Маркин А.Л., Ермолицкий А.В. (ЗАО «МЦСТ»)

АНАЛИЗ УКАЗАТЕЛЕЙ В ПРОГРАММАХ С ВЫЗОВАМИ БИБЛИОТЕЧНЫХ ФУНКЦИЙ

Аннотация

В статье рассмотрена методика ускорения анализа указателей в оптимизирующих компиляторах, позволяющая уменьшить затраты и увеличить точность анализа вызовов функций стандартной библиотеки за счёт использования модели их поведения.

Ключевые слова

оптимизирующий компилятор, стандартная библиотека, анализ указателей

Вступление

Важной частью оптимизирующих компиляторов является анализ указателей - набор техник для вычисления возможных пересечений указателей в памяти, т.е. определяется, могут ли два указателя ссылаться на один участок памяти. Такая информация необходима для полноценной работы всех оптимизаций, работающих с операциями чтения/записи в память [1].

Принцип работы анализа указателей.

Анализ указателей является технически сложным и ресурсоёмким процессом, который даёт максимальный эффект при межпроцедурном анализе всей программы. На практике же большинство программ применяют технологию помодульной сборки - компилятор видит только информацию из текущей единицы компиляции и не обладает информацией о программе в целом. Более того, почти всегда в программах используются функции, предоставляемые стандартной библиотекой, код которых может быть недоступен компилятору. Это приводит к ухудшению точности анализа в случае программ с библиотечными вызовами, что негативно сказывается на эффективности оптимизаций.

Наиболее точным межпроцедурным анализом в компиляторе для архитектуры Эльбрус является анализ, основанный на построении графа указателей. Данный алгоритм начиная с функции *main* рекурсивно просматривает содержимое каждой функции. Для каждой операции чтения/записи в функции он находит множество указателей с которыми эта операция гипотетически может работать. Если множества указателей двух операций не пересекаются, то операции считаются независимыми. Т.к. данный анализ является межпроцедурным, то во время его работы могут создаваться графы с миллионами узлов, а само время работы может занимать большую часть от всего времени компиляции.

В компиляторе для архитектуры Эльбрус межпроцедурный анализ указателей показывает наибольшую эффективность в режиме “вся программа”. В этом режиме компилятор видит все исходные файлы как один модуль и имеет доступ к коду всех функций, использующихся в

программе.

При этом независимо от режима в программах часто встречаются вызовы функций, определённых стандартной библиотекой языка. Их поведение всегда неизменно, но компилятор вынужден каждый раз заново производить их анализ. На это тратятся ресурсы во время компиляции, а также, учитывая сложность некоторых библиотечныйх функций, происходит уменьшение точности всего анализа. В рамках данной работы было предложено создать модель поведения функций, определённых стандартом для устранения необходимости анализировать содержимое таких функций. [2]

Структура модели поведения функций

Модель библиотечных функций представляет из себя, описание свойств множества функций, поведение которых определено стандартом и не может быть изменено. Модель функции позволяет анализировать программу без точного анализа содержимого этой функции. В неё включается информация об аргументах функции, работе функции с указателями и глобальными переменными. Ниже приведено подробное описание свойств функций, описываемых данной моделью.

Режим возврата - свойство, отвечающее на вопрос “всегда ли функция возвращает управление?”. Данное свойство важно для оптимизации графа вызовов, а также для отсечения лишних узлов при построении графа указателей.

Наличие побочных эффектов - такие функции, для которых выполняются два условия: функция с одинаковыми значениями аргументов всегда возвращает одинаковый результат, и функция не меняет глобальное состояние программы.

Выделение функцией памяти - наличие этого свойства свидетельствует о том, что результатом функции будет уникальный указатель, ведущий на выделенный участок памяти.

Чтение глобальных переменных - свойство говорит о зависимости записываемого в аргументы значения или результата вызова функции от значения, находящегося в глобальной переменной.

Запись в глобальные переменные - свойство означает, что значение из аргументов функции может быть записано в глобальную переменную и оно станет доступным из функций, читающих глобальные переменные.

Запись в переменную errno - означает, что функция может изменить значение системной глобальной переменной errno. Часто это свойство можно не учитывать

Запись по аргументам-указателям - список аргументов-указателей, в которые может быть записано какое-либо значение

Чтение из аргументов-указателей - список аргументов-указателей, которые считываются функцией

Использование аргументов-указателей - список аргументов-указателей, которые могут попасть в результат функции или быть записаны в глобальную переменную

Вызов функции по аргументу-указателю - список аргументов, которые являются указателем на функцию и могут вызываться внутри библиотечной функции

Возврат указателя - результатом функции является указатель.

