УДК 004.056.55

Н.А. ПОЛУЯНЕНКО, А.В. ПОТИЙ, д-р техн. наук

ИСПОЛЬЗОВАНИЕ ТЕХНОЛОГИЙ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ В ГРАФИЧЕСКИХ ПРОЦЕССОРАХ ДЛЯ ГЕНЕРАТОРОВ ПОТОКОВОГО ШИФРОВАНИЯ

Введение

Практически все программно-аппаратные методы защиты информации неразрывно связаны с криптографией [1]. Криптография существует уже много веков, однако в том виде, в котором мы ее знаем и используем сейчас, она насчитывает несколько десятилетий. Можно сказать, что основным (хотя и не единственным) направлением развития современной криптографии является создание стойких алгоритмов шифрования, которое неразрывно связано с вычислительными мощностями.

В последние несколько лет перспективным направлением увеличения вычислительной мощности компьютерных систем стал перенос вычислений с центрального процессорного устройства (CPU – от англ. central processing unit) на графические процессоры (GPU – от англ. Graphics processing units). Эту концепцию поддержали основные производители графических ускорителей, в частности компания NVIDIA.

В настоящий момент процессоры, используемые в компьютерах общего назначения, подошли к пределу тактовой частоты – около 3 ГГц. Поэтому дальнейший рост производительности осуществляется путем распараллеливания [2]. Средства параллельного выполнения процессоров от Intel и AMD включают в себя наборы команд для выполнения векторных операций, а также реализацию на одном кристалле нескольких процессорных ядер – реальных или виртуальных (с использованием HyperThreading). Однако возможности по распараллеливанию традиционных процессоров ограничены. Этому мешают их изначально (начиная с 8086) последовательная архитектура и достаточно сложный набор команд, требующий, для быстрого выполнения реализации, достаточно сложных элементов процессора – конвейера, системы прогнозирования переходов, кешей и т. п.

Архитектура видеокарт NVIDIA, позволяющая использовать их для неграфических вычислений, получила название CUDA (от англ. Common Unified Device Architecture – общая унифицированная архитектура устройства), а разработанная для нее спецификация [3] облегчила процесс создания алгоритмов, работающих на GPU. Технология CUDA была впервые представлена компанией NVIDIA в 2007 г. [4]. CUDA может использоваться на GPU производства NVIDIA, таких как GeForce, Quadro, Tesla. Последняя линейка устройств (Tesla) вообще не имеют видеовыхода и предназначены исключительно для высокопроизводительных вычислений.

NVIDIA распространяет ряд примеров, демонстрирующих решение некоторых задач из различных областей математики, физики и информатики. Прирост производительности в этих задачах при переходе от CPU к GPU составляет от 10 до 100 раз.

Поскольку графическая карта – это отдельный вычислительный модуль с собственным вычислительным узлом и оперативной памятью, то в программе для CUDA код разделяется на две части: host – выполняется на CPU; device – выполняется на GPU.

Параллелизм в CUDA обеспечивается следующим образом: при запуске кода на выполнение создается не один, а группа потоков – рис. 1. Вся группа потоков, созданных при запуске кода, называется сеткой (grid). Элементами сетки являются блоки (block). Они формируют одномерную или двухмерную структуру. Каждый блок, в свою очередь, состоит из группы нитей (thread), объединенных в одномерный, двумерный или трехмерный массив. Число блоков в сетке и нитей в блоке задается при старте кода.

