На правах рукописи

Адинец Андрей Викторович

Высокоуровневая система программирования графических процессорных устройств

05.13.11 – Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

АВТОРЕФЕРАТ
диссертации на соискание учёной степени
кандидата физико-математических наук

Москва – 2009

Работа выполнена в научно-исследовательском вычислительном центре МГУ имени М.В. Ломоносова.

Научный руководитель:	д.ф.-м.н., чл.-корр. РАН, Воеводин Владимир Валентинович
Официальные оппоненты:	д.ф.-м.н. профессор, Машечкин Игорь Валерьевич
	д.т.н., профессор, Гергель Виктор Павлович
Ведущая организация:	Институт прикладной математики имени М.В. Келдыша РАН

Защита состоится 23 октября 2009 года в 15 часов на заседании диссертационного совета Д501.002.09 Московского государственного университета имени М.В. Ломоносова, расположенного по адресу: 119992, г. Москва, ул. Ленинские горы, д. 1, стр. 4, НИВЦ МГУ, конференц-зал

С диссертацией можно ознакомиться в библиотеке НИВЦ МГУ.

Автореферат разослан «_____» _____ 2009 г.

Учёный секретарь
диссертационного совета, Суворов В.В.

Общая характеристика работы

Актуальность работы. Современные графические процессорные устройства (ГПУ, GPU, Graphics Processing Unit) являются высокопроизводительными вычислительными системами, потенциально сравнимыми с суперкомпьютерами в решении представительных классов вычислительно ёмких задач. По ряду характеристик системы, построенные на базе графических процессоров, превосходят традиционные суперкомпьютеры: более высокая энергоэффективность, лучшее соотношение цена/производительность, меньшие размеры, пониженные требования к инженерной инфраструктуре, более высокая распространённость и доступность. Однако главной проблемой, препятствующей массовому внедрению ГПУ в вычислительную практику, является отсутствие адекватных высокоуровневых средств программирования для подобных архитектур.

Активные исследования по разработке методов решения вычислительных задач на графических процессорах начались не так давно, в начале 2000-х годов. Они велись по ряду направлений, в частности, адаптация существующих программ к использованию вычислительных ресурсов графических процессоров, исследование производительности графических процессорных устройств на определённых классах задач, разработка средств и систем программирования графических процессоров.

Сегодня для решения задач на графических процессорах от различных производителей (прежде всего, NVidia и AMD) используется целый ряд средств программирования, в частности, OpenGL, CUDA, RapidMind, Brook+. Однако все они обладают целым рядом недостатков. Все они являются низкоуровневыми средствами, и их использование требует знания многих деталей архитектуры графического процессора. Более того, подобные средства программирования работают только с конкретными семействами архитектур. В результате создаваемые с их помощью программы являются низкоуровневыми, трудными для сопровождения и непереносимыми. Необходим высокоуров-

невый инструментарий для программирования графических процессоров, который позволит создавать эффективные программы, переносимые между семействами архитектур графических процессоров с сохранением высокого уровня эффективности.

Целями данной диссертационной работы являются исследование подходов к созданию высокоуровневых средств программирования графических процессоров, обоснование принципов построения высокоуровневого языка программирования ГПУ, разработка архитектуры и системы программирования, опирающейся на предложенный язык и учитывающей особенности различных семейств ГПУ, разработка методов оптимизации программ для учёта иерархии памяти и уровней параллелизма различных графических процессоров.

Научная новизна диссертации состоит, во-первых, в разработке принципов построения и реализации высокоуровневого языка программирования C$ для решения вычислительно ёмких задач на графических процессорах, и, во-вторых, в разработке методов оптимизации программ для семейства архитектур графических процессоров. Разработанный язык обладает свойством высокоуровневости. Создаваемые с его помощью программы обладают свойствами переносимости с сохранением эффективности и простоты интеграции в уже существующие приложения.

Практическая значимость. Применимость и востребованность результатов данной диссертационной работы определяется несколькими факторами. Графические процессоры быстро развиваются, предлагая экономически выгодные средства решения вычислительно сложных задач. Развитие компьютерных методов исследований в таких областях, как вычислительная астрофизика, финансовое моделирование, биоинженерия и в ряде других ведёт к росту числа приложений, которые могут эффективно выполняться на графических процессорах. При этом архитектуры графических процессоров эволюционируют, их число растет, что делает создание программ для каждой из архитектур ГПУ крайне трудоёмкой задачей.

