Реализация алгоритмов шифрования "Магма"и "Кузнечик"с помощью HIP

А.И. Борисов[1], Е.В. Мясников[1]

[1]Самарский национальный исследовательский университет им. академика С.П. Королева, Московское шоссе 34А, Самара, Россия, 443086

Аннотация. В данной статье приводятся реализации российских криптографических стандартов симметричного шифрования для графических процессоров, написанные на языке HIP - открытой технологии, позволяющей создавать код, переносимый между платформами AMD и NVIDIA. Рассматриваются вопросы эффективной реализации копирования между памятью графического процессора и оперативной памятью. Полученные результаты позволяют достичь 8-кратного увеличения скорости шифрования относительно центрального процессора.

1. Введение

В настоящее время криптографическая защита информации является неотъемлемой частью современной IT-инфраструктуры. Постоянно возрастают как объемы обрабатываемой информации, так и вычислительная мощность компьютеров. Соответственно, растут требования, как к стойкости алгоритмов, так и к их скорости.

Идея использования графических процессоров для ускорения существующих алгоритмов шифрования появилась практически одновременно с самой идеей их использования для вычислений общего назначения. Как известно, наибольший эффект от применения графических процессоров достигается при возможности полного распараллеливания задачи. Неудивительно, что наиболее заметные результаты в этой сфере были получены для блочного шифрования в режимах ECB (электронной кодовой книги) и CTR (режим гаммирования) [1], поскольку блоки открытого текста в данном случае обрабатываются независимо.

К настоящему времени написан целый ряд работ, посвященный ускорению блочных шифров с помощью CUDA, по большей части посвященных реализации AES и других известных шифров [2, 3, 4]. В работе [3] исследуется зависимость скорости шифрования от места расположения раундовых ключей, при этом удается достичь скорости шифрования более 250 Гбит/с без учета копирования данных между памятью графического ускорителя и оперативной памятью. В работе [4], являющейся продолжением работы [3], удается достичь 80 Гбит/с с учетом копирования.

Алгоритм "Кузнечик"[5] является новым российским стандартом симметричного шифрования, веденным в 2015 году в дополнение к более старому алгоритму "Магма"(в англоязычных статьях так же известного как GOST)[5], используемому с 1989 года.

"Кузнечик"отличается от "Магмы"как по длине блока (128 бит вместо 64), так и по общей структуре(SP-сеть вместо сети Фейстеля).

Алгоритму "Кузнечик"посвящено существенно меньшее число работ, по большей части сконцентрированных на криптоанализе алгоритма. В работе [6] приводится оптимизация алгоритма на основе таблиц поиска, скорость, достигаемая на 4-хпоточном ЦП составляет 54 МБ/с. Работа [7] является прямым продолжением работы [6], но посвящена уже криптоанализу алгоритмов "Магма"и "Кузнечик". CUDA используется как средство ускорения поиска пар для скользящей атаки на шифр "Магма". Хотя относительно версии, исполняемой на обычном процессоре, декларируется ускорение в 129 раз, полученные для "Магмы"результаты неприменимы для "Кузнечика"в силу их различий.

В работе [8] исследуются возможные варианты реализации алгоритма "Кузнечик"на основе таблиц подстановки на платформе NVIDIA CUDA, итоговая максимальная скорость шифрования составила 30 Гбит/с на графическом процессоре NVIDIA GeForce GTX 1070.

Следует отметить, что подавляющее большинство статей, в которых рассматривается использование графических процессоров NVIDIA и соответствующей технологии NVIDIA CUDA. В данной статье рассматривается реализация российских алгоритмов шифрования "Кузнечик"и "Магма"в режиме гаммирования с помощью открытой технилогии HIP, позволяющей писать программы как для графических процессоров NVIDIA, так и для графических процессоров AMD. В качестве исполнительного устройства выступает AMD Radeon RX Vega 56.

