УДК 004.021
DOI: 10.12737/22100

А.А. Колпаков, Ю.А. Кропотов

РАСШИРЕННЫЙ АЛГОРИТМ МИКШИРОВАНИЯ АУДИОПОТОКОВ ДЛЯ МНОГОПРОЦЕССОРНЫХ УСТРОЙСТВ В ТЕЛЕКОММУНИКАЦИЯХ

Представлен многопотоковый алгоритм аудиомикширования для вычислений на графических картах, который для уменьшения времени работы использует двухпроходной рендеринг. Показано, что разработанный алгоритм дает существенный прирост производительности по сравнению с традиционными, использующими центральные процессоры.

Ключевые слова: GPGPU, параллельные вычисления, гетерогенные вычислительные системы, графические процессоры, микширование аудиоданных.

A.A. Kolpakov, Yu.A. Kropotov

EXTENDED ALGORITHM OF AUDIO STREAMS MIXING FOR MULTIPROCESSOR DEVICES IN TELECOMMUNICATIONS

In this paper is presented an extended algorithm of audio streams mixing for computations on graphic processors which combines many stages of mixing by means of two-pass rendering use which reduces considerably switching time between buffers.

By the method of experimental computer comparative researches was carried out the assessment of the algorithm developed productivity. The results of investigations have shown that the application of the algorithm developed results in computation productivity increase up to six times.

Key words: GPGPPU, parallel computations, heterogeneous computer systems, graphic processors, audio data mixing.

Введение

С развитием современных компьютерных сетей такие мультимедиа-Internet-приложения, как удаленная работа по сети, совместные разработки и видеоконференции, прочно вошли в повседневную жизнь. Согласно исследованиям [1], 75% межпользовательского общения по сети приходится на голосовой трафик. Поскольку голосовое общение играет такую важную роль, нельзя исключать и голосовые конференции с несколькими участниками.

Простейший сценарий организации голосовой коммуникации заключается в том, что каждый источник звука шлет свой аудиопоток каждому приемнику независимо. Такой метод прост и удобен, но требует высокой пропускной способности сети, что не всегда можно обеспечить. Поэтому лучшим методом может служить аудиомикширование, что означает комбинирование аудиопотоков от всех источников в один. Исходя из возможности звуковых волн накладываться друг на друга данный метод может обеспечивать приемлемое качество звука, при этом обеспечивая снижение загруженности сети. Однако применение данного метода может существенно увеличить нагрузку на центральный процессор сервера, что может негативно сказаться на производительность системы в целом. Выходом из ситуации может стать применение графических процессоров для решения данной задачи.

Проблемы использования графических процессоров для аудиомикширования

Хотя графические процессоры достаточно производительны, существует несколько проблем в использовании их для аудиомикширования, которые связаны с архитектурой и ограниченностью функционала [2].

Первая проблема – это пропускная способность шины между графическим процессором и основной памятью, которая меньше, чем между основным процессором и основной памятью. Например, чипсет Intel 975X обеспечивает теоретическую

пропускную способность для CPU 10,7 ГБ/с, а для GPU - только 8 ГБ/с. Практика показывает, что отсутствие поддержки асинхронного ввода/вывода требует больших временных затрат для дополнительных операций, таких как блокировка/разблокировка буфера. Поскольку общие вычисления на GPU базируются на 3D-рендеринге, скорость записи обычно выше, чем скорость чтения. Такая асимметрия делает процедуру считывания результата достаточно продолжительной [3; 4].

Во-вторых, общие вычисления на GPU базируются на 3D-моделях; различные задачи требуют различных настроек GPU, таких как 3D-модели, трансформирующие матрицы и программы шейдеров. Во время загрузки настроек вычислительные потоки GPU не задействованы. Хуже всего то, что GPU не сообщает CPU о завершении выполнения задания, поэтому CPU вынужден периодически проверять статус GPU. Это довольно времязатратная операция, так как она нарушает параллелизм между GPU и CPU.

