Babenko Lyudmila Klimentevna
Taganrog Institute of Technology – Federal State-Owned Autonomy Educational Establishment of Higher Vocational Education "Southern Federal University".
E-mail: blk@fib.tsure.ru.
Block "I", 2, Chehov street, Taganrog, 347928, Russia.
Phone: +78634312018.

Ischukova Evgeniya Aleksandrovna
E-mail: jekky82@mail.ru.
Phone: +78634371905.

УДК 004.421.4

Л.К. Бабенко, И.Д. Сидоров, А.С. Кириллов

УСКОРЕНИЕ ВЫЧИСЛЕНИЙ ДИСКРЕТНОГО ЛОГАРИФМА С ПОМОЩЬЮ ТЕХНОЛОГИИ CUDA

Рассматриваются возможности дальнейшего ускорения реализации дискретного логарифмирования. Анализируется возможность применения технологии CUDA для ускорения вычислений на различных этапах. Рассматривается эффективная реализация необходимых арифметических операций. Приведены графики, построенные по результатам проведенных экспериментов.

Криптоанализ; дискретное логарифмирование; технология CUDA; параллельное программирование; вычислительно сложные задачи.

L.K. Babenko, I.D. Sidorov, A.S. Kirillov

SPEEDING UP DISCRETE LOG COMPUTATIONS USING CUDA TECHNOLOGY

Different possibilities for further discrete log performance increase are considered. The capabilities of CUDA technology for speeding up different parts of computation process are analyzed. Here we represent effective implementation of needful arithmetic operations. Some graphic materials illustrate results of experiments.

Cryptanalysis; discrete log problem; CUDA technology; parallel programming; computation intensive tasks.

Введение. Постановка задачи. В настоящее время широко распространены алгоритмы шифрования и цифровой подписи (такие, как Эль-Гамаль, DSA, ГОСТ), стойкость которых основана на сложности решения задачи дискретного логарифмирования в мультипликативной группе числового поля и в группе точек эллиптической кривой над конечным полем. Для произвольной циклической группы G эта задача формулируется следующим образом: по известным $a, b \in G$ найти такой логарифм x, что $a^b = x$.

Особенностью данной задачи является быстрый рост времени выполнения при увеличении размера задачи. Авторами был разработан ряд эффективных параллельных алгоритмов, предназначенных для ускорения решения задачи дискетного логарифмирования в различных группах с помощью распределённых многопроцессорных вычислений. Алгоритмы, предназначенные для мультипликативной группы числового поля, рассматриваются в работах [1,2], а алгоритмы для группы точек эллиптической кривой – в работе [3].

Тем не менее, многопроцессорные вычислительные системы на сегодняшний день по соотношению цены и производительности проигрывают специализированным ускорителям. Наиболее перспективными по соотношению таких параметров, как вычислительная мощность, цена, энергопотребление и удобство программирования, являются системы, построенные на базе видеокарт компании NVIDIA с поддержкой технологии CUDA.

Технология CUDA(*Compute Unified Device Architecture*) предоставляет возможность программирования видеокарт с помощью расширения языка Си. Видеокарты, поддерживающие эту технологию, являются одновременно и SIMD вычислителями (так как один поток может выполнять вычисления над набором данных), и MIMD вычислителями (так как в видеокарте находятся несколько мультипроцессоров, способных выполнять несколько потоков одновременно). Применение CUDA даёт особенно хорошие результаты для задач, в которых присутствует обработка векторных данных.

В сравнение с GPU, CPU ориентирован на максимально быстрое выполнение одного потока данных со случайным доступом к памяти. Все существующие на сегодняшний день способы распараллеливания являются надстройкой к общей идее и неспособны обеспечить кратный рост производительности. Таким образом, большинство процессоров, за исключением специализированных, не ориентированы на массивно-параллельные вычисления. GPU, в свою очередь, изначально разрабатывались под параллельные вычисления. Из структуры GPU можно увидеть, что есть большая память DRAM, доступ к которой возможен из любой вычислительной единицы. Вычислительные единицы называются потоковыми процессорами(SP), они объединяются в мультипроцессоры(MP). Один мультипроцессор представляет собой SIMD вычислитель. В мультипроцессоре также присутствует специализированный вид памяти(shared) быстрого доступа, расположенной на кристалле. Задача разбивается на участки (Warp), равные кол-ву SP в MP. Переключение потоков идет Warp'ами. Количество мультипроцессоров определяется моделью используемого GPU. Таким образом, внутренняя архитектура вычислительных блоков GPU напоминает архитектуру специализированных процессоров, ориентированных на параллельные вычисления.

