Вестник Сыктывкарского университета.
Серия 1: Математика. Механика. Информатика. 2023.
Выпуск 1 (46)
Bulletin of Syktyvkar University.
Series 1: Mathematics. Mechanics. Informatics. 2023; 1 (46)

Научная статья

УДК 517.9, 539.3
https://doi.org/10.34130/1992-2752_2023_1_30

УСКОРЕНИЕ КРИПТОГРАФИЧЕСКИХ ВЫЧИСЛЕНИЙ ПУТЕМ НИЗКОУРОВНЕВОЙ ОПТИМИЗАЦИИ БАЗОВЫХ БЛОКОВ

Юрий Валентинович Гольчевский[1], **Дмитрий Александрович Ушаков**[2]

[1,2]Сыктывкарский государственный университет им. Питирима Сорокина, yurygol@mail.ru

Аннотация. В данной работе представлено исследование проблемы оптимизации программного кода при реализации алгоритмов шифрования. Выделены базовые блоки криптографического алгоритма на примере алгоритма «Кузнечик». Реализованные варианты алгоритма с использованием различных версий векторных инструкций и их комбинаций протестированы на процессорах различных микроархитектур. Некоторые разработанные варианты реализации алгоритма показывают большую скорость шифрования, чем существующие программные продукты.

Ключевые слова: криптографические вычисления, низкоуровневая оптимизация, базовые блоки, алгоритм «Кузнечик»

Для цитирования: Гольчевский Ю. В., Ушаков Д. А. Ускорение криптографических вычислений путем низкоуровневой оптимизации базовых блоков // *Вестник Сыктывкарского университета. Сер. 1: Математика. Механика. Информатика.* 2023. Вып. 1 (46). С. 30–49. https://doi.org/10.34130/1992-2752_2023_1_30

Article

Cryptographic Calculations Acceleration by Low-Level Optimization of Basic Blocks

Yuriy V. Golchevskiy[1], **Dmitriy A. Ushakov**[2]
[1,2]Pitirim Sorokin Syktyvkar State University, yurygol@mail.ru

Abstract. The paper presents a study of optimizing the program code problem when implementing encryption algorithms. The basic blocks of the cryptographic algorithm are highlighted on the example of the Kuznechik algorithm. Implemented variants of the algorithm using different versions of vector instructions and their combinations have been tested on processors of various microarchitectures. Some developed algorithm implementation variants show a higher encryption speed than existing software products.

Keywords: cryptographic computing, low-level optimization, basic blocks, algorithm Kuznechik

For citation: Golchevskiy Yu. V., Ushakov D. A. Cryptographic Calculations Acceleration by Low-Level Optimization of Basic Blocks. *Vestnik Syktyvkarskogo universiteta. Seriya 1: Matematika. Mekhanika. Informatika* [Bulletin of Syktyvkar University, Series 1: Mathematics. Mechanics. Informatics], 2023, no 1 (46), pp. 30−49. https://doi.org/10.34130/1992-2752_2023_1_30

Введение

В настоящее время оптимизация программного обеспечения (ПО) важна как для программ общего назначения (веб-браузеры, офисные пакеты), так и для специализированных программ (монтаж видео, моделирование различных процессов). К последним можно также отнести системы криптографической защиты информации (СКЗИ). Они в силу своей специфики (например, многократное применение блоков преобразований над открытым текстом) нуждаются в оптимизации участков кода, называемых базовыми блоками.

В качестве примера работ, представляющих результаты изучения проблемы оптимизации криптографических вычислений с различных точек зрения, можно привести работы [1–7]. В [1; 2] рассматривается комплексный подход к ускорению криптографических вычислений

и метод оптимизации целочисленного деления посредством ассемблерных вставок. В [3–5] описаны способы оптимизации блочных алгоритмов шифрования, включая ГОСТ 28147-89 («Магма») с использованием технологии SSE. В работе [6] представлен подход к оптимизации программных решений. Ускорению ГОСТ 34.10-2015 («Кузнечик») посвящена работа [7], где предлагается усовершенствованный алгоритм умножения вектора на столбец матрицы без применения низкоуровневых инструкций.

Сказанное выше обусловливает некоторые задачи данного исследования:

- изучить основные направления оптимизации программного кода при реализации алгоритмов шифрования и выделить базовые блоки криптографического алгоритма на примере исследуемого ГОСТ Р 34.12-2015;
- исследовать код, содержащий intrinsic функции и сгенерированный компиляторами с разными уровнями оптимизации;
- реализовать оптимизированные версии алгоритма с использованием различных версий векторных инструкций и их комбинаций и протестировать их на процессорах различных микроархитектур, сравнить полученные результаты и сделать выводы о возможности и эффективности низкоуровневой оптимизации на основе численной оценки.

Способы оптимизации программ

Существует несколько направлений оптимизации программного кода: алгоритмическая, оптимизация потребления аппаратных ресурсов и низкоуровневая. Стоит подчеркнуть, что оптимизация не всегда возможна. Например, для программ, реализующих графический пользовательский интерфейс, скорость выполнения может оказаться низкой из-за недостаточной производительности графической подсистемы компьютера.

