© 2012 г. П.А. ЛЕБЕДЕВ
(Московский институт электроники и математики
Национального исследовательского университета
«Высшая школа экономики»)

Сравнение старого и нового стандартов РФ на криптографическую хэш-функцию на ЦП и графических процессорах NVIDIA

В работе представлены пути оптимизации и соответствующие реализации семейства криптографических хэш-функций ГОСТ Р 34.11-94 и ГОСТ Р 34.11-2012. Приводятся результаты для процессоров архитектуры x86_64 и видеокарт NVIDIA с поддержкой технологии CUDA для авторской и других хорошо известных реализаций. Показано, что новая хэш-функция может быть вдвое быстрее чем старая на современных ЦП, но медленее на графических картах и системах с ограниченными ресурсами. Результаты авторских реализаций показывают наибольшую производительность среди всех протестированных на обеих платформах.

COMPARISON OF OLD AND NEW CRYPTOGRAPHIC HASH FUNCTION NATIONAL STANDARDS OF RUSSIAN FEDERATION ON CPUS AND NVIDIA GPUS / P.A. Lebedev (Moscow Institute of Electronics and Mathematics NRU HSE, B. Tryokhsvyatitel'skiy 3, Moscow 109028, Russia, E-mail: cygnus@michiru.ru). The following work presents optimization guidelines and implementations of cryptographic hash functions GOST R 34.11-94 and GOST R 34.11-2012. Results for x86_64 CPUs and NVIDIA CUDA-capable GPUs are provided for author's and several other well-known implementations. It is shown that the new standard can be twice as fast as the old one on modern CPUs, but it can be slower on embedded devices and GPUs. The results given for the author's implementations are the fastest among all the tested implementations on both platforms.

1. Введение

Стрибог — проектное название семейства криптографических хэш-функций, принятое в качестве замены национальному стандарту Российской Федерации ГОСТ Р 34.11-94 [1] в виде ГОСТ Р 34.11-2012 [2]. Для краткости в данной работе проектное название данной хэш-функции будет использоваться наравне с наименованием стандарта. Данное в стандарте описание, к сожалению, не может быть напрямую использовано для создания высокопроизводительной реализации. В этой работе будут рассмотрены необходимые для достижения максимальной производительности оптимизации на уровне алгоритма и конкретных платформ.

Исследование реализации данных хэш-функций с использованием графических процессоров (GPGPU) важно для проверки, насколько хорошо выбранные основные единицы данных и примитивные операции пригодны для реализации алгоритма на современных высокопараллельных архитектурах. Платформа NVIDIA CUDA [5] была выбрана для данной работы как наиболее зрелая и простая в использовании.

В данной работе будут рассмотрены детали реализации и результаты для хэш-функций Стрибог и ГОСТ Р 34.11-94 на графических картах NVIDIA с применением технологии CUDA, а также процессорах архитектуры x86_64 с использованием SIMD-инструкций. В результаты включено сравнение авторской и других известных реализаций (при их наличии).

2. Семейство хэш-функций Стрибог

Устройство новой хэш-функции во многом следует старому стандарту. Входное сообщение разбивается на блоки фиксированного размера, для сообщений размером не кратным длине блока используется дополнение. Начальное внутреннее состояние хэш-функции обновляется последовательной обработкой блоков сообщения функцией сжатия. Параллельно с этим вычисляются число обработанных бит и контрольная сумма блоков. После всех блоков сообщения функция сжатия обрабатывает блок с общей длиной сообщения и блок с контрольной суммой для завершения вычисления значения хэш-функции. Размер блоков сообщения и внутреннего состояния хэш-функции составляет 512 бит. Для целей представления длины сообщения и контрольной суммы блоков данные блока рассматриваются как числа с порядком бит little-endian по модулю 2^{512}.

Семейство состоит из двух хэш-функций с длинами результирующего значения в 256 и 512 бит, которые отличаются начальным внутренним состоянием и его частью, принимаемой за результат вычислений. Результаты данной работы одинаково применимы к обеим функциям, поскольку их производительность, очевидно, одинакова.

Основное отличие хэш-функции Стрибог от своего предшественника — функция сжатия. Её основная операция состоит из трёх преобразований: подстановки на байтах, транспонирования матрицы байт и умножения 64-битных векторов на матрицу 64×64 в GF(2).