Как говорилось выше, используется расширенная версия языка Си. По большей части, расширения представляют собой набор дополнительных директив, основные конструкции языка не меняются. Это достаточно сильно облегчает программирование, так как недостатком многих параллельных вычислительных систем является сложность модели их программирования и множество специфических конструкций языка.

CUDA обладает очень хорошей масштабируемостью под задачу – достаточно указать размерность, а выполнение и переключение потоков берет на себя оборудование. В случае приобретения нового, более производительного GPU тот же код будет задействовать все возможные ресурсы без каких-либо вмешательств со стороны программиста. Прирост производительности в этом случае будет равняться приросту количества вычислительных единиц. Размерность задачи может быть указана множеством разных путей, исходя из программной модели вычислителя. Самым высоким уровнем является сетка(Grid), это двумерный массив. Его ширина и длина равна 65535. Далее, в каждом элементе этого массива находится блок(Block), максимальной размерностью 512*512*64, а в каждом блоке набор нитей(Threads) из 512 штук. Для определения того, какая именно часть задачи выполняется, введены специальные переменные, отражающие местонахождение потока в многомерной структуре.

При реализации алгоритмов с большим количеством ветвлений, и при интенсивной работе с большими объёмами данных в оперативной памяти, эффективность CUDA снижается. В первом случае это происходит из-за того, что ветвление внутри Warp'а приводит к многократному просчету всех возможных вариантов для данного Warp'а (Один Warp это один MP с SIMD архитектурой), и в результате получается «последовательное» выполнение. Во втором случае снижение эффективности обусловлено медленной DRAM, в отличие от, например, Shared-памяти.

Этапы дискретного логарифмирования. Рассмотрим, какие этапы дискретного логарифмирования можно эффективно реализовать с помощью CUDA. Дискретное логарифмирование на эллиптической кривой осуществляется с помощью методов встречи посередине и встречи на случайном дереве. Для обоих методов критическим ресурсом является не мощность процессора, а объём оперативной памяти [2]. Так как объём памяти видеокарты меньше, чем у компьютера общего назначения, а скорость обмена между видеокартой и оперативной памятью относительно невелика, то эффективность реализации подобных алгоритмов на CUDA на данный момент представляется невысокой.

Вычисление логарифма в мультипликативной группе F_p^* обычно выполняется с помощью субэкспоненциальных методов, например, решета числового поля. Эти методы содержат два вычислительно-сложных этапа – нахождения достаточного количества гладких чисел (просеивание) и построение с помощью этих соотношений уравнения, из которого и можно найти искомый логарифм. Второй этап вычислений обычно требует большого количества вычислений над элементами матрицы, которую необходимо разместить в оперативной памяти. Поэтому обычно эти вычисления эффективно проводятся на многоядерных вычислительных машинах, обладающих большим объёмом оперативной памяти.

Вычисления отдельных элементов на этапе просеивания независимы, и вычислительная сложность этого этапа при большом размере модуля значительно превышает сложность этапа обработки матрицы. Поэтому имеет смысл рассмотреть возможность ускорения вычислений на этапе просеивания с применением параллельных вычислений по технологии CUDA

Просеивание. Задача этапа просеивания состоит в нахождении достаточно большого количества чисел, гладких по базису из простых чисел для метода базы разложения, и гладких по базисам из простых чисел и простых идеалов для метода "решета" числового поля. Число называют гладким по заданному базису, если в его разложении на простые числа присутствуют только элементы базиса. После нахождения гладкого числа его нужно разложить на элементы базиса в виде (1) для метода базы разложения или в виде (2,3) для метода "решета" числового поля.

$$\prod_{i=1}^{n} p_i^{e_i} = a^{e_{n+1}} b^{e_{n+2}}, \tag{1}$$

$$\prod_{i=1}^{n_1} p_i^{e_i} = (c + dm) (\mathrm{mod}\, p), \tag{2}$$

$$\prod_{i=n_1+1}^{n_2} q_i^{e_i} = (c + d\alpha). \tag{3}$$

В формулах (1–3) a – основание логарифма, b – степень, p_i – элементы базиса из простых чисел, q_i – элементы базиса из простых идеалов в расширенном поле. На этапе просеивания отдельные кандидаты проверяются независимо, поэтому нет

необходимости в хранении большого объёма данных в оперативной памяти. Рассмотрим, как можно выполнить проверку чисел на гладкость.

