УДК 004.021

Савин Андрей Сергеевич
ООО «Махуру»
Россия, Москва[1]
Программист
E-Mail: assavin90@gmail.com

Хохлов Алексей Анатольевич
ФГБОУ ВПО «Российский университет дружбы народов» (РУДН)
Россия, Москва
Доцент, кандидат физико-математических наук
E-Mail: khokhlov_aa@pfur.ru

Оптимизация алгоритма Singular Spectrum Analysis для ARM процессоров мобильных устройств

Аннотация: В настоящей работе рассматривается оптимизация вычислительных алгоритмов, используемых в распространенных методах анализа временных рядов. В качестве примера рассматривается метод Singular Spectrum Analysis, также известный как метод «Гусеница», реализованный и оптимизированный для процессоров, используемых в популярных сегодня смартфонах и планшетах. Для оптимизации были учтены рекомендации производителя процессоров – были использованы векторные регистры, произведена развертка вложенных циклов в наборы последовательных операций, а также подобран оптимальный набор флагов компиляции. Это позволило значительно увеличить производительность используемых алгоритмов, одним из которых является широко известный метод вращений Якоби. Данный метод используется для поиска собственных значений и собственных векторов матриц и является наиболее ресурсоемким в методе «Гусеница». Авторам статьи удалось увеличить скорость работы данного метода до 20 процентов в зависимости от размера окна («лага») исследуемого временного ряда, требуемой точности вычислений и других параметров, зависящих от условий конкретной задачи. В статье приведены описание и результаты вычислительного эксперимента, в рамках которого алгоритм метода «Гусеница» был реализован на языке программирования Objective C и представлены сравнительные результаты работы оптимизированного и неоптимизированного алгоритмов.

Ключевые слова: SSA; метод «гусеница»; оптимизация для ARM; метод вращений Якоби; диагонализация матриц; векторный регистр; Objective C; временной ряд.

Идентификационный номер статьи в журнале 110TVN214

[1] Москва, Орджоникидзе, 3, к.118

Повсеместное распространение смартфонов и планшетных компьютеров, повышение их вычислительной мощности, а также доступность мобильного интернета неизбежно привели к тому, что круг применения мобильных устройств расширился от использования для звонков, обмена сообщениями и несложных игр до использования в качестве устройств для видеосвязи, редактирования сложных документов и решения других задач самой разной направленности, в том числе инженерных, экономических и других. В частности, планшетные компьютеры давно используются как удобный и комфортный инструмент биржевых трейдеров и брокеров, включающий в себя возможность анализа и инструменты принятия решений.

На данный момент существует определенное количество мобильных приложений, решающих широкий круг аналитических и прикладных задач. Среди брокерского и биржевого программного обеспечения в качестве примеров можно привести мобильный торгово-аналитический терминал для iPhone с возможностью тестовой торговли «iSmart» и профессиональную торгово-аналитическую систему «SmartPAD» для устройств Apple iPad. Эти программы, помимо того, что предоставляют доступ к различной информации – состоянию рынков, истории котировок, текущим котировках, также позволяют проводить простую аналитику – выделять тренды, оптимизировать портфели и выполнять некоторые другие операции.

Помимо этих программ существуют и другие, однако их функциональность ограничена вычислительными ресурсами мобильных устройств. Поэтому задача оптимизации вычислительных алгоритмов для более эффективной работы на смартфонах и планшетных компьютерах является крайне актуальной в настоящее время.

В данной работе рассматривается метод «Гусеница» анализа временных рядов [2], известный также, как SSA (Singular Spectrum Analysis), реализованный на языке программирования Objective C [3] для работы на устройствах Apple iPhone и Apple iPad.

SSA не требует присутствия каких-либо особенных характеристик у исследуемого временного ряда, будь то стационарность, знания модели тренда, наличия периодических составляющих и других. При этом SSA успешно решает такие задачи, как, выделение трендов, обнаружение периодик, сглаживание ряда, построение полного разложения ряда в сумму тренда, периодик и шума и задачи фильтрации. Существуют модификации метода для анализа многомерных временных рядов (так называемый Multi-channel SSA), а также развита теория прогнозирования временных рядов при помощи алгоритмов, лежащих в основе SSA. В данной работе для примера используется классический метод SSA, применяемый к одномерному временному ряду.

