Модель и реализация распределения регистров с помощью расширенного линейного сканирования на базе событий реального времени

К.В. Кононенко

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

Мы предлагаем модель распределения регистров для продвинутой компиляции на лету, которая представляет данную задачу, как проблему обработки потока заявок на обслуживание в реальном времени в некоторой заданной области. В качестве прикладного примера можно привести осуществление слежения за появляющимися объектами в радиолокационном поле, подразумевающее первичное обнаружение цели и следующее за этим ее сопровождение с некоторым показателем надежности и устойчивости. Эффективность данной модели была продемонстрирована в эксперименте, в котором решение задачи распределения регистров было выполнено с использованием трех различных алгоритмов. Эти алгоритмы были рекомендованы различными источниками для быстрой динамической компиляции и решают задачу путем подсчета частоты использования переменных, методом простого линейного сканирования и применяя его более продвинутую вариацию - second-chance binpacking. Все три алгоритма показали, что использование только этой модели позволило значительно повысить качество результатов.

Модель реализована в виде платформенно-независимого программного пакета. Данный пакет создан для встраиваемых систем на базе POSIX-совместимых операционных систем. Представлены результаты исследования метода решения задачи распределения регистров с помощью расширенного линейного сканирования применительно к стандарту ECMA-335 в компиляторе времени выполнения GNU Portable.NET. На базе предложенной модели были реализованы указанные выше эвристические алгоритмы распределения, осуществлено их сравнение. Полученные результаты исследований сравнены с различными версиями Novell Mono, Microsoft .NET 2.0 и несколькими вариантами реализации той же межъязыковой среды выполнения, в которых используется прямая интерпретация на базе указателей на обработчики, простая развертка цикла интерпретатора и режим интерпретации без контроля над распределением регистров.

1. Введение

Различные источники исследований оптимального распределения регистров говорят, что эта задача в продвинутых компиляторах времени выполнения программы не может быть решена путем непосредственной раскраски графов при компиляции «на лету» в реальном времени из-за полиноминально недетерминированной сложности ее выполнения. В этом случае в системах динамической компиляции, компиляторах «на лету» и интерактивных средах разработки рекомендовано использование линейного сканирования. Предлагается его реализация посредством последовательного упорядочивания и анализа инструкций. Регистры выделяются в данном линейном порядке с начала до конца сканирования промежуточного представления программы. В случае недостатка свободных регистров переменная с самым дальним окончанием времени жизни сохраняется в локальный стек. Данное описание обобщает распределение регистров простого линейного сканирования. Конкретная реализация и детали могут значительно отличаться. Распределение регистров на базе second-chance binpacking является вариацией линейного сканирования, которая инвестирует больше времени в оптимизацию и генерацию машинного кода. Распределение регистров с помощью расширенного линейного сканирования пытается снять ограничения, возникающие при раскраске графа, и предлагает еще один вариант решения задачи, который требует еще больше времени, давая машин-

ный код более высокого качества, при этом затрачивает меньше времени, чем раскраска графа.

Предметом нашего изучения стала простая математическая модель обработки заявок на обслуживание, когда имеется два класса ресурсов. Во-первых, мы располагаем ограниченным количеством ресурсов первого класса, которыми являются машинные регистры, во-вторых, есть практически неограниченное число ресурсов второго класса, представленных локальной памятью стека. Данные ресурсы должны быть распределены оптимальным образом в реальном времени. Наиболее сомнительной концепцией в распределении регистров в алгоритмах, применяемых в настоящее время, как представляется, является концепция раскраски графа. Возникает вопрос, возможно ли эффективно генерировать машинный код без использования концепции спилинга. Данная идея представлена ниже. Мы изучили новую математическую модель. В ней задействованы списки событий реального времени, которые используются для решения двух разделенных задач распределения регистров. Во-первых, глобальный алгоритм присваивает регистры и локальную память виртуальным регистрам в рамках всего метода целиком до генерации машинного кода. Во-вторых, локальный алгоритм присваивает регистры и локальную память стека в рамках блока инструкций в промежуточном представлении во время генерации кода. Вводится механизм синхронизации обоих алгоритмов на основе концепции дырок в промежутках жизни аппаратных регистров.

2. Модель распределения ресурсов в реальном времени

Рассмотрим модель распределения ресурсов в реальном времени, когда ресурсы первого класса ограничены, а ресурсы второго класса практически неограниченны. Задействование ресурсов первого вида, предположительно, существенно сокращает время выполнения куска машинного кода на аппаратном процессоре и программы в целом. Однако их ограниченность определяет необходимость подключения в некоторые моменты ресурсов второго класса. В частности, это является одной из классических задач простой радиолокации. В данном случае обнаруживаются объекты в заданной области слежения и осуществляется сбор сведений об их свойствах. В случае, если какое-либо воздушное судно имеет приоритет, необходимо сфокусировать все лучшие ресурсы на обработке информации о нем. Если объект не приоритетен, то для анализа информации по нему допустимо использование ресурса второго класса. Таким образом, каждому анализируемому объекту присваивается область видимости. Объект «рождается», когда он появляется в зоне контроля в момент, когда он был обнаружен. С другой стороны, объект «умирает», когда покидает зону нашей ответственности. В таком случае допустимо просто игнорировать этот объект и перераспределять ресурсы на другие задачи. Также это случается, когда теряется контроль над объектом. Если объект потерян, то целесообразно, однако, некоторое время сохранять предварительно накопленную по нему информацию. Возможно, она окажется полезной, если он вновь будет обнаружен. Проблема определения времени сохранения и обработки информации по потерянному в нашей зоне ответственности объекту, по-видимому, является отдельной задачей. Другими словами, каждый анализируемый объект имеет свой вес важности в любой момент времени.

