На правах рукописи

Краснов Михаил Михайлович

СЕТОЧНО-ОПЕРАТОРНЫЙ ПОДХОД К ПРОГРАММИРОВАНИЮ ЗАДАЧ МАТЕМАТИЧЕСКОЙ ФИЗИКИ

05.13.11 - математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

АВТОРЕФЕРАТ

диссертации на соискание ученой степени

кандидата физико-математических наук

Москва – 2016

Работа выполнена в Федеральном государственном учреждении "Федеральный исследовательский центр Институт прикладной математики им. М.В. Келдыша Российской академии наук" (ИПМ им. М.В. Келдыша РАН)

Научный руководитель:
Жуков Виктор Тимофеевич, доктор физико-математических наук.

Официальные оппоненты:
Гаранжа Владимир Анатольевич, доктор физико-математических наук, профессор РАН, зав. сектором параллельных вычислений отдела прикладных проблем оптимизации Вычислительного центра им. А.А. Дородницына ФИЦ ИУ РАН.

Баранов Антон Викторович, кандидат технических наук, доцент, ведущий научный сотрудник Межведомственного суперкомпьютерного центра РАН – филиала ФГУ ФНЦ НИИСИ РАН.

Ведущая организация: ФГБУН «Институт вычислительной математики и математической геофизики Сибирского отделения Российской академии наук».

Защита состоится «___» _____ 2017 г. в ___ часов на заседании диссертационного совета Д 002.024.01 в ИПМ им. М.В. Келдыша РАН по адресу: 125047, Москва, Миусская пл., 4.

С диссертацией можно ознакомиться в библиотеке ИПМ им. М.В.Келдыша РАН, http://keldysh.ru .

Автореферат разослан «____» _____ 2017 г.

Учёный секретарь диссертационного совета
кандидат физико-математических наук А.Е. Бондарев

Общая характеристика работы

Объект исследования и актуальность темы

При решении задач математического моделирования наблюдается разительное отличие между описанием алгоритмов в математической литературе и реализацией этих алгоритмов в программах. В литературе при описании алгоритмов активно используются операторы, такие, например, как оператор Лапласа, операторы дивергенции, градиента и ротора, операторы производных и интегралов. Кроме того, при описании алгоритмов на двумерных и трёхмерных областях оперируют с векторами. Всё это позволяет записывать алгоритмы компактно и просто для понимания. При записи тех же самых алгоритмов на языках программирования эти короткие и ясные формулы часто превращаются в громоздкие многострочные операторы, разобраться в которых весьма непросто. При работе с векторами одна относительно короткая формула может превратиться даже не в один громоздкий оператор, а в целых три (по числу измерений).

Ещё одна проблема, с которой сталкиваются математики в последние годы – необходимость освоения новых мощных суперкомпьютеров с нетрадиционной (не фон-Неймановской) архитектурой, в высшей степени параллельных, методы программирования для которых весьма отличаются от традиционных. Основу вычислительной мощности таких суперкомпьютеров могут составлять графические ускорители NVIDIA CUDA (Compute Unified Device Architecture), на которых одновременно может быть запущено тысяча нитей, или новейшие процессоры Intel Xeon Phi с 60 ядрами на кристалле, в каждом из которых одновременно может исполняться 4 нити.

Имеются программные средства, позволяющие компактно записывать сложные формулы, и средства, облегчающие программирование на

графических ускорителях и других нетрадиционных архитектурах. Но нет единой системы, решающей сразу обе задачи.

Цели работы

Целями данной диссертационной работы являются:

- Разработка подхода к программированию, позволяющего:
    - Компактно записывать и эффективно программно реализовывать класс математических формул, в частности, введение в программы понятия «сеточного оператора», аналогичного математическому понятию оператора.
    - Единообразно реализовывать подход на разных типах сеток и для различных вычислительных архитектур;
- Разработка экспериментального программного комплекса, показывающего возможность реализации данного подхода для решения широкого класса задач математической физики.

Научная новизна работы

1. Разработан новый подход к программированию для класса математических вычислений на различных типах сеток, как регулярных, так и произвольных нерегулярных. Основой данного подхода является введённое понятие программного сеточного оператора, аналогичного математическому понятию оператора. Возможности программного оператора существенно превышают возможности традиционных шаблонов (таких, как, например, шаблон частной производной нужной степени и точности). По сути, программный оператор – это функция произвольной сложности с нужным числом аргументов.

2. В рамках разработанного подхода вводится новая система понятий, обеспечивающая эффективную реализацию подхода. Эта система

понятий, помимо ключевого понятия программного сеточного оператора, также включает понятия вычисляемого объекта (evaluable object), вычислителя (evaluator), сеточной функции (grid function), исполнителя вычислений (executor), индексатора (index), заместителя (proxy). Введение в подход чётко определённых понятий позволяет единообразно реализовать его на разных типах сеток и для различных параллельных вычислительных архитектур с общей памятью. В частности, реализации подхода для локально-адаптивных сеток и для трёхмерных нерегулярных тетраэдральных сеток были написаны на основе реализации для регулярных трёхмерных сеток. Процесс переноса подхода на новый тип сетки, конечно, неформальный и творческий, но несложный.