Проверку числа на гладкость можно осуществить с помощью пробного деления, при этом в случае успеха сразу получается разложение числа – показатели при элементах базиса в формуле (1). Однако данный способ является не очень быстрым из-за необходимости осуществлять целочисленное деление длинных чисел — достаточно медленную операцию.

Существует более быстрый способ проверки числа на гладкость. На этапе инициализации вычисляется проверочное число checker по формуле (4).

$$chec\ker = \prod_{i=2}^{n} p_i^{\left\lfloor \frac{\log p}{\log p_i} \right\rfloor}. \qquad (4)$$

Первый элемент базиса пропускается, так как это -1. Затем при проверке гладкости числа-кандидата cand производится вычисление НОД(checker,cand). Если НОД(checker,cand)=cand, то данное число является гладким. Действительно, в разложение checker (4) входят все элементы базиса с максимально возможными для данного числа (cand < p) степенями, поэтому, если в разложение cand входят только элементы базиса, то НОД(checker, cand)=cand.

Реализация арифметических операций на видеокарте. Для вычисления НОД используется бинарный алгоритм. Этот алгоритм замечателен тем, что использует только операции сравнения, вычитания и деления на два, которые очень легко реализовать на вычислительной машине даже для длинных чисел. Данный способ проверки на гладкость рекомендуется в [4]. Эксперименты, проведённые авторами, показали, что скорость просеивания с помощью НОД превосходит скорость просеивания пробным делением в 4 раза уже при длине модуля p, равной 75 битам [1], и преимущество возрастает при увеличении длины модуля, так как доля гладких чисел падает.

Для первых экспериментов был выбран метод базы разложения, так как организация просеивания в нём проще, чем в методе решета числового поля. Фактически, для генерации чисел-кандидатов необходимо реализовать лишь умножение по модулю (и производную операцию возведения в степень) для получения правой части выражения (1). Рассмотрим особенности реализации для видеокарт операций умножения по модулю и нахождения НОД.

Длинная арифметика в экспериментальных алгоритмах и программах реализована следующим образом. Число представляется в виде одномерного массива, первый элемент которого хранит количество разрядов, а остальные элементы – само число, записанное в обратном порядке, по правилу – младший разряд, младший адрес. Основание системы счисления выбиралось как максимально возможное для целых размером 24 бита, так как в текущей версии CUDA работа с такими целыми осуществляется в несколько раз (до 5 раз на видеокарте GTX260) медленнее, чем с 32-разрядными целыми.

Особенностью реализации арифметических операций на CUDA было то, что для ускорения вычислений операнды, изначально находящиеся в памяти DRAM, перед началом вычислений размещались в shared-памяти. Shared-память находится непосредственно в процессоре видеокарты, и обращение к ней занимает намного меньше времени, чем обращение к DRAM [5]. Следовательно, когда большинство операций происходят в shared памяти, это значительно ускоряет процесс вычислений. Также, поскольку shared память выделяется на блок, и после вычислений данные теряются, то результат необходимо скопировать обратно в DRAM.

Ещё одна особенность заключается в том, что при проведении экспериментов в одном вычислительном блоке размещалось не множество нитей, а одна. При этом вычислительных блоков может быть много. Это обусловлено тем, что shared-память выделяется на один блок. Если использовать множество нитей, то не исключены коллизии при обращении к памяти, приводящие к падению производительности. Чтобы их избежать, было выбрано размещение по правилу один блок – одна нить.

Результаты экспериментов. Для проведения экспериментов авторами были разработаны программы, реализующие умножение, возведение в степень и нахождение НОД бинарным алгоритмом. Приведём результаты экспериментов для нахождения НОД, так как во-первых, эта операция является определяющей, а во-вторых, другие операции показали схожие результаты.