В-третьих, недостатком GPU является производительность в логических операциях. Как известно, CPU отслеживает ветвления, GPU же работает по-другому: каждая ветка ветвления сначала выполняется, а потом уже выбирается нужный результат. Это делает распараллеливание легче, но требует больше ресурсов.

И наконец, набор инструкций GPU несовместим с CPU. Кроме того, время выполнения и длина кода лимитированы. Все это делает сложным перенос существующих алгоритмов на графические процессоры.

Структура разрабатываемого алгоритма

Базовый алгоритм аудиомикширования состоит из пяти этапов. Первый шаг – это суммирование аудиосемплов от различных источников, что может быть представлено следующей формулой:

$$\overrightarrow{M}_t = \sum_{k=0}^{n} \vec{u}_{k,t},$$

где $\vec{u}_{k,t}$ – вектор семпла, т.е. вектор отсчетов, полученных микрофоном k за время t; $\overrightarrow{M}_t$ – итоговый вектор микширования.

Второй этап – это эхокомпенсация, которая в базовом виде заключается в исключении семпла i-го устройства из итогового вектора. Данный этап представляется в виде

$$\overrightarrow{M}_{i,t} = \overrightarrow{M}_t - \vec{u}_{i,t},$$

где $\overrightarrow{M}_{i,t}$ – итоговый вектор микширования для i-го устройства; $\vec{u}_{i,t}$ – семпл i-го устройства.

Для корректности итогового вектора $\overrightarrow{M}_{i,t}$ необходимо, чтобы его размерность была равна размерности входящих векторов. Однако после проведения этапов 1 и 2 вектор $\overrightarrow{M}_{i,t}$ может быть переполнен, что приведет к нежелательным шумам. Для того, чтобы использовать изначально вектор $\overrightarrow{M}_{i,t}$ больших размеров, необходимо проводить его сжатие для дальнейшего использования. Это делается на третьем этапе по формуле

$$\overrightarrow{M}_{i,t} = \overrightarrow{M}_{i,t} \times frac_{i,t},$$

где $frac_{i,t}$ – коэффициент ослабления для i-го устройства. Этот коэффициент необходимо вычислять автоматически, исходя из максимального сжатого семпла. Поиск максимума среди сжатых семплов происходит на четвертом этапе, а корректировка коэффициента ослабления – на пятом.

Блок-схема базового алгоритма микширования представлена на рис. 1.