Алгоритм стандартного метода SSA хорошо известен и изучен [4], поэтому опишем его вкратце. Из исходного одномерного временного ряда строится траекторная матрица, размерность которой определяется важным параметром, зависящим от условий конкретной задачи – длина гусеницы (иногда встречается название «лаг» гусеницы или «окно» гусеницы). Выбор окна гусеницы является важным этапом решения задач методом SSA. Небольшая длина гусеницы позволяет учесть меньше информации о ряде, большая длина гусеницы требует больших вычислительных ресурсов. Столбцами траекторной матрицы являются скользящие отрезки длиной, равной длине гусеницы. После некоторых преобразований, опционально включающих в себя процедуры нормирования и центрирования, строится квадратная матрица, содержащая в себе информацию об исходном временном ряде. Далее производится сингулярное разложение этой матрицы на сумму элементарных матриц, каждая из которых задается набором из собственного числа и двух сингулярных векторов — собственного и факторного. Таким образом, исходный временной ряд разлагается на интерпретируемы аддитивные составляющие. В зависимости от условий задачи производится

отбор главных компонент, по которым при помощи процедуры ганкелизации восстанавливается временной ряд.

Те или иные части алгоритма (например, нормирование или центрирование) могут отсутствовать, но его неотъемлемой частью является решение задачи нахождения собственных значений и собственных векторов для матрицы, полученной из траекторной. Известно, что эта задача является достаточно сложной, особенно для случая кратных собственных значений и необходимо со всей ответственностью подходить к выбору метода диагонализации матрицы.

Одним из наиболее точных и устойчивых методов для нахождения собственных значений и собственных векторов матриц является метод вращений Якоби [5], поэтому он используется авторами при вычислительной реализации алгоритма SSA. Однако, в то же время, метод Якоби достаточно ресурсоемкий, что в некоторой степени ограничивает его использование на мобильных устройствах: не всегда удается провести анализ ряда с необходимой длиной окна гусеницы, что, в свою очередь, не позволяет точно выбирать диапазоны частот фильтрации данных и другие параметры анализа. Помимо этого, для некоторых наборов исходных данных, алгоритм Якоби не обеспечивает сходимость за приемлемое число итераций, поэтому в работе используется устойчивая модификация алгоритма, разработанная коллективом кафедры систем телекоммуникаций РУДН для эффективного решения задач компьютерного дизайна многослойных оптических покрытий нанометровых размеров [6].

Авторами была проведена работа по оптимизации данного алгоритма для эффективной работы на ARM процессорах, которые используются в современных смартфонах и планшетных компьютерах. Были учтены рекомендации производителей процессоров для обеспечения эффективной работы с регистрами [7]. Подобран наиболее эффективный набор флагов компиляции программы, что позволило значительно увеличить производительность метода. Ниже описаны конкретные примеры действий по оптимизации работы алгоритма.

Для ускорения работы с циклами компилятору явно дается команда оптимизировать тот или иной участок исходного кода. Например, если известно, что в цикле производится работа с массивами дины, кратной четырем, то при написании кода для ARM процессора конструкцию вида:

```
1    for (i = 0; i < n; i ++)
2    {
3    // …
4    }
```

необходимо заменить на

```
1    for (i = 0; i < ((n / 4) * 4); i ++)
2    {
3    // …
4    }
```

Описанная выше конструкция укажет компилятору, что необходимо применять векторизацию, и это существенно уменьшает время выполнения операций с массивами внутри цикла.

Если в алгоритме часто производится работа с небольшими циклами, то можно дать явную команду компилятору для «разворачивания» данных циклов. Для этого можно воспользоваться следующей конструкцией:

```
1       #pragma unroll(n)
```

Особенно это полезно для вложенных циклов (отметим, что в используемой реализации алгоритма Якоби уровень вложенности достигает пяти). Развернув внутренний цикл в набор последовательно выполняемых команд, компилятор применяет векторизацию к «развернутым» из цикла данным, что также может дать ощутимый прирост в производительности.

Сравнительные результаты времени работы для оптимизированного и неоптимизированного алгоритмов представлены на рисунке 1.