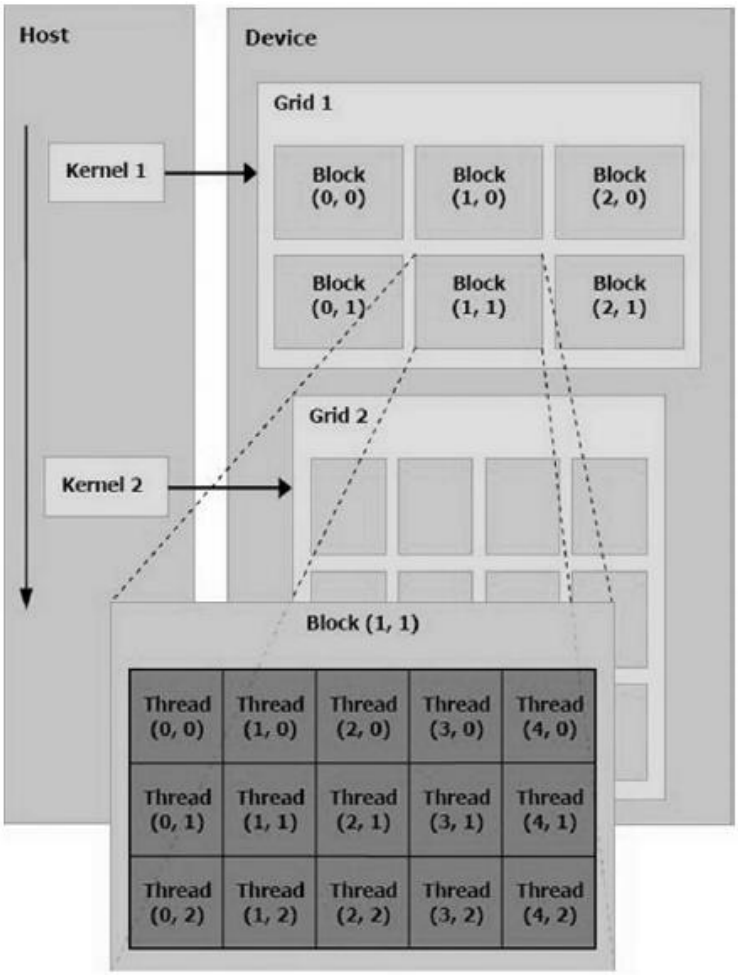

Рис. 1. Структура организации потоков на GPU

Параллельная программа в модели передачи сообщений представляет собой набор обычных последовательных программ, которые обрабатываются одновременно. Обычно каждая из этих последовательных программ выполняется на своем процессоре и имеет доступ к своей, локальной памяти [5]. Явным достоинством при такой организации вычислений является возможность написания и отладки программы на однопроцессорной системе.

Для достижения высокой эффективности при массивно-параллельных вычислениях, нужно учитывать множество факторов: архитектурные особенности GPU, быстродействие и порядок доступа к памяти, механизмы синхронизации между вычислительными потоками. Важную роль играет также приспособленность самого алгоритма к исполнению в параллельном режиме. Перенос простых арифметических операций на GPU дает выигрыш в производительности по сравнению с CPU при использовании даже самых простых моделей графических адаптеров [6].

Архитектура CPU позволяет использовать практически любые шаблоны параллельных вычислений, в то время как архитектура CUDA в каждый определенный момент времени позволяет выполнять на всех своих ядрах только одну инструкцию [7]. Поэтому архитектура CUDA может эффективно применяться только при вычислениях с большим параллелизмом и интенсивной арифметикой. Все функции, выполнимые на GPU, не поддерживают рекурсии и имеют некоторые другие ограничения, которых нет в архитектуре CPU.

Современные GPU предоставляют выгодное соотношение цены, производительности и энергопотребления. Многие суперкомпьютерные кластеры имеют высокий рейтинг в ТОП500, благодаря использованию GPU [8]. Однако, будучи специализированными устройствами, рассчитанными на потоковую обработку однотипных данных, GPU на многих

алгоритмах не показывают значимого преимущества перед процессорами традиционной архитектуры [9]. Это обусловлено тем, что современные GPU используют SIMD-архитектуру [10] и специально рассчитанные на работу с ней контроллеры памяти [11]. GPU являются «массивно-параллельными» процессорами, т.е. вычислительными устройствами, на которых одновременно выполняется большое (по сравнению с CPU) количество вычислительных потоков.

В [12] проводится сравнение эффективности CPU и GPU реализаций некоторых комбинаторных алгоритмов, используемых в криптоанализе. Показывается, что применение специальных техник трансформации потока управления позволяет существенно компенсировать потери производительности, возникающие из-за неэффективного исполнения условных переходов на SIMD-устройстве. Однако ограничения, которые накладывают механизмы работы с памятью, применяемые в современных GPU, для рассматриваемых алгоритмов оказываются непреодолимыми.

