Прогнозирование производительности при реализации алгоритмов генерации случайных последовательностей больших размерностей на реконфигурируемых архитектурах с сопроцессорами

Д.В. Быков, А.Д. Неретин

Генераторы случайных последовательностей — это важный элемент информационной безопасности. Они могут быть использованы как для генерации ключей, которыми шифруются данные, выполняется аутентификация и т.д., так и для безопасной очистки секторов диска при шифровании. Выделим два подхода к подобным генераторам [1]:

1. Малой размерности: для генерации малого количества ключей.

В данном подходе ключи должны обладать максимально возможным запасом стойкости, быть статистически неотличимыми от случайных, а так же необходимо, чтобы невозможно было алгоритмически предсказать сгенерированную последовательность даже по большому количеству ранее полученных чисел.

2. Большой размерности: для генерации большого количества ключей и для безопасной очистки диска.

В этом случае большее внимание уделяется достижению максимальной производительности: в основном используются генераторы псевдослучайных чисел, с последующим шифрованием по симметричному алгоритму, для достижения большей энтропии.

Поскольку, в настоящее время широко используются системы информационной безопасности, а так же учитывая, что часто используются ключи с коротким сроком действия, возникает необходимость генерировать огромный объем новых ключей, что занимает довольно продолжительное время. Поэтому особо остро возникает проблема повышения производительности генераторов случайных последовательностей большой

размерности, которую не могут обеспечить традиционные универсальные ЭВМ.

Для решения подобных проблем повышения производительности в настоящее время используются параллельные и/или конвейерные системы на базе сопроцессоров, в качестве которых могут использоваться графические процессоры (Graphics Processing Unit, GPU), либо программируемые логические интегральные схемы (ПЛИС или FPGA, Field Programmable Gateway Array).

До недавнего времени разработка под архитектуру ПЛИС была довольно трудоемкой задачей (даже по сравнению с разработкой под GPU), которая требовала дополнительных знаний, как в программировании, так и в схемотехнике, даже не смотря на текущие возможные средства разработки [2]. Однако, с выходом стандарта Open Computing Language 2.0 (OpenCL) в июле 2013 года, а так же добавлением его поддержки Altera Corporation (USA, San Jose) для работы с их платами, разработка приложений под ПЛИС становится крайне актуальной задачей. К тому же, как показано в [3, 4], использование OpenCL позволяет достичь значительного прироста производительности.

Генераторы случайных последовательностей большой размерности в основном используют довольно простые алгоритмы, получающие какое-либо начальное состояние (seed), из которого генерируется вся последовательность. Данные алгоритмы довольно просто распараллелить (достаточно с максимальной энтропией получать значения начального состояния для каждого потока), однако, из-за простоты алгоритма не всегда есть возможность создать полноценный конвейер, а поскольку разработка под ПЛИС по большей части предполагает именно конвейерную обработку данных, а не параллельную, необходимо получить предварительную оценку прироста производительности, чтобы удостоверится в целесообразности разработки [5].

Естественно, подобные методы предварительной оценки обладают теми или иными недостатками, такими как сложность и длительность расчетов,

низкая точность оценки, отсутствие учета особенностей архитектуры ПЛИС. Одной из методик, позволяющих быстро, просто и эффективно рассчитать оценку прироста производительности реализации алгоритмов на ПЛИС является Reconfigurable Amenability Test (RAT), предложенная в 2009 году B. Holland, K. Nagarajan и A. D. George в [6].

В RAT используются следующие критерии для определения целесообразности реализации алгоритма на ПЛИС: пропускную способность, числовую точность и необходимые ресурсы. Данные факторы являются доминирующими при оценке эффективности реализации алгоритмов на ПЛИС. Методология RAT предполагает, что первоначально определяется необходимая точность приложения с учётом типа и количества доступных ресурсов, затем проводится тест пропускной способности для получения прогнозируемой производительности алгоритма.