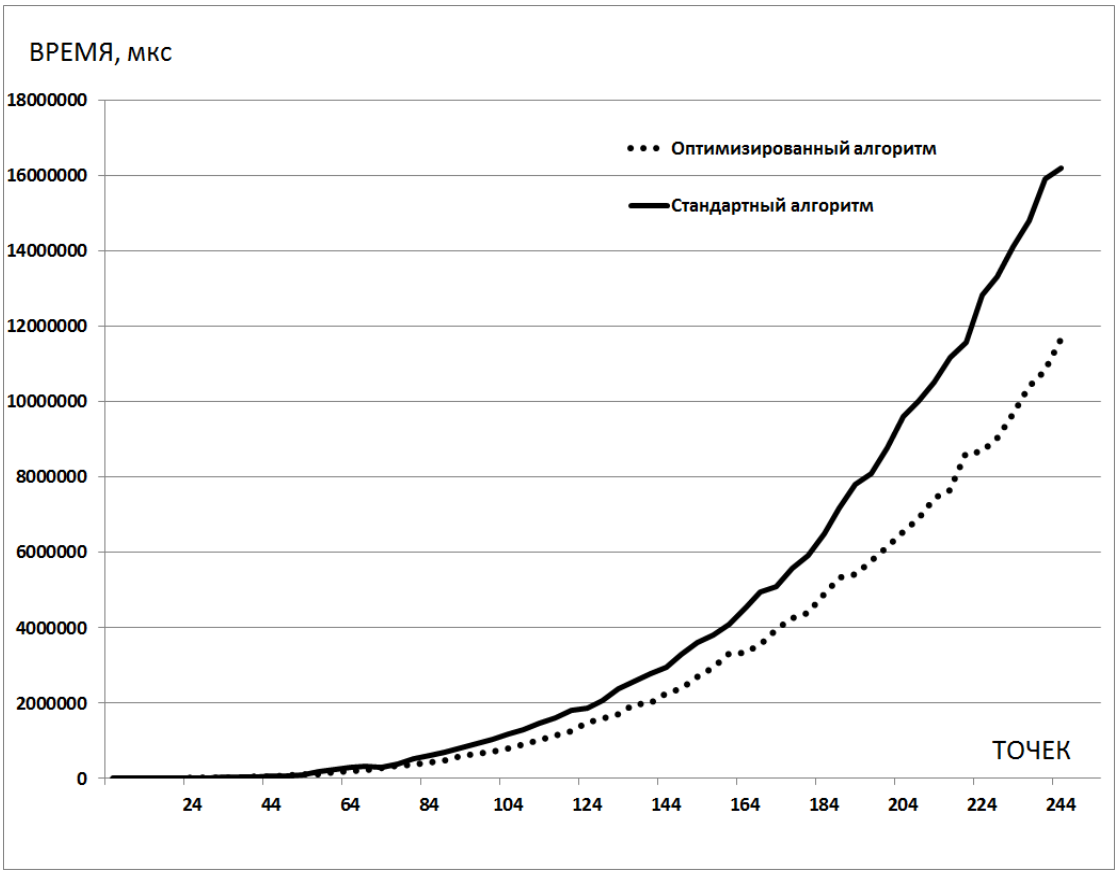

Рис. 1. *Сравнение оптимизированного и неоптимизированного алгоритмов диагонализации матрицы методом вращений Якоби.*

Далее было проведено сравнение скорости работы оптимизированного и неоптимизированного алгоритмов диагонализации методом вращений Якоби в зависимости от количества итераций метода и размерности матрицы, получаемой из траекторной. Заметим, что количество итераций задается изначально и является одним из условий остановки метода – если необходимая точность не достигается за заданное количество итераций, вычисления прекращаются. На рисунке 2 представлена разница времени работы оптимизированного и неоптимизированного алгоритмов.