Постановка задачи

Цель данной работы – изучение возможности применения GPU для систем потокового шифрования, в частности исследование возможностей переноса части вычислений по генерации псевдослучайной последовательности с CPU на GPU и сравнение производительности полученных решений. Под производительностью будем понимать количество генерируемых бит в единицу времени.

Для оценки возможностей GPU использовались регистры сдвига с нелинейной обратной связью генерирующие последовательность максимального периода (M-PCHOC) взятые из алгоритма, представленного в 2008 году на международном конкурсе eStream Achterbahn-128/80 version 1.1 [13].

Следует учитывать, что потоковый шифр Achterbahn в первую очередь разрабатывался для аппаратной реализации. Некоторые его возможности, такие как применение многократной реализации функции обратной связи с использованием параллельной реализации PCHOC, позволяющее увеличить количество генерируемых бит за такт, нами не рассматривались.

Функции обратных связей для всех 13 M-PCHOC (A_0, …, A_{12}), которые используются в потоковом шифре Achterbahn-128/80 были выполнены в соответствии с оптимизированной реализацией, приведенной в [14]. Функции обратных связей для каждых из 13 регистров имеют одинаковую логическую глубину [12], что обеспечивает примерно равную скорость генерации каждым регистром.

Кроме M-PCHOC, взятых из конструкции потокового шифра Achterbahn-128/80, был также исследован M-PCHOC найденный нами, и имеющий нелинейность второго порядка. Функция обратной связи, используемая в нашем M-PCHOC второго порядка, задается следующим соотношением:

$$A_{30} = x_{30} + x_{28} + x_{27} + x_{26} + x_{25} + x_{22} + x_{21} + x_{17} + \\ + x_{15} + x_{12} + x_{11} + x_4 + x_{27} \cdot x_{28} + x_{27} \cdot x_{29},$$

где x_i – значение x_i ячейки в регистре. Отсчет идет от ячейки, в которой помещается новое значение (определяемое функцией обратной связи A_{30}) и начинается с номера 1, и заканчивается последней (30 ячейкой), значение которой является выходным битом в следующей итерации.

Для компиляции программного кода использовалось программное обеспечение Microsoft Visual Studio 2015 и программно-аппаратная платформа для организации параллельных вычислений на графических процессорах NVIDIA CUDA 8.0. Вычисления проводились на персональном компьютере под управлением 64-разрядной Windows 7 SP 1, с процессором Intel Core i5-3210M CPU 2,5GHz и видеопроцессором NVIDIA GeForce GT 630M.

Полученные результаты

Вначале приведем результаты измерения скорости генерации М-РСНОС на CPU и GPU. В табл. 1 приведено время (t_{GB}), затраченное на генерацию 1 Гбайта (8 589 934 592 итераций) данных каждым из М-РСНОС A_0, …, A_{12} как по отдельности, так и для всех вместе.

Таблица 1

Временные затраты на генерацию 1 Гбайта на CPU (t_{GB}^C) и GPU (t_{GB}^G)

М-РСНОС	A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}	A_0...A_{12}
t_{GB}^C (с.)	107,4	112,5	118,1	115,6	117,7	114,2	115,3	118,3	128,4	130,5	140,9	106,8	133,8	133,8
t_{GB}^G (с.)	2408	2457	2310	2719	2621	2736	2506	2834	2556	2589	2687	2556	2294	26033

Необходимо оговорить тот факт, что здесь и далее под генерацией мы понимаем холостой прогон всех регистров необходимое число раз без полной передачи или записи полученных значений. При реальной генерации необходимо учитывать передачу генерируемых значений, на что также будет затрачено определенное время.

Как видим, благодаря оптимизированной реализации, время выполнения каждого из регистров как на CPU так и на GPU примерно равно (разброс не более 14 % для CPU и 11 % для GPU). Таким образом, если мы будем выполнять на GPU каждый из РСНОС параллельно в отдельном потоке, мы будем получать выходной бит без существенного простоя какого-либо из регистров.

