Распределённый итеративный поиск опций компиляции

Леонид Евгеньевич Брусенцов
(Институт автоматики и электрометрии СО РАН, ЗАО «Интел А/О», г. Новосибирск)
e-mail: leonid.brusencov@intel.com

Аннотация

Рассматривается проблема поиска оптимальных значений опций компиляции, обеспечивающих создание исполняемого кода с заданными характеристиками. Обсуждается класс методов, основанных на итеративном поиске, требующих массированных вычислений. Предлагается архитектура распределённой децентрализованной системы поиска в многомерном пространстве для решения проблемы, приводятся результаты эксплуатации прототипа.

Введение

На текущий момент в науке и промышленности активно используются оптимизирующие компиляторы. Это обусловлено тем, что исследователям и производителям необходимо создавать эффективные программные системы, удовлетворяющие современным, высоким стандартам. Однако, количество параметров оптимизации, наборов опций и их значений в современных оптимизирующих компиляторах настолько велико, что невозможно, проанализировав программу, ответить на вопрос: какой набор значений параметров будет самым оптимальным для той или иной программы.

Эвристические алгоритмы компиляторов также не справляются с данной задачей. Это вполне объяснимо: невозможно придумать универсальный алгоритм определения значений параметров оптимизации для любой программы, любой архитектуры, набора опций, среды исполнения программы и входных данных. Такая задача является неразрешимой.

Поиском решения для описанной проблемы занимается множество исследователей по всему миру и одной из популярных техник для решения задачи стал итеративный перебор значений параметров оптимизации компилятора [1, 2, 3, 4]. Эта техника решает проблему подбора оптимального набора значений для параметров оптимизации при компиляции программы, производя несколько десятков, сотен или тысяч итераций (циклов построения – запуска программы – измерения производительности) и меняя от итерации к итерации набор параметров компиляции, как всей программы, так и отдельных её элементов. При этом количество итераций зависит от многих факторов, среди которых подход к поиску решения, алгоритмы перебора, уровень детализации поиска, количество учитываемых участков программы, и т. д. Однако описанная техника существенно увеличивает время, необходимое, чтобы получить оптимизированный код из исходных текстов программы в сотни – тысячи раз.

Решение проблемы уменьшения времени поиска при использовании итеративного перебора или других техник оптимизации с обратной связью привлекает внимание специалистов по всему миру. Однако были найдены лишь частичные решения, например, выполнение нескольких итераций поиска в одном запуске, применение различных алгоритмов, ускоряющих поиск, и другие. Для решения проблемы необходимо либо избавиться от итераций совсем, либо существенно сократить их необходимое количество, либо уменьшить время выполнения каждой итерации.

Однако, несмотря на большое количество опубликованных методик, количество итераций для совершения полного перебора настолько велико (числа порядка 2^{3000}), что даже применение всех этих подходов не помогает в полной мере, всё равно остаётся огромное число необходимых итераций для получения отчётливых полезных результатов.

Компания Интел тоже внесла свой вклад в исследования данной области, разработав итеративный статистически-эвристический инструмент поиска с обратной связью наилучших значений опций своего компилятора для оптимизируемого кода [1]. Но, несмотря на то, что удалось добиться некоторых стабильных результатов, в работе рассматривалось лишь очень малое количество опций компилятора для перебора. Поэтому мы решили провести работы по его адаптации для возможности работы со всеми опциями компилятора сразу и на длительных промежутках времени – месяцами или годами.

В свете постоянного удешевления компьютерной аппаратуры, помещения всё нарастающего количества ядер на подложках процессоров, появляются широкие возможности по организации распределённого поиска.

Работа посвящена проблемам создания и организации систем итеративного поиска эффективных значений оптимизирующих опций компиляции.

В статье приводится специфика задачи, формулируются требования к системе итеративного поиска, предлагается распределённая архитектура системы и обсуждаются результаты испытаний реализации первого прототипа.

Специфика задачи

Оптимизация конкретных приложений для конкретных вычислительных устройств предполагает проведение измерений производительности получаемых вариантов исполняемого кода на устройствах идентичных целевому.

Оптимизируемые приложения бывают различными, неизвестно заранее сколько может понадобиться машинных ресурсов для исполнения того или иного, код может быть как работающим параллельно (возможно даже автоматическое распараллеливание при компиляции), так и строго последовательно.