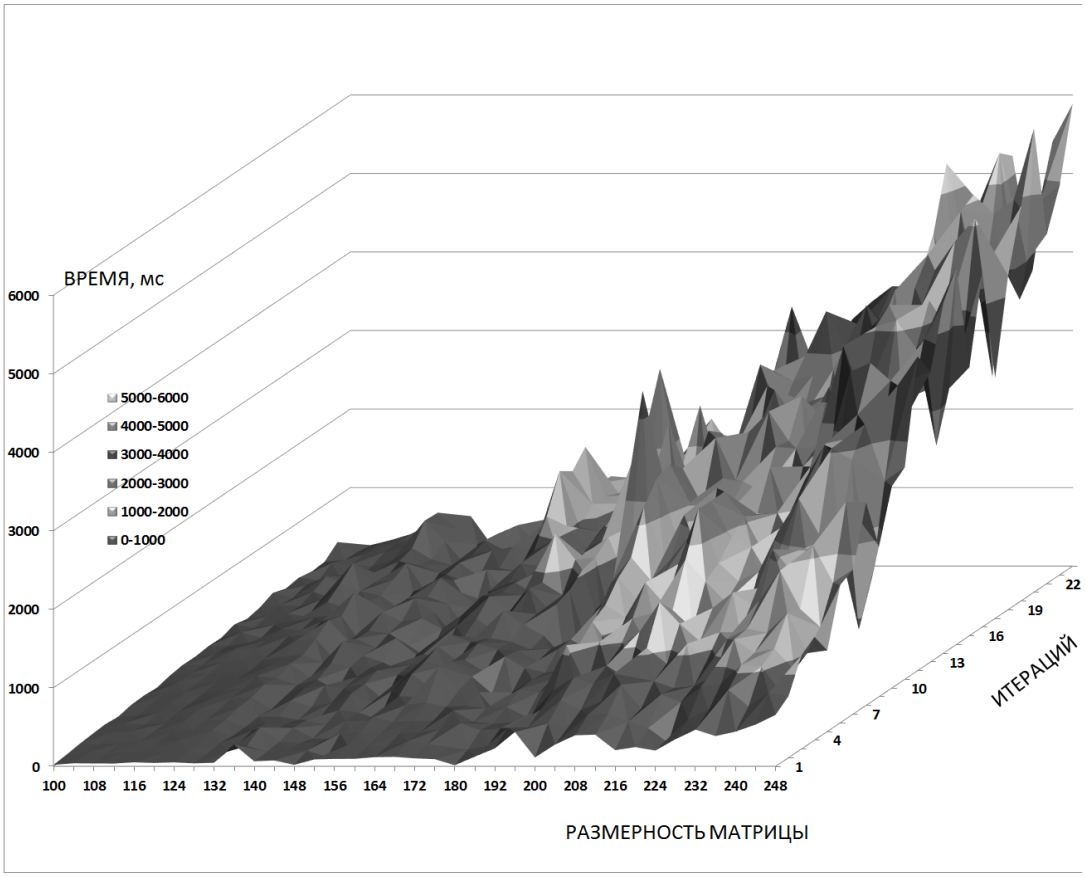

***Рис. 2**. Разница времени работы оптимизированного и неоптимизированного алгоритмов в зависимости от количества итераций и размерности матрицы.*

Помимо алгоритма Якоби, в методе SSA неоднократно производится суммирование элементов массивов различной длины (в частности, при преобразовании траекторной матрицы и при выполнении ганкелизации), что также нельзя оставлять без внимания. Благодаря применению оптимизации работы с регистрами процессора, производительность процедур, выполняющих данные операции, удалось увеличить до двух раз, что также положительно сказалось на скорости работы алгоритма SSA.

В частности, для ускорения суммирования массивов большой длины можно использовать SIMD (Single instruction, Multiple data), а также векторные регистры. В ARM процессорах SIMD расширение реализовано в виде дополнительного набора инструкций и называется ARM NEON. Рассмотрим пример, в котором происходит суммирование элементов массива длины n. Стоит отметить, что значение переменной n должно быть кратно четырем.

```
1    double sum = 0;
2    for (i = 0; i < n; i += 1)
3    {
4    sum += someArray[i];
5    }
```

Данный программный код можно переписать в ином виде:

```
1    double sum1 = 0;
2    double sum2 = 0;
```

```
3       double sum3 = 0;
4       double sum4 = 0;
5       for (i = 0; i < n; i += 4)
6       {
7       sum1 += someArray [i];
8       sum2 += someArray [i+1];
9       sum3 += someArray [i+2];
10      sum4 += someArray [i+3];
11            }
12      sum1 += sum2;
13      sum3 += sum4;
14      sum1 += sum3;
```

В реализованном примере переменные, суммируемые внутри цикла будут складываться в векторный регистр, содержащий в себе 4 32-битных значения. Далее временные регистры выполняют операции суммирования, используя SIMD инструкции, что дает ощутимый прирост в производительности.