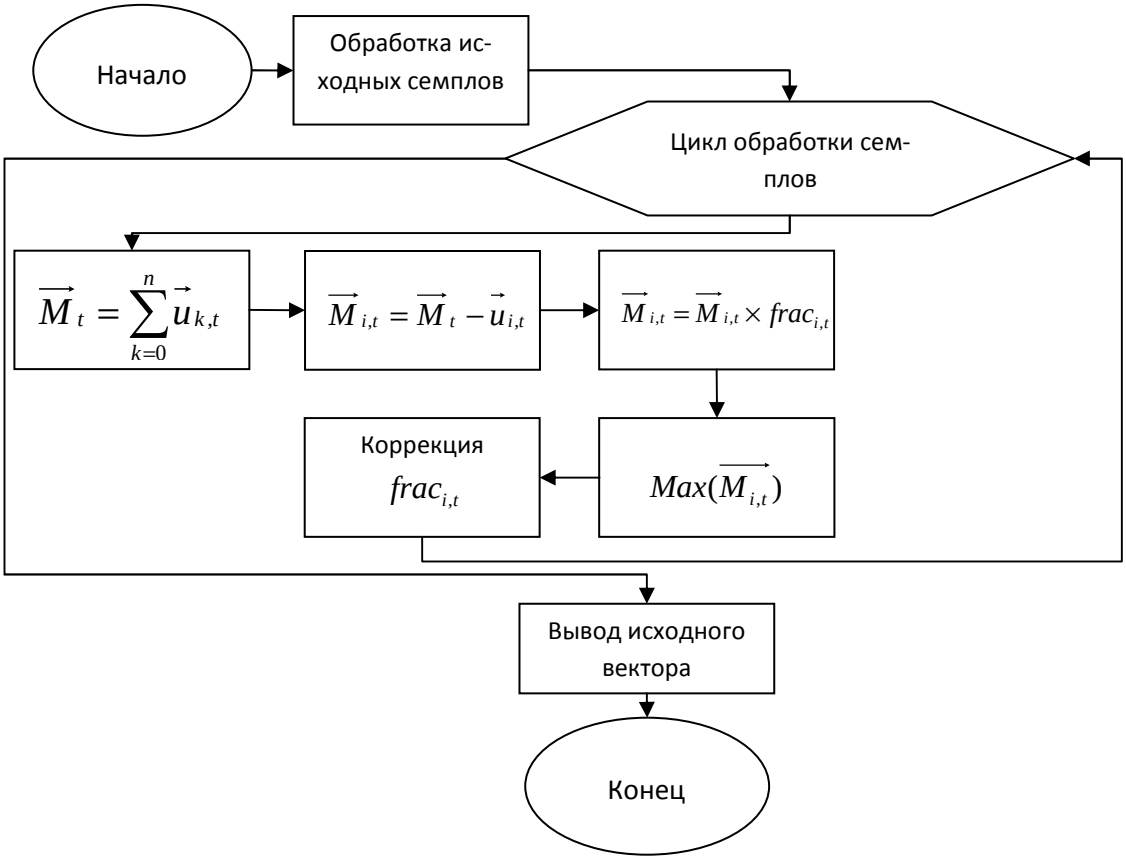

Рис. 1. Блок-схема базового алгоритма микширования

Для лучшего описания алгоритма возможности GPU представляются в виде f – функции текстуры X и координат пикселя p_i:

$$f(X,P) = \{y_i \mid \forall p_i \in P_i y_i = f(X, p_i)\}, \quad (1)$$

где y_i – проекция пикселя p_i на текстуру X, P – проекция 3D-модели.

Для каждого пикселя в проекции 3D-модели P будет рассчитана функция (1), а затем результат будет записан в буфер рендера.

Из формулы (1) видно, что образец вычислений на GPU может быть представлен как $A=(X,P,f)$.

Пусть n обозначает общее количество источников звука в сессии, а L – длину одного аудиосемпла. Каждый из трех первых шагов микширования (накопление семплов, эхокомпенсация и сжатие) выдает n последовательностей по L байт. В то же время два последних шага (поиск максимального семпла и адаптация коэффициента затухания) выдают только n целых чисел. Поскольку первые три шага могут быть выполнены в рамках одной проекции для вычислений на GPU, они могут быть скомбинированы в один шаг:

$$\vec{m}_i = \left(\sum_{j=0}^{n-1} \vec{u}_j - \vec{u}_i \right) \times frac_i . \quad (2)$$

Поскольку четвертый шаг использует другое измерение, отличное от первых трех, он не может быть объединен в рамках формулы (2). Блок-схема расширенного алгоритма микширования представлена на рис. 2.