В табл. 2 приведены результаты испытаний для одновременного расчета итераций с A_0 по A_{12} регистр. Так как в A_{12} используется регистр, состоящий из 33 ячеек, то для универсальности во всех регистрах применялись 64 битные переменные, с которыми GPU работает медленнее чем с 32 битными. Вычисления проводились в один поток. Каждый регистр

вычислялся в своей нити, т.е. всего для расчета одного набора из 13 регистров запускалось 13 нитей. Общее число используемых нитей обозначим символом n_{trd}.

Таблица 2

Временные затраты на генерацию 1 Гбайта на GPU при одновременной генерации нескольких М-РСНОС

n_{trd}		13	26	39	52	65	78	91	104	117	130	143	156	169	182
t_{GB}^G (с.)	один блок	26033	26476	26509	26509	26689	29474	29540	30113	30572	30867	34258	34373	34603	34865
	два блока	26476	26492	26492	26492	26705	29720	29605	30244	30523	30900	34717	34504	34799	34881

Как видим из табл. 2? с ростом числа одновременно вычисляемых регистров время, затраченное на проведения расчетов для каждой из итерации, растет по нелинейному закону. В связи с чем, более целесообразно рассматривать производительность генерации τ, которую определим как количество значений, генерируемых одновременно регистрами во всех потоках в единицу времени:

$$\tau = \frac{n_{trd}}{t_{GB}} \left[\frac{\text{Гбайт}}{\text{сек}} \right].$$

Более наглядно результаты табл. 2 отображены на графике рис. 2. Как видим, с ростом числа одновременно просчитываемых регистров производительность достигает своего предельного значения и дальнейшее увеличения числа одновременно рассчитываемых регистров не дает прироста производительности.

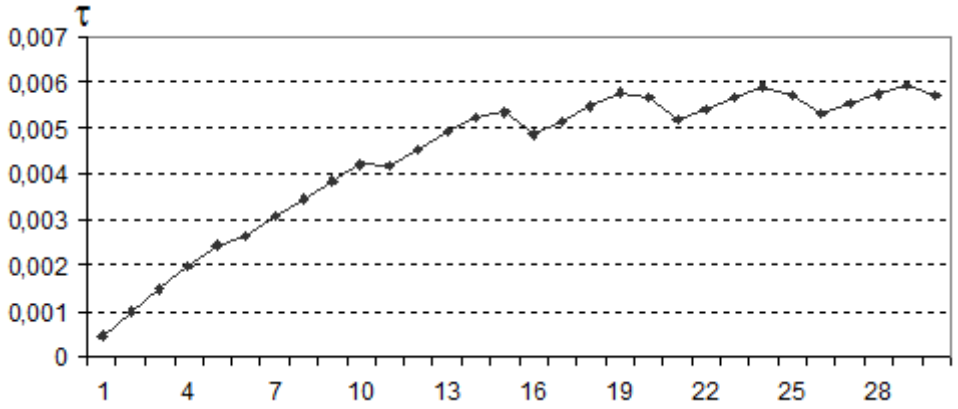

Рис. 2. Зависимость производительности τ генерации на GPU от числа одновременно рассчитываемых регистров с A_0 по A_{12}

На рис. 3 и 4 отображены результаты производительности при различном количестве задействованных блоков и нитей в каждом блоке для 32-битных и 64-битных переменных, используемых в качестве регистров.