Для объяснения результата рассмотрим более подробно описание и принцип работы ARM NEON. SIMD – это название процесса параллельного оперирования множественными данными в рамках одной инструкции процессора. В расширении NEON данные могут быть сложены в длинные регистры размером 64 или 128 бит. Эти регистры также называются векторными и могут содержать в себе 8, 16, 32 или 64 – битные элементы. Более наглядно это представлено на рисунке 3:

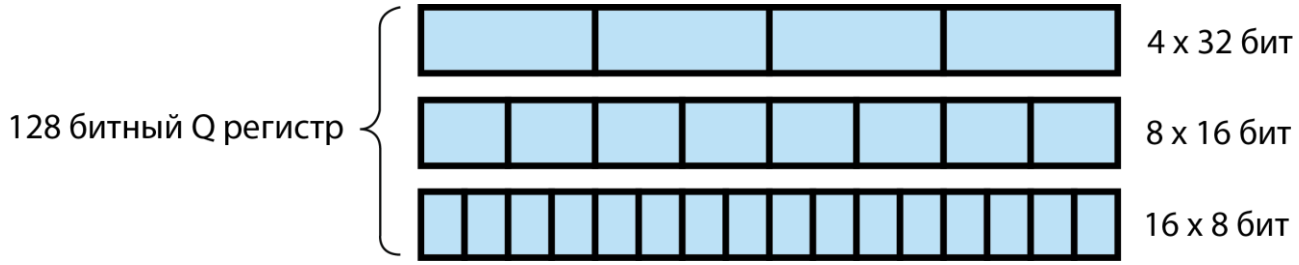

Рис. 3. *128 – битные векторные регистры*

NEON представляет программисту два набора регистров: 64 – битные D регистры и 128 – битные Q регистры. В действительности же это деление является условным и используется для более рационального размещения данных в памяти регистров, так как и D, и Q регистры физически ссылаются на одни и те же области физической памяти. Для того, чтобы более оптимально оперировать данными размером менее 64 бит, рациональнее использовать D – регистры. В иных случаях выгоднее будет использовать Q регистры. Структура адресации представлена на рисунке 4:

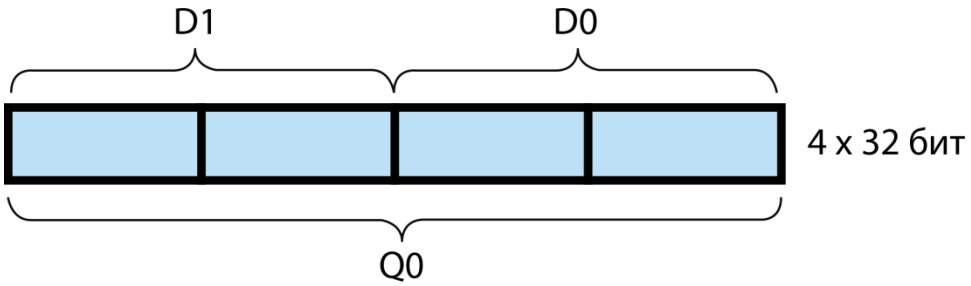

***Рис. 4**. Q и D регистры ARM процессоров*

Непосредственно перед выполнением операций над данными процессор сам определяет расположение данных в регистрах. Например, требуется посчитать сумму восьми 16-битных чисел. Первые 4 из них будут положены в некоторый 64 – битный векторный регистр D2, остальные 4 укомплектуются в 64 – битный регистр D1. Результат сложения будет также находиться в 64-битном векторном регистре, например, D0. Схематично это представлено на рисунке 5:

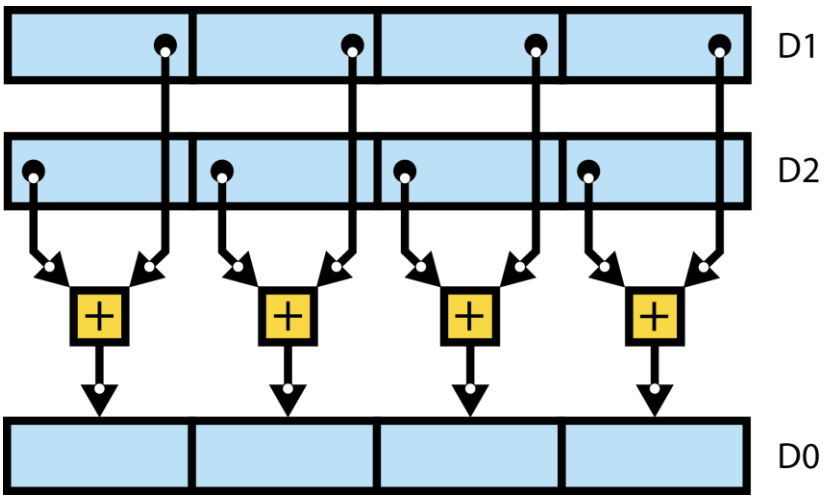

***Рис. 5**. 128 – схематичный пример сложения набора 16-битных чисел с использованием векторных регистров.*

Аналогично, с помощью векторных регистров, оптимизируются операции умножения. Таким образом нетрудно заметить, что с помощью технологии NEON в разы ускоряются операции сложения и умножения, что влечет за собой существенное уменьшение времени работы выполнения вычислительных алгоритмов. Сравнительные результаты времени работы представлены на рисунке 6.