Такая формулировка сводит задачу к распределению ограниченных ресурсов в конечное и бесконечное время. В нашем программном пакете каждая программа-кандидат анализируется с ее начала. Во время данного процесса каждая инструкция промежуточного кода индексируется и выделяется регистрами или локальной памятью в течение времени анализа программы. Каждая переменная является объектом. Каждый объект имеет свой вес. Объект с наибольшим весом использует ресурсы первого класса и, конечно, вес можно измерять по-разному. Анализ различных вариантов показал, что целесообразно принять вес как разность между оптимистичными доходами и пессимистичными потерями, в случае использования элементов из конечного множе-

ства вариантов B, вместо использования элементов из бесконечного множества вариантов A. Другими словами, необходимо рассмотреть, какой может быть потенциальный результат в будущем, если в настоящий момент больше не имеется свободных ресурсов первого класса, но в наличие бесконечное множество ресурсов второго класса. Возникает вопрос о том, каковы будут последствия присваивания ресурсов второго классу объекту, где в настоящий момент используются ресурсы первого класса, и присвоения ресурсов первого класса другому объекту, если это предполагается более выгодным.

3. Реализация CLR с помощью libjit

Дизайн библиотеки libjit содержит набор средств, которые заботятся о процессе динамической компиляции во время выполнения программы. Эти средства независимы от используемого языка программирования и специфичного типа байт-кода виртуальной машины. Большая часть работы динамического компилятора связана с арифметическими операциями, преобразованием типов данных, записью, чтением из памяти, операциями ветвления, анализом потока данных, анализом графа потока управления, распределением регистров. Только очень небольшая часть компилятора времени выполнения зависима от языка. Задачей аппаратно-переносимого интерфейса libjit и libjit вообще является предоставление множества низко уровневых утилит для компиляции на лету, которые независимы от языка программирования и его специфических особенностей.

Libjit содержит множество платформенно-независимых функций с префиксом jit_insn_*. Последовательность вызовов данных функций генерирует множество инструкций в представлении, близком к SSA форме. Например, в GNU Portable.NET компилятор времени выполнения вызывает множество данных функций во время синтаксического анализа сборки с управляемым байт-кодом. Данное представление может быть скомпилировано и выполнено. Это происходит в двух случаях. В первом, когда вызывается метод jit_function_apply. Во втором, процесс компиляции выполняется, когда данная сгенерированная функция вызывается из другой функции. Процесс компиляции на лету состоит из следующих шагов. Во-первых, для каждой переменной выделяется виртуальный регистр. Во-вторых, выполняется анализ пространства жизни переменной. В конце данного анализа генерируется множество критических точек для каждой инструкции в промежуточном операционном коде. Другими словами, каждая критическая точка является либо точкой, когда переменная впервые жива на входе, либо точкой, когда переменная жива последний раз на выходе из инструкции.

4. Экспериментальная платформа

1. Машина на основе платформы IA-32, Intel Celeron M, processor 1.50 GHz; 0.98 GB of RAM; исходные и бинарные коды LibJIT Linear Scan Register Allocator 0.1.2.5; Debian GNU / Linux Etch Operating System, Microsoft Windows XP Home Edition Service Pack 3, тест производительности Pnetmark, исходные и бинарные коды Portable.NET 0.8.1 source code, Novell / Ximian Mono 2.0, 2.2, 2.4 (JIT compiler), Mono 1.1.11 и интерпретатор mint, Microsoft .NET Framework 2.0, утилита ‘timeIt.exe’ для замера времени выполнения на платформе Windows XP, утилита ‘time’ для замера времени выполнения на Debian GNU/Linux Etch.
2. Тест производительности №1 (pnetmark). Замеряет время выполнения сгенерированного кода. Это не включает времени, затраченного на компиляцию и время затраченного на анализ графа потока данных и графа потока управления. Результатом микротестов pnetmark являются целые числа. Каждый тест выполняет серию операций заданного типа в течение некоторого промежутка времени. Возвращается число раз, которое выполнилась заданная серия операций за фиксированное время.
3. Тест производительности №2. Включает время прохождения регрессионных тестов в библиотеке Portable.NET. Замеряет, как быстро проходят регрессионные тесты.