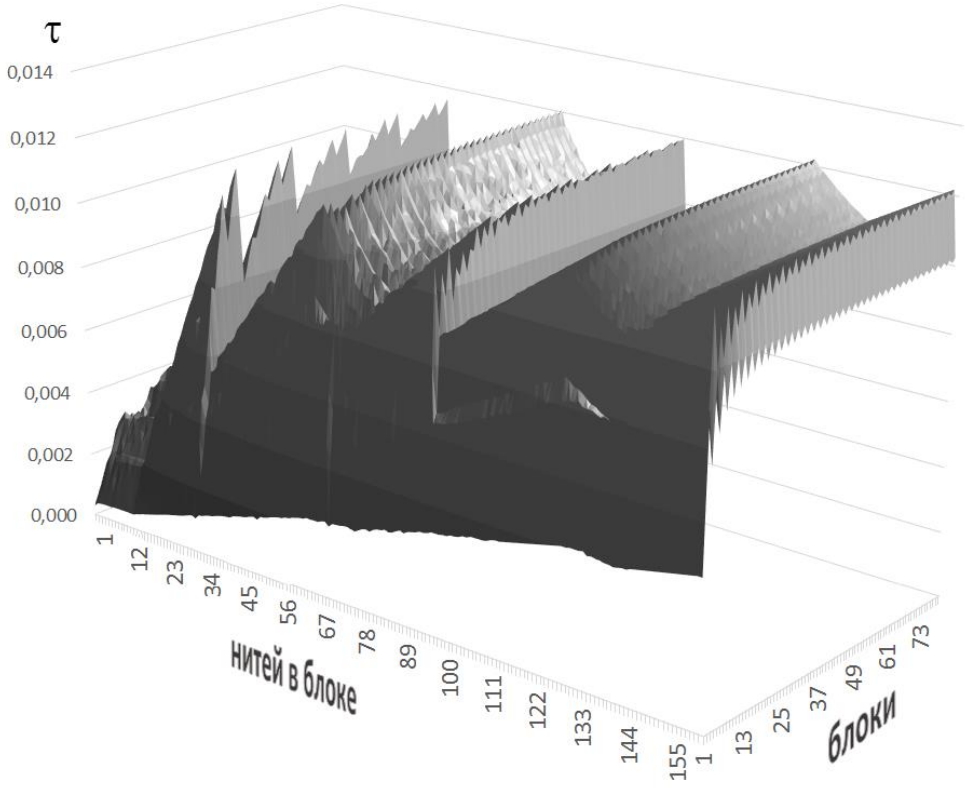

Рис. 3. Зависимость производительности τ генерации на GPU от количества блоков и нитей в блоке при размерности регистров в 64 бита

Как видим, на GPU генерация при размере используемых переменных в 64 бита значительно медленнее чем в 32 бита.

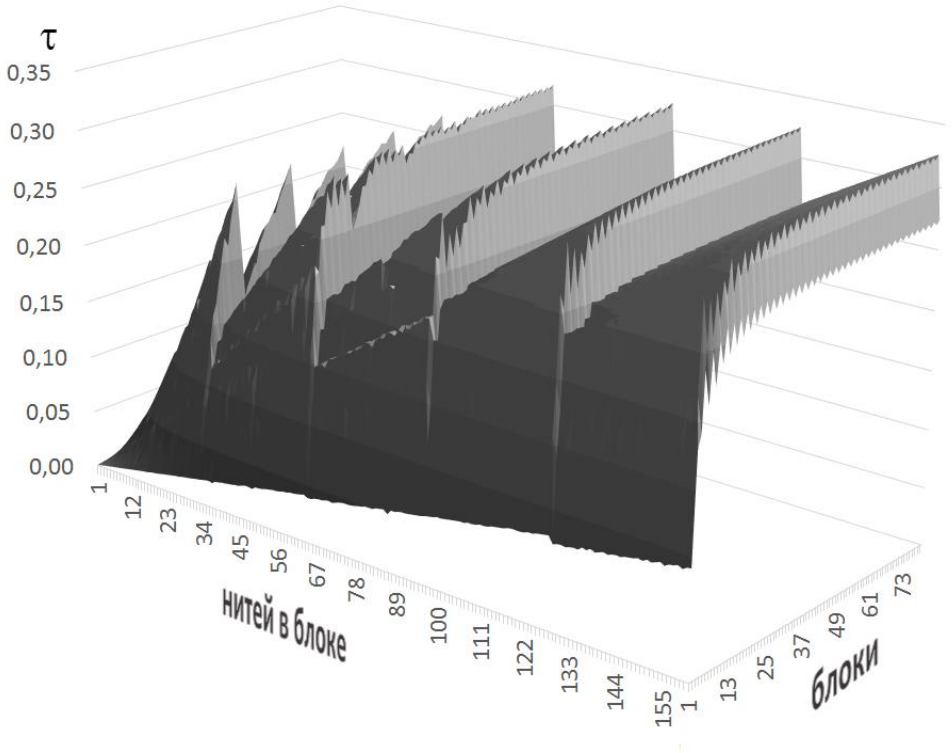

Рис. 4. Зависимость производительности τ генерации на GPU от количества блоков и нитей в блоке при размерности регистров в 32 бита

