Модельное профилирование при построении векторных архитектур процессоров

Д. В. Богаевский[1], С. Н. Ежов[2], Д. И. Каплун[3], А. Д. Кошкин, А. Д. Шахов
Санкт-Петербургский государственный электротехнический университет
«ЛЭТИ» им. В.И. Ульянова (Ленина)
[1] dan4ezz94@gmail.com, [2]sn_ezhov@mail.ru, [3]dikaplun@etu.ru

***Аннотация*. В работе представлен подход к реализации системного профилирования виртуальной архитектуры на эмуляторе QEMU. Описанный подход отличается возможностью исследования гостевых моделей векторных процессоров. Также, приводятся рекомендации по оптимизации векторизации, полученные на модели векторного процессора в ходе профилирования.**

Ключевые слова: обработка изображений; процессорная архитектура; профилирование; производительность; имитационное моделирование

I. Введение

Разработка и создание современных микропроцессорных решений требует больших трудозатрат, именно поэтому для повышения эффективности их разработки необходимо применение комплексных средств, позволяющих проводить оценку эффективности на тестовой выборке.

Такой подход позволяет быстро сопоставлять альтернативные подходы и обоснованно выбирать оптимальные решения при создании и модификации новых микропроцессорных архитектур.

Актуальность работы заключается в построении подходов и выборе методов и средств практического применения оптимизационных подходов, повышающих качество проектирования отечественных высокопроизводительных векторных процессоров на этапе проектирования.

Таким образом, целью работы является создание оптимизационных подходов, предназначенных для применения в процессе проектирования на базе новых и стандартных архитектур, а также формирование технологий, существенно повышающих эффективность разработки аппаратуры и программного обеспечения.

Задачи, которые позволят приблизиться к поставленной цели, сводятся к исследованию потока выполнения команд, отслеживанию работы с памятью и эмпирической оценке полученных данных. Для оценки производительности и оптимальности программных решений удобно использовать статистические методы применимо к различным метрикам исследуемого объекта, например, время, цикломатическая сложность, ошибка отклонения и другие.

II. Профилирование с использованием QEMU

В рамках данной работы исследование процессорной архитектуры проводится с использованием эмулятора QEMU [1, 2], позволяющего эмулировать отдельные пользовательские приложения, написанные для одной архитектуры, на другой. Открытый исходный код программы позволяет внедрять необходимые для исследования метрики прямо в эмулируемую модель.

На рис. 1 изображена схема, иллюстрирующая модельное профилирование в инструментальном комплексе на основе виртуальной машины QEMU. Инструкции гостевых программ интерпретируются симулятором, представляющим собой модель микропроцессора и его частей, формирующих структуру вычислительной системы. Сам симулятор представляет из себя прикладную программу, исполняющуюся на инструментальной платформе (host machine) под инструментальной операционной системой.

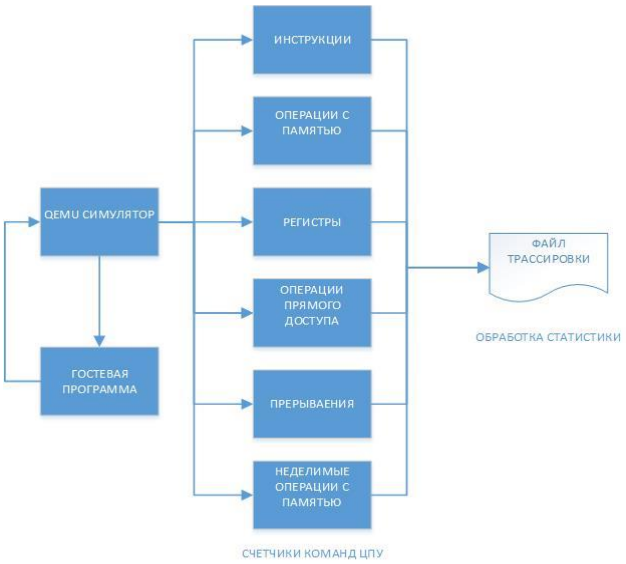

Рис. 1. Цикл трансляции и эмуляции кода в QEMU

Организация имитационного воспроизведения в QEMU основывается на взаимодействии с моделью аппаратного окружения [3] и осуществляется через: контроллер памяти, порты ввода\вывода, прерывания, платформенно-зависимые регистры, специальные инструкции.