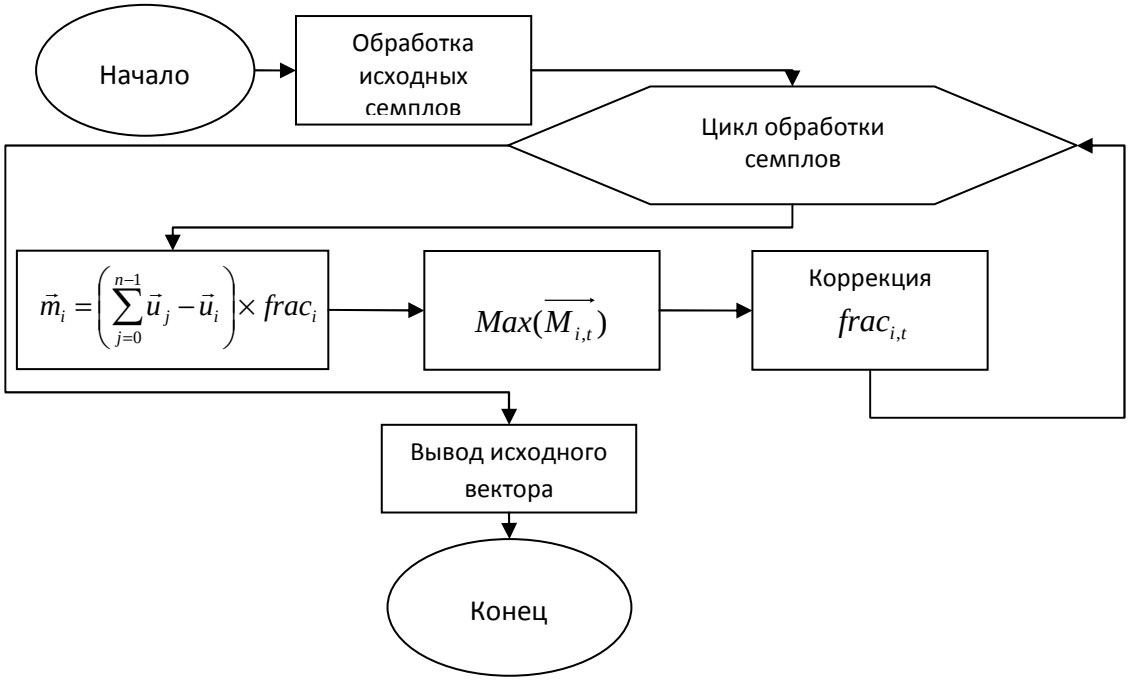

Рис. 2. Блок-схема расширенного алгоритма микширования

Обычно вычисления, проекции которых не совпадают и не имеют пересечений, могут быть объединены по общим признакам. Поэтому если эти вычисления обозначить как *A1=(Xl,PI,f1)* и *A2=(X2,P2,f2)*, то их можно объединить следующим образом:

$$X = \langle X_1, X_2 \rangle,$$
$$P = P_1 \cup P_2, \tag{3}$$
$$f(\langle X_1, X_2 \rangle, p) = \begin{cases} f_1(X_1, p), p \in P_1, \\ f_2(X_2, p), p \in P_2. \end{cases}$$

Как видно из (3), основным алгоритмом является проверка координаты каждого пикселя для выбора функции выполнения модуля. Так как GPU выполняет все ветви до выбора нужной, каждая ветвь будет выполнена для каждого пикселя, что займет много времени.

В разрабатываемом алгоритме представлен альтернативный метод выполнения большого количества функций, который выводит в один буфер рендера выходные данные различной длины с помощью многопотокового рендеринга. Здесь основным правилом является перемещение проекции путем модифицирования проекционной матрицы, чтобы вычислительная площадь каждой функции ограничивалась необходимой областью, а не всем буфером.

В зависимости от пикселя на него может приходиться разный объем информации без потери универсальности. Обозначим через w общее число пикселей, необходимое для хранения *L* байт. Таким образом, буфер рендера может быть представлен как (w+1)·n. 3D-модель разрабатываемого алгоритма представляет собой прямоугольник, который лежит на плоскости *Z*. Он имеет размеры: 2w/(w+1) единиц по ширине и 2 единицы по высоте. Координаты вершин: (-1,-1,0), (1-2/(*w*+1),-1,0), (-1,1,0), (1-2/(*w*+1),1,0).

При первом проходе рендеринга в качестве матрицы проецирования выбирается единичная матрица. Это обеспечивает проекции совпадение с 3D-моделью. После преобразования видимой области, в этом проходе для выполнения первых трех шагов микширования применяется формула (2). Преобразования 3D-модели, производимые в первых трех шагах, представлены на рис. 3.

