Павлов Владислав Эдуардович
студент
Удальцов Валерий Анатольевич
магистрант
ФГАОУ ВО «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»
г. Санкт-Петербург

DOI 10.21661/r-129822

ОПТИМИЗАЦИЯ СКОРОСТИ РАБОТЫ БЛОЧНЫХ АЛГОРИТМОВ ШИФРОВАНИЯ

Аннотация*: по мнению исследователей, задача оптимизация работы алгоритмов шифрования является востребованной и актуальной в настоящее время. В данной статье рассматривается симметричный алгоритм блочного шифрования «Магма» ГОСТ Р 34.12–2015 и технологии nVidia CUDA, OpenCL, OpenGL. Особенности данных технологий используются для оптимизации работы алгоритма шифрования «Магма». Авторами статьи произведены контрольные замеры скорости работы алгоритма шифрования с использованием различных технологий, результаты которых представлены в виде таблицы. Представлен анализ полученных результатов, позволяющий оценить преимущество в скорости работы алгоритма шифрования использующего для расчетов возможности рассматриваемых технологий. Результаты исследования, представленные в данной статье, в дальнейшем могут быть использованы для увеличения скорости процесса шифрования при применении отечественных стандартов шифрования, таких как ГОСТ Р 34.12–2015, ГОСТ 28147–89, а также при проведения различных исследований, направленных на выявление недостатков в области надежности шифров, с целью сокращения общего времени, затрачиваемого на исследование шифра.*

***Ключевые слова**: криптографическое преобразование, криптография, ГОСТ, nVidia, CUDA, OpenCL, OpenGL, параллельные вычисления.*

Введение

Конфиденциальность информации, наряду с целостностью и доступностью, является важнейшим свойством информации, без соблюдения которого невозможно достижение информационной безопасности. Конфиденциальность обеспечивается различными методами, одним из таких методов является криптография. Использование криптографических алгоритмов шифрования требует большого количества вычислительных и временных ресурсов, поэтому задача оптимизации скорости работы алгоритмов шифрования в настоящее время является актуальной.

В статье рассматриваются способы ускорения вычислений с применением неспецифических вычислений на графических процессорах.

Постановка задачи

На сегодняшний день блочные шифраторы, которые используют для своих вычислений мощности центрального процессора (далее – CPU), преобразуют информации со скоростью до 50 МБ/с. Использование ресурсов графического процессора (далее – GPU) позволяет увеличить эту цифру в несколько раз.

Перед нами поставлена цель оптимизировать скорость работы блочного алгоритма шифрования за счет использования неспецифических вычислений на GPU.

Описание используемой модели

В данной статье для достижения поставленной цели применяются неспецифические вычисления на графических процессорах, которые реализованы для алгоритма шифрования «Магма», описанного в ГОСТ Р 32.12–2015. Преимуществом данного метода оптимизации скорости работы криптографических алгоритмов шифрования является возможность использования большого количества одновременно выполняемых потоков. Для осуществления вычислений на графических процессорах применены следующие технологии: CUDA, OpenCL и вычислительные шейдеры OpenGL.

Рассмотрим ГОСТ Р 34.12–2015 и используемые технологии. Алгоритмы базовых блочных шифров, применяемые в криптографических методах обработки и защиты информации, в том числе для обеспечения конфиденциальности, аутентичности и целостности информации при ее передаче, обработке и хранении в автоматизированных системах определяются ГОСТ 34.12–2015.

Алгоритм шифрования «Магма» описан в ГОСТ 34.12 2015 г., как алгоритм с длиной блока 64 бит, длинной секретного ключа 256 бит и 32 итерационными ключами по 4 бита каждый [1].

В качестве нелинейного биективного преобразования выступает функция подстановки *t*:

$$t(a) = \pi_7(a_7)|| \dots ||\pi_0(a_0), (1)$$

где a – вектор длинной 32 бита, a_i – вектор длинной 4 бита, π_i – таблица подстановки.

Преобразование g имеет вид:

$$g(k, a) = t(a + k) \ll 11, (2)$$

где a и k – вектора длинной 32 бита, представленные в виде целочисленных значений, итогом сложения которых является вектор длинной 32 бита, << n циклический сдвиг на n компонентов.