2. Алгоритм "Кузнечик"

«Кузнечик» представляет собой 10-раундовый шифр на основе SP-сети с длиной ключа 256 бит и длиной блока 128 бит. Полное описание может быть найдено в [5].

2.1. Алгоритм расширения ключа

Перед началом шифрования на основе главного 256-битного ключа генерируются 10 128-битных раундовых ключей $K_1, ..., K_{10}$. Внутри процедура расширения представляет собой сеть Фейстеля, функция трансформации в которой аналогична раунду основного шифра с фиксированным ключом. Первые два раундовых ключа K_1, K_2 получаются из половин основного. Остальные ключи получаются шифрованием сетью Фейстеля. На вход каждого раунда сети подаются ключи (K_{2i}, K_{2i-1}), на выходе получаются (K_{2i+2}, K_{2i+1}).

Процедура расширения имеет строго последовательный характер, кроме того, она может исполняться только один раз для получения массива раундовых ключей, и потому в данной работе не рассматривается.

2.2. Основные преобразования

В основе шифра лежат два преобразования:

(i) Нелинейное преобразование $\pi : GF(2^8) \to GF(2^8)$, реализуемое через таблицу поиска

(ii) Линейное преобразование

$$\ell(a_{15}, \ldots, a_0) = 148 \cdot a_{15} + 32 \cdot a_{14} + 133 \cdot a_{13} + 16 \cdot a_{12} + 194 \cdot a_{11} + 192 \cdot a_{10} + 1 \cdot a_9 + \\ +251 \cdot a_8 + 1 \cdot a_7 + 192 \cdot a_6 + 194 \cdot a_5 + 16 \cdot a_4 + 133 \cdot a_3 + 32 \cdot a_2 + 148 \cdot a_1 + 1 \cdot a_0, \quad (1)$$

где $a_i \in GF(2^8)$,
операции происходят в поле $GF(2^8)$(в частности, + эквивалентен операции XOR)

На основе этих двух преобразований определяются преобразования, использующиеся непосредственно в шифровании:

$$\begin{aligned} S(a) &= S(a_{15}||\dots||a_0) = \pi(a_{15})||\dots||\pi(a_0), && (2)\\ R(a) &= R(a15||\dots||a0) = \ell(a15,\dots,a0)||a15||\dots||a1, && (3)\\ L(a) &= R^{16}(a), && (4)\\ X[k](a) &= k \oplus a, && (5) \end{aligned}$$

где $||$ означает конкатенацию,
$a_i \in GF(2^8)$,
$k, a \in GF(2^{128})$,
$R^{16}(a)$ означает 16-кратное применение функции R над a.

В этих обозначениях алгоритм «Кузнечик» может быть описан следующим образом:

$$E_{K_1,\dots,K_{10}}(a) = X[K_{10}]LSX[K_9]\dots LSX[K_2]LSX[K_1](a), \quad (6)$$

где запись вида $LSX[K](a)$ эквивалентна $L(S(X[K](a)))$.

3. Алгоритм "Магма"

Алгоритм "Магма"представляет собой шифр, основанный на сети Фейстеля. Длина ключа - 256 бит, длина блока - 64 бита.

3.1. Алгоритм расширения ключа

256-битный ключ разбивается на 8 32-битных частей k_1,...,k_8. Раундовые ключи K_1, ..., K_{24} генерируются циклическим повторением k_1,...,k_8, ключи K_{25}, ..., K_{32} равны k_8,...,k_1 (обратный порядок частей).

3.2. Функция шифрования

Основным элементом сети Фейстеля является раундовая функция, определяющая свойства конкретного шифра. Для алгоритма "Магма"раундовая функция имеет вид

$$f(x,k) = \pi(x+k) \lll 11 \quad (7)$$

где x - 32-битный полублок текста,
k - 32-битный раундовый ключ,
π - нелинейное преобразование, реализуемое табличной подстановкой,
$+$ - беззнаковое арифметическое сложение с переполнением,
$\lll$ - циклический сдвиг влево.