Разработанная в рамках данной диссертационной работы система C$ отражает высокоуровневые особенности графических процессоров, которые остаются неизменными уже в течение достаточно длительного времени, например, параллельную обработку массивов данных. При этом низкоуровневые особенности архитектур ГПУ, меняющиеся между различными поколениями ГПУ, скрыты от пользователя. В совокупности это обеспечивает эффективный перенос программ с платформы на платформу, что показали проведённые расчёты на большом числе современных ГПУ.

Все исследования, выполненные в рамках данной работы, ориентированы на применение технологий организации высокопроизводительных вычислений на графических процессорах для решения реальных вычислительно сложных задач из различных областей науки.

На защиту выносятся следующие основные результаты и положения:

1. На основании сравнительного анализа архитектуры современных ГПУ и технологий параллельного программирования разработан высокоуровневый язык C$ для программирования графических процессоров. Программы на языке C$ обладают свойством переносимости между различными семействами ГПУ с сохранением высокой эффективности.

2. Для предложенного языка спроектирована и реализована система поддержки времени выполнения. Система обеспечивает единое внутреннее представление программы, её оптимизацию, генерацию и исполнение кода для различных архитектур графических процессоров (NVidia и AMD).

3. Разработаны и интегрированы в систему программирования C$ методы оптимизации программ для графических процессоров. Разработанные методы оптимизации учитывают иерархию памяти и уровни параллелизма различных графических процессоров.

4. Разработанная система программирования успешно прошла апробацию при решении вычислительно сложных задач линейной алгебры, финансового моделирования, астрофизики и ряда других с использованием большого числа различных архитектур графических процессоров.

Апробация работы. Основные положения работы прошли апробацию на научных семинарах НИВЦ МГУ. Результаты работы представлялись на международной конференции "SYRCoSE'07" (г. Москва, июнь 2007 г.), на научной конференции "Ломоносовские чтения-2008" (Москва, апрель 2008 г.), на научной конференции "Ломоносовские чтения-2009" (Москва, апрель 2009 г.), всероссийской научной конференции "Научный сервис в сети Интернет: многоядерный компьютерный мир" (г. Новороссийск, сентябрь 2007 г.), всероссийской суперкомпьютерной конференции "Научный сервис в сети Интернет: решение больших задач" (г. Новороссийск, 2008 г.), международной научной конференции "Параллельные вычислительные технологии-2009" (г. Нижний Новгород, март-апрель 2009 г.).

Публикации. По теме диссертации опубликовано 5 научных работ, из них две в журналах, входящих в список ВАК.

Структура и объем диссертации. Диссертация состоит из введения, 4-х глав, заключения, списка литературы и 2-х приложений. Общий объём диссертации — 124 страницы.

Содержание работы

Первая глава посвящена обзору текущего состояния дел в области программирования графических процессоров. Графические процессоры, по сравнению с традиционными высокопроизводительными архитектурами, обладают лучшими показателями энергоэффективности и производительности в расчёте на стоимость, а также меньшими требованиями к инженерной инфраструктуре. Поэтому, начиная с 2000-х годов, они активно используются для решения вычислительно

ёмких задач. Среди решаемых на ГПУ задач есть задачи вычислительной астрофизики, финансового моделирования, молекулярной динамики, квантовой химии, линейной алгебры, а также многие другие. Однако основным препятствием на пути внедрения графических процессоров в широкую вычислительную практику является отсутствие для них адекватных средств программирования.

Средства программирования ГПУ можно разделить на интерфейсы программирования трёхмерной графики, архитектурно-зависимые интерфейсы и архитектурно-независимые интерфейсы. Первые в настоящее время практически не используются, в то время как вторые используются наиболее активно. К архитектурно-зависимым интерфейсам относятся средства программирования NVidia CUDA и AMD CAL/Brook+. К архитектурно-независимым — язык OpenCL, а также системы метапрограммирования наподобие RapidMind.

Однако все указанные средства программирования ГПУ обладают рядом недостатков. Прежде всего, все они являются низкоуровневыми, что означает, что для создания эффективных программ они требуют хорошего знания архитектуры конкретного ГПУ и ручной оптимизации. Создаваемые при этом программы получаются громоздкими и трудными для сопровождения. И хотя в ряде случаев они формально переносимы на другие архитектуры ГПУ и программируемых ускорителей, при таком переносе не сохраняется эффективность выполнения.