1) S — нелинейная биекция. 512 бит аргумента рассматриваются как массив из 64 байт, каждый из которых заменяется по заданной стандартом таблице подстановки.

2) P — переупорядочивание байт. Байты аргумента меняются местами по определённому в стандарте порядку.

3) L — линейное преобразование. Аргумент рассматривается как 8 64-битных векторов, каждый из которых заменяется результатом умножения на определённую стандартом матрицу 64×64 над GF(2).

В функции сжатия используются только преобразование LPS и побитовое исключающее ИЛИ над 512-битными блоками. Вместе со сложением по модулю 2^{512} они составляют полный набор операций, использующихся в данной хэш-функции.

Следующие алгоритмические реализации были разработаны Казимировым и Шевчуком в [4]:

1) Используется таблица подстановки для побайтного умножения на матрицу в линейном преобразовании, размер этой таблицы — 16 КБ.

2) Переупорядочивание байт, определённое стандартом, является операцией транспонирования матрицы байт размером 8×8. Оно заменяется чтением с шагом в 8 байт и записью результатов в другую область памяти.

3) Подстановка во время вычисления хэш-функция заменяется соответствующей перестановкой таблицы побайтного умножения. Данная оптимизация возможна за счёт выбора байта как единицы развёртки цикла умножения.

Перечисленные оптимизации являются достаточными для создания переносимой реализации. Соответствующий код на языке C приведён в листинге 1.

Листинг 1. Функция сжатия ГОСТ Р 34.11-2012

```
const uint8_t pi[256] = { /* pi constant */ };
const uint64_t A[64] = { /* A constant */ };

uint64_t mul_table[8][256];

void precalc_mul_table()
{
  for(int i=0;i<8;++i)
    for(int j=0;j<256;++j){
      uint64_t t = 0;
      uint8_t p = pi[j];
      for(int k=0;k<8;++k)
        if(p&(1<<k))
          t ^= A[(i<<3)|k];
        mul_table[i][j] = t;
    }
}

void lps(uint64_t* out,const uint8_t* in)
{
  for(int i=0;i<8;++i){
    uint64_t t = mul_table[0][in[i]];
    for(int k=1;k<8;++k)
      t ^= mul_table[k][in[i+k*8]];
    out[i] = t;
  }
}
```

На основе данных идей автором была разработана оптимизированная реализация для архитектуры x86_64. Использование её расширенного регистрового файла позволяет хранить почти всё промежуточное состояние вычислений в регистрах.

Контрольная сумма блоков обновляется сложением по модулю 2^{512}, которое может быть эффективно выполнено серией инструкций сложения с переносом над регистрами общего назначения. Для хранения 512-битных промежуточных результатов используются четвёрки регистров SSE. В процессе выполнения преобразования LPS отдельные байты извлекаются из них и используются в табличных подстановках. Результаты аккумулируются в восьми регистрах общего назначения, а в конце снова собираются в регистры SSE. Вышеперечисленные операции требуют поддержки SSE 4.1 для максимальной эффективности, но достаточно легко эмулируются инструкциями SSE 2 (доступными на всех процессорах x86_64) с небольшой потерей производительности.

Поскольку значение функции сжатия на каждом шаге зависит от предыдущего шага, параллельно обрабатывать блоки одного потока данных невозможно. Это свойство

не является особенностью Стрибога, а присуще большинству хэш-функций, включая ГОСТ Р 34.11-94.

Размер промежуточного состояния Стрибога слишком велик, чтобы уместиться в регистры одного потока графического процессора, поэтому использование одного потока вычислений на один поток данных приведёт к использованию локальной памяти. Тем не менее, данный вариант («простой») был реализован. Также автором была реализована «гибридная» схема, в которой группа из нескольких потоков обрабатывает блоки данных последовательно. Это позволяет разделить 512-битные промежуточные значения между потоками, храня их в переменных встроенных скалярных типов. При необходимости обмена информацией между потоками в группе используется выделенная для каждой группы область разделяемой памяти. При этом необходимости в синхронизации потоков не возникает, поскольку все потоки выполнения, обрабатывающие один поток данных, являются частью одного ворпа и выполняются синхронно по шагам. Данная схема была реализована с использованием 32- и 64-битных переменных.

Несмотря на то, что для хэш-функции Стрибог наиболее удобной единицей хранения являются 64 бита, 32-битная реализация описанной выше гибридной схемы оказалась на 30% быстрее. Это объясняется тем, что для архитектуры графического процессора именно 32-битные значения являются основной аппаратной единицей. Простая схема реализации с одним потоком вычислений на один блок данных показывает производительность на уровне 64-битной гибридной.