Использование алгоритмической оптимизации в большинстве случаев повышает производительность прикладных программ. Но использование этого метода не всегда оправдано для системного и специализированного ПО. Для него могут потребоваться низкоуровневые оптимизации путем ассемблерных вставок или полностью ассемблерной реализации «горячих точек» программы.

Повышение эффективности использования ресурсов процессора и памяти дает возможность возложить часть работы по увеличению быстродействия программы на компилятор с незначительными изменениями исходного кода приложения.

Основными препятствиями на пути повышения быстродействия программ являются вызовы функций, циклы, условные переходы и зависимости по данным [8–10]. Снижение производительности при использовании циклов обусловлено накладными расходами на инициализацию цикла, проверку условия выполнения, совершение условных переходов. Одним из приемов оптимизации циклов является их раскрутка – за одну итерацию совершается больше полезной работы и уменьшается доля накладных расходов.

Проблема условных переходов или ветвлений может решаться с помощью процесса предвыборки. В современных процессорах существует блок предсказания переходов, который может повысить точность предсказания переходов при достаточном количестве совершенных переходов.

Проблему зависимости данных (если для начала работы над одними данными нужны предварительные вычисления над другими данными, и процессор должен дождаться завершения работы над ними) можно решать путем использования нескольких временных переменных, операции над которыми могут быть исполнены параллельно.

Также существует проблема оптимизации программ по использованию памяти, тесно связанная с эффективным использованием ресурсов процессора. Это обусловлено тем, что обращение к памяти разных типов (регистры, кэш, оперативная память) занимает определенное количество тактов процессора [11]. Для оптимизации программы целесообразно отказаться от частого использования указателей, а значит, и обращений к медленной оперативной памяти, в пользу таких концепций, как локальность и «дружелюбный к кэшу» (cache-friendly) код [8].

Низкоуровневую оптимизацию следует производить после изучения кода, сгенерированного компилятором. Тогда появляется возможность использовать регистры процессора более эффективно, так как компиляторы не всегда используют все доступные регистры. Примером этого метода может служить использование векторных регистров XMM, YMM, ZMM для выполнения операций одновременно над несколькими наборами чисел, хранящихся в данных регистрах [12; 13].

Анализ кода, сгенерированного компиляторами

В ГОСТ Р 34.12-2015 определены три этапа преобразований входного вектора [14]:

1. Побитовое сложение 128-битного входного вектора с итерационным ключом с помощью операции исключающее «ИЛИ».

2. Нелинейное преобразование, заключающееся в применении к каждому 8-битному подвектору входного вектора фиксированной подстановки.

3. Линейное преобразование, сводящееся к операциям умножения вектора-строки промежуточного текста на коэффициенты в поле Галуа.

Кроме алгоритмических оптимизаций для ускорения производительности можно учитывать особенности ассемблера и планировщика, который может поставить инструкции на параллельное исполнение на разных исполняющих устройствах [15; 16].

Предлагается реализация алгоритмической оптимизации алгоритма, основанная на использовании набора инструкций SSE2 и XMM регистров. Данный выбор объясняется тем, что регистры XMM могут вместить блок данных объемом 128 бит целиком, что равно принятой длине входного блока ГОСТ Р 34.12-2015. Кроме того, набор инструкций SSE2 доступен на всех современных процессорах [17; 18].

Для того чтобы сделать выводы о целесообразности приведенного метода оптимизации, необходимо определить реализации шифра «Кузнечик», производительность которых будет оцениваться:

- версия на языке C без использования intrinsic функций (далее src-версия);

- версия на языке C с использованием intrinsic функций, приведенная в статье [19] (https://github.com/aprelev/libgost15) (далее itr-версия).

Тестируемые версии компилировались при помощи GNU Compiler Collection 4.9.2 (GCC) и Intel C++ Compiler из набора Intel Parallel Studio XE 2016 (ICL). При компиляции обеих версий высокоуровневой реализации использовались следующие опции компиляции GCC [20]:

- -shared -c -m32 (-m64) -mmmx -msse -msse2 -Wall -std=c99 для всех версий;
- -Os для оптимизации программы по размеру;
- -O3 для оптимизации программы по скорости выполнения.

По результатам проведенного анализа были сделаны следующие выводы:

- версии, оптимизированные по размеру кода (ключ -Os), показывают неудовлетворительную производительность, поэтому были исключены из дальнейшего тестирования;
- src-версии используют только регистры общего назначения (РОН), т. е. компилятор не смог распознать возможность оптимизации и использовать более эффективные технологии (MMX, SSE, AVX);
- в 32-битной itr-версии компилятор произвел оптимизацию, что позволило сэкономить 1 регистр;
- в 64-битной itr-версии компилятор использовал все доступные РОН для сохранения указателей на данные и раундовые ключи шифрования.

Для дальнейших исследований было решено оставить 64-битную itr-версию, так как ее код был сгенерирован наиболее оптимальным образом.