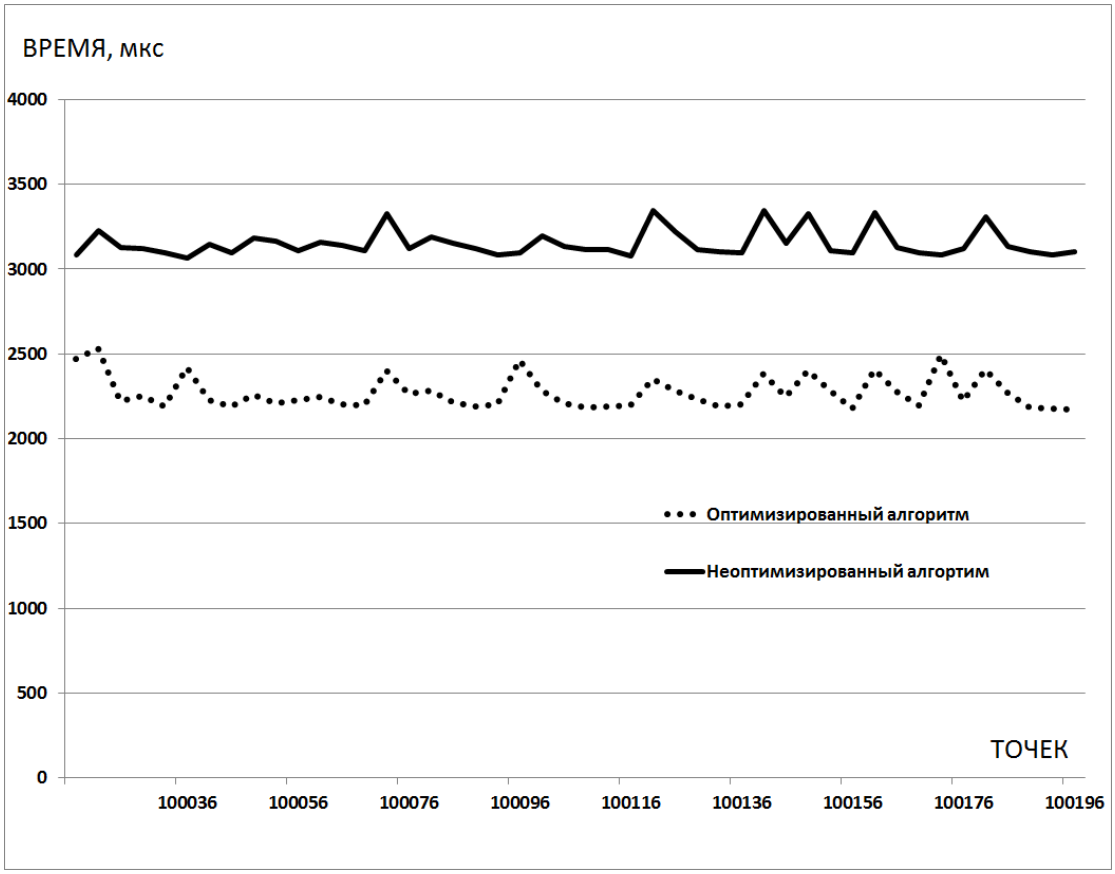

Рис. 6. *Сравнение времени работы оптимизированного и неоптимизированного алгоритмов при выполнении операций суммирований больших массивов*

Представленные результаты показывают, что оптимизация вычислительных алгоритмов позволила достичь заметного прироста производительности при работе на смартфонах и планшетных компьютерах. В реальных прикладных задачах это позволяет увеличивать длину гусеницы и сокращать время, требуемое на вычисления. Таким образом, становится возможным решать более широкий круг задач.