По результатам экспериментов для доступа к таблице умножения используется текстура, а для константы раундового ключа C — константная память, как наиболее быстрые. В простой версии для реализации сложения по модулю 2^{512} используются аппаратные инструкции сложения с переносом, а в гибридной — параллельный сумматор с использованием вычисления префиксной суммы. Учитывая объём памяти графической карты, вычисление числа обработанных бит сводится к одной операции сложения, а побитовые исключающие ИЛИ распределяются по потокам тривиальным образом. По сравнению с преобразованием LPS эти операции занимают незначительное время.

В целом, хэш-функция Стрибог достаточна проста в реализации на разном оборудовании и довольно легко позволяет модифицировать себя для параллельного вычисления.

3. Хэш-функция ГОСТ Р 34.11-94

Старый стандарт хэш-функции отличается меньшим размером блока и внутреннего состояния — 256 бит. Его функция сжатия использует симметричный шифр ГОСТ 28147-89 и различные перемешивания, основанные на линейных регистрах сдвига. Хотя она также использует только операции подстановки и побитового исключающего ИЛИ, они применяются для элементов данных размером 1, 2 или 4 байта.

Большинство открытых реализаций ГОСТ Р 34.11-94 основаны на работе Саринена [8], содержащей следующие алгоритмические оптимизации:

1) Восемь 4-битных s-блоков объединены в четыре 8-битных. Их значения расширены до 32 бит и повёрнуты на 11 бит, что позволяет избежать этой операции в процессе вычислений, размер таблицы подстановки 4 КБ.

2) Многократные применения преобразования ψ заменены результирующим выражением с использованием 32-битных значений, используя маски и сдвиги для выбора нужных 16-битных элементов.

Авторская реализация ГОСТ Р 34.11-94 использует данные оптимизации, но, поскольку все значения хранятся в 128-битных регистрах, для вычисления всех преобразований используются более сложные конструкции из сдвигов и операций перемешивания, требующих поддержки SSSE3.

В отличие от Стрибога, структура алгоритма ГОСТ Р 34.11-94 не позволяет легко разделить его выполнение между несколькими потоками. «Гибридная» реализация автора для данной хэш-функции использует 4 потока ГП на поток данных, что позволяет выполнять 4 шага шифрования параллельно (после генерации и сохранения всех ключей). Для выполнения преобразований потоки группы обмениваются требуемыми частями 256-битных значений через общую память. Для сложных преобразований необходимые действия задаются управляющей таблицей. К сожалению, подобный подход требует значительного усложнения и так не простого в реализации алгоритма и на практике не даёт преимущества в скорости выполнения по сравнению с «простым» вариантом.

В общем, хэш-функция ГОСТ Р 34.11-94 является более сложной в реализации, чем новый стандарт.

4. Результаты

Все реализации тестировались на процессоре Intel Core i7-920 CPU @ 2.67 GHz и видеокарте NVIDIA GTX 580. Использовалась авторская гибридная 32-битная схема для ГОСТ 34.11-2012 (Стрибог) и простая последовательная для ГОСТ Р 34.11-94. Для сравнения использовались следующие реализации:

— openssl-ccgost — ccgost engine из OpenSSL toolkit [6], версия 1.0.1c.

— rhash — утилита RHash [7], версия 1.2.9.

— kazymyrov — Реализация хэш-функции Стрибог Александром Казимировым [4], ревизия 400a39cc126d04f660cbcdcdeb8060177a563ea3.

Результаты приведены в таблице 1.

Таблица 1. Результаты производительности

	ГОСТ Р 34.11-94		Стрибог-512	
Реализация	МБ/с	Тактов/байт	МБ/с	Тактов/байт
ЦП, 1 ядро/поток				
openssl-ccgost	18	143	-	-
rhash	49	52	-	-
kazymyrov	-	-	38	67
lebedev	64	40	94	27
ГП, 8192 потока данных				
lebedev	1697	-	608	-

Реализация ГОСТ Р 34.11-94 из OpenSSL медленная, поскольку не содержит ни алгоритмических, ни программных оптимизаций. Реализация из RHash их содержит, включая некоторые ассемблерные оптимизации, но большая часть кода написана на переносимом C. Реализация автора в переносимом варианте в целом идентична RHash, а оптимизированная для x86_64 на 30% быстрее.