Хотя при решении задачи необходимо использовать компьютеры, аналогичные целевому, существенная часть вычислений не привязана к архитектуре вычислителя.

Существует множество способов ускорения поиска, выработаны различные алгоритмы работы в пространствах, однако не решена проблема поиска теоретически достижимых величин оптимизации исследуемого кода для целевой машины. Это значит, что работу приходится вести практически вслепую, нет никаких достоверных признаков того, что поиск пора заканчивать. А поскольку совершение полного перебора в пространстве в данный момент является невыполнимой задачей, поиски запускаются на длительное время, во время которых возможны отказы узлов и появление новых требований к системе.

На результат измерения могут влиять системные процессы, которые непредсказуемо изменяют время исполнения исследуемого приложения.

Работа системы итеративного поиска сопряжена с накоплением большого числа разнородной информации.

Требования к системе итеративного поиска

1. Машины, на которых производятся измерения результатов компиляции, должны быть идентичными целевой платформе.
2. Не допускается запуск каких-либо посторонних задач, по возможности требуется ограничить количество работающих фоновых служб. Также недопустим параллельный запуск нескольких версий оптимизируемого приложения.
3. Операционные системы и наборы динамических библиотек должны быть одинаковыми, поскольку потенциально принимают участие в работе приложения.
4. Для обеспечения адекватности получаемых данных необходимо, чтобы трансляция приложения происходила с применением одного и того же компилятора и версией его библиотек, опции компилятора говорили об одной и той же целевой архитектуре (то есть

чтобы компилятор собирал приложение, предназначенное для запуска при одних и тех же условиях).

5. Система должна быть рассчитана на длительное время работы и предусматривать устойчивость к аппаратным отказам компьютеров и возможным сбоям программного обеспечения, а также возможность внесения изменений в программное обеспечение системы во время работы.

6. Требуется обеспечить наглядное представление текущего состояния системы и возможность её управления.

7. Система должна предусматривать сохранение статистики о найденных решениях в долгосрочной перспективе.

8. Для хранения и эффективной обработки больших массивов собранной информации требуется организовать хранилище с быстрым и гибким способом доступа.

Распределённая архитектура системы итеративного поиска

В работе предлагается распределённая децентрализованная система запусков построений и профилирований на имеющемся наборе машин, организованная модульным образом, с предусмотренными элементами поддержки своих отдельных узлов, допускающая восстановление работы в случае чрезвычайных ситуаций, а также управляемая с помощью веб-интерфейса. Синхронизация работы осуществляется с помощью разделяемой всеми машинами памяти (файловое хранилище с предоставлением доступа через сервисы стандартных протоколов обмена данными, например NFS).