Помимо этого, устойчивая модификация алгоритма Якоби, оптимизированная для использования в приложениях для мобильных устройств, может быть применена в решении широкого круга других задач, требующих применения алгоритмов нахождения собственных значений и собственных векторов матриц.

ЛИТЕРАТУРА

1. Данилов Д. Л., Жиглявский А. А. Главные компоненты временных рядов: метод "Гусеница". Под редакцией Санкт-Петербургский университет, 1997.
2. Golyandina N., Nekrutkin V., Zhigljavsky A. Analysis of Time Series Structure: SSA and Related Techniques, CHAPMAN & HALL/CRC, 2001.
3. Knaster S., Malik W., Dalrymple M. Learn Objective-C on the Mac For OS X and iOS 2nd Edition. Apress, 2012. ISBN: 978-1-4302-4188-1.
4. Golyandina N., Zhigljavsky A. Singular Spectrum Analysis for Time Series, Springer, 2013.
5. Уилкинсон Дж. Х., Райнш С. Справочник алгоритмов на языке Алгол. Линейная алгебра. М.: Машиностроение, 1976. 390 с.
6. Хохлов А. А. Устойчивый метод восстановления тензора диэлектрической проницаемости по спектрофотометрическим данным. Письма в ЭЧАЯ. 2011, Т.8 №5 (168). С. 828-832.
7. ARM NEON support in the ARM compiler. September 2008. http://www.arm.com/files/pdf/neon_support_in_the_arm_compiler.pdf

Рецензент: Ловецкий Константин Петрович, кандидат физико-математических наук, доцент кафедры систем телекоммуникаций РУДН.

Andrey Savin
«Mahuru», Ltd
Russia, Moscow
E-Mail: assavin90@gmail.com

Alexey Khokhlov
Peoples' Friendship University of Russia (PFUR)
Russia, Moscow
E-Mail: khokhlov_aa@pfur.ru

SSA algorithm optimization for ARM mobile processors

Abstract: The present research dwells upon the optimization of the computing algorithms used in widespread methods of time series analysis.

The "Singular Spectrum Analysis" method is considered as an example. This method is also known as the "Caterpillar" method, realized and optimized for the processors used in smartphones and tablets that are popular today. The recommendations of the producer of processors were taken into account for analysis – also there were used vector registers, there were made the evolvement of the home loops in sets of consecutive operations, and also the optimal set of compilation markers/flags is picked up. All of this allowed to considerably increase the productivity of the used algorithms - one of which isthe wide-spread Jacobi eigenvalue method. This method is used for search of eigenvalues and eigenvectors of a real symmetric matrix. This method is the most resource-intensive in the Caterpillar method. The authors of article managed to increase the work speed of this method to 20 percent depending on the size of a window of the studied time series, the demanded accuracy of calculations and the other parameters depending on the conditions of a specific task. In the research there are given the description and results of computing experiment, within which the algorithm of the Caterpillar method was realized in the Objective C computer language. There is also presented the comparative work results of the optimized and non-optimized algorithms.

Keywords: SSA; Caterpillar; optimization for ARM; Jacobi eigenvalue algorithm; matrix diagonalization; Objective C; time series.

Identification number of article 110TVN214

REFERENCES

1. Danilov D. L., Zhigljavskij A. A. Glavnye komponenty vremennyh rjadov: metod "Gusenica". Pod redakciej Sankt-Peterburgskij universitet, 1997.
2. Golyandina N., Nekrutkin V., Zhigljavsky A. Analysis of Time Series Structure: SSA and Related Techniques, CHAPMAN & HALL/CRC, 2001.
3. Knaster S., Malik W., Dalrymple M. Learn Objective-C on the Mac For OS X and iOS 2nd Edition. Apress, 2012. ISBN: 978-1-4302-4188-1.
4. Golyandina N., Zhigljavsky A. Singular Spectrum Analysis for Time Series, Springer, 2013.
5. Uilkinson Dzh. H., Rajnsh S. Spravochnik algoritmov na jazyke Algol. Linejnaja algebra. M.: Mashinostroenie, 1976. 390 s.
6. Hohlov A. A. Ustojchivyj metod vosstanovlenija tenzora dijelektricheskoj pronicaemosti po spektrofotometricheskim dannym. Pis'ma v JeChAJa. 2011, T.8 №5 (168). S. 828-832.
7. ARM NEON support in the ARM compiler. September 2008. http://www.arm.com/files/pdf/neon_support_in_the_arm_compiler.pdf