяем абстрактную интерпретацию для определения оптимального алгоритма, который бы предсказывал свойства заданной программы до момента её выполнения на аппаратном обеспечении. Каждое бинарное свойство имеет вероятность слу-

читься или не случиться. Каждое из данных бинарных свойств мы храним в одном бите. Мы предсказываем свойства программы с помощью абстрактной интерпретации, которая должна создать множество данных битов. Наши алгоритмы распределения регистров используют данную информацию для оптимизации и генерации машинного кода. Вероятность предсказать свойства заданной программы до момента её выполнения в линейное время обратно пропорционально размеру анализируемой программы. Мы также описываем ниже необходимые преобразования для вычисления заданной вероятности и ее свойства. Эту вероятность мы называем уровнем абстракции. Поскольку практически сложно создать простой однозначный процесс для вычисления всех возможных вариантов поведения заданной программы, мы также просматриваем заданные свойства сгенерированного машинного кода. Каждый кусочек машинного кода имеет ожидаемый вес на основе линейной зависимости приближённого значения на основе вычисленного уровня абстракции.

Свойства определены в некотором домене оптимизации. Например, как мы упомянули выше, они могут быть числом обращений к памяти, безопасностью машинного кода, размером сгенерированного машинного кода. В обоих случаях, должен иметься интервал машинного кода для достижения одной заданной стратегии оптимизации или их множества. Таким образом, мы рассматриваем процесс распределения регистров в компиляторах на лету для CLR как результат заданного процесса абстрактной интерпретации. В заданном процессе каждый управляемый метод отображается на множество случайных величин в заданном вероятностном пространстве. Эта векторная случайная величина является множеством возможного машинного кода и множеством данных, для которых компилируется исходный код. Другими словами, каждый скомпилированный объект является реализацией упорядоченного множества случайных величин. Мы рассматриваем их распределения.

4. Формализация

Вычислим вес как различие между ожидаемой оптимистичной прибылью и ожидаемыми пессимистичными потерями. Другими словами, мы смотрим на ожидаемый лучший вариант и на ожидаемый худший вариант. Таким образом, мы определяем функцию побед и потерь, которые определены на примитивных цепочках событий <X>:

$$\mathbf{E}[\mathrm{losses}(\ll X \gg)] \stackrel{\text{def}}{=} \int_{t=0}^{t=T} \mathrm{abstraction}(\ll X(t) \gg)\mathbf{E}[\mathrm{losses}\ (\ll X(t) \gg)]\ \mathrm{dt}$$

$$\mathbf{E}[\mathrm{wins}(\ll X \gg)] \stackrel{\text{def}}{=} \int_{t=0}^{t=T} \mathrm{abstraction}(\ll X(t) \gg)\mathbf{E}[\mathrm{wins}\ (\ll X(t) \gg)]\ \mathrm{dt}$$

$$\mathbf{E}[\mathrm{weight}(\ll X \gg)] \stackrel{\text{def}}{=} \mathbf{E}[\mathrm{wins}(\ll X \gg) - \mathrm{losses}(\ll X \gg)] = \mathbf{E}[\mathrm{wins}(\ll X \gg)] - \mathbf{E}[\mathrm{losses}(\ll X \gg)] =$$

$$= \int_{t=0}^{t=T} \mathrm{abstraction}(\ll X(t) \gg)\mathbf{E}[\mathrm{weight}\ (\ll X(t) \gg)]\ \mathrm{dt}$$

E[weight(‹X(t)›) представляет ожидаемый вес блока с индексом i(t) в случае корректного принятия решения с вероятность 1. Более того, если блок действительно имеет заданное свойство оптимизации, то вся информация о свойствах заданного сигнала была использована корректно для получения ожидаемой выгоды. Отметим, что мы ожидаем, что, если была использована вся доступная информация, а не только её часть, то может быть принято корректное решение.

Мы представляем промежуточный код в виде независимых множеств, в которых используется каждая из переменных. Каждой переменной присваивается виртуальный регистр на все промежутки ее жизни. Имеется ограниченное множество ресурсов для присваивания регистров. Мы оптимизируем распределение регистров, и, таким обра-

зом, нашими ограниченными ресурсами являются машинные регистры. Покажем ниже ожидаемый вес каждой переменной при выделении данной переменной регистра.

Рассматрим шаг, когда уже имеется информация о множестве промежутков жизни. Предположим, что распределение свойств промежуточного кода содержать свойства для оптимизации является равномерным и уровень абстракции является константой. Таким образом:

abstraction (<X(t)>) = $\frac{1}{\text{Power}(<X(t)>)}$.

Функция power мощности множества возвращает значения в битах, которые могут быть использованы для представления множества всех возможных вариантов поведения машинного кода и данных. Каждый блок битов может использовать переменную или нет. Мы вводим функцию Δ(t), которая равна ожидаемому числу раз заданная переменная используется в множестве бит для доступа к памяти в промежутке между t to t +Δt. Мы получаем:

E[weight(<X(t)] = $\int_A^B \frac{1}{\text{Power}(<X(t)>)} \Delta(t)\, dt$.

5. Заключение

Результаты нашего исследования могут быть интересны для разработки компиляторов времени выполнения, интерпретаторов, и компиляторов смешанных режимов выполнения программы для таких виртуальных машин, как CLI, Java, Python, Perl, Ruby, LLVM, Parrot. Дальнейшие направления работы могут включать исследования в области динамической компиляции, динамической верификации, создания новых доменно-ориентированных алгоритмов распределения регистров, но не ограничиваются ими. Предлагаемые алгоритмы могут быть оптимизированы для применения в различных доменах, например, для компрессии машинного кода и безопасности регистровых файлов.