На основе сравнительного анализа архитектуры современных ГПУ и технологий параллельного программирования формулируются основные требования к разрабатываемой системе программирования ГПУ:

- **Высокоуровневость**. Система должна позволять прикладному программисту писать код в терминах, приближенных к его предметной области.

- **Эффективность**. Производительность программ, созданных при помощи разрабатываемой системы, должна быть сравнима с производительностью программ, полученных в результате ручной

оптимизации.

- **Переносимость**. Программы должны быть переносимы между различными архитектурами ГПУ с сохранением эффективности.

- **Простота интеграции** разрабатываемых при помощи создаваемой системы программ в существующие приложения и программные пакеты.

Вторая глава посвящена описанию разработанного в рамках данной работы языка программирования C$ (читается “си-эс”). При разработке языка за основу был взят язык C#, который был расширен конструкциями для программирования ГПУ.

Основная идея, которая привела к созданию языка C$, заключается в следующем: *вычисление на ГПУ - это независимое параллельное вычисление элементов некоторого целевого массива при помощи чистой функции*. При этом данная чистая функция, которая в терминах графических процессоров является шейдером, может быть построена как последовательность применений некоторых других чистых функций друг к другу. В качестве таких чистых функций могут выступать как массивы, содержащие данные, так и функции, используемые для их обработки. При этом отдельные функции создают сравнительно небольшую вычислительную нагрузку, и их непосредственное вычисление на ГПУ ведёт к высоким накладным расходам. Чтобы избежать их, используется концепция *ленивых вычислений*: применение функций друг к другу не вызывает вычислений, а приводит лишь к построению *графа функциональных операций*. Как только требуется результат подобного графа, выражаемая им функция вычисляется на ГПУ на всей её области определения, а результат её вычисления становится доступным в виде массива.

На основании этой идеи в основу разработанного языка были положены следующие концепции:

- **Высокоуровневость**. Это требование следует из требования высокоуровневости разрабатываемой системы.

- **Модель глобального адресного пространства и единого потока управления**. Эти требования вытекают из архитектуры современных ГПУ. Также эти модели являются наименьшим общим знаменателем, доступным на других нетрадиционных архитектурах.

- **Функциональное и массивное программирование**. Шейдер, исполняемый на ГПУ, можно представить как чистую функцию, а обрабатываемые на ГПУ данные – в виде массивов, что обуславливает применение данных концепций.

- **Неизменяемые объекты**. Эта концепция вытекает из способа организации параллельных вычислений, характерных для ГПУ. Потоки работают независимо друг от друга, создавая за один проход массив данных, который далее не модифицируется.

Исходя из вышеперечисленных требований, для программирования ГПУ в языке C$ введены типы *чистых функций* и *массивов*, *неизменяемые структурные классы*, а также операции над этими типами.

Функциональные типы — абстрактные типы чистой функции. Данные типы определяют две операции: вычисление в точке и получение области определения. В качестве функций могут выступать функции, объявленные в программном коде, массивы и функции-выражения, используемые для хранения построенного графа функциональных операций. Таким образом, массивы в языке C$ являются частным случаям функций.

Массивы языка C$ являются одно- или многомерными индексируемыми совокупностями однородных данных. В языке C$ массивы обладают особенностями, которые делают их использование в неоднородной системе, состоящей из ГПУ и ЦПУ, более эффективным. Массивы могут передаваться из памяти ГПУ в ОЗУ ЦПУ и обратно по необходимости. При этом формат хранения массивов в памяти не фиксирован и может меняться в зависимости от конкретной решаемой задачи. Массив C$ может быть использован на ЦПУ на запись сколько угодно

раз, но до первого использования в программе, исполняемой на ГПУ; после этого он становится неизменяемым. Массив, созданный в результате исполнения программы на ГПУ, изначально является неизменяемым. Такое поведение массивов соответствует типичным способам их использования в программе на ГПУ.

Неизменяемые структурные классы языка C$ являются аналогами структур других языков программирования. Объекты этих типов являются *неизменяемыми*. Это означает, что значения полей могут быть заданы только один раз, при создании объекта; после этого они не могут быть изменены. При помощи неизменяемых структурных классов в языке C$ реализованы такие типы, как комплексные числа и двух- и трёхкомпонентные векторы. Формат хранения массивов структурных типов также не фиксирован; однако чаще всего различные поля структурных типов физически хранятся в отдельных массивах. Членами структурных типов могут быть не только поля, но и методы и перегруженные операции; они могут использоваться для построения графов функциональных операций.