При компиляции обеих версий высокоуровневой реализации использовались следующие опции компиляции ICL в различных комбинациях [21]:

- /QaxSSE2,SSE3,SSSE3,SSE4.1,SSE4.2,AVX,CORE-AVX-I,CORE-AVX2;
- /Qipo для включения межпроцедурной оптимизации;
- /O3 для оптимизации программы по скорости выполнения;
- /Od для отключения оптимизации.

По итогам проведенного анализа сгенерированного кода было решено оставить 32-битную и 64-битную src-версии. Это объясняется тем, что в 32-битной версии компилятор распознал возможность оптимизации и использовал младшие 64 бита XMM регистров для вычислений. В 64-битной версии компилятор не использовал XMM регистры, а производил вычисления на 64-битных РОН.

Ручная оптимизация (ассемблерная реализация)

Согласно [22], низкоуровневую оптимизацию необходимо начинать после анализа листингов, сгенерированных компиляторами, поэтому в тестирование были включены высокоуровневые версии на языке C. Современные компиляторы могут генерировать достаточно оптимальный код с точки зрения производительности, используя возможности векторных расширений, даже если это явно не указано в коде. Однако возможности для оптимизации остаются. С другой стороны, прирост быстродействия от техник «ручной» низкоуровневой оптимизации должен составлять не менее 20 %. Если прирост производительности меньше этого порогового значения, то усилия, потраченные на оптимизацию, не имеют смысла согласно [23]. Это утверждение позволяет сделать вывод об успешности примененных методов.

Для ассемблерных реализаций выбор компилятора не имеет значения, так как весь код оформлен в виде объектного файла, созданного FASM, содержимое которого переносится компилятором в исполняемый файл без изменений. Компиляция таких версий проводилась без использования оптимизации [24].

Приведем несколько пояснений для дальнейшего изложения проблемы.

Ядром будет называться минимальная совокупность команд и используемых ими регистров, достаточная для осуществления табличной подстановки.

Используемые регистры можно разделить:

1) на регистры-накопители (далее – РН) – содержат результат одного раунда, который затем складывается с ключом;

2) регистры-маски (далее – РМ) – хранят маску (могут отсутствовать, если маска загружается из памяти);

3) регистры-экстракторы (далее – РЭ) – из них извлекаются байты для подстановки по таблице LSP-преобразований.

Всего было подготовлено несколько десятков ассемблерных реализаций алгоритма как для 32-битного, так и для 64-битного режима работы процессора. Наименования всех исполняемых файлов версий алгоритма, приведенных на графиках, будем строить по определенной схеме:

ex-архитектура-[компилятор][используемые наборы векторных расширений]-[ядро][модификаторы].exe,

которая для экономии места будет преобразована в следующую схему:

ex-архитектура-[g|i][r,m,s,s3,s4,a,a2]-
[Ln|Wn|Bn|Un|Mn][Pn,A,E,R,C,M,X,G,K[n],f,n,l,p,t].exe.

Компилятор:
— g – GCC, i – ICL.

Используемые наборы векторных расширений:
— r – регистры общего назначения;
— m – MMX;
— s – SSE или SSE2, s3 – SSSE3, s4 – SSSE4.x;
— a – AVX, a2 – AVX2.

Ядро (n – число задействованных регистров для одного раунда):
— L – из РЭ байты извлекаются командой pextrw;
— W – из РЭ байты извлекаются командой pmovzxbw;
— B – из РЭ байты извлекаются командой pextrb;
— U – из РЭ байты извлекаются командами punpcklbw/punpckhbw;
— M – РЭ отсутствуют, байты извлекаются прямо из специально выделенной области памяти;
— R – в качестве РЭ используются регистры общего назначения;
— F – из РЭ байты извлекаются командой pshufb.

Модификаторы:
— K – раундовый ключ «кеширован» в XMM регистрах;
— E – раундовый цикл полностью развернут;
— Pn – за одну итерацию раундового цикла обрабатываются n блоков;
— A – вместо умножения/сдвига на 16 используется сложение с самим собой и множитель 8 при обращении к памяти;
— C – константы-смещения для таблицы подстановки помещаются в РЭ вместо указания их при обращении к памяти;
— X – группировка блоков вычисления РЭ;

— G – группировка блоков PH;
— n – ближний порядок, f – дальний порядок, l – реассоциация регистров;
— t – использование свободных регистров для «кеширования» значений памяти.

Тестирование производительности

Итоговое тестирование проводилось путем измерения скорости шифрования сгенерированного случайным образом массива данных (усредненной по многим испытаниям).

Вычисление скорости шифрования производилось путем деления объема шифруемых данных на среднее время выполнения цикла шифрования.

Данные для оценки скорости шифрования генерируются случайным образом на основе алгоритма «Вихрь Мерсенна» (Mersenne Twister) и его свободно распространяемой реализации TinyMT (https://github.com/MersenneTwister-Lab/TinyMT).

На первом этапе проводилось шифрование полного объема тестовых данных размером 512 МБ, затем его половины и т. д. — до размера блока. Также следует учитывать, что с уменьшением размера шифруемых данных время шифрования уменьшается, поэтому количество испытаний должно увеличиваться пропорционально.