Прогнозируемое время выполнения алгоритма определяется исходя из двух показателей: времени обмена данными между CPU и сопроцессором и времени непосредственных вычислений. Время обмена данными учитывает объём данных и скорости чтения и записи, время непосредственных вычислений - опять же объём данных, количество операций над ними, частоту и количество одновременно выполняемых сопроцессором операций. Кроме того, при вычислении времени выполнения алгоритма предлагается учитывать использование одиночной или двойной буферизации. Двойная буферизация подразумевает одновременное осуществление передачи данных и непосредственных вычислений [7]. Более подробная информация о методике приведена в [6]. Ниже приведены основные расчетные формулы:

$$t_{comm} = t_{read} + t_{write},$$

$$t_{read} = \frac{N_{elements} * N_{bytes/element}}{\alpha_{read} * throughput_{ideal}},$$

$$t_{read} = \frac{N_{elements} * N_{bytes/element}}{\alpha_{write} * throughput_{ideal}},$$

где t_{comm} – время обмена данными между CPU и FPGA; t_{read} – время чтения данных с CPU; t_{write} – время записи данных на CPU; $N_{elements}$ – количество

передаваемых элементов; $N_{bytes/element}$ – размер элемента данных; $throughput_{ideal}$ – идеальная пропускная способность при чтении и записи; α_{read}, α_{write} – коэффициенты соотношения реальной и идеальной пропускной способности.

$$t_{comp} = \frac{N_{elements} * N_{operations/element}}{f_{clock} * throughput_{process}},$$

где t_{comp} – время непосредственных вычислений на ПЛИС; $N_{elements}$ – количество обрабатываемых элементов; $N_{operations/element}$ – количество операций над элементов данных, необходимое для достижения конечного результата; f_{clock} – тактовая частота; $troughput_{process}$ – количество операций за такт.

Рассмотрим аппаратные реализации алгоритмов генерации случайных последовательностей, предложенных Altera Corporation (USA, San Jose):

1. Linear Feedback Shift Register.
2. C Library Random Number Generator.
3. Race Condition-Based True Random Numbers.
4. Word Stream Scrambling.

Linear Feedback Shift Register (LFSR, Регистр сдвига с линейной обратной связью) – один из самых простых генераторов случайных чисел. Состоит всего из двух частей: функции битового сдвига (которая обеспечивает сдвиг регистров в ту или иную сторону) и обратной связи. Выходная последовательность зависит только он начального состояния и ассоциированного многочлена [8]. Ассоциированные многочлен изначально выбирается для генератора определенной размерности (списки рекомендованных значений можно найти в интернете). Достаточно просто реализуется аппаратно, и подобная реализация имеет высокую производительность. Однако сам по себе алгоритм плохо поддается конвейеризации.

C Library Random Number Generator (CLRNG, Генератор случайных чисел библиотеки C) – генератор случайных чисел, основанный на линейном конгруэнтном методе [9]. Так же довольно простой метод, в котором каждое следующее значение рассчитывается следующим образом:

$X_{n+1} = (a * X_n + c)\ mod\ m,$

где a, c и m – некоторые константы, к которым предъявлены определенные требования (список рекомендованных значений так же можно найти в интернете).

Исходя из формулы, очевидно, что выходная последовательность будет зависеть только от начального состояния генератора.

Для повышения производительности m обычно берут равной 2^{32}, 2^{64} и т.д. в зависимости от разрядности чисел, чтобы избежать операции деления.

Race Condition-Based True Random Numbers (RCBTRN, Генератор истинных случайных чисел на базе состояния гонки) – в отличии от большинства генераторов истинно случайных чисел для ПЛИС, использующих в своей основе несколько независимых сигналов кольцевого генератора, данный генератор, предложенный компанией Altera Corporation (USA, San Jose), построен на основе создания состояния гонки на нескольких несущих цепях.

Данный генератор состоит из трех основных модулей:

1. Схема гонки: задержка несущих цепей, для получения состояния гонки, осуществляется с помощью one-hot дешифратора, после чего защелками определяется с какой из цепей пришел первый сигнал (либо ни с одной, если время прибытия достаточно близко).

2. Конечный автомат регулировки: производит настройку one-hot дешифраторов. Для того чтобы в схеме гонки на выходе были именно случайные числа, необходимо подбирать нестабильные параметры для one-hot дешифратора.

3. Выходной фильтр фон Неймана: необходим для того, чтобы убрать наиболее вероятные результаты.

Подобный алгоритм куда более сложен и менее быстро действен, чем предыдущие два, однако, в отличие от них, он хорошо поддается

конвейеризации, а так же не требует дополнительного шифрования выходной последовательности.

Word Stream Scrambling (WSS, Скремблинг потока слов) – может быть использован как для генерации случайной последовательности из определенного потока данных, так и для шифрования этого потока. Поток данных поступает на сдвиговый регистр, а затем мультиплексируется в псевдослучайном порядке, чтобы получить искаженный поток данных, в котором каждое значение находится в пределах определенного смещения [10].

Изначально создается случайный шаблон перетасовки, который гарантирует, что данные не будут потеряны и/или дублированы. Для обеспечения периода в 64 цикла, данный шаблон записывается в 6-разрядную таблицу поиска. Это позволяет ускорить время выполнения за счет замены операций длительный вычислений на более быструю операцию поиска, при этом сохраняя достаточную энтропию.