Литература

1. Презентация - Исследование оптимального распределения регистров в программах на MS-CIL. Conference Information Systems and Technology 2009, Obninsk State Technical University for Nuclear Power Engineering (INPE), 2009 . www.libjit.com/downloads
2. Документация и материалы Научных Сессий МИФИ и PLDI, посвящённых libjit linear scan и GNU Portable.NET, www.libjit.com/downloads
3. Baoding Liu, Uncertainty Theory, Third Edition, Department of Mathematical Sciences, Tsinghua University, http://orsc.edu.cn/liu
4. Kolmogorov, A. N. (1933) Grundbegriffe der Wahrscheinlichkeitsrechnung. Springer, Berlin. Publish in English as Foundations of the Theory of Probability, Kolmogorov, Chelsea Publishing Company New York. 1st edn, 1950; 2nd edn, 1956, Library of Congress Catalogue Card Number 56-11512.
5. Kolmogorov Complexity: Sources, Theory and Applications, Alexander Gammerman and Vladimir Vovk.
6. A. N. Kolmogorov, "Three approaches to the quantitative definition of information".
7. Kolmogorov, A. N. (1968) Logical basis for information theory and probability theory. IEEE Tran. Inform. Theory IT-14, 662-664.
8. Kolmogorov, A. N. (1983) Combinatorial foundations of information theory and the calculus of probabilities. Russian Math. Surveys, 38, 29-40.
9. E. A. Feinberg and A. N. Shiryaev 2006 Statist Decisions 24 (4) 445-470.
10. G. Peskir and A. N. Shiryaev 2006 Optimal stopping and free-boundary problems (Lectures Math. ETH Zurich) (Birkhauser, Basel).

11. M.Probst, A.Krall, B.Scholz, "Register Liveness Analysis for Optimizing Dynamic Binary Translation, Ninth Working Conference on Reverse Engineering (WCRE 2002), 2002.
12. Massimiliano Poletto and Vivek Sarkar. Linear scan register allocation. ACM Transactions on Programming Languages and Systems, 21(5):895–913, 1999.
13. Omri Traub, Glenn H. Holloway, and Michael D. Smith. Quality and speed in linear scan register allocation. In SIGPLAN Conference on Programming Language Design and Implementation, pages 142–151, 1998.
14. Abstract interpretation**,** Patrick Cousot, ACM Computing Surveys (CSUR), Volume 28 , Issue 2 (June 1996), ISSN:0360-0300
15. Standard ECMA-335, Common Language Infrastructure; ISO/IEC 23271:2006.
16. R. Wilkes, NGen Revs Up Your Performance with Powerful New Features, http://msdn.microsoft.com/msdnmag/issues/05/04/NGen/
17. Extended Linear Scan: an Alternate Foundation for Global Register Allocation, Vivek Sarkar and Rajkishore Barik.
18. Steven Muchnick. Advanced Compiler Design and Implementation. Morgan Kaufmann Publishers, 1997.
19. Representation-based Just-in-time Specialization and the Psyco prototype for Python, Armin Rigo http://psyco.sourceforge.net/theory_psyco.pdf
20. Towards Device Emulation Code Generation, T.Heinz, R.Wilhelm, LCTES'09, Proceedings of the 2009 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems, Pages 109-118 ISBN:978-1-60558-356-3.
21. Addressing the Challenges of DBT for the ARM Architecture, Ryan W. Moore, Jose A. Baiocchi, Bruce R. Childers, Jack W. Davidson, Jason D. Hiser, LCTES'09, Proceedings of the 2009 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems.
22. C.Wimmer: Linear Scan Register Allocation for the Java HotSpot™ Client Compiler. Master's thesis, Institute for System Software, Johannes Kepler University Linz, 2004.
23. P.Briggs, K.D.Cooper , L.Torczon, Improvements to graph coloring register allocation, ACM Transactions on Programming Languages and Systems (TOPLAS), v.16 n.3, p.428-455, May 1994.
24. R.Cytron, J.Ferrante, B.K.Rosen , M.N.Wegman , F.K.Zadeck, Efficiently computing static single assignment form and the control dependence graph, ACM Transactions on Programming Languages and Systems (TOPLAS), v.13 n.4, p.451-490, Oct. 1991.
25. M. Probst, A. Krall, B. Scholz, "Register Liveness Analysis for Optimizing Dynamic Binary Translation," wcre, pp.0035, Ninth Working Conference on Reverse Engineering (WCRE 2002), 2002.
26. A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, Compilers: Principles, Techniques, and Tools, Second Edition. Addison-Wesley, 2007. ISBN 978-0-32148-681-3.
27. James Larus. Spending Moore's Dividend. ISSN:0001-0782. Communications of the ACM. Volume 52, Issue 5 (May 2009).
28. Standard ECMA-334, C# Language.
29. Andrew W. Appel, Maia Ginsburg, Modern Compiler Implementation in C, Cambridge University Press (August 21, 2008), ISBN-10: 0521607655.
30. Preston Briggs. Register Allocation via Graph Coloring. PhD thesis, Rice University, April 1992.
31. Minimal Description Length, http://www.mdl-research.org/
32. J. Rissanen*,* An Introduction to the MDL Principle, http://www.mdl-research.org/jorma.rissanen/pub/Intro.pdf/
33. Jongeun Lee, Aviral Shrivastava. Jongeun Lee, Aviral Shrivastava: A compiler optimization to reduce soft errors in register files. LCTES 2009: 41-49.