Схема проведения исследования представлена в работе [25].

Тестирование производительности реализованных версий производилось на процессорах различных микроархитектур (Haswell, Vishera, Sandy Bridge, Arrandale, Wolfdale).

Результаты тестирования

На рис. 1, 2 представлены значения скорости шифрования тестируемых версий для 32- и 64-битного режима работы процессора соответственно.

Близость кривых позволяет говорить о том, что особенности организации конвейера, присущие конкретной микроархитектуре, являются наиболее существенным фактором, определяющим эффективность тестируемых способов низкоуровневой оптимизации. Напротив, такие факторы, как различия в типе процессоров, рабочих частотах, размерах кэшей, не оказывают существенного влияния.

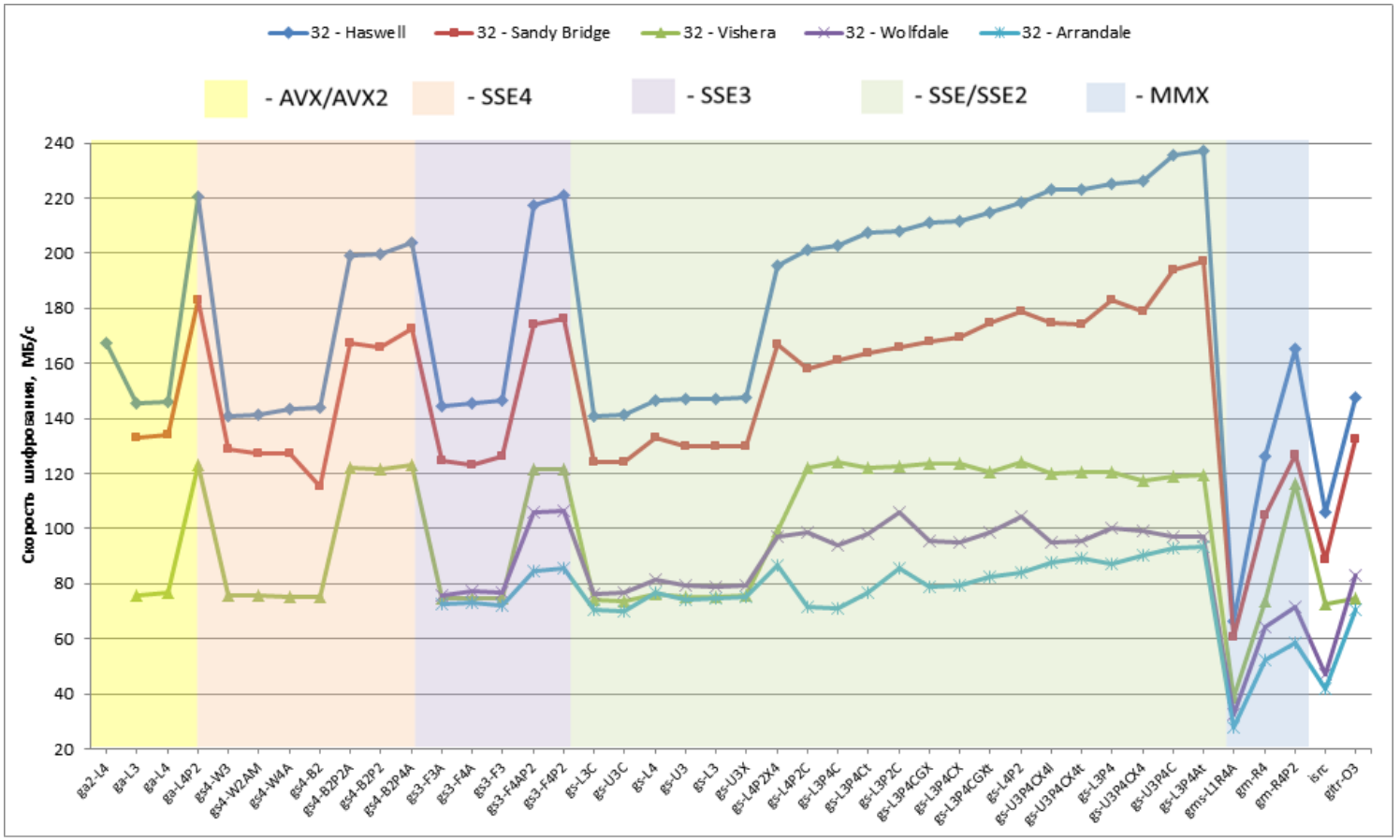

Рис. 1. Результаты тестирования для 32-битного режима работы процессора

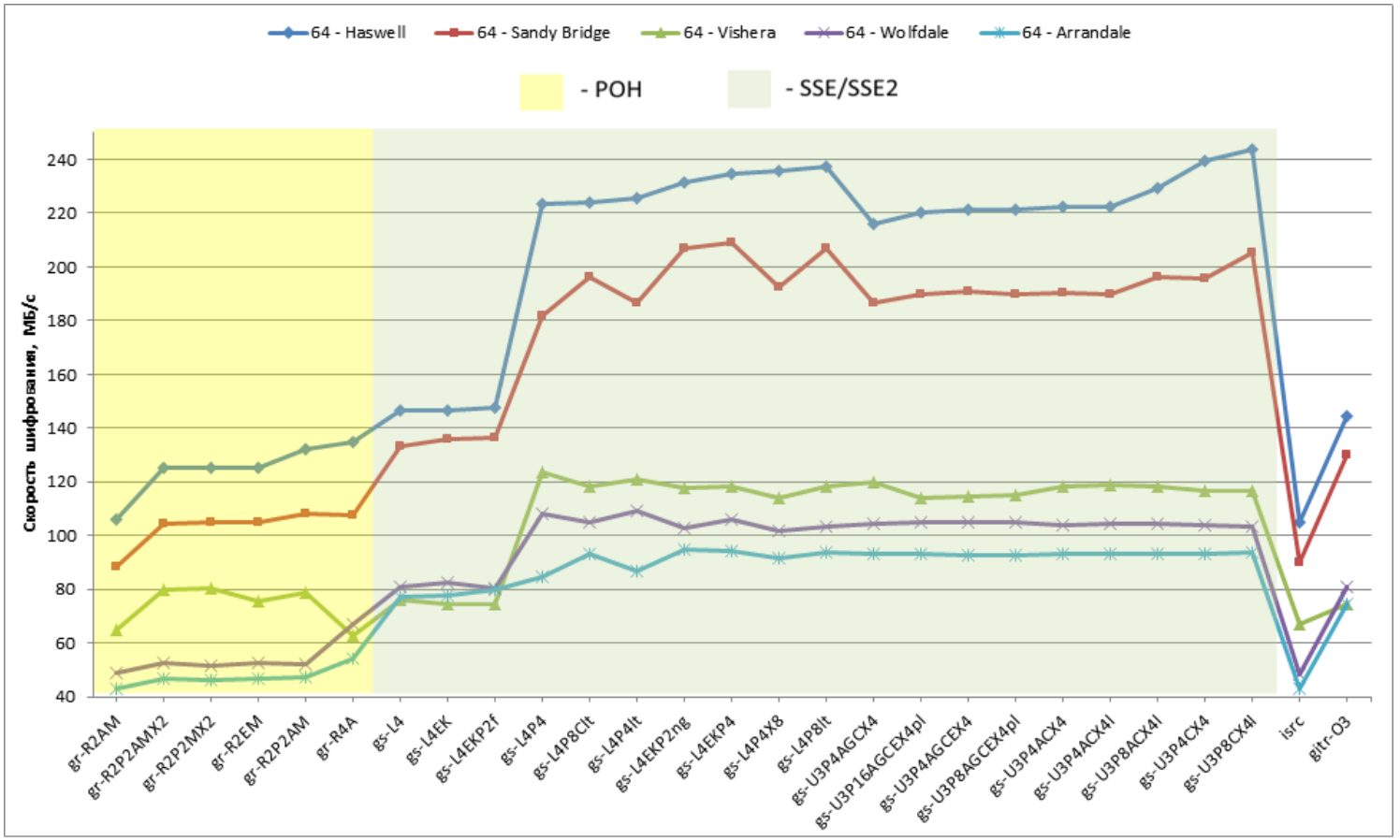

Рис. 2. Результаты тестирования для 64-битного режима работы процессора

Рассмотрим скорости шифрования для 32-битных версий, реализованных с использованием разных наборов векторных инструкций (рис. 1). Тестирование показывает, что производительность алгоритма шифрования «не пропорциональна» версии используемых векторных расширений, что является следствием ее принадлежности к классу «преимущественно