Данный алгоритм довольно прост в реализации, однако, в таком виде использовать его для шифрования данных не рекомендуется. Несмотря на то, что в отличие от остальных алгоритмов, в данном алгоритме на ПЛИС записывается большой объем данных (чтобы получить определенное количество случайных чисел, необходимо передать такое же количество на ПЛИС), за счет того, что алгоритм очень хорошо конвейеризуется, достигается значительный прирост производительности. Так же стоит отметить, что запас стойкости данного алгоритма выше, чем у первых двух.

Поскольку для всех алгоритмов кроме Word Stream Scrambling записываемые данные очень малы (параметры инициализации, начальное значение и т.д.), время записи на ПЛИС можно считать равным 0. В соответствии с этим схемы взаимодействия CPU и ПЛИС принимают следующий вид (стандартный вид представлен в [6]):

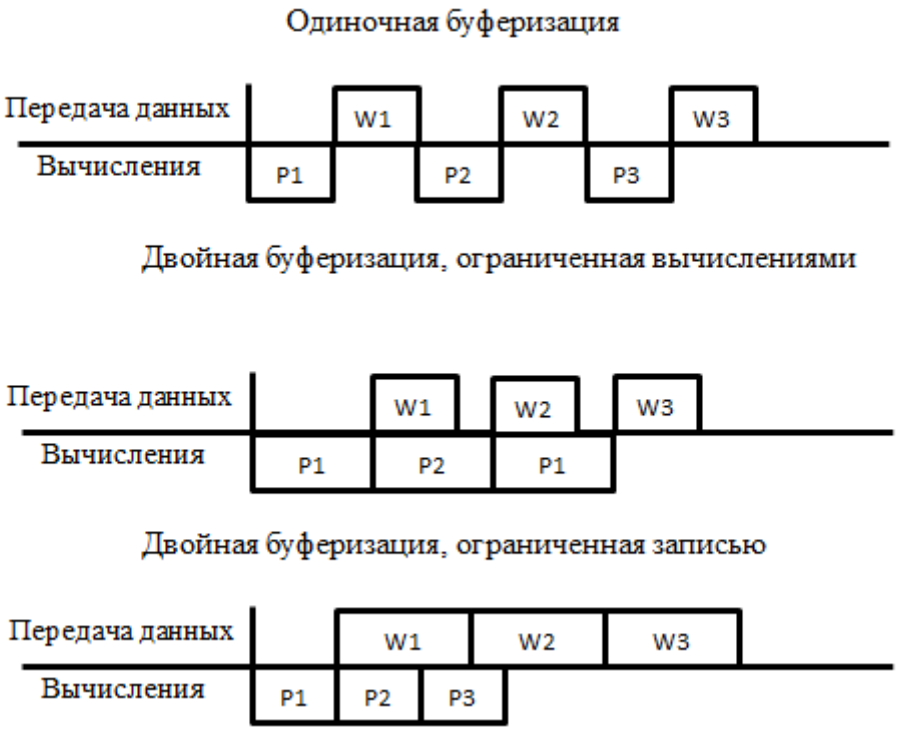

Рис. 1. – Схемы процесса взаимодействия CPU и ПЛИС.

Ниже в Таблице № 1 приведены результаты расчетов прогнозируемого ускорения. Расчет производился для плат Altera серии Cyclone V в сравнении с Intel Core i7 920 @ 2.67 GHz.

Таблица № 1

Прогнозируемое ускорение

Параметр	**LFSR**	**CLRNG**	**RCBTRN**	**WSS**
$N_{elements}$	100000000			
$N_{byte/elements}$, бит	32			
$throughput_{write}$, Гбит /с	6,144			
α_{write}	0,7			
f_{clock}, МГц	550			
$throughput_{process}$	32	255	68	505
$N_{elements/process}$	32	511	2176	1674

t_{comm}, с	0,73			1,46
t_{comp}, с	0,18	0,36	5,81	0,60
Время выполнения на CPU, с	2,78	4,17	75,61	10,68
Прогнозируемое ускорение (одиночная буферизация)	3,05	3,83	11,56	5,18
Прогнозируемое ускорение (двойная буферизация)	3,81	5,71	13,01	7,32

Из полученных результатов видно, что даже при реализации Linear Feedback Shift Register можно получить прирост производительности. Стоит учитывать, что во всех случаях кроме Race Condition-Based True Random Numbers, время записи с ПЛИС на CPU превышает время вычислений. Это связано с тем, что платы серии Cyclone V не поддерживают PCI Express третьего поколения. Соответственно, увеличение скорости передачи данных в 2 раза за счет использования PCI Express третьего поколения для этих трех алгоритмов при двойной буферизации повысит прирост производительности так же в 2 раза.