Команды целевой архитектуры транслируются во внутреннее представление, из которого впоследствии генерируется машинный код для реальной архитектуры. Для подсчета количества выполненных инструкций предлагается ввести глобальную переменную внутреннего представления для хранения соответствующего значения. Во время трансляции во внутреннее представление необходимо увеличивать счётчик текущей инструкции на единицу. Обращение к платформенно-зависимым регистрам и выполнение специальных инструкций в QEMU транслируется в вызовы вспомогательных функций.

Предлагается модифицировать функции, отвечающие за машинно-зависимые регистры и специальные инструкции таким образом, чтобы значения, полученные на первом проходе, сохранялись в журнал событий, а во время воспроизведения считывались из него. В QEMU за контроллер памяти и порты ввода\вывода отвечает одна и та же подсистема работы с памятью, в которой вся память рассматривается как набор областей четырёх типов. Наиболее распространённым способом передачи большого объёма данных между устройством и операционной системой является прямой доступ к памяти. Для сохранения данных, записанных по указателю виртуальным устройством QEMU напрямую, предлагается инструментировать каждое виртуальное устройство отдельно.

Прерывания в эмуляторе QEMU обрабатываются в основном цикле и, соответственно, вызов соответствующих обработчиков прерываний ассоциирован с переменной interrupt_request. Предлагается сохранять значение данной переменной перед условным оператором, определяющим наличие необработанного прерывания, в том случае, если её значение изменилось по сравнению с предыдущим. Также необходимо сохранять номера аппаратных прерываний. Для реализации данной возможности необходимо инструментировать код обработчика аппаратных прерываний, который также находится в основном цикле QEMU.

Задачей повышения производительности кода на векторных архитектурах MIPS [3] является обнаружение параллелизма на уровне операций, планирование операций с учетом найденного параллелизма, обеспечивающее высокую логическую скорость, и одновременно с тем проведение оптимизаций, снижающих количество блокировок исполнения.

Выполнение модельного профилирования позволяет определить следующие характеристики исследуемого приложения:

- Метрики по отдельным функциям.
- Метрики по графам вызовов.
- Метрики по отдельным строкам исходного кода.
- Метрики по отдельным типам данных.
- Шкала времени с отображением различных событий в программе.

При оптимизации [4] необходимо учитывать особенности использования векторных регистров, развертки вложенных циклов в наборы последовательных операций, а также оптимальный набор флагов компиляции.

При проведении исследования нами использовались следующие метрики программного кода с последующей статистической обработкой:

- Затрачиваемое время.
- Количество обращений к памяти.
- Ошибка при упрощении алгоритма.
- Хронология и количество вызванных команд.
- Цикломатическая сложность.
- Влияние флагов компиляции.
- Частота вызовов векторных команд.

Проведение профилирования с использованием описанных метрик позволило выделить ряд проблем и рекомендаций по их устранению для наиболее оптимальной работы программного кода на исполняющем устройстве.

Причины уменьшения скорости работы циклов:

- Шаги цикла зависимы и не могут исполняться параллельно.
- Тело цикла или количество шагов мало, и выгоднее отказаться от использования цикла, чтобы избежать лишних условных переходов и операций инкремента.
- Тело цикла не помещается в память, так как используется слишком много регистров.
- Цикл содержит вызовы функций и процедур из сторонних библиотек.
- В цикле имеются условные переходы.

III. Рекомендации по программированию

Оценка эффективности рекомендаций для набора приложений существенно зависит от специфики предложенных к рассмотрению задач. В нашем случае для однородных тестовых примеров графического отображения анализ состоит в изучении границ области применимости и исследования качества реализации в компиляторе фиксированной цепочки оптимизаций. В то же время, целью анализа производительности на небольшом наборе реальных приложений является дополнение возможностей компилятора в целом, рекомендациями по организации дополнительных цепочек оптимизаций, позволяющих создавать наиболее эффективный код для отдельных значимых фрагментов каждого приложения.