конвейеризируемых» программ [26]. Решения SSE/SSE2 в целом наиболее оптимальны с точки зрения производительности. Использование MMX инструкций в ряде случаев приводит к снижению производительности в несколько раз по сравнению с наиболее оптимизированными реализациями.

Далее необходимо привести сравнение версий, использующих инструкцию pextrw для извлечения данных из РЭ и четыре регистра для обработки блока открытого текста, одна из которых сгенерирована компилятором, а другая реализована на ассемблере. Версия girt-O3 (сгенерированная компилятором) дает прирост скорости шифрования для микроархитектур Haswell и Wolfdale и ее падение до 6 МБ/с для остальных микроархитектур по сравнению с версией gs-L4 (ассемблерная реализация).

Для 64-битного режима работы процессора (рис. 2) результаты аналогичны представленным ранее, за исключением версий gr-R4A, gs-L4, gs-L4EK и gs-L4EKP2f для микроархитектуры Vishera. Такое поведение может быть объяснено особенностями микроархитектуры Vishera (более низкая ассоциативность кеша данных L1 и его меньший объем по сравнению с процессорами Intel).

Для процессоров Intel прирост производительности составляет в среднем 2,25 раза. Для процессоров микроархитектуры Vishera прирост производительности ниже как для 32-битного, так и для 64-битного режима работы процессора.