Принадлежность к стандарту - стандарт языка, в котором данная функция определена.

Приведённых выше свойств достаточно для корректной работы анализов и оптимизаций без анализа кода библиотечных функций, описанных моделью. Вся необходимая информация извлекается непосредственно через запросы к модели. Это позволяет уменьшить время компиляции программ, количество потребляемой памяти во время компиляции и увеличить точность анализа указателей за счет уменьшения количества анализируемого кода.

Исследование экспериментальных результатов.

Предложенная модель библиотечных функций была внедрена в промышленный оптимизирующий компилятор для семейства микропроцессоров Эльбрус. Влияние модели функций на процесс компиляции и исполнения было экспериментально измерено на задачах из пакетов тестов SPEC-CPU2000 и SPEC-CPU2006 [3].

В рамках данного исследования проводится замер скорости работы и потребляемой памяти анализа указателей в 2 режимах сборки с 3 разными наборами опций.

Первый набор называется «*black_box*». В этом наборе опций компилятор не видит код библиотечных функций, и, следовательно, всегда делает консервативные предположения касательно их поведения.

Второй набор называется «*full_lib*». В нём компилятору становится доступен для анализа промежуточный код библиотечных функций.

Третий режим называется «*fast_lib*». В этом режиме используется модель поведения библиотечных функций. Если компилятор находит в ней исследуемую функцию, то он не производит её анализ, а получает все данные основываясь на информации из модели.

Исследование влияния на процесс компиляции.

Параметры компиляции по вышеописанным наборам опций измеряются в режимах помодульной сборки и сборки всей программы. Помодульная сборка представляет из себя наиболее привычный способ сборки программ, при котором сначала отдельно собирается каждый файл с исходным кодом, а далее происходит их связывание. В режиме «вся программа» компилятор видит все исходные файлы как одно целое и компилирует всю программу с библиотеками как единственный модуль.

Замеры затрат ресурсов показывают, что при использовании модели библиотечных функций затраты памяти и время выполнения анализа указателей сокращаются по

отношению к набору опций «*full_lib*», а на некоторых задачах близки к набору опций «*black_box*». Но точность анализа такая же или выше чем с набором опций компиляции «*full_lib*». Результаты тестов для наборов задач SPEC-CPU2000 и SPEC-CPU2006 можно увидеть на Рис. 1 а,б и Рис. 2 а,б. Т.к. для данных замеров сборка в помодульном режиме не имеет смысла, рассматриваются только замеры в режиме сборки «вся программа». Погрешность измерений в среднем составляет 4.5%.