Над дополнительными типами, введёнными в язык C$, определён ряд операций. Прежде всего, это операция *актуализации*, которая используется для вычисления результата уже построенного графа функциональных операций, и необходима для функционирования механизма ленивых вычислений. С точки зрения синтаксиса языка, она представляет собой приведение функции-выражения к массиву или скалярному значению.

Далее, существуют операции и конструкции для построения графа функциональных операций. К ним относятся операции *суперпозиции* и *редукции*, а также конструкции *связанных переменных* и *вызова функции по образцу*. Также имеется *условная* операция, которая рассматривается как частный случай суперпозиции. Именно такой набор операций был выбран по той причине, что с его помощью можно, с одной стороны, создавать программы для решения достаточно большого класса задач, а с другой стороны, созданные программы можно

эффективно отображать на различные архитектуры ГПУ.

Операция *суперпозиции* позволяет применять функции к другим функциям, в том числе к массивам. С точки зрения синтаксиса она выглядит как применение обычной функции к аргументам, некоторые из которых являются функциями. Следует заметить, что благодаря широкой трактовке понятия функции в языке C$ операция суперпозиции покрывает ряд важных случаев: индексирование массивов, применение функций к массивам, применение стандартных арифметических операций и т.д. С другой стороны, суперпозиция не позволяет получить скалярное значение из функционального объекта.

Операция *редукции* позволяет получать скалярное значение из функционального объекта при помощи последовательного применения некоторой бинарной операции, стандартной или определённой пользователем, к значениям этого функционального объекта.

Конструкция *связанных переменных* позволяет явно указывать, каким образом индексы пробегают значения из областей определения двух и более функций в одном графе функциональных операций. Таким образом, конструкция связанной переменной представляет собой цикл без заголовка, при этом область значений связанных переменных вычисляется как пересечение явных ограничений на её область определения и областей определения функций, в которых она используется. Связанные переменные позволяют описывать в программе на C$ более сложные конструкции, такие как вычисление функции только на части области определения, или редукцию по части измерений. По сути, именно связанные переменные позволяют создавать на языке C$ наиболее вычислительно ёмкие программы, такие, как умножение матриц.

Конструкция *вызова функции по образцу* позволяет задавать различные способы вычисления функции для различных подмножеств области определения. Например, определяемая функция может вычисляться по-разному на границе области определения и внутри неё. По сути, данная конструкция представляет более структурированный спо-

соб выражения условных операций. Конструкция вызова по образцу показывает, что условие зависит только от значений индексов, и не зависит от значений функций-аргументов, и эта информация учитывается системой C$ при трансляции программы.

Третья глава посвящена описанию среды времени выполнения языка C$. Система времени выполнения состоит из трёх уровней: уровня собственно системы времени выполнения, уровня семейства архитектур и уровня конкретной архитектуры. Общая структурная схема системы программирования C$ изображена на рис. 1.

Уровень *системы времени выполнения* определяет структуру направленного ациклического функционального графа, который строится в результате применения операций языка C$ к функциям. Также система времени выполнения отвечает за выделение памяти под массивы и вычисление областей определения функций.

Уровень *семейства архитектур* определяет способ трансляции графа функциональных операций в разработанный для системы C$ промежуточный низкоуровневый код программ на ГПУ. На этом уровне выполняется наиболее содержательная часть оптимизаций программ для ГПУ. Кроме того, уровень семейства архитектур отвечает за оптимальное размещение данных в памяти ГПУ и ЦПУ и автоматическую синхронизацию между ними по необходимости.

Уровень *конкретной архитектуры* является прослойкой между уровнем семейства архитектур и интерфейсом к конкретному ускорителю. Его задача — обеспечить стандартный интерфейс доступа ко всем ускорителям данного семейства для вышестоящего уровня, а также транслировать программу из языка промежуточного представления в машинный код конкретного ускорителя. Помимо этого, он обеспечивает вышестоящему уровню информацию о конкретном используемом ускорителе семейства и сбор вспомогательной информации о выполнении программ на ускорителе.