Поддерживаются пять уровней оптимизации. Уровень оптимизации 0 использует бинарный интерфейс cdecl, глобальное распределение регистров на основе числа использований, операции с плавающей точкой на основе FPU. Уровень оптимизации 1 присутствует только локальное распределение регистров в каждом блоке инструкций. Уровень оптимизации 2 использует линейное сканирование, быстрый анализ жизни переменных на основе анализа графа потока управления. Только один интервал жизни переменных для каждого виртуального регистра. Уровень оптимизации 3 использует second-chance binpacking на базе множества интервалов жизни переменных для каждого виртуального регистра. Уровень оптимизации 4 включает алгоритм удаления мёртвого кода. Новый генератор машинного кода также использует расширения SIMD и регистры XMM для представления операций с плавающей точкой.

	CLR	**Вариант среды выполнения**	**Уровень оптимизации с libjit**
1	GNU Portable.NET	internal ABI	4
2	GNU Portable.NET	cdecl ABI	4
3	GNU Portable.NET	internal ABI	3
4	GNU Portable.NET	cdecl ABI	3
5	GNU Portable.NET	internal ABI	2
6	GNU Portable.NET	cdecl ABI	2
7	GNU Portable.NET	internal ABI	1
8	GNU Portable.NET	cdecl ABI	1
9	GNU Portable.NET	cdecl ABI Распределение регистров на основе числа использованных переменных. Операции над числами с плавающей точкой выполняются на процессоре x87	0
10	GNU Portable.NET	раскрутка цикла интерпретатора	
11	GNU Portable.NET	использует поток указателей на машинный код с обработчиками инструкций. CIL преобразуется в промежуточное представление виртуальной машины	
12	GNU Portable.NET	режим интерпретации libjit	
13	Microsoft .NET Framework 2.0		
14	Mono	2.4	
15	Mono	2.2	
16	Mono	2.0	
17	Mono	1.1	
18	Mono	Mint 1.1.11	

5. Заключение

В работе были достигнуты результаты в следующих областях. Во-первых, была представлена модель распределения регистров на основе событий реального времени. Во-вторых, на основе данной модели было исследовано и проанализировано, насколько эффективны различные эвристики распределения регистров. Экспериментальные данные испытанных алгоритмов распределения регистров представлены на рисунке 1. Тест pnetmark замеряет качество сгенерированного машинного кода, поскольку время, затраченное на компиляцию, в нем не включено. Тест “Execution time of regression tests”

для Portable.NET с libjit linear scan 0.1.2.5 показывает повышение производительности на 26%, по сравнению с Mono 2.0, и на 12,7% лучшие результаты, по сравнению с Mono 2.4. Тест производительности №1 для Microsoft .NET 2.0 показывает на 31,5% лучшие результаты, чем компилятор времени выполнения Portable.NET 0.8.0, который использует libjit linear scan 0.1.2.5, и на 48,3% лучшие результаты, чем Mono 2.4.

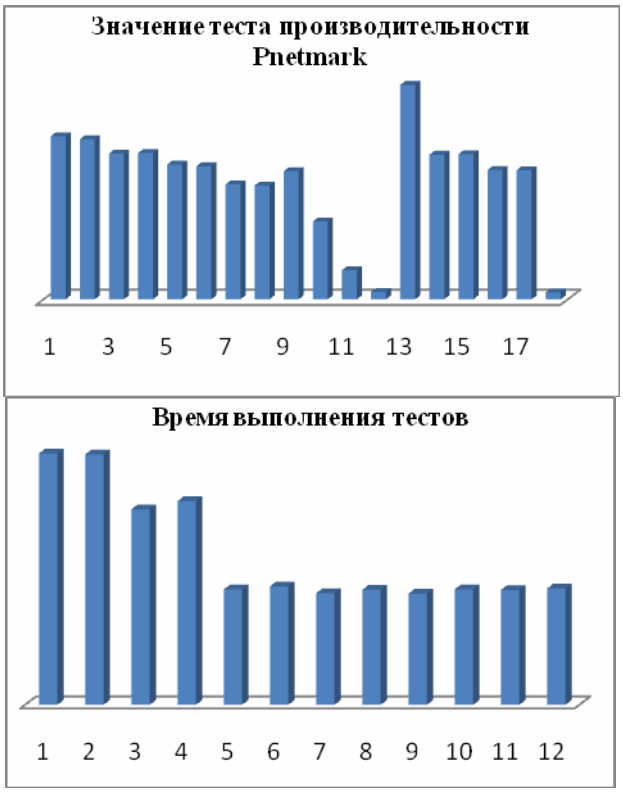

Рисунок 1. Среднее значение теста производительности и время выполнения тестов.

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. Abstract interpretation, Patrick Cousot, ACM Computing Surveys (CSUR), Volume 28 , Issue 2 (June 1996), ISSN:0360-0300
5. Standard ECMA-335, Common Language Infrastructure; ISO/IEC 23271:2006.
6. Extended Linear Scan: an Alternate Foundation for Global Register Allocation, Vivek Sarkar and Rajkishore Barik.
7. Steven Muchnick. Advanced Compiler Design and Implementation. Morgan Kaufmann Publishers, 1997.
8. James Larus. Spending Moore's Dividend. ISSN:0001-0782. Communications of the ACM. Volume 52, Issue 5 (May 2009).