Структура сети Фейстеля в алгоритме "Магма"не подвергалась никаким изменениям, ее общий вид и принцип работа описаны в [9].

4. Режим гаммирования

Блочные алгоритмы шифрования могут эксплуатироваться в нескольких режимах. Наиболее простым режимом является режим простой замены (или режим электронной кодовой книги, ECB), при использовании которого каждый блок открытого текста шифруется независимо. Однако данный режим не обеспечивает защиты от статистических атак, поскольку два одинаковых блока открытого текста преобразуются в одинаковый шифротекст.

Для преодоления данной проблемы были созданы другие режимы шифрования: сцепления блоков, с обратной связью по выходу, с обратной связью по шифротексту. Их недостатком является невозможность распараллеливания обработки блоков текста.

В отличие от вышеприведенных режимов, режим гаммирования допускает параллельную обработку блоков шифротекста и обеспечивает защиту от статистических атак, путем превращения блочного шифра в аналог потокового. В режиме гаммирования шифруется не блок открытого текста, а значение некоторого определенным образом изменяющегося счетчика, стартовая позиция которого задается перед процедурой шифрования. Результат шифрования затем складывается по модулю 2 с блоком открытого текста. Расшифрование, в силу обратимости сложения по модулю 2, производится так же, как шифрование.

Поскольку алгоритм изменения счетчика известен заранее (обычно, это простое увеличение на 1), каждый блок открытого текста может быть зашифрован независимо от других, что дает возможность использовать графические процессоры для ускорения шифрования в режиме гаммирования.

5. HIP

HIP [10] - новый инструмент программирования для графических процессоров, разработанный под эгидой инициативы GPUOpen. HIP играет роль прослойки, предоставляющей язык написания программ для графических процессоров и интерфейс взаимодействия с графическими процессорами, дублирующие их аналоги в CUDA (в основном посредством макросов C++). Программы, написанные на CUDA, могут буть перенесены на HIP с относительно небольшими изменениями (AMD предоставляет средство автоматической миграции hipify). В зависимости от используемой в системе платформы, HIP может производить компиляцию либо с помощью HCC (AMD), либо с помощью NVCC (NVIDIA). Таким образом, код, написанный на HIP, может быть скомпилирован как для графических процессоров AMD, так для графических процессоров NVIDIA.

Терминология и программная модель HIP совпадают с терминологией и программной моделью CUDA. Взаимодействие с устройством производится с помощью посыла команд графическому процессору через API. Каждая команда привязывается к определенному потоку команд (stream). Команды в разных потоках команд, по возможности, исполняется параллельно. К командам относятся команды копирования, синхронизации и исполнения функций. Функция, предназначенная для запуска на графическом процессоре, называется функцией ядра. Каждая функция ядра исполняется множеством потоков, или в устоявшемся переводе терминологии CUDA - нитей (thread). Нити объединяются в блоки, блоки образуют сетку блоков. Из доступных программисту типов памяти достаточно отметить глобальную память - основную память графического процессора, и разделяемую память - сверхбыструю память малого объема.

6. Реализация алгоритмов с помощью HIP

6.1. Алгоритм "Кузнечик"

В статье [8] был проведен анализ различных реализаций алгоритма "Кузнечик"на основе замены LS-части алгоритма на поиск по ряду предвычисленных таблиц (таблиц подстановок). В результате было получено 16 таблиц суммарным объемом 64КБ, для которых наиболее эффективным оказалось размещение в глобальной памяти.