Оптимальное значение производительности соответствует 16 блокам и количеством нитей в блоке кратным 32. На рис. 5 приведен график зависимости производительности M-PCHOC A_{11} и A_{30} для 32-битной конфигурации при 16 блоках в зависимости от количество нитей, запускаемых в блоке.

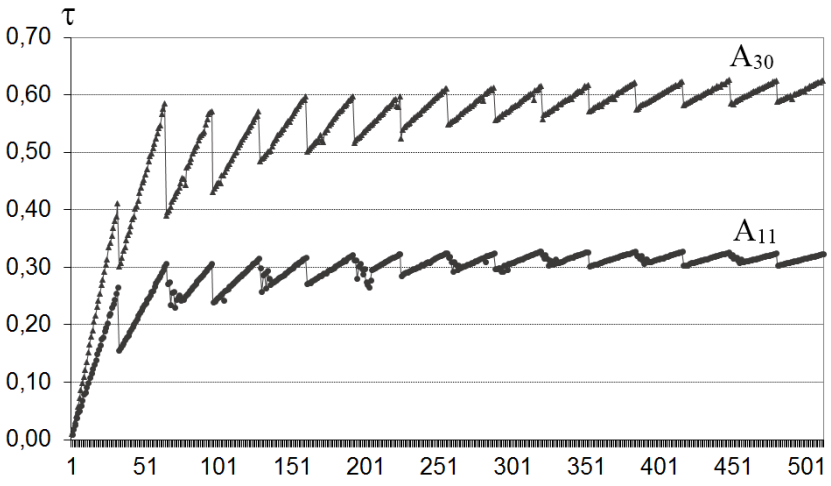

Рис. 5. Производительность τ генерации на GPU в зависимости от числа нитей в блоке при 16 блоках

Максимальное значение для A_{11} достигается в 384 нитей и соответствует генерации со скоростью 0,328 Гбайт/с (что эквивалентно генерации 1 Гбайта за 3 сек.). Как видим, максимальные значения образуются при использовании числа нитей кратных 32, что

соответствует размеру варпа (warp) – количеству физически одновременно выполняемых нитей [15].

Использование вместо одного потока (Stream) двух или четырех не увеличивает общую максимальную производительность.

При использовании для генерации только CPU, генерация производилась со скоростью 0,0119 Гбайт/с (эквивалентно генерации 1 Гбайта за 84 сек.). Таким образом, при использовании GPU, производительность увеличилась в 28 раз.

Стоит отметить, что при использовании машинно-ориентированного языка низкого уровня (assembler) возможно повысить скорость генерации РСНОС на CPU. Как отмечалось в [16] возможно получить скорость генерации 1 Гбайта за несколько десятков секунд, в зависимости от вида образующего полинома. Перевод части кода на GPU в PTX Assembler не показал заметного прироста производительности.

При замене М-РСНОС из алгоритма Achterbahn полиномом с нелинейностью второго порядка – A_{30}, и равных прочих условий, средняя скорость генерации на CPU составила 0,015 Гбайт/с (эквивалентно генерации 1 Гбайта за 65,8 сек.). При GPU – 0,626 Гбайт/с (эквивалентно генерации 1 Гбайта за 1,6 сек.).

Таким образом при использовании GPU и М-РСНОС второго порядка, имеющего более простую, с точки зрения вычислительных затрат структуру, был получен прирост производительности более чем в 41 раз.

Выводы

С помощью переноса части вычислений с CPU на GPU возможно существенное увеличение скорости генерации псевдослучайной последовательности. В ходе вычислительных экспериментов прирост производительности составил до 2756 % на полиноме A_{11} потокового шифра Achterbahn-128 и до 4173 % на М-РСНОС, имеющем более простую структуру.

