Для цитирования: Горелов М. А. Оптимизация доступа к элементам массива при аппаратной поддержке архитектуры «Эльбрус» // Вопросы радиоэлектроники. 2018. № 2. С. 51–54.
УДК 004.4'422

М. А. Горелов[1, 2]

[1] АО «МЦСТ», [2] ПАО «ИНЭУМ им. И. С. Брука»

ОПТИМИЗАЦИЯ ДОСТУПА К ЭЛЕМЕНТАМ МАССИВА ПРИ АППАРАТНОЙ ПОДДЕРЖКЕ АРХИТЕКТУРЫ «ЭЛЬБРУС»

Производительность современных научных приложений во многом определяется эффективностью исполнения вычислительных циклов. Поэтому оптимизация таких циклов является одной из наиболее острых задач оптимизирующей компиляции. Особенно это характерно для архитектур со статическим планированием кода, в частности для архитектуры «Эльбрус». В данной работе рассмотрена оптимизация, называемая Array Access, которая позволяет снизить количество операций, необходимых для вычисления адреса при регулярном обращении к элементам массивов, и за счет этого добиться более эффективного планирования цикла с точки зрения аппаратных ресурсов. Подробно описаны алгоритм работы оптимизации и аппаратные средства архитектуры «Эльбрус», на которых она основана. Также приведены результаты экспериментальных замеров производительности на задачах пакета SPEC CPU2006, подтверждающие эффективность оптимизации.

Ключевые слова: *оптимизирующий компилятор, архитектура «Эльбрус», цикловые оптимизации, array access.*

Введение

Существует широкий класс приложений, основную часть времени исполнения которых занимают вычислительные циклы. К ним относятся, например, научные и высокопроизводительные задачи. Для циклов в подобных приложениях характерны активная работа с памятью, обращение в которую происходит по регулярно изменяющимся адресам, и отсутствие побочных эффектов и глобальных меток.

Для архитектур со статическим планированием, в том числе для архитектуры «Эльбрус», производительность вычислительной системы во многом зависит от компилятора. Для ускорения описанных выше задач в оптимизирующий компилятор включен набор цикловых оптимизаций. Среди них есть как оптимизации, свойственные большинству компиляторов: раскрутка цикла, слияние циклов, программная конвейеризация (software pipelining), вынос инвариантов из цикла, – так и специфичные для компилятора «Эльбрус» оптимизации, в основе которых лежат свойства архитектуры: аппаратная конвейеризация (overlap), использование буфера асинхронной предподкачки данных, оптимизация доступа к элементам массива и другие. Применение цикловых оптимизаций позволяет наиболее «плотно» спланировать вычислительный цикл и способствует его более эффективному исполнению. Статья посвящена одной из перечисленных выше архитектурно-зависимых оптимизаций – Array Access, оптимизации доступа к элементам массива с использованием аппаратных средств архитектуры «Эльбрус».

Аппаратная поддержка доступа к элементам массива в микропроцессорах с архитектурой «Эльбрус»

Значительное время доступа во внешнюю память является одним из основных препятствий для обеспечения непрерывной работы и полной загрузки вычислительных устройств микропроцессора. Наиболее популярным методом преодоления этой проблемы является кэширование данных, однако оно имеет ряд недостатков, которые не позволяют преодолевать блокировки по ожиданию данных из памяти на многих современных научных приложениях. Для преодоления недостатков кэширования и оптимизации работы с подсистемой памяти используют технику предварительной подкачки данных, как программной, так и аппаратной. В качестве альтернативы автоматической аппаратной подкачке в архитектуру «Эльбрус» было включено специальное программируемое устройство обращения к массивам AAU (Array Access Unit). Подробнее с его устройством и особенностями реализации метода предварительной подкачки можно ознакомиться в [1, 2]. При этом кроме использования AAU по прямому назначению существует возможность использовать его для экономии вычислений числовых регистров при регулярном обращении к массивам. Именно о таком использовании AAU пойдет речь в данной работе.

Обозначим необходимые для дальнейшего изложения термины.