Анализ зависимости скорости шифрования от частоты процессора, не углубляясь в особенности микроархитектуры, которые обусловливают различия производительности тестируемых версий, показал, что в данном эксперименте процессор AMD уступал по характеристикам быстродействия сопоставимым процессорам Intel, несмотря на более высокую тактовую частоту. Возможно, одна из основных причин заключается в более низкой ассоциативности кеша данных L1 процессора и его меньшем объеме, что приводит к снижению эффективности кеширования, поскольку одни и те же кеш-линейки вынуждены попеременно замещать свое содержимое данными из разных участков памяти.

Выводы

Проведенный анализ указанных работ и данных работ [27; 28] позволяет сказать, что некоторые разработанные варианты реализации алгоритма шифрования «Кузнечик» показывают большую скорость шифрования, являясь как минимум не менее эффективными, чем существующие программные продукты. Оптимизации, основанные на использовании различных версий векторных расширений, позволяют добиться значительного (от 1,7 до 3,2 раз) улучшения производительности шифрования ассемблерных реализаций над src-реализациями, это означает, что усилия, потраченные на низкоуровневую оптимизацию, согласно [23], имеют смысл.

Список источников

1. **Северин П. А., Гольчевский Ю. В.** Комплексный подход к ускорению криптографических вычислений // *Информационные технологии в управлении и экономике.* 2012. № 2. С. 36–39.

2. **Гольчевский Ю. В., Северин П. А.** Об оптимизации криптографических алгоритмов посредством ассемблерных вставок при целочисленном делении // *Известия ТулГУ. Технические науки.* 2013. Вып. 3. С. 295–301.

3. Скоростное поточное шифрование // SecurityLab.ru [Электронный ресурс]. URL: http://www.securitylab.ru/analytics/436620.php (дата обращения: 03.12.2022).

4. **Павлов В. Э., Удальцов В. А.** Оптимизация скорости работы блочных алгоритмов шифрования // *Приоритетные направления развития образования и науки : материалы междунар. науч.-практ. конф. (Чебоксары, 9 апр. 2017 г.).* Чебоксары: ЦНС «Интерактив плюс», 2017. Т. 2. С. 76–80. doi: 10.21661/r-129822.

5. Особенности национальной криптографии // SecurityLab.ru [Электронный ресурс]. URL: http://www.securitylab.ru/analytics/480357.php (дата обращения: 03.12.2022).

6. **Гашин Р. А., Гольчевский Ю. В.** Разработка криптобиблиотеки на базе алгоритмов проекта eSTREAM // *Безопасность информационных технологий.* 2015. № 4 (22). С. 52–57.

7. **Ищукова Е. А., Кошуцкий Р. А., Бабенко Л. К.** Разработка и реализация высокоскоростного шифрования данных с использованием алгоритма «Кузнечик» // *Auditorium.* 2015. № 4 (8).

8. Оптимизация кода: процессор // Хабрахабр [Электронный ресурс]. URL: https://habrahabr.ru/post/309796/ (дата обращения: 03.12.2022).

9. **Bryant R., Hallaron D.** Computer Systems A Programmer's Perspective. Printice Hall, 2015. 1120 p.

10. **Гербер Р., Бик А., Смит К., Тиан К.** Оптимизация ПО : сборник рецептов. СПб.: Питер, 2010. 352 с.

11. **Кунин Р.** Оптимизация кода: память [Электронный ресурс]. URL: https://habrahabr.ru/post/312078/ (дата обращения: 03.12.2022).

12. Intel Instruction Set Extensions Technology [Электронный ресурс]. URL: https://www.intel.com/content/www/us/en/support/articles/000005779/processors.html (дата обращения: 07.12.2022).

13. **Киреев С. Е., Калгин К. В.** Эффективное программирование современных микропроцессоров и мультипроцессоров [Электронный ресурс]. URL: https://ssd.sscc.ru/sites/default/files/content/attach/317/lecture2016_01_intro.pdf (дата обращения: 03.12.2022).

14. ГОСТ Р 34.12-2015 Информационная технология. Криптографическая защита информации. Блочные шифры.

15. Intel 64 and IA-32 Architectures Software Developer Manuals [Электронный ресурс]. URL: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html (дата обращения: 07.12.2022).

16. Intel 64 and IA-32 Architectures Optimization Reference Manual [Электронный ресурс]. URL: https://cdrdv2-public.intel.com/671488/248966-046A-software-optimization-manual.pdf (дата обращения: 07.02.2023).

17. Intel SSE4 Programming Reference [Электронный ресурс]. URL: https://www.intel.com/content/dam/develop/external/us/en/documents/d9156103-705230.pdf (дата обращения: 07.02.2023).

18. Intel Architecture Instruction Set Extensions Programming Reference [Электронный ресурс]. URL: https://cdrdv2-public.intel.com/671368/architecture-instruction-set-extensions-programming-reference.pdf (дата обращения: 07.02.2023).

19. ГОСТ Р 34.12 '15 на SSE2, или Не так уж и плох Кузнечик // Хабрахабр [Электронный ресурс]. URL: https://habrahabr.ru/post/312224/ (дата обращения: 03.12.2022).