Каждая таблица подстановки содержит результат комбинированного LS-преобразования для всех возможных значений для одного из 16 байт блока текста - итого 16 таблиц, в каждой по 256 записей длиной 16 байт. Вариант алгоритма "Кузнечик использующий данное разложение, обозначен ниже как "Кузнечик-1"

Объем таблиц подстановок препятствует их расположению в быстрой разделяемой памяти. С целью уместить таблицы подстановок в разделяемую память, был рассмотрен

иной вариант разложения на таблицы. В этом случае каждая таблица подстановки содержит 16 записей, по одной записи на возможное значение тетрады (4 бита). Итого получается 32 таблицы (по одной на каждую тетраду входного блока), каждая из 16 записей длиной 16 байт.

Итоговый размер таблиц позволяет расположить их в быстрой разделяемой памяти. Более того, возможно разложить их в порядке, исключающем конфликты доступа (конфликты банков, bank conflicts), связанные с одновременным доступом 2-х потоков к разным значениям внутри одного банка разделяемой памяти. Вариант алгоритма "Кузнечик использующий данное разложение, обозначен как "Кузнечик-2".

Раундовые ключи шифрования и начальное значения счетчика режима гаммирования копируются на устройство до начала шифрования и располагаются в разделяемой памяти.

6.2. Алгоритм "Магма"

Таблицы преобразования π располагаются в разделяемой памяти. Раундовые ключи шифрования и начальное значения счетчика режима гаммирования копируются на устройство до начала шифрования и также располагаются в разделяемой памяти.

6.3. Организация копирования

Достаточно часто фактором, ограничивающим общую производительность, является скорость копирования данных между памятью графического процессора и оперативной памятью. В данной статье рассматриваются два варианта работы с данными:

(i) копирование данных в память графического процессора;

(ii) прямой доступ к данным в оперативной памяти.

Кроме того, рассматривается так же использование нескольких потоков команд устройства. В этом случае общий тестовый набор данных делится на равные части, каждая из которых затем независимо шифруется в отдельном потоке команд.

Для ускорения копирования используется закрепление страниц памяти, содержащих данные. В этом случае гарантируется отсутствие ошибок страниц, что позволяет драйверу производить копирование оптимальным образом без использования защитных мер. Закрепление памяти занимает некоторое время, однако оно чаще всего производится на этапе подготовки к шифрованию, и потому временные затраты на закрепление памяти не включаются в общие замеры времени.

7. Результаты экспериментов

В целях измерения производительности приведенных реализаций алгоритмов было проведено несколько экспериментов.

В первом эксперименте замерялась скорость шифрования 256 МБ данных без учета копирования. Шифруемые данные находились в памяти графического процессора.

Во втором эксперименте замерялась скорость шифрования данных с учетом копирования в зависимости от объема данных. Шифруемые данные либо копировались в память графического процессора и обратно, либо оставались в оперативной памяти (прямой доступ). Использовался размер блока нитей, позволяющий достичь максимальной скорости шифрования в первом эксперименте.

В третьем эксперименте замерялась скорость шифрования 1 ГБ данных с учетом копирования при использовании нескольких потоков команд. Шифруемые данные либо копировались в память графического процессора и обратно, либо оставались в оперативной памяти (прямой доступ).

Дополнительно, в сравнительных целях были написаны реализации алгоритмов для центрального процессора, использующие AVX-регистры.

Все тесты запускались на следующей конфигурации:

ЦП: AMD Ryzen 7 3700X, 8 ядер, 3,6 ГГц;
ГП: AMD Radeon RX Vega 56, 3584 потоковых процессоров, 8 ГБ HBM2;
ОЗУ: 16 ГБ;
ОС: Ubuntu 18.04 LTS.

Результаты экспериментов приведены в таблицах 1, 2, 3 и 4.