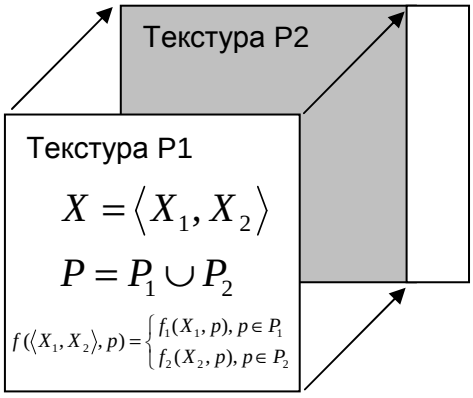

Рис. 3. Первый проход алгоритма

Во втором проходе рендеринга проекция смещается влево на *L* пикселей. В то же время программа шейдера переключается в режим поиска максимального семпла. В этом проходе может быть записана только одна колонка, поскольку большинство частей проекции лежат вне буфера рендера и будут автоматически игнорированы GPU. Так как отсечение проекции было выполнено в начале рендера, этот метод вызывает функцию только для корректных пикселей, а не для всего буфера. Действия, производимые на втором проходе алгоритма, представлены на рис. 4.

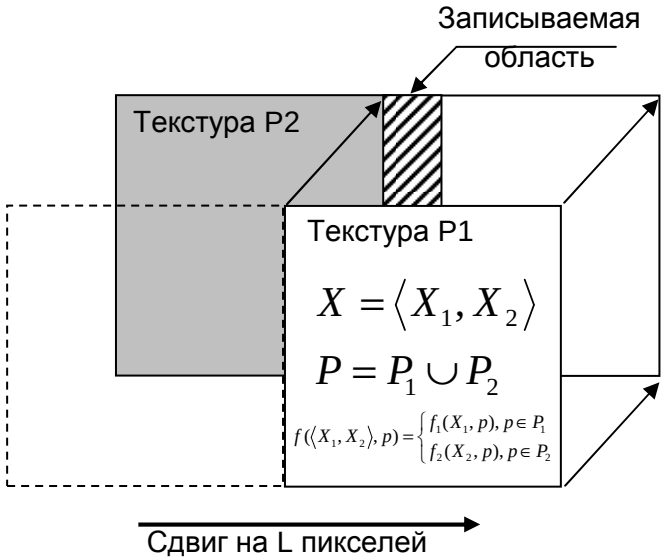

Рис. 4. Второй проход алгоритма

Использование алгоритма с одной текстурой

Как замечено выше, оба прохода алгоритма в качестве входных данных требуют *n* последовательностей семплов каждый. Для каждой последовательности должна быть выделена своя уникальная текстура. Всего требуется *n* текстур размерностью *L*. Такая многотекстурная технология не подходит для аудиомикширования. Во-первых, для каждого источника звука требуется своя текстура. Поэтому могут потребоваться дополнительные проходы. Во-вторых, загрузка множества маленьких текстур гораздо медленнее, чем загрузка одной большой.

В разрабатываемом алгоритме предлагается загрузка единой текстуры. В качестве примера представлен первый проход алгоритма. Входные данные содержат

n последовательностей семплов размером *L* байт и n коэффициентов затухания. Используются два текстурных буфера формата RGBA. Текстура T1 имеет размерность [*L*/4] n и хранит все семплы последовательностей в линию. Текстура T2 имеет размерность 1n и хранит коэффициенты затухания. Координаты всех текстур приведены в табл. 1.

Таблица 1

Координаты текстур, используемых в разработанном алгоритме

Вершины	Координаты текстуры T1	Координаты текстуры T2
(-1, -1, 0)	(1/2w, 1)	(0.5, 1)
(1-2/(w+1), -1, 0)	(1, 1)	(0.5, 1)
(-1, 1, 0)	(1/2w, 0)	(0.5, 0)
(1-2/(w+1), 1, 0)	(1, 0)	(0.5, 0)

