Четверина О.А.
(ЗАО «МЦСТ»)

Статический выбор оптимальной стратегии оптимизации.

В докладе рассматривается задача выбора оптимальной стратегии компиляции на основе статистических данных о времени компиляции и исполнения, а также значении ключевых параметров процедур. Предложен алгоритм кластерной минимизации ошибки и представлен результат его применения на реальных задачах.

Ключевые слова: оптимизирующий компилятор, кластерная минимизация ошибки выбора, optimizing compiler, optimization sequence solver

Введение.

Оптимизация кода является важным методом повышения производительности вычислительных систем. Как правило, для принятия решения по применению оптимизирующих преобразований компиляторы используют статистически подобранные эвристики. При этом ряд преобразований или применяются только при возведении дополнительных опций, или в некоторых случаях приводят к ухудшению производительности. Проводится много исследований по вопросам создания оптимального набора оптимизаций для отдельно взятой задачи, составляющего стратегию оптимизации [1]. Некоторые исследования направлены на разработку набора стратегий оптимизации для пакета задач, позволяющего получить хорошие результаты на задачах в среднем [2], [3]. На данный момент задача выбора оптимизационной линейки решается итерационно: либо производится многократное исполнение задач, либо для разных стратегий оценивается результат исполнения по спланированному в каждом случае коду. В представленной работе изучается вопрос статического выбора последовательности оптимизаций на раннем этапе компиляции. Как будет показано, такой подход позволяет не только повысить производительность, но и сократить время, затрачиваемое на компиляцию.

В разделе 1 будет поставлена задача выбора наилучшей стратегии из имеющийся по вычислимым заранее свойствам (параметрам) процедуры, и в разделе 2 будет предложен способ ее решения. Кроме того, в разделе 3 будут приведены результаты теоретической оценки эффективности и результаты применения на реальных задачах.

1. Задача выбора стратегии.

Как уже упоминалось, создание оптимального набора стратегий является отдельной

задачей, и есть разные подходы к ее решению. В данном докладе основное внимание будет уделено выбору стратегии из имеющегося, тем не менее стоит отметить, что разработка стратегий во многом зависит от требуемых свойств набора. Так, часто используемые в работах настройки фактора раскрутки [1-3] не представляет особого интереса для статического предсказания, поскольку неоптимальность применения раскрутки обычно связана с неточностью предсказанного значения числа итераций цикла, либо с наличием только среднего значения при отсутствии распределения числа итерации. При многократном исполнении задачи с различными факторами раскрутки неявно производится уточнение профиля, что и позволяет произвести настройку .

Для постановки задачи выбора оптимальной стратегии нужно построить критерий оценки качества выбора стратегий и зафиксировать набор используемых для выбора входных параметров, соответствующих свойствам процедур. Необходимость введения числового критерия качества довольно очевидна при наличии требования учета времени компиляции, но в действительности она возникает даже в том случае, если единственное, что требуется, это ускорить время исполнения. Предположим, что выявилось 3 процедуры, у которых все значения параметров совпадают, но оптимальные стратегии различны (Таблица 1.). Если для них будет выбрана стратегия, являющаяся оптимальной для большинства процедур, то суммарное время исполнения будет больше, чем если выбрать стратегию, выбор которой более значим с точки зрения разницы времени исполнения. В качестве функционала, соответствующего оптимальному выбору при минимизации времени исполнения, можно выбрать сумму времен исполнения. Если же предполагается равнозначность тренировочных процедур, то можно дополнительно ввести нормировку относительно стартовых значений.