Также в этой главе описываются методы оптимизации, используемые при трансляции программ. Задача используемых методов оптими-

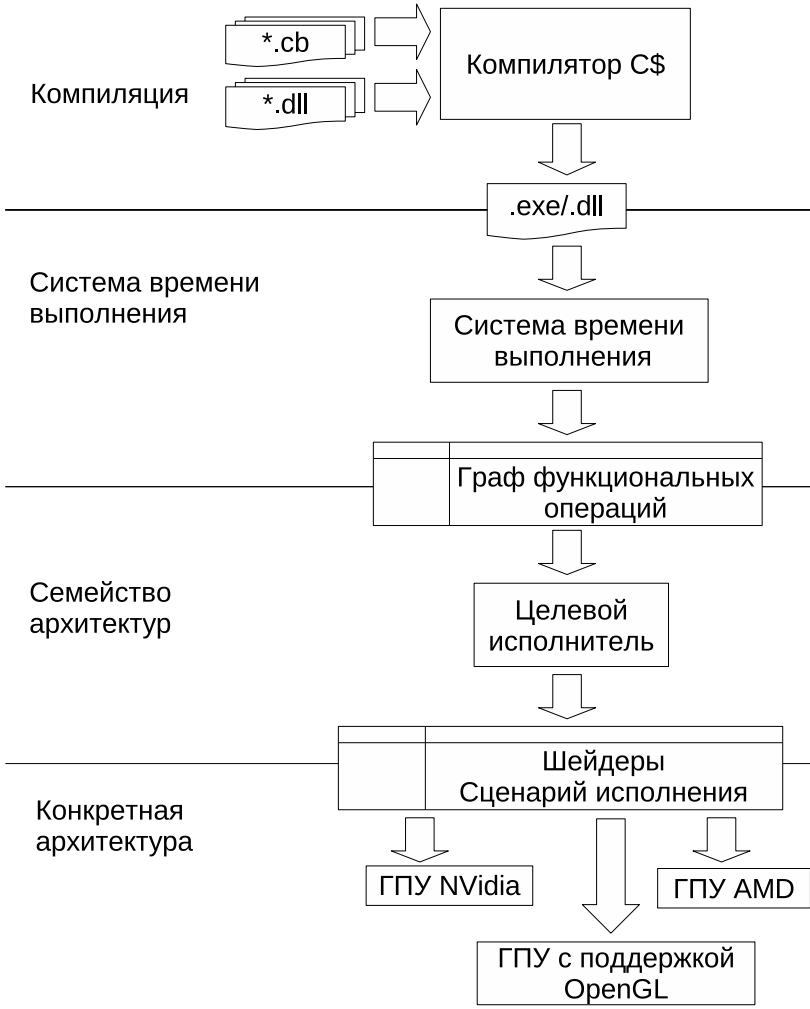

Рис. 1. Общая структурная схема системы программирования C$.

зации - задействовать низкоуровневые особенности конкретной архитектуры ГПУ, прежде всего, иерархию памяти и систему команд, для повышения эффективности работы программы.

В используемых алгоритмах оптимизации граф функциональных операций преобразуется в тесновложенное гнездо циклов. При этом корень дерева преобразуется в параллельный цикл без зависимостей, а вершины, соответствующие операциям редукции - в последовательные циклы. Для полученного гнезда цикла рассматриваются преобразования блочной развёртки, которые позволили бы минимизировать общую задержку на обмен данными с видеоОЗУ за счёт использования иерархии памяти. Заметим, что увеличение глубины развёртки сокращает объём передаваемых данных, но в то же время уменьшает количество одновременно работающих потоков ГПУ, и как следствие, возможности аппаратуры по сокрытию задержек доступа в видеоОЗУ.

Поэтому для нахождения оптимальной глубины развёртки каждого из циклов ставится задача оптимизации, в которой целевой функцией выступает суммарная задержка при доступе в видеоОЗУ, параметрами оптимизации - значения глубины развёртки циклов, а основным ограничением - объём вспомогательной памяти на чипе. Поскольку объём и, как следствие, суммарная задержка пропорциональны произведению значений глубины развёртки циклов, полученная задача оптимизации является нелинейной. Однако возникающая задача является задачей геометрического программирования, которая сводится к задаче выпуклой оптимизации и может быть эффективно решена при помощи метода барьеров. Несмотря на нелинейность задачи и кажущуюся в связи с этим сложность, время, затрачиваемое на решение задачи оптимизации, на порядок меньше времени, затрачиваемого на последующую генерацию кода.

Найденные значения глубины развёртки циклов передаются на этап генерации кода. На этом этапе выполняется устранение повторных обращений к памяти и группировка чтений по соседним адресам с целью дальнейшего сокращения количества операций с памятью. Кроме того,

в результате блочной развёртки циклов возникает большое количество независимых однородных операций. Они преобразуются в векторные команды, если векторные операции поддерживаются конкретной архитектурой ГПУ. Таким образом, векторизация возникает как побочный эффект блочной развёртки циклов.