Регистровая память, входящая в состав AAU, содержит следующие регистры:

1. AAD (Array Access Descriptors) – 32 дескриптора доступа к массиву, каждый из которых описывает массив, а именно: содержит виртуальный адрес базы или индекс базы массива относительно общей базы стека, размер массива, права доступа и т.д.
2. AAINCR (Array Access Increment registers) – 8 регистров приращений, каждый из которых содержит значение приращения номера элемента массива. Регистр AAINCR.0 всегда содержит 1.
3. AASTI (Array Access Store Index registers) – 16 индексных регистров, каждый из которых содержит значение смещения, соответствующее индексу текущего элемента массива.

Данные регистры инициализируются в предцикле с помощью особых операций – AAURW и AAURR (Array Access Register Write/Read).

В системе команд также предусмотрены операции записи и чтения элемента массива – STAA и LDAA (Store/Load Array Access), адрес которых формируется из регистров доступа к массиву и выглядит следующим образом:

$$AAD.i + [AASTI.j \mathrel{+}= AAINCR.k] + lit\_arg,$$

где AAD.i описывает массив; AASTI.j содержит смещение, соответствующее считываемому или записываемому элементу; AAINCR.k содержит значение приращения AASTI.j на каждой итерации цикла; lit_arg – литеральный аргумент.

Кроме того, каждая операция содержит флаг am, который может принимать значения 0 или 1. Если он содержит 1, то вместе с выполнением операции будет инкрементирован регистр AASTI.j.

Оптимизация доступа к элементам массива с использованием аппаратной поддержки

Основной принцип оптимизации доступа к элементам массива заключается в том, чтобы использовать операции STAA и LDAA, описанные выше, вместо операций чтения и записи, обращающихся по регулярно изменяющимся адресам. Таким образом, поскольку AA-операции используют аппаратные регистры доступа к массиву, можно значительно сократить количество адресной арифметики в цикле и, как следствие, уменьшить его ресурсный размер, что позволит в итоге более оптимально его конвейеризовать [3].

Рассмотрим в качестве примера простой цикл с одним чтением:

```
for (i = 1; i < 10; i+=2)
{
    bar += foo[i];
}
```

После оптимизации он примет следующий вид:

```
AAD.0 = foo;
AASTI.0 = 0;
AAINCR.1 = 2;
for (i = 1; i < 10; i+=2)
{
    bar += LDAA (AAD.0 + [AASTI.0 += AAINCR.1]);
}
```

В предцикле будут построены операции инициализации регистров доступа к массиву, а операция чтения будет заменена на LDAA.

Алгоритм анализа и оптимизации можно разделить на пять смысловых частей:

1. Отбраковываются все циклы, которые содержат операции, обладающие побочными эффектами, в частности операции CALL, поскольку при вызове значения AA-регистров в стеке не сохраняются.
2. Происходит анализ операций – из всех операций чтения/записи выбираются те, для которых справедливо следующее:
   - операция выполняется на каждой итерации цикла;
   - известен шаг индексного выражения операции;
   - адрес операции выровнен.
3. Отобранные ранее операции обращения в память разбиваются на группы, в пределах которых индексы всех операций различаются на константу и имеют одинаковое приращение. Как было указано выше, положительный эффект от применения оптимизации обусловлен снижением количества адресной арифметики. Как правило, операции чтения и записи, индексы которых отличаются на константное значение, обращаются к элементам одного и того же массива, и, соответственно, какой-либо заметный эффект будет достигнут, только если оптимизация будет применена ко всем таким операциям. Кроме того, данный подход позволяет для каждой такой группы инициализировать один регистр AASTI. Это важно, поскольку индексных регистров всего 16, тогда как в некоторых вычислительных циклах может вестись работа с очень большим количеством массивов, и операций обращения к их элементам может быть заметно больше.
4. Если в результате применения оптимизации к операциям группы ресурсный размер цикла станет меньше, операции меняются на аналогичные AA-операции. При этом для всех создаваемых операций выставляется значение флага am, равное 0.
5. Последняя часть оптимизации работает непосредственно после планирования цикла. Для

каждой группы проводится поиск операции, постдоминирующей все остальные операции группы, и для нее выставляется значение флага am, равное 1.