Таблица 1

	Время на стратегии 1	Время на стратегии 2
Процедура 1	1000	500
Процедура 2	200	220
Процедура 3	300	305
Суммарное время исполнения	1500	1025

В представленном докладе будет дана общая формулировка, при которой оценка качества производится с помощью функционала, зависящего от времен компиляции и исполнения всех использующихся для тренировки процедур. Набор параметров — это набор J параметров с областью значений в R, определенных для каждой процедуры. При этом часть параметров могут иметь непрерывную область значений, часть могут принимать только целые значения. Каждой стратегии соответствует набор последовательность оптимизаций, то есть *оптимизационная линейка*. В этом случае задачу выбора стратегий можно

сформулировать следующим образом:

Пусть есть вектор $(l_{i1}, l_{i2} \dots l_{in})$, где N — число процедур, l_{ik} — номер оптимизационной линейки для k-ой процедуры p_k. И пусть есть таблицы $\text{comp}(p_k, i), \text{exe}[p_k, i]$ размерами $N \times I$, где I - число стратегий. Определим некоторый функционал качества $F(l_{i1}, l_{i2} \dots l_{in})$ на множестве L^N, зависящий от $\text{comp}[p_k, l_{ik}], \text{exe}[p_k, l_{ik}]$.

Кроме того, каждой процедуре p_k соответствует точка в пространстве параметров R^J, то есть построено отображение $U: P \to R^J$.

Нужно построить отображение из множества параметров в множество линеек $G: R^J \to L$ таким образом, чтобы на нем функционал $F(G(U(p_1)), \dots G(U(p_N)))$ достигал минимума при условии, что для любой процедуры p_k в пространстве параметров есть некоторая окрестность, содержащая заданное число S точек p_s из P, применение на которой линейки $G(U(p_K))$ не увеличивает среднее значение функционала по сравнению с применением базовой на множестве всех процедур.

Последнее требование связано с необходимостью обеспечить некоторую непрерывность выбора линеек и увеличить объем выборки для более точной оценки результата применения линеек на процедурах, не входящих в тренировочных набор.

2. Решение задачи выбора стратегии.

Поскольку критерием качества выбора является значение общего функционала, а не вероятность выбора оптимальной линейки для отдельной процедуры, то для решения задачи плохо подходят вероятностные модели, такие как Байесовские сети, или в явном виде нейронные сети. Кластеризация по метрике тоже невозможна, поскольку метрика в данном случае определена только при фиксации общей для процедур линейки. Вместо этого был разработан *алгоритм кластерной минимизации ошибки* функционала, выделяющий классы с преобладанием выбора одной линейки.

Вначале для всех процедур была подсчитана таблица *ошибок выбора линеек* размера $N \times I$. Для этого была взята подсчитанная методом градиентного спуска точка минимума функционала $F(G(U(p_1)), \dots G(U(p_N)))$. Затем для каждой процедуры и каждой линейки был подсчитан логарифм отношения значения функционала, подсчитанного для отклонения от минимальной точки в сторону соответствующей линейки для процедуры, и значения

минимума функционала . Взятие логарифма полезно, чтобы в дальнейшем вместо вычисления среднего геометрического можно было перейти к суммам. Итого, для каждой процедуры есть оптимальная линейка с значением ошибки равным 0 , остальные – с большими значениями. Далее, была подсчитана стартовая ошибка $Wstart$, то есть сумма ошибок применения базовой линейки ко всем процедурам.

Алгоритм:

1)Назначаем всем процедурам базовую линейку, текущая ошибка $W=Wstart$.

2)Выбираем процедуру с максимальной ошибкой на текущей для нее линейке среди непомеченных флагом. Если таких нет, прекращаем работу. Берем оптимальную для нее линейку $line$.

3)Вычисляем суммарную ошибку $Wtmp$ на всех процедурах при применении линейки $line$. Для каждого параметра нижнее и верхнее расстояние до границ множества параметров.

4)Исполняем цикл по параметрам. С коэффициентом $t_1<1.0$ уменьшаем расстояние до верхней или нижней границы одного из параметров. Считаем суммарную ошибку $Wtmp$. Если она уменьшилась или осталась такой же, оставляем границы, иначе возвращаем предыдущие. Если границы для параметров перестали уменьшаться, идем в 5), иначе повторяем 4).

5)Исполняем цикл по параметрам: С коэффициентом $t_2>1.0$ увеличиваем расстояние до верхней или нижней границы одного из параметров. Считаем суммарную ошибку $Wtmp$. Если она уменьшилась или осталась такой же, оставляем границы, иначе возвращаем предыдущие. Если границы для параметров перестали увеличиваться, идем в 6), иначе повторяем 5).