На рис. 1 показана зависимость времени вычислений от количества проверяемых чисел. На рис. 2 – зависимость времени вычислений от разрядности проверяемых чисел.

Рис. 1. Зависимость времени выполнения от числа кандидатов

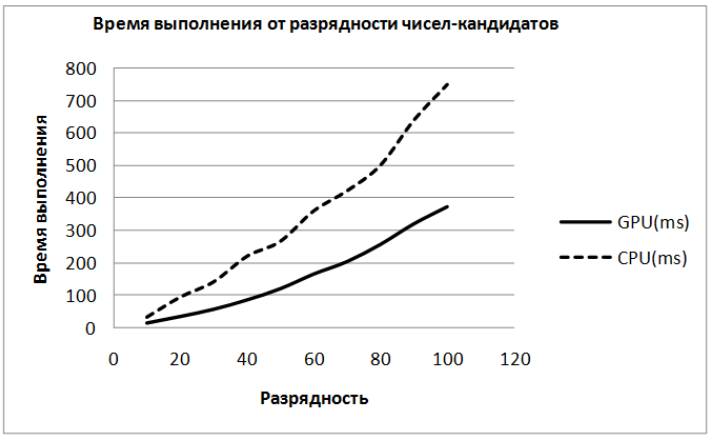

Рис. 2. Зависимость времени выполнения от разрядности чисел

Из графиков можно видеть, что скорость вычислений плавно уменьшается при росте размера числа и при увеличении количества проверяемых чисел. Также из графика видно, что производительность при использовании в данном эксперименте GPU GT250M возросла примерно в два раза по сравнению с процессором Intel Core 2 Duo 2 GHz.

Заключение. Проведённые эксперименты показывают, что технология CUDA может эффективно использоваться для решения определённых задач асимметричной криптографии. Представляется, что проведя оптимизацию разработанных программ, можно добиться более существенного прироста производительности. Также перспективным направлением исследования является интеграция приложений для CUDA с уже разработанными распределёнными программами для получения полноценного инструмента дискретного логарифмирования.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Бабенко Л.К., Сидоров И.Д.* Параллельный алгоритм дискретного логарифмирования методом решета числового поля // Известия ЮФУ. Технические науки. – 2008. – № 8 (85). – С. 199-203.
2. *Babenko L.K., Sidorov I.D.* Parallel algorithms for discrete log solving in GF(p) and elliptic curves // Proceedings of the Workshop on Computer Science and Information Technologies (CSIT'2008), Antalya, Turkey, September 15-17, 2008. Volume 1. Ufa State Aviation Technical University, 2008.
3. *Сидоров И.Д.* Анализ эффективности параллельных алгоритмов дискретного логарифмирования на эллиптической кривой // Молодежь и современные информационные технологии / Сб. трудов VII Всероссийской научно-практической конференции студентов, аспирантов и молодых ученых «Молодежь и современные информационные технологии». – Томск, 2009. – Ч. 1. – Томск: Изд-во СПБ Графикс.
4. *Ростовцев А.Г., Маховенко Е.Б.* Теоретическая криптография. – СПб.: АНО НПО «Профессионал», 2005. – 480 с.
5. NVIDIA CUDA — неграфические вычисления на графических процессорах. [Электронный ресурс]. – Режим доступа: http://www.ixbt.com/video3/cuda-1.shtml, свободный.

Бабенко Людмила Климентьевна
Технологический институт федерального государственного автономного образовательного учреждения высшего профессионального образования «Южный федеральный университет» в г. Таганроге.
E-mail: blk@fib.tsure.ru.
347928, г. Таганрог, ул. Чехова, 2, корпус "И".
Тел.: 88634312018.

Сидоров Игорь Дмитриевич
E-mail: idsidorov@gmail.com.

Кириллов Алексей Сергеевич
E-mail: LeX_90@bk.ru.

Babenko Lyudmila Klimentevna
Taganrog Institute of Technology – Federal State-Owned Autonomy Educational Establishment of Higher Vocational Education “Southern Federal University”.
E-mail: blk@fib.tsure.ru.
Block “I”, 2, Chehov street, Taganrog, 347928, Russia.
Phone: +78634312018.

Sidorov Igor Dmitrievich
E-mail: idsidorov@gmail.com.

Kirillov Alexey Sergeevich
E-mail: LeX_90@bk.ru.