Аудиомикширование производится независимо для каждого пикселя. Процесс микширования для пикселя с координатами (*x*,*y*) в виде псевдокода:

```
    float2 ptCur=pt.t0;
for(int
y=0;y<TOTAL_SOURCE;y++)
if(y!=pt.v.Y)
{
ptCur.y=(Y+0.5)/TOTAL_SOURCE;
iSum + = de-
codePCMU(tex2D(s[0],
ptCur)*256);
}
int4 w = tex2D(s[1],
pt.t1)*256;
iSum=iSum * w.b/16;
```

Текстурные координаты пикселя (*x*,*y*) рассчитываются путем интерполяции текстурных координат вершин. Согласно табл. 1, текстурная координата для T1 или pt.t0 должна быть (X,Y), а для T2 или pt.t1 – (0.5,Y). Pt.t0 указывает на семпл, для которого производятся текущие преобразования микширования. Данный семпл назовем «точкой прицеливания». Для исключения появления эха другая текстурная координата ptCur ограничена по доступу T1 вместо pt.t0. Составляющая *x* текстуры ptCur идентична текстуре pt.t0, а составляющая *y* рассчитывается из циклической переменной, которая обозначает каждый источник звука. В цикле регистр положения *v* используется, чтобы пропустить «точку прицеливания». Наконец, коэффициенты затухания считываются из текстуры pt.t1. Поскольку коэффициент хранится в первом байте, выборка производится только по синей составляющей.

Экспериментальное исследование разработанного алгоритма

Тестовые входные данные для микширования представляют собой последовательности по 320 семплов. Все семплы сгенерированы случайно. В качестве тестового стенда использован компьютер с процессором Intel Core i3-4130 (4 ГБ оперативной памяти, графическая карта NVIDIA GeForce GT730). Варьировалось количество последовательностей *M*. Результаты для базового и разработанного алгоритмов на выходе идентичны. Результаты экспериментального исследования приведены на рис. 5.

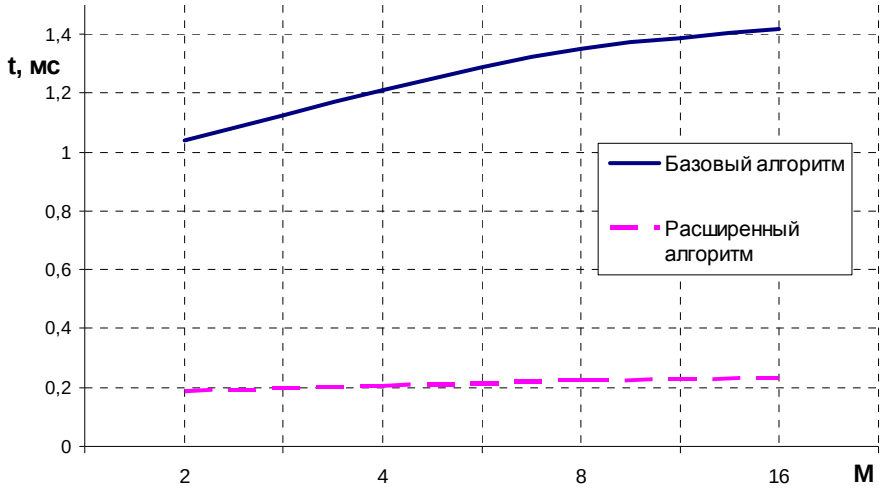

Рис. 5. Результаты экспериментального исследования расширенного алгоритма аудио микширования

Как видно из результатов тестирования, представленных на рис. 5, применение расширенного алгоритма аудиомикширования позволяет увеличить производительность компьютерной системы в 5-6 раз.

Время работы алгоритма *t* и дисперсия результатов для 8 последовательностей семплов приведены в табл. 2.

Таблица 2

Результаты исследования расширенного алгоритма микширования аудиопотоков для 8 последовательностей семплов

Применяемый алгоритм	Время работы алгоритма, мс	Дисперсия результатов
Базовый	1,351	3,757
Разработанный	$2{,}226 \cdot 10^{-1}$	$1{,}426 \cdot 10^{-3}$

Как видно из результатов тестирования, представленных в табл. 2, применение разработанного алгоритма в гетерогенной компьютерной системе уменьшает время на обработку данных с $1{,}351 \cdot 10^{-3}$ с – время обработки данных базовым алгоритмом до $0{,}2226 \cdot 10^{-3}$ с.