На языке псевдокода алгоритм для цикла может выглядеть следующим образом:

```
Цикл по операциям чтения/записи oper
    Если oper не выполняется на каждой итерации или
    индекс oper не удовлетворяет условиям применения оптимизации
        oper не подходит
    Конец если
    index = получить индекс операции
    is_useful = сократит ли замена операции ресурсный размер цикла
    Цикл по группам операций group
        group_index = получить общую часть индексов операций group
        Если index – group_index == const
            Добавить oper в group
            Если is_useful == TRUE
                Пометить group для применения оптимизации
            Конец если
        Конец если
    Конец цикла
    Если подходящая группа не нашлась
        Создать новую группу group
        Если is_useful == TRUE
            Пометить group для применения оптимизации
        Конец если
    Конец если
Конец цикла
Цикл по группам операций group
    Если group помечена для оптимизации
        Цикл по операциям oper группы group
            step = найти шаг индекса oper
            Если есть AAINCR для такого шага step
                aaincr_num = номер этого AAINCR
            Иначе
                aaincr_num = инициализировать новый AAINCR
            Конец если
            nconst_ind = получить неконстантную часть индекса oper
            const_ind = получить константную часть индекса oper
            Если есть AASTI для такого nconst_ind
                aasti_num = номер этого AASTI
            Иначе
                aasti_num = инициализировать новый AASTI
            Конец если
            Если есть AAD для массива
                aad_num = номер этого AAD
            Иначе
                aad_num = инициализировать новый AAD
            Конец если
            Замена операции на AA-аналог
        Конец цикла
    Конец если
Конец цикла
```

На практике оказалось, что использование MOVA – операций пересылки данных из буфера асинхронной предподкачки массива APB (Array Prefetch Buffer) – дает больший прирост производительности, чем LDAA. А поскольку условия для применения APB и Array Access почти идентичны, LDAA практически не используются. Таким образом, оптимизация Array Access направлена в основном на применение к записям.

Производительность

Эффективность оптимизации доступа к элементам массива была проверена на задачах пакета SPEC CPU2006. Тестирование проводилось на машине «Эльбрус 401-PC» с одним четырехъядерным микропроцессором «Эльбрус-4C», работающим на тактовой частоте 800 МГц. Результаты (рис. 1, 2) представлены только для тех задач, для которых применение оптимизации дало заметный эффект.

При компиляции в режиме -O3 (гистограмма на рис. 1) наибольший прирост производительности наблюдается на задачах 434.zeusmp и 470.lbm и составляет около 3%; исполнение задачи 454.calculix ускоряется примерно на 2%. Деградация производительности также имеет место для некоторых задач, однако она не превышает 0,5%, и в среднем эффект от применения оптимизации положителен.

В пиковом режиме компиляции (гистограмма на рис. 2) наиболее заметный эффект применение оптимизации дает для следующих задач: 433.milc, которая ускоряется почти на 12%; 410.bwaves – на 5%; 462.libquantum и 454.calculix – примерно на 4% каждая. В среднем эффект также положительный.

Заключение

В данной работе была рассмотрена оптимизация доступа к элементам массива, основанная на аппаратных средствах архитектуры «Эльбрус». Использование этой оптимизации позволяет более рационально спланировать вычислительный цикл вследствие уменьшения его ресурсного размера. Экспериментальные данные подтверждают эффективность оптимизации для вычислительных задач.