Преобразования с участием итерационных ключей имеют вид:

$$G(k, a_1||a_0) = a_0||(g(k, a_0) \oplus a_1), (3)$$

$$G^*(k, a_1||a_0) = (g(k, a_0) \oplus a_1)||a_0, (4)$$

где k – итерационный ключ длинной 32 бита, a_i – часть блока подвергаемого преобразованию длиной 32 бита.

Итерационные ключи вычисляются из секретного ключа, по формуле:

$$K_i = \begin{cases} k_i, \text{при } 0 \le i < 8 \\ k_{i\text{-}8}, \text{при } 8 \le i < 16 \\ k_{i\text{-}16}, \text{при } 16 \le i < 24 \\ k_{7\text{-}(i\text{-}24)}, \text{при } 24 \le i < 32 \end{cases}, (5)$$

где K_i – итерационный ключ длинной 32 бита, k_i – часть секретного ключа длинной 32 бита.

CUDA (Compute Unified Device Architecture) – это программно-аппаратная платформа для параллельных вычислений, использующая ресурсы графического процессора nVidia для неграфических вычислений. Разработка CUDA была начата в 2006 году. Поддерживаются языки программирования C, C++ и Fortran, операционные системы Windows 7, Windows XP, Windows Vista, Linux, Mac OS X. Все поддерживаемые графические процессоры nVidia разбиты на несколько классов в зависимости от аппаратных средств и поддерживаемых операций, поддерживается обратная совместимость.

CUDA использует модель SIMD, что накладывает ограничения на алгоритмы, которые можно эффективно исполнять на такой платформе. Так, например, при выполнении условного оператора возникает проблема с обработкой альтернативной ветки – приходится выполнять операции, результат которых будет отброшен. Однако CUDA маскирует эту проблему, программисту не приходится решать её самому. Из-за особенностей процессора CUDA может работать только с ограниченным набором данных – одномерные, двумерные и трёхмерные блоки с данными одного типа. Каждый блок может обрабатываться несколькими потоками, использующими разделяемую память. CUDA позволяет выполнять вычисления одновременно на центральном и графическом процессорах за счёт асинхронности вызова вычислительного ядра. С точки зрения программиста программа с использованием CUDA состоит из нескольких фаз, каждая из которых выполняется на центральном или графическом процессоре. Функции, которые могут быть исполнены на графическом процессоре, помечаются специальными ключевыми словами (расширение языка) и называются ядрами. Задействованные структуры данных также помечаются ключевыми словами. При компиляции специальный препроцессор разделяет фазы на две части, эти части компилируются отдельно, то есть модификация компилятора исходного языка программирования не требуется. Адресные пространства центрального и графического процессоров независимы друг от друга, что вызывает задержку при копировании данных между ними.

Негативными сторонами являются необходимость наличия графического процессора именно от компании nVidia [2].

OpenCL (Open Computing Language) – открытый стандарт для универсального параллельного программирования различных типов процессоров. Стандарт предоставляет программистам переносимый и эффективный доступ ко всем возможностям гетерогенных вычислительных платформ. Для координации работы всех устройств гетерогенной системы всегда есть одно главное устройство, которое взаимодействует со всеми остальными посредством OpenCL API. Такое устройство называется «хост» и определяется вне OpenCL. Хост посылает пакеты с информацией и исполнительные команды на ускорители и получает готовые данные. Хост обрабатывает и выполняет программный код, представляющий собой тело программы, на центральном процессоре под управлением операционной системы, используя C++ или другой язык программирования. Ускорители выполняют OpenCL код, написанный на языке OpenCL C99. Существует определенный OpenCL компилятор для центрального процессора, для графического процессора и для специальных карт-ускорителей.

Плюсы технологии несомненны: кроссплатформенность, способность исполнять код как под GPU, так и под CPU, поддержка стандарта целым рядом компаний: Apple, AMD, Intel, nVidia и некоторыми другими. Минусов не так много, но и они есть: необходимость установки и настройки драйверов для работы OpenCL на различных устройствах. Для использования OpenCL в проекте требуется установить соответствующий SDK в зависимости от выбора аппаратного ускорителя – CPU, GPU. Получить SDK можно на сайте производителя – Intel, nVidia, AMD и пр. Программирование на OpenCL включает подготовку кода для запуска в ядре (распараллеливаемые операции) и на хосте (программа, управляющая ядрами). Подробное описание использования данной технологии можно найти в официальной документации. В качестве примера приведен фрагмент кода, выполняющий инициализацию OpenCL [3].