Таблица 1. Скорость шифрования без учета копирования в зависимости от размера блока нитей

Число нитей	Скорость шифрования, Гбит/с		
	Кузнечик-1	Кузнечик-2	Магма
16	33,47	57,54	229,82
32	43,34	115,5	451,15
64	48,64	110,68	508,48
128	55,64	142,9	696,5
256	60,17	167,74	853,67
384	64,2	176,45	921,19
512	66,52	165,91	876,84
768	69,68	175,56	940,16
1024	67,97	172,81	928,84

Таблица 2. Скорость шифрования с учеитом копирования в зависимости от объема данных

Объем данных, МБ	Скорость шифрования, Гбит/с					
	Кузнечик-1		Кузнечик-2		Магма	
	Копирование	Прямой доступ	Копирование	Прямой доступ	Копирование	Прямой доступ
2	17,789	28,305	18,695	37,891	23,734	39,938
4	19,047	37,695	20,914	47,422	24,578	47,273
8	20,367	44,727	22,547	54,211	25,195	57,961
16	20,57	43,078	23,742	53,445	25,883	55,313
32	20,93	44,766	24,625	59,195	27	55,313
64	21,234	38,398	25,531	55,242	28,289	54,758
128	21,102	39,422	26,016	58,313	29,25	57,273
256	21,242	41,992	26,023	64,5	29,625	63,242
512	20,836	46,078	26,141	73,031	29,75	70,117
1024	20,313	49,094	25,883	79,273	29,305	74,742

Как видно из результатов экспериментов без учета копирования, "Кузнечик-2"в 2,5 раза быстрее, чем "Кузнечик-1"за счет использования быстрой разделяемой памяти. Алгоритм "Магма за счет своей простоты оказывается быстрее, чем "Кузнечик".

Если учитывать копирование, то, хотя пиковая пропускная способность интерфейса PCIe 3.0 равна 126 Гбит/с (15,8 ГБ/с), ни один из алгоритмов на используемой конфигурации не способен пересечь границу в 80 Гбит/с (10 ГБ/с), но крайне близки к этому. Результаты статьи [4] так же близки к 80 Гбит/с, что может указывать на влияние накладных расходов на коммуникацию с графическим ускорителем. В общем случае, прямой доступ к данным в оперативной памяти по шине оказывается быстрее, чем копирование в память

Таблица 3. Скорость шифрования с использованием нескольких потоков команд

Число потоков команд	Скорость шифрования, Гбит/с					
	Кузнечик-1		Кузнечик-2		Магма	
	Копирование	Прямой доступ	Копирование	Прямой доступ	Копирование	Прямой доступ
2	36,164	52,266	53,703	79,391	57,578	74,984
4	41,828	54,914	66,563	79,727	62,688	75,313
6	43,563	55,414	73,219	78,945	67,945	75,789
8	44,594	55,641	76,359	79,797	69,719	75,555
10	45,43	55,156	66,164	78,938	70,406	75,664
12	46,969	56,453	72,703	78,852	73,297	75,688
14	48,078	58,219	73,156	78,813	75,125	75,789
16	48,844	59,539	75,328	79,68	76,43	75,609
32	58,305	62,813	78,82	79,258	79,32	75,531

Таблица 4. Сравнение максимальных скоростей шифрования для центрального и графического процессоров

Алгоритм шифрования	Скорость шифрования, Гбит/с	
	Ryzen 3700X	Vega 56
Кузнечик-1	10,229	62,813
Кузнечик-2	8,585	79,797
Магма	5,082	79,32

ускорителя. Однако использование нескольких потоков команд практически нивелирует разницу в производительности, поскольку позволяет перекрывать операции шифрования и копирования в различных потоках команд. В любом случае, из результатов экспериментов следует, что основным ограничивающим фактором является пропускная способность интерфейса PCIe 3.0, и с выходом новых версий интерфейса общая скорость шифрования будет только возрастать.

8. Заключение

В данной статье были рассмотрены реализации российских алгоритмов шифрования "Кузнечик"и "Магма"с помощью языка HIP. На графическом процессоре AMD Radeon Vega 56 реализация алгоритма "Кузнечик"с использованием разделяемой памяти развивает скорость шифрования в 176 Гбит/с без учета копирования, реализация алгоритма "Магма"позволяет достичь скорости шифрования в 940 Гбит/с без учета копирования. Если учитывать копирование, максимальная скорость шифрования равна 79 Гбит/с при использовании нескольких потоков команд, и ограничивается пропускной способностью интерфейса PCIe 3.0.