20. Using the GNU Compiler Collection // GCC Online Documentation [Электронный ресурс]. URL: https://gcc.gnu.org/onlinedocs/gcc.pdf (дата обращения: 03.12.2022).

21. Intel C++ Compiler 19.1 Developer Guide and Reference // Intel Developer Zone [Электронный ресурс]. URL: https://www.intel.com/content/www/us/en/develop/documentation/cpp-compiler-developer-guide-and-reference/top.html (дата обращения: 07.12.2022).

22. **Fog A.** Optimizing Subroutines In Assembly Language: An Optimization Guide For x86 Platforms [Электронный ресурс]. URL: http://www.agner.org/optimize/optimizing_assembly.pdf (дата обращения: 03.12.2022).

23. **Касперски К.** Техника оптимизации программ. Эффективное использование памяти. СПб.: БХВ-Петербург, 2003. 464 с.

24. flat assembler 1.73 Programmer's Manual | flat assembler [Электронный ресурс]. URL: http://flatassembler.net/docs.php?article=manual (дата обращения: 03.12.2022).

25. **Гольчевский Ю. В.** Автоматизация исследования оптимизации скорости шифрования информации // *Информационные технологии в моделировании и управлении: подходы, методы, решения : сборник научных статей V Всероссийской научной конференции с международным участием.* Тольятти: Изд-во ТГУ, 2022. С. 208–215.

26. **Северин П. А., Гольчевский Ю. В.** Низкоуровневая оптимизация производительности на примере функции хеширования по ГОСТ Р 34.11-2012 // *Системный администратор.* 2017. № 1–2. С. 170–171.

27. **Ahmetzyanova L. R., Alekseev E. K., Oshkin I. B. и др.** On the properties of the CTR encryption mode of the Magma and Kuznyechik block ciphers with re-keying method based on CryptoPro Key Meshing // *IACR Cryptol.* ePrint Arch., 2016, 628 p.

28. **Гафуров И. Р.** Высокоскоростная программная реализация алгоритмов шифрования из ГОСТ Р 34.12-2015 // *Ученые записки УлГУ. Серия: Математика и информационные технологии.* 2022. Вып. 2. С. 38–48.

References

1. **Severin P. A., Golchevskiy Yu. V.** Comprehensive Approach for Cryptographic Computation Acceleration. *Informatsionnyye tekhnologii v upravlenii i ekonomike* [Information technologies in management and economics]. 2012, no 2, pp. 36–39. (In Russ.)

2. **Golchevskiy Yu. V., Severin P. A.** Cryptographic Algorithms Optimization by Means of Assembly Inserts in Integer Division. *Izvestiya TulGU. Tekhnicheskiye nauki* [News of TulGU. Technical sciences]. 2013, vol. 3, pp. 295–301. (In Russ.)

3. Fast Stream Encryption. *SecurityLab.ru.* Available at: http://www.securitylab.ru/analytics/436620.php (accessed: 03.12.2022).

4. **Pavlov V. E., Udaltsov V. A.** Optimizing the performance of block encryption algorithms. *Prioritetnyye napravleniya razvitiya obrazovaniya i nauki* [Priority directions of science and education development]. Cheboksary: SCC "Interaktiv plus LLC, 2017, 2(2), pp. 76–80. DOI: 10.21661/r-129822. (In Russ.)

5. Features of national cryptography. *SecurityLab.ru.* Available at: http://www.securitylab.ru/analytics/480357.php (accessed: 03.12.2022). (In Russ.)

6. **Gashin R. A., Golchevskiy Yu. V.** Development of Cryptographic Library Based on Algorithms from eSTREAM Project. *Bezopasnost' informatsionnykh tekhnologiy* [Information Technology Security], 2015, no 4 (22), pp. 52–57. (In Russ.)

7. **Ischukova E. A., Koshutsky R. A., Babenko L. K.** Development and implementation of high-speed data encryption using the Kuznechik algorithm. *Auditorium*, 2015, no 4 (8). (In Russ.)

8. Code Optimization: CPU. *Habrahabr.* Available at: https://habrahabr.ru/post/309796/ (accessed: 03.12.2022). (In Russ.)

9. **Bryant R., Hallaron D.** *Computer Systems A Programmer's Perspective.* Printice Hall, 2015. 1120 p.

10. **Gerber R., Bik A. J. C., Smith K., Tian X.** The Software Optimization Cookbook. St. Petersburg: Piter, 2010. 352 p. (In Russ.)

11. **Kunin R.** *Optimizatsiya Koda: pamyat'* [Code Optimization: Memory]. Available at: https://habrahabr.ru/post/312078/ (accessed: 03.12.2022). (In Russ.)

12. *Intel Instruction Set Extensions Technology.* Available at: https://www.intel.com/content/www/us/en/support/articles/000005779/processors.html (accessed: 07.12.2022).

13. **Kireev S. E. Kalgin K. V.** *Effektivnoye programmirovaniye sovremennykh mikroprotsessorov i mul'tiprotsessorov* [Efficient programming of modern microprocessors and multiprocessors]. Available at: https://ssd.sscc.ru/sites/default/files/content/attach/317/lecture2016_01_intro.pdf (accessed: 03.12.2022). (In Russ.)