Инструментация исходного кода QEMU и применение различных методов профилирования [5], в том числе описанных выше, позволили создать и эмпирически

подтвердить ряд рекомендаций по написанию программ для векторных архитектур.

Перемещение базовых блоков, что позволит расположить код часто исполняемых базовых команд близко друг к другу и сократит время вычисления адресов переходов.

Декомпозиция часто встречающихся блоков команд, имеющих много входящих и исходящих ребер, вероятнее всего, говорит о неоптимальной работе с памятью. В этом случае, возможно, получится отказаться от излишних загрузок данных и ускорить процесс исполнения программы.

Встраивание функций простых функций, основанное на частоте исполнения, позволяет использовать не стек при «вызове» такой функции, что в ряде случаев увеличивает производительность алгоритма.

Развертка циклов, применима к циклам с маленьким размером тела. Она похожа на ручную векторизацию. В таком случае можно эффективнее использовать каждую. Поэтому многократно дублируют тело цикла в зависимости от количества исполняющих устройств. Но такая оптимизация может вызвать зависимость по данным, чтобы от нее избавиться, вводятся дополнительные переменные.

Хорошей вариантом является переупорядочивание веток условий, основываясь на их логике и частоте исполнения, с целью минимизации затрат по предсказанию. Наиболее вероятные ветви рекомендуется располагать в начале ветвления. При этом, часть логических условий можно заменить на арифметические выражения. Очевидно, что это позволяет проверять меньше условий, а также реже совершать условные переходы, которые являются одной из самых ресурсоёмких операций.

Проверка предложенных решений производилась на следующей выборке алгоритмов обработки изображений:

- Фильтрация изображения методом свертки с окном.
- Преобразование цветовых пространств (RGB-YUV).
- Предварительная и постобработка FDCT и IDCT (forward/inverse discrete cosine transform).
- Квантование и деквантование.

Предложенные решения позволяют оценить параметры алгоритмов для векторного процессора, и определяют формирование набора команд, вносящих существенный вклад в производительность и пригодных к реализации на разрабатываемой архитектуре.

Для оценки затрачиваемого времени был использован high_precision_timer из библиотеки chrono C++11 и тестовое изображение с радужным градиентом, обеспечивающее максимальный цветовой охват. Выбор лучшей версии алгоритма производился с учётом минимизации затрачиваемого времени. Пример оценки среднего статистического значения времени работы алгоритмов представлен на рис. 2.

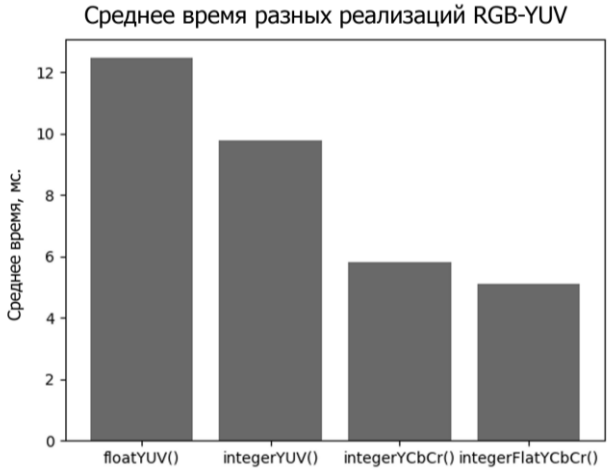

Рис. 2. Время работы алгоритмов

Измерение памяти при сохранении уровня допустимой погрешности позволяет оценить возможные искажения при преобразовании алгоритмов в их целочисленный аналог.

Применение оценки с использованием среднеквадратического отклонения позволяет учесть размер изображения и уменьшить фактор индивидуальности восприятия:

$$error = \frac{1}{W*H}\sqrt{\sum_{i=1}^{H}\sum_{j=1}^{W}(a-b)^2},$$

где “W” и “H” – размеры изображения в пикселях, “a” – значение в эталонном алгоритме, “b” – значение в его целочисленной версии.