6) Если $W-Wtmp>p*Wstart$, где $p<1.0$, и число элементов в подмножестве больше S , то подмножеству точек в пространстве параметров, ограниченного текущими значениями границ, назначаем линейку $line$. Стартовую процедуру помечаем флагом. Идем в 2.

Полученные в результате работы алгоритма в пункте 6) границы параметров используются как границы кластеров для соответствующих линеек. При этом кластеры могут иметь пересечения, в таком случае элемент будет принадлежать тому кластеру, который был построен позже.

Оптимальные параметры для имеющейся тренировочной базы $t_1=0.9$, $t_2=1.05$, $p=0.01$ были подобраны эвристически . Выделение кластеров можно начинать с любой имеющейся линейки. В описанном алгоритме стартовый класс привязан к базовой линейке, поскольку в компиляторе, на котором проводилось исследование, она

является в среднем оптимальной для различных режимов. Вместо фиксации кластера при выборе «образующей» точки, можно формировать кластеры с различными начальными точками, и выбирать тот, для которого $W - Wtmp$ максимальна, либо тот, для которого максимально среднее уменьшение ошибки, то есть $(W - Wtmp)/s$, где s - число элементов в получившемся кластере.

3. Результаты применения алгоритма.

Алгоритм кластерной минимизации ошибки выбора тестировался на базе компилятора Эльбрус. В качестве тренировочной базы были использованы 9183 процедур пакета spec2000 и 4 различные стратегии. При построение кластеров был использован функционал, минимизирующий общее время исполнения. Начальное значение общей ошибки 0.143064 , при последовательном построении для s=3 были зафиксированы 7 кластеров (см Рис.1), при этом суммарная ошибка изменялась следующим образом: 0.0752633, 0.0629314, 0.0495115, 0.0409881, 0.0400379, 0.0386413 , 0.0345678, еще через 21 шаг 0.0270214.

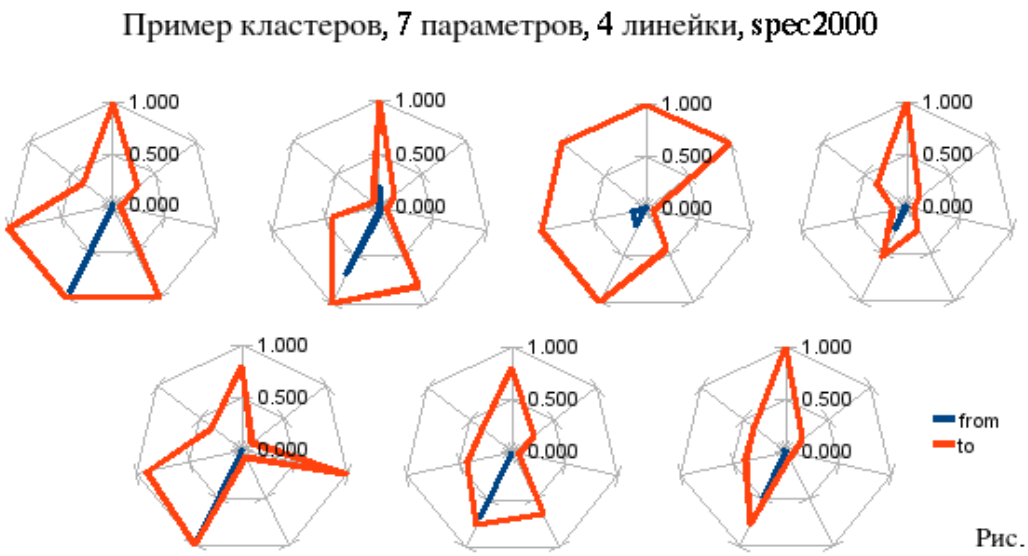

Рис. 1

При построении обучающей базы для оценки времени исполнения процедур был использован профиль исполнения на полных данных. На Рис.2 приведен результат сравнения применения кластеризации при построении 7 кластеров с теоретическим оптимумом, ошибка функционала на котором равна 0. Причина ухудшения исполнения ряда задач при работе в режиме теоретического оптимума связана с эффектами работы памяти. Получен средний прирост производительность 7.34% на кластерной модели по сравнению с 8.35% теоретическим оптимумом для тренировочных данных, то есть достигнуто 88% потенциального ускорения. Стоит отметить, что полученный эффект не является максимально возможным и зависит от качества построения оптимизационных линеек и выбора параметров.