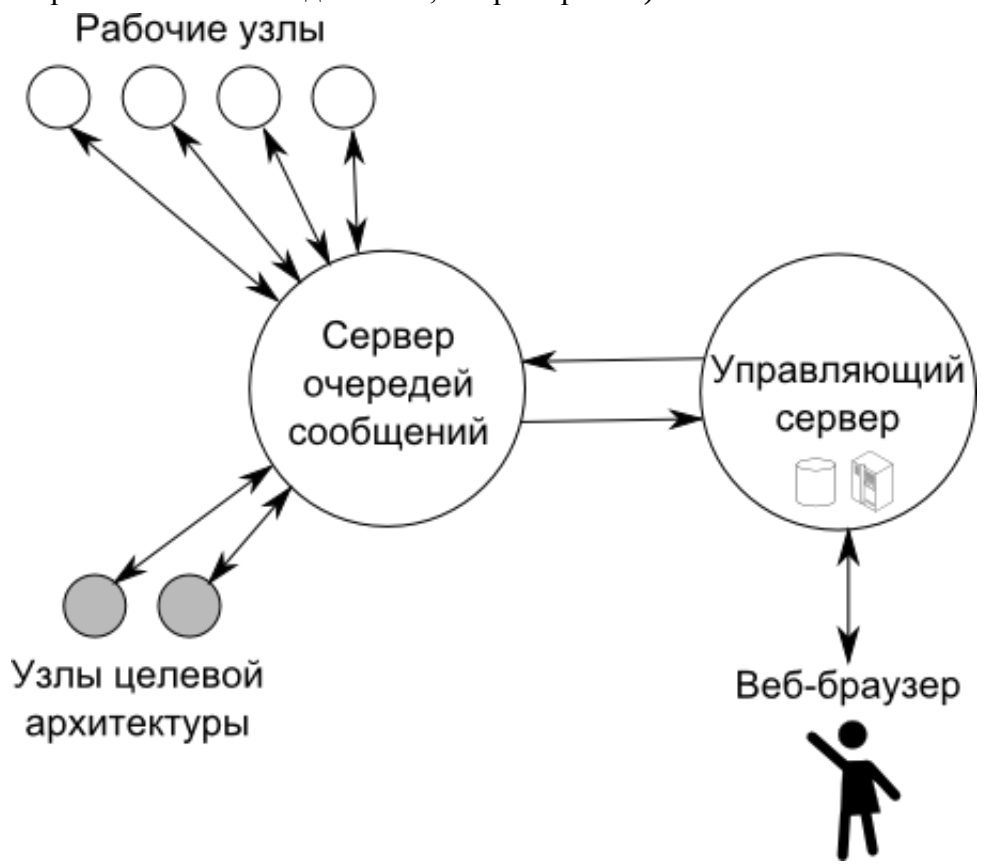

Рис. 1 Программная архитектура системы

Разделяемая память объёмом 50 Гб – сервер очередей сообщений, была организована на выделенном компьютере с установленными стандартными сервисами доступа. В основном он служит для общения отдельных узлов системы, но помимо того хранит различные версии компиляторов и библиотек, наборы опций компиляторов, а также свежие дистрибутивы узлов системы.

Поскольку сервер очередей сообщений содержит в себе историю текущих запусков, необходимую при восстановлении системы в случае отказов, он снабжён источником бесперебойного питания и средствами инкрементального резервирования информации. Для минимизации сетевого трафика, а также экономии места на диске, при передаче данных и сообщений используется архиватор WinZip.

Другая выделенная для управляющего сервера машина организует мониторинг системы, общение с пользователем, а также хранит результаты всех уже завершённых потоков. На ней установлен веб-сервер, хранятся веб-страницы интерфейса отображения, а также база данных MySQL для истории всех производимых когда-либо запусков. Для надёжности компьютер управляющего сервера также защищён источником бесперебойного питания. База данных предоставляет быстрый и гибкий доступ к большим массивам сохранённых данных.

Узел системы (программа, функционирующая на собственной машине) обменивается командами и данными с помощью текстовых сообщений, помещаемых на сервер очередей сообщений. Центральными командами являются выполнение построения приложения с заданными значениями опций компилятора, профилирование построенного экземпляра и запуск нового потока.

Рабочий узел следит за появлением нового задания и при наличии свободных ресурсов на несущей машине исполняет его. В заданиях описываются необходимые ограничения для выполнения, такие как операционная система, версии библиотек, а также характеристики вычислительного устройства (для совершения запусков на однородных компьютерах).

В процессе исполнения производится периодическое обновление информации о ходе выполнения каждой из задач для организации мониторинга. В конце исполнения все необходимые данные помещаются в разделяемую память для дальнейшей обработки.

Рабочий узел представляет собой модульный сервис, поддерживающий обновление, каждый модуль которого исполняется в отдельном потоке, а код расположен в динамически загружаемой библиотеке. Например, таким модулем является менеджер библиотек, предоставляющий пути до нужных версий файлов, а также устанавливающий их из сети в случае отсутствия. Отдельный модуль следит за появлением новых версий динамических библиотек для организации частичного самообновления сервиса.

Модули общаются через память процесса. На время организации запуска для измерения характеристик исследуемого варианта работа большинства потоков модулей приостанавливается.

В случае ошибок или неожиданных проблем предпринимаются попытки повторного исполнения задания с ведением истории. При определённом количестве неудач задача считается невыполнимой и в конечном итоге какая-то итерация пропускается.

Специальные модули сервиса рабочих машин поддерживают машины в работоспособном состоянии (в частности, очищают ненужные временные файлы, периодически перезагружают машину, осуществляют мониторинг состояния машины с передачей данных в распределённую память и многое другое).