9. Благодарности

Работа выполнена при финансовой поддержке РФФИ в рамках научного проекта № 20-37-70053 в частях «1. Введение» - «6. Реализация алгоритмов с помощью HIP» и Министерства науки и высшего образования РФ в рамках госзадания ФНИЦ «Кристаллография и фотоника» РАН в частях «7. Результаты экспериментов» и «8. Заключение».

10. Литература

[1] ГОСТ 34.13 - 2015. Криптографическая защита информации. Режимы работы блочных шифров – Москва: Стандартинформ, 2016. – 24 с.

[2] Lee, W.-K. Fast implementation of block cipheis and PRNGs in Maxwell GPU aichitectuie / W.-K. Lee, H.-S. Cheong, R.C.-W. Phan, B.-M. Goi // Clustei Comput. – 2016. – Vol. 19(1). – P. 335-347.

[3] Abdeliahman, A.A. Analysis on the AES Implementation with Vaiious Gianulaiities on Diffeient GPU Aichitectuies / A.A. Abdeliahman, M.M. Fouad, H.M. Dashan // Advances in Electiical and Electionic Engineeiing. – 2017. – Vol. 15(3).
[4] Abdeliahman, A.A. Enhancing the Actual Thioughput of the AES Algoiithm on the Pascal GPU Aichitectuie / A.A. Abdeliahman, H.M. Dashan, G.I. Salama // 3id Inteinational Confeience on System Reliability and Safety. – 2018. – P. 97-103.
[5] ГОСТ 34.12 - 2015. Криптографическая защита информации. Блочные шифры – Москва: Стандартинформ, 2015. – 21 с.
[6] Ищукова, Е.А. Разработка и реализация высокоскоростного шифрования с использованием алгоритма «Кузнечик» / Е.А Ищукова, Р.А. Кошуцкий, Л.К. Бабенко // Журнал Auditoiium. – 2015. – Т. 4, № 8.
[7] Ishchukova, E. Fast Implementation and Ciyptanalysis of GOST R 34.12-2015 Block Cipheis / E. Ishchukova, L. Babenko, M. Anikeev // Pioceedings of the 9th Inteinational Confeience on Secuiity of Infoimation and Netwoiks, 2016 – P. 104-111.
[8] Boiisov, A.N. The implementation of "Kuznyechik" enciyption algoiithm using NVIDIA CUDA technology / A.N. Boiisov, E.V. Myasnikov // CEUR Woikshop Pioceedings. – 2019. – Vol. 2416. – P. 308-313.
[9] Menezes, A.J. Handbook of Applied Ciyptogiaphy / A.J. Menezes, P.V. Ooischot , S.A. Vanstone – CRC Press, 1996. – 816 p.
[10] Sun, Y. Evaluating Peifoimance Tiadeoffs on the Radeon Open Compute Platfoim / Y. Sun, S. Mukheijee, T. Baiuah, S. Dong, J. Gutieiiez, P. Mohan, D. Kaeli // IEEE Inteinational Symposium on Peifoimance Analysis of Systems and Softwaie – Belfast, 2018. – P. 209-218.

Implementation of “Magma” and “Kuznyechik” ciphers using HIP

A.N. Borisov[1], E.V. Myasnikov[1]

[1]Samara National Research University, Moskovskoe Shosse 34A, Samara, Russia, 443086

Abstract. In this paper we present an implementation of Russian cryptographic standards using HIP - an open source technology that allows writing an C++ code, which can be compiled for AMD and NVIDIA GPUs. Details of effective CPU-GPU data copy is considered. The 8 times perfomance gain over CPU AVX version is reached.