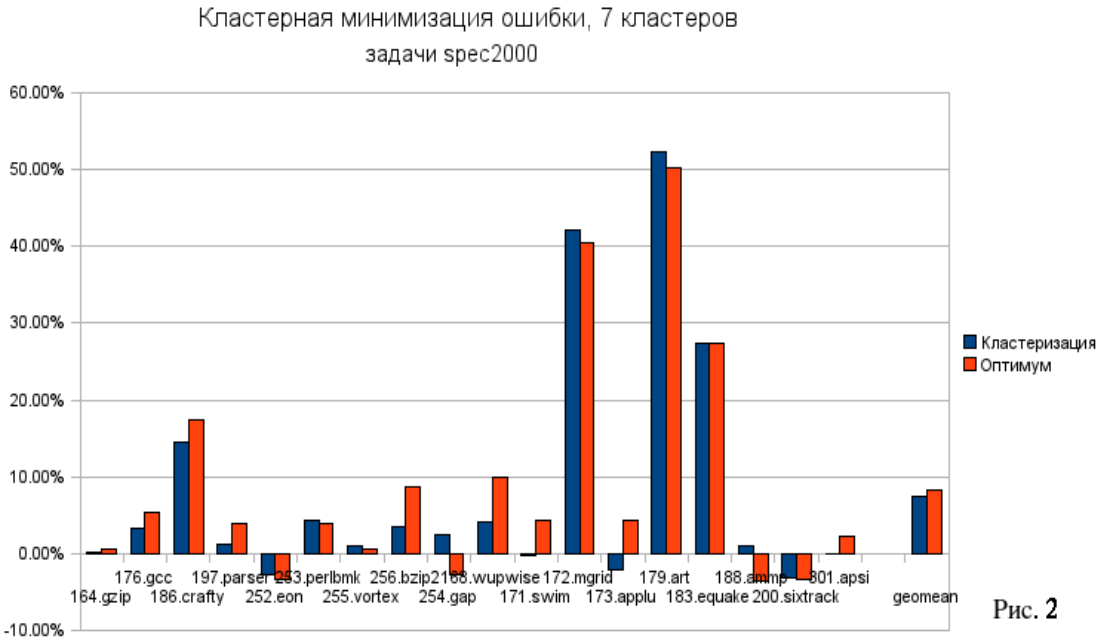

Рис. 2

Выводы.

В представленном докладе описан разработанный механизм кластерной минимизации ошибки для статического выбора оптимальной стратегии компиляции процедуры. Было показано, что он позволяет выделить весьма незначительное число кластеров, получив при этом основную часть возможного эффекта минимизации функционала качества, что говорит об устойчивости механизма выбора. Так, на задачах spec2000 было достигнуто 88% потенциального прироста производительности при выделении всего 7 кластеров. Дополнительное удобство разработанного алгоритма заключается в том, что он не требует введения в пространстве параметров равномерной метрики.

Литература

1. Prasad A. Kulkarni , David B. Whalley , Gary S. Tyson, Evaluating Heuristic Optimization Phase Order Search Algorithms, Proceedings of the International Symposium on Code Generation and Optimization, p.157-169, March 11-14, 2007

2. Spyridon Triantafyllis , Manish Vachharajani , Neil Vachharajani , David I. August, Compiler optimization-space exploration, Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization, March 23-26, 2003, San Francisco, California

3. Suresh Purini, Lakshya Jain, Finding good optimization sequences covering program space, Transactions on Architecture and Code Optimization (TACO), January 2013