Заключение

В данной работе представлен расширенный алгоритм микширования аудиопотоков для вычислений на графических процессорах. Главное его достоинство – это комбинирование множества этапов микширования путем использования двухпроходного рендеринга, что существенно снижает время переключения между буферами. Использование для расчетов одной текстуры повышает эффективность операций ввода/вывода. Хотя операции ввода/вывода занимают приблизительно половину времени вычислений, экспериментальные исследования разработанного алгоритма показали увеличение производительности до 6 раз.

СПИСОК ЛИТЕРАТУРЫ

1. Lindholm, E. NVIDIA Tesla: A unified graphics and computing architecture / E. Lindholm, J. Nickolls, S. Oberman, J. Montrym. IEEE Micro. – 2008. – 28(2). – С. 39-55.

2. Luebke, D. GPGPU: general purpose computation on graphics hardware / D. Luebke, M. Harris, J. Kr'uger, T. Purcell, N. Govindaraju, I. Buck, C. Woolley, A. Lefohn // In SIGGRAPH '04: ACM

SIGGRAPH 2004 Course Notes, New York, USA. – 2004. – С. 33.
3. Колпаков, А.А. Оптимизация генетических алгоритмов при использовании вычислений на графических процессорах на примере задачи нулевых битовых векторов / А.А. Колпаков // Информационные системы и технологии. - 2013. – №2(76). – С. 22-28.
4. Колпаков, А.А. Теоретическая оценка роста производительности вычислительной системы при использовании нескольких вычислительных устройств / А.А. Колпаков // В мире научных открытий. - 2012. – №1. – С. 206-209.

1. Lindholm, E. NVIDIA Tesla: A unified graphics and computing architecture / E. Lindholm, J. Nickolls, S. Oberman, J. Montrym. IEEE Micro. – 2008. – 28(2). – С. 39-55.
2. Luebke, D. GPGPU: general purpose computation on graphics hardware / D. Luebke, M. Harris, J. Kr'uger, T. Purcell, N. Govindaraju, I. Buck, C. Woolley, A. Lefohn // In SIGGRAPH '04: ACM SIGGRAPH 2004 Course Notes, New York, USA. – 2004. – С. 33.
3. Kolpakov, A.A. Optimization of genetic algorithms using computations on graphic processors by example of problem of zero bit vectors / A.A. Kolpakov // *Information Systems in Technology*. - 2013. – №2(76). – pp. 22-28.
4. Kolpakov, A.A. Theoretical assessment of increase in efficiency of computation system at use of some computation devices / A.A. Kolpakov // *In the World of Scientific Discoveries.* - 2012. – №1. – pp. 206-209.

Статья поступила в редакцию 10.03.16.
Рецензент: д.т.н., профессор кафедры Владимирского государственного университета им. А.Г. Столетова и Н.Г. Столетова Орлов А.А.

Сведения об авторах:

Колпаков Александр Анатольевич, ст. преподаватель Муромского института (филиала) Владимирского государственного университета имени Александра Григорьевича и Николая Григорьевича Столетовых, e-mail: desT.087@gmail.com.

Кропотов Юрий Анатольевич, д.т.н., профессор Муромского института (филиала) Владимирского государственного университета имени Александра Григорьевича и Николая Григорьевича Столетовых, e-mail: kaf-eivt@yandex.ru.

Kolpakov Alexander Anatolievich, Senior lecturer of Murom Institute (Branch) of Stoletovs State University of Vladimir, e-mail: desT.087@gmail.com.

Kropotov Yury Anatolievich, D.Eng., Prof. of Murom Institute (Branch) of Stoletovs State University of Vladimir, e-mail: kaf-eivt@yandex.ru.