Разработанные методы оптимизации позволяют задействовать низкоуровневые особенности конкретных ГПУ. По сути, они обеспечивают сохранение эффективности при переносе C$-программ с платформы на платформу.

Четвёртая глава описывает применение разработанной в рамках данной работы системы программирования C$ для решения задач линейной алгебры, вычислительной астрофизики, финансового моделирования и обработки изображений. При решении большинства из них она позволяла создавать компактный программный код, который переносим между различными архитектурами ГПУ с сохранением эффективности. При этом высокая эффективность исполнения достигалась автоматически и не требовала усилий со стороны разработчика прикладной программы. Разработанные методы оптимизации позволяют задействовать низкоуровневые особенности конкретных ГПУ и повысить производительность в несколько раз по сравнению с трансляцией, выполняемой без использования разработанных методов оптимизации. На задачах умножения матриц, моделирования системы N тел и вычисления цены опционов Блэка-Шоулза была достигнута эффективность, составляющая 65-90% от той, что достигалась в результате длительной ручной оптимизации. Производительность системы C$, в ГФлоп/с с одинарной точностью, на тестируемых задачах на различных графических процессорах приведена в табл. 1.

В **заключении** сформулированы основные результаты работы:

1. На основании сравнительного анализа архитектуры современных ГПУ и технологий параллельного программирования разработан высокоуровневый язык C$ для программирования графических процессоров. Программы на языке C$ обладают свойством пере-

	AMD HD 4850	NVidia GeForce 8800 GTX
Пиковая производительность	1000	340
Умножение матриц 2048*2048	339,58	78,56
Фильтрация 3*3 2048*2048	89,24	6,7
Фильтрация 5*5 2048*2048	143,31	2,41
Блэк-Шоулз 10^6 опционов	273,8	203,82
Гаусс 9*9 2048*2048	65,96	7,44
Задача N тел 110592 тел	716,53	211,60

Таблица 1. Производительность системы C$ (ГФлоп/с) на ряде вычислительных задач.

носимости между различными семействами ГПУ с сохранением высокой эффективности.

2. Для предложенного языка спроектирована и реализована система поддержки времени выполнения. Система обеспечивает единое внутреннее представление программы, её оптимизацию, генерацию и исполнение кода для различных архитектур графических процессоров (NVidia и AMD).

3. Разработаны и интегрированы в систему программирования C$ методы оптимизации программ для графических процессоров. Разработанные методы оптимизации учитывают иерархию памяти и уровни параллелизма различных графических процессоров.

4. Разработанная система программирования успешно прошла апробацию при решении вычислительно сложных задач линейной алгебры, финансового моделирования, астрофизики и ряда других с использованием большого числа различных архитектур графических процессоров.

Список публикаций

Основные результаты диссертации опубликованы в следующих работах:

[1] *А.В.Адинец, Вл.В.Воеводин.* Графический вызов суперкомпьютерам // *Открытые системы.* — 2008. — 05. — № 04. — С. 32–41.

[2] *А.В.Адинец.* Анализ эффективности решения задачи N тел на различных вычислительных архитектурах // *Вестник ННГУ имени Н.И.Лобачевского (принята к публикации).*

[3] *Andrew V. Adinetz.* A Higher-Level and Portable Approach to GPGPU Programming // Proceedings of SYRCoSE'07 Conference. — Lomonosov Moscow State University, Moscow: 2007. — 06.

[4] *А.В.Адинец, Н.А.Сахарных.* О программировании вычислений общего назначения на современных графических процессорах // Труды конференции "Научный сервис в сети Интернет-2007: многоядерный компьютерный мир". — Новороссийск: 2007. — 09.

[5] *А.В.Адинец, Н.А.Сахарных.* О системе программирования вычислений общего назначения на современных графических процессорах // Численные методы, параллельные вычисления и информационные технологии: Сборник научных трудов. — Москва: Издательство Московского Университета, 2008. — 01. — С. 25–52.

Подписано в печать 14.09.2009 г. Формат 60x84/16.
Бумага офс. № 1. Печать ризо.
Усл. печ. л. 1. Тираж 100 экз.
Заказ № 7.

Участок оперативной печати НИВЦ МГУ.
119991, ГСП-2, Москва, НИВЦ МГУ имени М.В.Ломоносова