Пример получаемых данных для алгоритма преобразования цветовых пространств приведён на рис. 3.

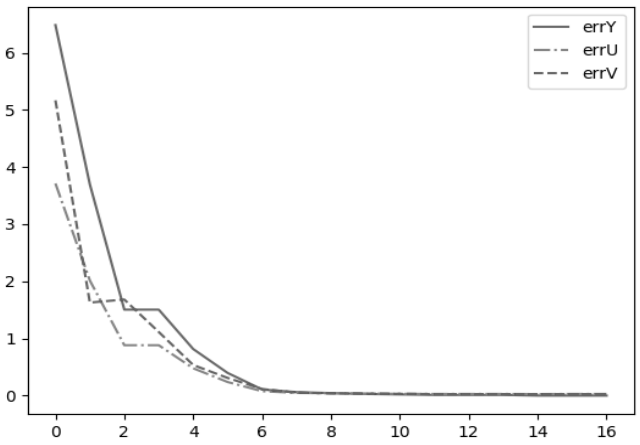

Рис. 3. Зависимость ошибки от разрядности

Полученные данные говорят о том, что память, выделяемую под временные переменные, можно

уменьшить с 16 бит до 7 бит без потери качества преобразования.

Описанные оценки являются объективными критериями корректности, так как зависят исключительно от числовых данных. Тем не менее, эти критерии не всегда соответствуют субъективным оценкам.

Изображения предназначены для восприятия человеком, поэтому единственное, что можно утверждать: плохие показатели объективных критериев обычно соответствуют низким субъективным оценкам, но хорошие показатели объективных критериев не гарантируют высоких субъективных оценок.

IV. Заключение

К сожалению, предложенные рекомендации не решают полный спектр проблем, возникающих при оптимизации программных и архитектурных решений, но их применение доказывает продуктивность дальнейших исследований в данной области. Именно поэтому мы продолжаем изучать полученные результаты профилирования, чтобы выявить больше практических рекомендаций и способов их внедрения.

Перспективным направлением дальнейшей работы является улучшение методов оценки производительности компилятора с целью обеспечения оперативности и достоверности результатов в зависимости от уровня оптимизаций. Возможным решением является использование полученной статистической информации на наборе тестовых задач графической обработки. Отдельной проблемой, требующей внимательного изучения, является выбор представительного класса задач для проведения анализа производительности. Особый интерес представляет сравнение производительности исполняемых файлов, полученных в разных компиляторах [5, 6].

В частности, продолжение работы будет развиваться в следующих направлениях:

- Введение новых параметров и методов профилирования.
- Автоматизация процесса профилирования.
- Верификация предложенных решений с использованием различных компиляторов.
- Разработка метода автоматического выбора опций компилятора, обеспечивающих оптимизацию целевой программы в соответствии с выбранным критерием.

Список литературы

[1] QEMU Emulator user documentation, URL: http://wiki.qemu.org/download/qemu-doc.html.

[2] Bellard Fabrice. QEMU, a fast and portable dynamic translator. Proceedings of the annual conference on USENIX Annual Technical Conference. ATEC '05. Berkley, USA: USENIX Association, 2005, pp. 41-46

[3] Shen J.P. and M.H. LIPASTI. Modern Processor Design: Fundamentals of Su-perscalar Processors. New York: McGraw-Hill, 2005.

[4] Ш.Ф. Курмангалеев. Методы оптимизации Си/Си++ - приложений распространяемых в биткоде LLVM с учетом специфики оборудования. Труды ИСП РАН, том 24, стр. 127-144, 2013 г. DOI: 10.15514/ISPRAS-2013-24-7. с.

[5] R. Levin, I. Newman, G. Haber. Complementing missing and inaccurate profiling using a minimum cost circulation algorithm. Proceedings of the 3rd international conference on High performance embedded architectures and compilers. HiPEAC’08. Berlin, Heidelberg: Springer-Verlag, 2008. Pp. 291-304.

[6] GCC 4.8.2 Manual, URL: http://gcc.gnu.org/onlinedocs/gcc-4.8.2/gcc/.