14. *GOST R 34.12-2015 Informatsionnaya tekhnologiya. Kriptograficheskaya zashchita informatsii. Blochnyye shifry* [GOST R 34.12-2015. Information technology. Cryptographic data security. Block ciphers]. (In Russ.)

15. *Intel 64 and IA-32 Architectures Software Developer Manuals.* Available at: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html (accessed: 07.12.2022).

16. *Intel 64 and IA-32 Architectures Optimization Reference Manual*. Available at: https://cdrdv2-public.intel.com/671488/248966-046A-software-optimization-manual.pdf (accessed: 07.02.2023).

17. *Intel SSE4 Programming Reference*. Available at: https://www.intel.com/content/dam/develop/external/us/en/documents/d9156103-705230.pdf (accessed: 07.02.2023).

18. *Intel Architecture Instruction Set Extensions Programming Reference*. Available at: https://cdrdv2-public.intel.com/671368/architecture-instruction-set-extensions-programming-reference.pdf (accessed: 07.02.2023).

19. *GOST R 34.12 '15 na SSE2, ili Ne tak uzh i plokh Kuznechik* [GOST R 34.12 '15 on SSE2, or Not So Bad Kuznechik]. *Habrahabr*. Available at: https://habrahabr.ru/post/312224/ (accessed: 03.12.2022). (In Russ.)

20. Using the GNU Compiler Collection. *GCC Online Documentation*. Available at: https://gcc.gnu.org/onlinedocs/gcc.pdf (accessed: 03.12.2022).

21. Intel C++ Compiler 19.1 Developer Guide and Reference. *Intel Developer Zone*. Available at: https://www.intel.com/content/www/us/en/develop/documentation/cpp-compiler-developer-guide-and-reference/top.html (accessed: 07.12.2022).

22. **Fog A.** *Optimizing Subroutines In Assembly Language: An Optimization Guide For x86 Platforms*. Available at: http://www.agner.org/optimize/optimizing_assembly.pdf (accessed: 03.12.2022).

23. **Kaspersky K.** *Tekhnika optimizatsii programm. Effektivnoye ispol'zovaniye pamyati* [Program optimization technique. Efficient Memory Usage]. St. Petersburg, BHV-Petersburg, 2003. 464 p. (In Russ.)

24. *flat assembler 1.73 Programmer's Manual | flat assembler*. Available at: http://flatassembler.net/docs.php?article=manual (accessed: 03.12.2022).

25. **Golchevskiy Yu. V.** Automation of Encryption Speed Optimization Research. *Informatsionnyye tekhnologii v modelirovanii i upravlenii: podkhody, metody, resheniya : sbornik nauchnykh statey V Vserossiyskoy nauchnoy konferentsii s mezhdunarodnym uchastiyem* [Information technologies in modeling and management: approaches, methods, solutions: Collection of scientific articles: V All-Russian scientific conference with international participation]. Tolyatti: Publishing House of TSU, 2022, pp. 208–215. (In Russ.)

26. **Severin P. A., Golchevskiy Yu. V.** Low-Level Performance Optimization on the Example of the Hash Function GOST R 34.11-2012. *Sistemnyy administrator* [System Administrator]. 2017, no 1–2, pp. 170–171. (In Russ.)

27. **Ahmetzyanova L. R., Alekseev E. K., Oshkin I. B. et al.** On the properties of the CTR encryption mode of the Magma and Kuznyechik block ciphers with re-keying method based on CryptoPro Key Meshing. *IACR Cryptol.* ePrint Arch., 2016, 628 p.

28. **Gafurov I. R.** High-speed software implementation of encryption algorithms from GOST R 34.12-2015. *Uchenyye zapiski UlGU. Seriya: Matematika i informatsionnyye tekhnologii* [Scientific notes of UlGU. Series: Mathematics and Information Technology], 2022, no 2, pp. 38–48. (In Russ.)

Сведения об авторах / Information about authors
Гольчевский Юрий Валентинович / Yuriy V. Golchevskiy
к.ф.-м.н, доцент, заведующий кафедрой прикладной информатики / Ph.D. in Physics and Mathematics, Associate Professor, Head of Applied Informatics Department
Сыктывкарский государственный университет им. Питирима Сорокина / Pitirim Sorokin Syktyvkar State University
167001, Россия, г. Сыктывкар, Октябрьский пр., 55 / 167001, Russia, Syktyvkar, Oktyabrsky Ave., 55

Ушаков Дмитрий Александрович / Dmitriy A. Ushakov
технический специалист по информационным системам / Information Systems Technician
Сыктывкарский государственный университет им. Питирима Сорокина / Pitirim Sorokin Syktyvkar State University

167001, Россия, г. Сыктывкар, Октябрьский пр., 55 / 167001, Russia, Syktyvkar, Oktyabrsky Ave., 55

Статья поступила в редакцию / The article was submitted 06.02.2023
Одобрено после рецензирования / Approved after reviewing 14.02.2023
Принято к публикации / Accepted for publication 20.02.2023