Для универсальности, компонент системы, совершающий выбор последовательностей значений опций компилятора для каждой функции целевого приложения по отдельности, был реализован в виде отдельной программы – генератора, который приводится в действие специальным модулем при наличии соответствующего задания.

Генератор содержит в себе набор алгоритмов поиска, позволяет запускать, останавливать и продолжать прерванный процесс поиска, или поток. Для каждой отдельной функции допускается использование своего алгоритма, выбор определяется пользователем.

Генератор хранит актуальные результаты работы потока и генерирует по запросу очередные значения опций компиляции на основании анализа ранее полученных данных.

Последовательность действий определяется вне данного компонента и определяется числом свободных машин, временами построений и запусков.

Программа планировщик определяет, когда делать отправку задания, а когда принимать готовый результат. Она ищет выполненные задания, а также подсчитывает количество работающих машин, находящихся в обработке и в ожидании задач, и на основе этих сведений принимает решение о создании новых задач.

Результаты испытаний реализации системы

Предложенная архитектура была реализована на двадцати разнородных компьютерах. Из них шесть соответствовали целевой архитектуре – четырёх-ядерный процессор Intel® Xeon® 5160, предустановленные 8 Гб оперативной памяти и установленная Microsoft Windows® Server 2003 x64 Edition Service Pack 2.

На компьютерах для совершения запусков были отключены вспомогательные службы автоматического обновления Windows, интернет времени, сервер и многие другие. Тем самым удалось снизить разброс в измерениях времени исполнения (в дальнейшем – просто шум измерения) с 2% до сотых долей (данные по показаниям профайлера Intel® PTU®).

Через браузер был организован мониторинг и управление потоками поиска, а также мониторинг рабочих машин для организации коллективной работы.

Мониторинг потоков поиска включает в себя автоматическое построение графика на основе получаемых на каждой итерации измеренных данных с их шумами, а также отображение процента завершённости потока. Данная функциональность позволяет на ранней стадии определить неверно заданные параметры для потоков (например, слишком малое время для выполнения запусков, в этом случае приложение никогда не успевает закончиться и убивается модулем осуществившим запуск, это явно можно увидеть на графике).

Через веб-интерфейс оператор может запускать операции по управлению потоками (что сопряжено с автоматизацией управления).

Децентрализованная система позволила избежать проблем с узкими местами (распределённая память работает значительно быстрее, чем необходимо). Снижены риски связанные с возможными выходами из строя отдельных узлов или компьютеров (особенно центрального, если он имеется). При выходе из строя узла работа на остальных продолжается (выход из строя некоторых из них и вовсе не влияет на остальные).

Для возвращения в рабочее состояние узла достаточно перезапустить его несущую машину, не требуется прочего вмешательства в систему.

Заключение

Тестирование системы показало, что полученная система соответствует предъявляемым требованиям.

При случившемся внеплановом отключении электропитания (отключились все узлы кроме распределённой памяти и базы данных) система после восстановления электропитания продолжила выполнять начатые потоки.

Среднемесячная загрузка рабочих узлов системы стремится к 80%.

В будущих версиях мы планируем добавить на веб-страницы работу с базой данных, расширить возможности управления (в частности – продолжение работы потока после его остановки прямо со страницы), улучшить стабильность работы и оптимизировать некоторые узлы.

ЛИТЕРАТУРА

1. Artem Chirtsov, Leonid Brusencov, Ilya Cherny, Sergey Grebenkin, Sergey Ermolaev. “Maximizing Intel® Compiler Performance Using Iterative Feedback Directed Optimization”, SECR 2008

2. G. Fursin, M. O'Boyle, and P. Knijnenburg, "Evaluating iterative compilation," in Proceedings of the 15th Workshop on Languages and Compilers for Parallel Computing, (College Park, MD), July 2002.

3. F. Bodin, T. Kisuki, P.M.W. Knijnenburg, M.F.P. O’Boyle, and E. Rohou Iterative Compilation in a Non-Linear Optimization Space, Profile and Feedback Directed Compilation, PACT, 1998.

4. K. Chow and Y. Wu, Feedback-Directed Selection and Characterization of Compiler Optimizations, FDO, 1999.