Однако для Linear Feedback Shift Register даже в этом случае время вычислений останется довольно низкой по сравнению с передачей данных, к тому же прирост ускорения все равно останется достаточно низким. Исходя из выше сказанного, можно сделать вывод, что реализация Linear Feedback Shift Register на ПЛИС для генерации большого количества случайных чисел и передачи их на CPU не представляется целесообразной.

В Таблице № 2 приведены результаты симуляции трех оставшихся алгоритмов в Quartus II 13.1 Web Edition:

Таблица № 2

Результаты симуляции

	CLRNG	**RCBTRN**	**WSS**
Время выполнения, с	0,48	6,15	0,59
Ускорение	3,45	11,00	5,21
Разница с оценкой, %	11,00	5,10	0,60

Результаты, полученные в результате симуляции, оказались достаточно близкими к теоретически рассчитанным значениям прироста производительности, поэтому можно сделать вывод, что данные алгоритмы целесообразно реализовать под ПЛИС для генерации большого количества случайных чисел.

Литература:

1. Иванов, М.А. Теория, применение и оценка качества генераторов псевдослучайных последовательностей [Текст] / М.А. Иванов, И. В. Чугунков. – М.: КУДИЦ-ОБРАЗ, 2003. – 240 с.

2. Андреев, А.А. Методика выбора базовой архитектуры реконфигурируемой вычислительной системы на основе методов теоретико-игровой оптимизации [Электронный ресурс] / А.А. Андреев // «Инженерный вестник Дона», 2013, № 1. – Режим доступа: http://www.ivdon.ru/magazine/archive/n1y2013/1569 (доступ свободный) – Загл. с экрана. – Яз. рус.

3. Берилло, А.Б. Тестирование производительности в OpenCL [Электронный ресурс] / А.Б. Берилло, – Режим доступа: http://www.ixbt.com/video3/opencl_bench.shtml (доступ свободный) – Загл. с экрана. – Яз. рус.

4. Блохин, О.Д. Исследование высокопроизводительного решения задачи N тел на базе платформы OpenCL [Электронный ресурс] / О.Д. Блохин, Д.К. Боголепов, М.М. Захаров, Д.П. Сопин. – 2010. – С. 34-37. – Режим доступа: http://www.itlab.unn.ru/archive/MSConf10/msconf-2010_book.pdf (доступ свободный) – Загл. с экрана. – Яз. рус.

5. Димаки, А.В. Аппаратно-программный генератор случайных чисел, сопрягаемый с компьютером типа IBM PC [Текст] / А.В. Димаки, А.А. Светлаков // «Известия томского политехнического университета», 2004, № 1. – 144-148 с.

6. Holland B. RAT: RC Amenability Test for Rapid Performance Prediction / Brian Holland, Karthik Nagarajan, Alan D. George. – URL: http://www.cse.sc.edu/~jbakos/reading_list/trets_rat_final.pdf.

7. Андреев, А. Е. Прогнозирование производительности при реализации алгоритмов на гибридных архитектурах с сопроцессорами [Электронный ресурс] / А. Е. Андреев, И. М. Силкин, Ю. В. Шафран // Журнал «Современные проблемы науки и образования», 2012, №3. – Режим доступа: http://www.science-education.ru/103-6389 (доступ свободный) – Загл. с экрана. – Яз. рус.

8. Кузьмин, Е.В. Параметризованная модель генератора псевдослучайных последовательностей в OrCAD [Электронный ресурс] / Е.В. Кузьмин, Ф.Г. Зограф // «Инженерный вестник Дона», 2013, № 3. – Режим доступа: http://www.ivdon.ru/magazine/archive/n3y2013/1766 (доступ свободный) – Загл. с экрана. – Яз. рус.

9. Chung-Chih Li Using Linear Congruential Generators for Cryptographic Purposes / Chung-Chih Li. – URL: http://pdf.aminer.org/000/077/798/using_linear_congruential_generators_for_cryptographic_purposes.pdf

10. Каляев, И.А. Реконфигурируемые мультиконвейерные вычислительные структуры [Текст] / И.А. Каляев, И.И. Левин, Е.А. Семерников, В.И. Шмойлов. – М.:ЮНЦ РАН, 2008. – 320 с.