OpenGL (Open Graphics Library – открытая графическая библиотека) – спецификация, определяющая независимый от языка программирования крос-

сплатформенный программный интерфейс для написания приложений, использующих двумерную и трехмерную компьютерную графику. В стандарте OpenGL 4.3 вышедшем в 2012 году было добавлено расширение GL_ARB_compute_shader позволяющее создавать и использовать вычислительные шейдеры, применяемые для не графических вычислений, в данных шейдерах используется общий стандарт языка(GLSL). Преимуществом использования данной технологии является ее распространенность.

Явным недостатком данной технологии является неприспособленность GLSL для неграфических вычислений, проявляющаяся в отсутствии указателей, а также отсутствием явного доступа к различным видам памяти графического устройства.

Применение модели

С целью проведения сравнительного анализа скорости шифрования рассматриваемого алгоритма с использованием CPU и описанных технологий, алгоритм «Магма» был реализован в режиме электронно-цифровой книги. Время шифрования для данных реализаций приведено в таблице 1.

Для эксперимента были взяты файлы объемом 1, 10, 100, 512 и 1024 МБ.

Измерения проводились на персональной ЭВМ со следующими характеристиками:

– CPU Intel Core I7–3537U;

– GPU GeForce GT750M;

– актуальная версия видеодрайвера 372.70 от 30.08.2016.

Таблица 1

Результаты шифрования файлов с применением различных технологий

Объем данных, МБ	Время шифрования, мс.			
	CPU	CUDA	OpenCL	OpenGL
1	22	3	5	7
10	217	30	45	53
100	2149	290	425	466
512	10853	1281	1872	2997
1024	21026	2464	4186	6247

Анализ полученных результатов

С учетом данных, представленных в таблице 1 видно, что входящая информация обрабатывалась со средней скоростью, представленной в таблице 2.

Таблица 2

Средняя скорость работы алгоритма шифрования с применением различных технологий

Используемая технология	Средняя скорость обработки входящей информации, МБ/с
CPU	46.79
CUDA	365.35
OpenCL	235.13
OpenGL	176.18

Исходя из полученных результатов видно, что применение любой из рассматриваемых технологий значительно увеличивает скорость работы блочного алгоритма шифрования «Магма». Отдельно стоит сказать, использование технологии CUDA позволяет добиться увеличения скорости обработки входящих данных примерно в 8 раз.

Заключение

В ходе работы были изучены технологии, позволяющие значительно увеличить скорость работы алгоритмов шифрования. Использование таких технологий, как CUDA, Open CL и OpenGL позволяет добиться увеличения скорости обработки входящих данных примерно в 8 раз. Анализ этих результатов подтверждает правильность выбранного подхода для достижения поставленной цели.

Данная разработка может быть использована в дальнейшем для ускорения процесса шифрования при применении отечественных стандартов шифрования, таких как ГОСТ 34.12–2015, ГОСТ 28147–89, а также с целью проведения исследований, направленных на выявление недостатков в области надежности шифров с применением различных методов анализа, с целью сокращения общего времени, затрачиваемого на исследование шифра.

Список литературы

1. ГОСТ Р 34.12–2015. Информационная технология. Криптографическая защита информации. Блочные шифры. – Введ. 19.06.2015. – М.: Стандартинформ. – 25 с.

2. Ханкин К.М. Сравнение эффективности технологий OPENMP, nVidia CUDA и STARPU на примере задачи умножения матриц [Текст] / К.М. Ханкин // Вестник Южно-Уральского государственного университета. Серия: Компьютерные технологии, управление, радиоэлектроника. – 2013. – №1. – С. 34–41.

3. Круглов А.В., Круглов В.Н., Чирышев А.В., Чирышев Ю.В. Применение ресурсоемких алгоритмов в системах машинного зрения реального времени // Фундаментальные исследования. – 2013. – №10–12. – С. 2612–2619.