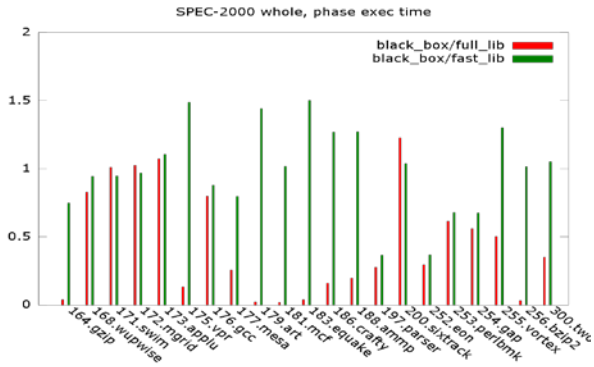

Рис 1.а Отношение времени исполнения фазы

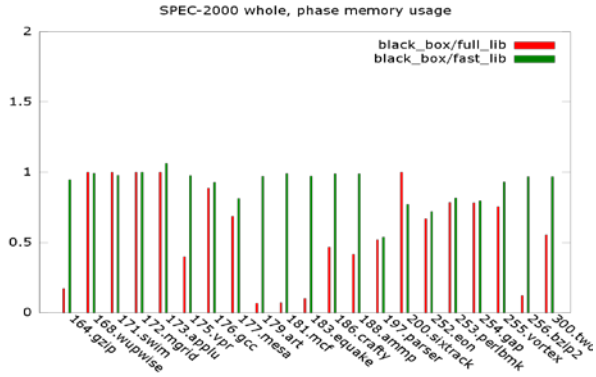

Рис 1.б Отношение расхода памяти

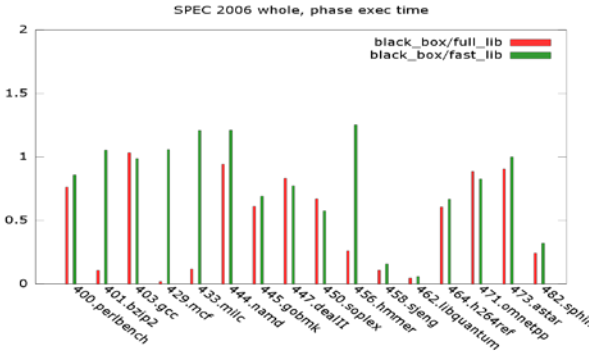

Рис 2.а Отношение времени исполнения фазы

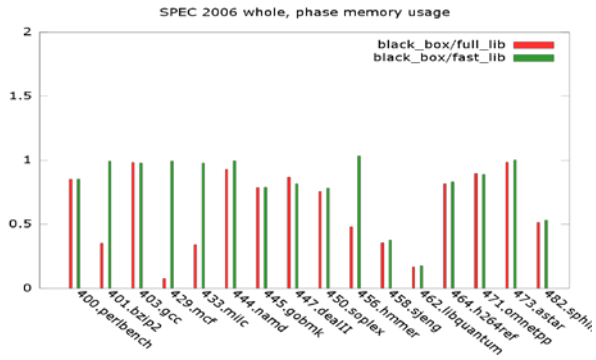

Рис 2.б Отношение расхода памяти

Красными столбцами обозначены замеры, показывающие отношение параметра в режиме «*black_box*» к режиму «*full_lib*», а зелёными - отношение параметра в режиме «*black_box*» к режиму «*fast_lib*». Т.е. чем меньше значение столбца тем хуже результат. Значение единица показывает, что отсутствуют изменения в затрачиваемых ресурсах.

Для набора задач SPEC-CPU2000 использование набора опций «*full_lib*» даёт деградацию в скорости анализа указателей в среднем на 55%, а расход памяти увеличивается в среднем на 40%. При использовании набора опций «*fast_lib*» деградации по времени работы анализа составляют в среднем 1%, что находится в пределах погрешности, а среднее увеличение использованной памяти составляет всего 9%.

Наибольшее ускорение анализа указателей произошло в задаче 183.equake на 50%, а максимальная деградация происходит в задаче 197.parser на 65%. При этом стоит отметить, что на задачах 164.gzip и 256.bzip2 при опциях «*full_lib*» происходят деградации времени работы анализа указателей на 96% и 97% соответственно, а при «*fast_lib*» в задаче 164.gzip

наблюдается деградация на 25%, а в задаче 256.bzip2 ускорение на 5%. Это связано с тем, что объём исходного кода задач сопоставим с объёмом кода используемых ими библиотечных функций.

Единственной задачей, уменьшившей потребление памяти при наборе опций «*fast_lib*» является 173.applu, показавшей улучшение на 6%. Наибольшую деградацию по использованной памяти показала задача 197.parser — анализ указателей на этой задаче стал потреблять на 46% больше памяти. Подобные эффекты происходят из-за отсутствия анализируемых функций в модели. Обычно это библиотечные функции, не описанные в стандарте. Например, в задаче 197.parser при работе анализа указателей было зафиксировано 421 уникальных обращений к модели, из которых только для 55 обращений содержалась информация.

Для набора задач SPEC-CPU2006 использование набора опций «*full_lib*» даёт деградацию в скорости анализа указателей в среднем на 50%, а расход памяти увеличивается в среднем на 37%. При использовании набора опций «*fast_lib*» деградации по времени исполнения составляют в среднем 20%, а среднее увеличение использованной памяти составляет 19%.

Наибольшее ускорение анализа указателей произошло в задаче 456.hmmer на 25%, а максимальная деградация происходит в задаче 462.libquantum на 94%. Время работы анализа на задаче 462.libquantum с набором опций «*fast_lib*» аналогично времени работы с набором опций «*full_lib*», что связано с большим количеством библиотечных функций, не включённых в модель — 328 из 367. На задаче 456.hmmer ускорение связано с тем, что все используемые библиотечные функции оказались в модели.

Как и в SPEC-CPU2006 уменьшения расхода памяти почти не наблюдается — только на задаче 456.hmmer на 3%, что находится в пределах погрешности. Наибольшее увеличение расхода памяти показала задача 462.libquantum на 82%. Также значительная разница между наборами опций «*full_lib*» и «*fast_lib*» наблюдается на задачах 401.bzip2 и 433.milc. Причины таких разрывов те же, что и для задач 164.gzip и 256.bzip2.