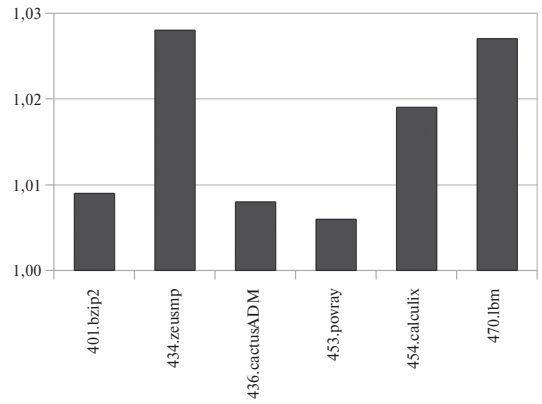

Рисунок 1. Прирост производительности при применении Array Access в режиме -O3

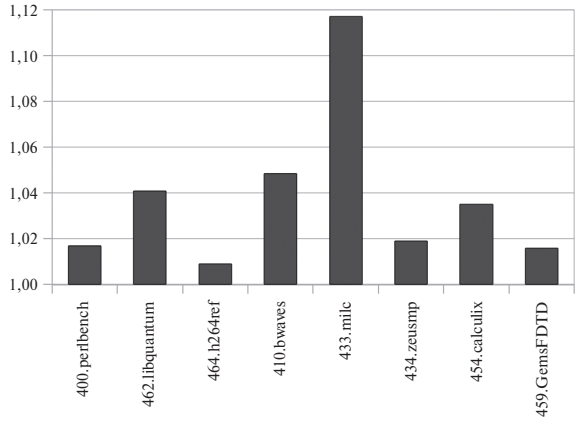

Рисунок 2. Прирост производительности при применении Array Access в пиковом режиме

СПИСОК ЛИТЕРАТУРЫ

1. Ким А.К., Перекатов В.И., Ермаков С.Г. Микропроцессоры и вычислительные комплексы семейства «Эльбрус». СПб.: Питер, 2013. 272 с.
2. Галазин А.Б., Грабежной А.В. Эффективное взаимодействие микропроцессора и подсистемы памяти с использованием асинхронной предварительной подкачки данных // Информационные технологии. 2007. № 5. С. 26–31.
3. Дроздов А.Ю., Степаненков А.М. Технология оптимизации цикловых участков процедур в компиляторах для архитектур с аппаратной поддержкой конвейеризации циклов // Информационные технологии и вычислительные системы. 2004. № 3. С. 52–62.

ИНФОРМАЦИЯ ОБ АВТОРЕ

Горелов Михаил Александрович, аспирант, ПАО «ИНЭУМ им. И.С. Брука»; инженер-программист, АО «МЦСТ», 119334, Москва, ул. Вавилова, д. 24, тел.: 8 (968) 096-94-61, e-mail: mikhail.m.gorelov@mcst.ru.

For citation: Gorelov M.A. Array Access optimization with hardware support of Elbrus architecture. Voprosy radioelektroniki, 2018, no. 2, pp. 51–54.

M.A. Gorelov

ARRAY ACCESS OPTIMIZATION WITH HARDWARE SUPPORT OF ELBRUS ARCHITECTURE

The efficiency of modern scientific and high-performance applications is mainly conditioned by the performance of computational loops. So computational loops optimization is one of the most crucial problems of optimizing compilation, especially for in-order architectures, such as Elbrus. Here we focused on the Array Access optimization, which allows to reduce the total amount of address operations and achieve more efficient loop scheduling in terms of hardware resources usage. In this paper, the optimization algorithm and necessary Elbrus architecture hardware means are described. The optimization is assessed on the SPEC SPU2006 benchmarks, and the results are given.

Keywords: optimizing compiler, Elbrus architecture, loop optimizations, Array Access.

REFERENCES

1. Kim A.K., Perekatov V.I., Ermakov S.G. *Mikroprocessory i vychislitelnye kompleksy semejstva Elbrus* [Elbrus microprocessors and computing systems]. Saint Petersburg, Piter Publ., 2013, 272 p. (In Russian).
2. Galazin A.B., Grabezhnoy A.V. Efficient interaction of the microprocessor and memory using asynchronous data prefetch. *Informatsionnye tekhnologii,* 2007, no. 5, pp. 26–31 (In Russian).
3. Drozdov A.Y., Stepanenkov A.M. Loop optimization with a compiler for the architectures with hardware support of loop pipelining. *Informatsionnye tekhnologii i vychislitelnye sistemy,* 2004, no. 3. pp. 52–62 (In Russian).

AUTHOR

Gorelov Mikhail, graduate student, PJSC Brook INEUM; software engineer, JSC MCST, 24, ulitsa Vavilova, Moscow, 119334, Russian Federation, tel.: +7 (968) 096-94-61, e-mail: mikhail.m.gorelov@mcst.ru.