Список литературы: 1. *Бабенко, Л. К., Ищукова, Е. А., Сидоров, И. Д.* Применение параллельных вычислений при решении задач защиты информации // Программные системы: теория и приложения. – 2013. – Т. 4, № 3(17). – С. 25–42. [Электронный ресурс]. – Режим доступа: http://psta.psiras.ru/read/psta2013_3_25-42. 2. *Бабенко, Л. К., Ищукова, Е. А., Сидоров, И. Д.* Параллельные алгоритмы для решения задач защиты информации. – 2-е изд., стереотип. – Москва : Горячая линия–Телеком, 2014. – 304 с. 3.*Что такое CUDA*? [Электронный ресурс]. – Режим доступа: http://www.nvidia.ru/object/what_is_cuda_new_ru.html. 4. *Сандерс, Д., Кэндрот, Э.* Технология CUDA в примерах. Введение в программирование графических процессоров. – ДМК Пресс, 2011. 5. *Немнюгин, С., Стесик, О.* Параллельное программирование для многопроцессорных вычислительных систем. – СПб : БХВ-Петербург, 2002. 6. *Верещак, М.И., Неласая, А.В.* Особенности реализации библиотеки арифметики произвольной точности на графических ускорителях для криптографических приложений // Системи обробки інформації. – 2011. – Вип. 7 (97). – С. 55-59. [Электронный ресурс]. – Режим доступа: http://www.hups.mil.gov.ua/periodic-app/article/9025/soi_2011_7_16.pdf 7. *Файзуллин, Р.Т., Свенч, А.А., Хныкин, И.Г.* Применение гибридной суперкомпьютерной системы в задачах криптоанализа // Доклады ТУСУРа. – 2010. – № 1 (21), ч. 1. – С. 61-63 [Электронный ресурс]. – Режим доступа: http://old.tusur.ru/filearchive/reports-magazine/2010-1/61-63.pdf 8. *TOP500* Supercomputer Site URL: http://www.top500.org 9. *Lee, V.W.* Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU / V.W. Lee et al // ACM SIGARCH Computer Architecture News. – ACM, 2010. – Vol. 38, No. 3. – P. 451–460. DOI: 10.1145/1816038.1816021 10. *Flynn, M.* Some computer organizations and their effectiveness / M. Flynn // Computers, IEEE Transactions on. – 1972. – Vol. 100, No. 9. – P. 948–960. DOI: 10.1109/tc.1972.5009071 11. *CUDA C* Best Practices Guide – CUDA SDK v.6.0 – NVIDIA corp. – 2014. URL: http://docs.nvidia.com/cuda 12. *Булавинцев, В.Г.* Сравнение эффективности CPU и GPU реализаций некоторых комбинаторных алгоритмов на задачах обращения криптографических функций // Вестник ЮУрГУ. Серия «Вычислительная математика и информатика». – 2015. – Т. 4, №. 3. – С. 67–84. DOI: 10.14529/cmse150306 [Электронный ресурс]. – Режим доступа: https://vestnik.susu.ru/cmi/article/viewFile/3412/3398 13. *Gammel, B., Gottfert, R., Kniffler, O* Achterbahn-128/80 [Электронный ресурс]. – Режим доступу: http://www.matpack.de/achterbahn/Gammel_Goettfert_Kniffler_Achterbahn-128-80.pdf 14. *Achterbahn* Stream Cipher Site [Электронный ресурс]. – Режим доступу: http://www.matpack.de/achterbahn/achterbahn-128-80.zip 15. *Казеннов, А.М.* Основы технологии CUDA и OpenCL. – Москва :

Учебное пособие, 2013. – 66 с. 16. *Полуяненко, Н.А., Потий, А.В.* Сравнение объема ансамбля М-РСЛОС и М-РСНОС, скорости генерации на их основе, для GF(2) и в расширениях поля GF(22) // Радиотехника. – 2016. – Вып. 186. – С. 153-159.

Харьковский национальный
университет имени В. Н. Каразина

Поступила в редколлегию 02.04.2017