Замеры времени исполнения тестов не дали значимого результата, что свидетельствует о незначительном влиянии результата анализа библиотечных функций на эффективность анализа указателей и, как следствие, скорость исполнения данных задач.

Замеры времени исполнения

Для оценки влияния внедрённой модели на общую скорость работы программ были произведены замеры времени исполнения задач из набора SPEC-CPU2000. Замеры производились для сборок в помодульном режиме и в режиме «вся программа». Сравнение производилось между тремя разными режимами, описанными выше: «*black_box*», «*full_lib*»,

«*fast_lib*». Погрешность измерений составила 0.44%, что объясняется малым влиянием внешних факторов на среду исполнения.

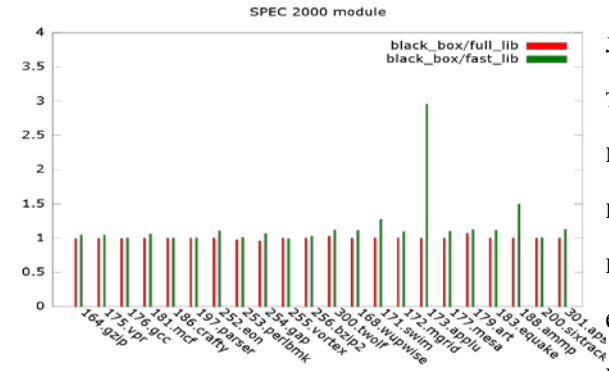

Рис. 3.а Скорость исполнения задач в помодульном режиме сборки

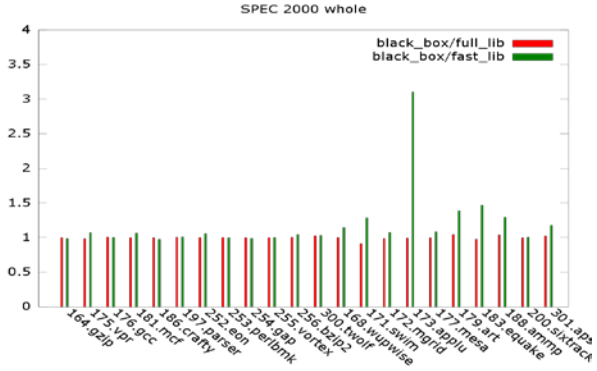

Рис. 3.б Скорость исполнения задач в режиме сборки «вся программа»

Результаты можно видеть на Рис. 2 а,б.

В помодульном режиме сборки разница в режимах «*black_box*» и «*full_lib*» отсутствует, что является закономерным следствием того, в обоих режимах отсутствует информация или код библиотек. Режим «*fast_lib*» показал в среднем ускорение на 18%. При этом максимальное замедление произошло в задаче 255.vortex на 2%, а максимальное ускорение произошло в задаче 173.applu на 295%, т. е. почти в три раза. Стоит отметить, что это единичный выброс и ускорение на остальных задачах не превышает 50%.

Для режима сборки «вся программа» наблюдаются схожие результаты. Средняя разница между режимами «*black_box*» и «*full_lib*» составляет 1%. Режим «*fast_lib*» показал среднее ускорение на 19%, также с выбросом на задаче 173.applu, ускорившейся на 310%. Ускорения по остальным задачам также не превышают 50%. Наибольшая деградация наблюдается на задаче 186.crafty, замедлившейся на 3%.

Выводы

В статье рассмотрено влияние использования модели поведения библиотечных функций вместо анализа их кода. Результаты исследований показали, что использование такой модели значительно уменьшает среднее время работы анализа указателей по сравнению с полным анализом содержимого библиотечных функций и сокращает расходы по памяти. Однако сокращение потребляемых ресурсов при анализе указателей не отразилось на времени исполнения задач.

Замеры времени исполнения при использовании модели поведения библиотечных функций прочими оптимизациями показали в среднем ускорение работы программ как в помодульном режиме, так и в режиме сборки «вся программа», что делает использование

подобной модели полезным для практического применения.

Литература

1. «Optimizing compilers for modern architectures», Randy Allen & Ken Kennedy
2. «Interprocedural Dataflow Analysis in the Presence of Large Libraries», Atanas Rountev, Scott Kagan, Thomas Marlowe
3. http://www.spec.org/ [Электронный ресурс]