Реализация алгоритма Стрибог Казимировым страдает от использования представления big-endian, делает несколько лишних копий данных и содержит ряд других недо-

статков, негативно влияющих на производительность. Авторская реализация в переносимом варианте сравнима с ГОСТ Р 34.11-94, а оптимизированная — вдвое быстрее.

Результаты на графическом процессоре даны для параллельной обработки 8192 потоков данных по 16 КБ. Эти величины были определены экспериментально, дальнейшее их увеличение не даёт значимого прироста производительности. Данные по числу тактов на байт для ГП не приведены, поскольку в условиях обработки тысяч потоков планировщиком ГП с перекрытием выполнения и другими деталями данная характеристика не имеет смысла, по крайней мере, при сравнении с таковой для ЦП. Обработка малого числа потоков на ГП нецелесообразна, т.к. производительность ГП в рассчёте на один поток данных на порядки медленнее ЦП. С этой точки зрения реализация ГОСТ Р 34.11-2012 является более выгодной, поскольку использует большее число потоков выполнения на поток данных и, следовательно, требует меньшего числа потоков данных для полной загрузки ГП.

На момент написания настоящей работы автору не известно о существовании других реализаций хэш-функции Стрибог на ГП. Единственной известной автору реализацией ГОСТ Р 34.11-94 на ГП является [3]. Её результат на видеокарте NVIDIA GTX 550 составляет $1.38 \cdot 10^6$ паролей в секунду. Учитывая, что пароль умещается в один блок данных, обработка одного пароля сравнима с обработкой трёх блоков данных, что даёт общую скорость в 126 МБ/с. Реализация автора на той же видеокарте показывает 269 МБ/с. К сожалению, точное сравнение произвести не удаётся, поскольку рассматриваемая реализация решает несколько иную задачу и не является ПО с открытым кодом.

На ЦП Стрибог оказывается быстрее за счёт использования только сложений по модулю 2 и инструкций пересылки данных. Старый ГОСТ, в свою очередь, содержит множество инструкций перемешивания, которые не лучшим образом отображаются на набор инструкций ЦП, в частности, SSE. Таблицы подстановки обоих алгоритмов умещаются в кэш процессора первого уровня. На ГП ГОСТ Р 34.11-2012 имеет превосходство за счёт использования только 4 потоков на один поток данных против 16 для Стрибога. Больший размер таблиц подстановки также может вносить вклад в отставание последнего.

В заключение отметим, что разработанные автором реализации являются наиболее производительными среди открытых реализаций криптографических хэш-функций, соответствующих прошлым и текущим стандартам Российской Федерации. Новый вариант стандарта проще в реализации и оптимизации как алгоритмически, так и для конкретной платформы. Оптимизированная версия на 47% быстрее на современных ЦП архитектуры x86_64 благодаря тому, что она хорошо соответствует набору инструкций процессора. Это даёт ускорение выполнения штатных операций. Увеличенный размер состояния и таблиц подстановки делает её медленнее на высокопараллельном оборудовании, например графических процессорах, в основном используемом для различного вида атак. Это также усложняет реализацию на ограниченных по ресурсам устройствах, в том числе встроенных системах.

СПИСОК ЛИТЕРАТУРЫ

1. ГОСТ Р 34.11-94: Государственный стандарт Российской Федерации. Информационная технология. Криптографическая защита информации. Функция хэширования. Издание официальное // М. Госстандарт России

2. ГОСТ Р 34.11-2012: Национальный стандарт Российской Федерации. Информационная технология. Криптографическая защита информации. Функция хэширования. Издание официальное. // М. Стандартинформ, 2012

3. *InsidePro Software* Extreme GPU Bruteforcer
`http://www.insidepro.com/eng/egb.shtml`

4. *Oleksandr Kazymyrov, Oleksii Shevchuk* Implementation of hash function Stribog.
`https://github.com/okazymyrov/stribog`

5. *NVIDIA Corporation* NVIDIA CUDA C Programming Guide Version 4.2 4/16/2012
`http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf`

6. OpenSSL Project
`http://openssl.org`

7. RHash Program
`http://rhash.anz.ru`

8. *Markku-Juhani O. Saarinen* Реализация GOST R 34.11-94 (с исправлениями David G. Hook)
`http://www.autochthonous.org/crypto/gosthash.tar.gz`