3. Разработан комплекс программных библиотек, демонстрирующих возможность эффективной программной реализации предложенного подхода к программированию для различных типов сеток. Данный подход апробирован на регулярных прямоугольных трёхмерных сетках на многосеточном методе, на локально-адаптивных сетках на задаче теплопроводности, на тетраэдральных трёхмерных сетках на разрывном методе Галёркина и на одномерных сетках на задаче ENO (Essentially Non-Oscillatory). Для каждого типа сетки была реализована своя программная сеточная библиотека, реализующая данный подход. Все версии библиотеки были перенесены на параллельную платформу NVIDIA CUDA, а также на OpenMP, что даёт возможность запуска программ на Intel Xeon Phi в «native» режиме. Путём простой перекомпиляции удаётся ускорить программы на CUDA в 4-8 раз по сравнению с последовательной версией (на суперкомпьютере К-100).

Практическая значимость

В соответствие с предлагаемым подходом написана сеточно-операторная библиотека gridmath для трёхмерных регулярных прямоугольных сеток, с помощью которой перенесена на графические ускорители NVIDIA CUDA программа, реализующая параллельный многосеточный метод в рамках проекта для Института проблем безопасного развития атомной энергетики РАН. Программа запускалась на кластере К-100 в ИПМ им. М.В. Келдыша РАН, причём процессы обменивались между собой по протоколу обмена сообщениями MPI, а внутри процесса счёт вёлся на графическом ускорителе NVIDIA CUDA. Каждый узел К-100 содержит по три таких ускорителя, использовались все три ускорителя на каждом узле.

На библиотеку gridmath (версия для регулярных трёхмерных сеток) был также перенесён тест MG из набора тестов NAS Parallel Benchmarks и задача решения системы квазигидродинамических уравнений QGD3D. Обе эти программы показали высокую переносимость исходного текста, написанного с использованием сеточно-операторной библиотеки, на ускорители NVIDIA CUDA, а также на OpenMP с последующим запуском на процессорах Intel Xeon Phi.

Затем подход был успешно реализован на локально-адаптивных сетках внутри прямоугольного параллелепипеда, и с помощью неё решена задача диффузии, в которой внутрь параллелепипеда помещалось другое тело (например, шар или цилиндр), на границе которого был скачок коэффициента диффузии. На границе внутреннего тела сетка сгущалась.

Библиотека была также перенесена на тетраэдральные трёхмерные сетки и с её помощью было решено уравнение Эйлера разрывным методом Галёркина. Также был реализован метод ENO для одномерных сеток. Последние три примера показывают, что в соответствие с предлагаемым

подходом можно относительно легко реализовать программную библиотеку для любых типов сеток.

На примере трёхмерной задачи теплопроводности реализованы несколько параллельных алгоритмов решения задач, включая методы Якоби, Гаусса-Зейделя, Чебышевские методы и многосеточный метод.

Положения, выносимые на защиту

1. Разработан новый подход к программированию для класса математических вычислений на различных типах сеток.
2. В рамках разработанного подхода введена новая система понятий, обеспечивающая эффективную реализацию подхода.
3. Разработан комплекс программных библиотек, демонстрирующих возможность эффективной программной реализации предложенного подхода к программированию для различных типов сеток.

Апробация работы и публикации

По теме диссертации были сделаны девять докладов на научных конференциях:

1. Международной научной конференции «Научный сервис в сети Интернет: экзафлопсное будущее», сентябрь 2011 г., г. Новороссийск.
2. Национальном суперкомпьютерном форуме (НСКФ-2013), ноябрь 2013 г., Переславль-Залесский, ИПС им. А.К. Айламазяна РАН.
3. XX Всероссийской конференции «Теоретические основы и конструирование численных алгоритмов решения задач математической физики», посвященная памяти К.И. Бабенко, сентябрь 2014 г., г. Новороссийск.
4. Международной конференции «Суперкомпьютерные дни в России», 28 - 29 сентября 2015 г., г. Москва.

5. XIV Международный семинар «Математические модели и моделирование в лазерно-плазменных процессах и передовых научных технологиях», 4 - 9 июля 2016 г., г. Москва.
6. VII всероссийская научная молодежная школа-семинар «Математическое моделирование, численные методы и комплексы программ» имени Е. В. Воскресенского с международным участием, 12 - 15 июля 2016 г., г. Саранск.
7. XXI Всероссийская конференция «Теоретические основы и конструирование численных алгоритмов для решения задач математической физики», посвященная памяти К.И.Бабенко, 5-11 сентября 2016 г., г. Новороссийск.
8. XV International seminar «Mathematical models & modeling in laser plasma processes & advanced science technologies», 26 - 30 September, 2016, Montenegro, Petrovac.
9. XVI Международная конференция «Супервычисления и математическое моделирование», 3-7 октября 2016, г. Саров.

Имеется 19 публикаций, из которых пять [7-11] – в журналах из списка ВАК.

Структура и объём работы

Диссертация состоит из введения, четырёх глав, заключения, списка литературы (66 наименований) и одного приложения. Общий объём работы составляет 113 страниц.

Содержание работы

Во **введении** обосновывается актуальность темы исследования, формулируются цель и задачи диссертации. Приводятся основные результаты работы, их практическая значимость. Перечисляются выступления на конференциях и семинарах, кратко излагается содержание работы.

В **первой главе** приводится обзор существующих работ, направленных, с одной стороны, на облегчение программирования на новых нетрадиционных компьютерных архитектурах, а с другой стороны, на сокращение способа записи сложных математических формул.

Данный подход к программированию стоит на стыке двух направлений. Первое направление связано с тем, что освоение методов программирования на новых нетрадиционных архитектурах является непростой задачей, особенно для прикладных математиков. С другой стороны, освоение этих архитектур необходимо, т.к. новейшие суперкомпьютеры оснащаются такими вычислителями и обидно их не использовать. Одним из важных направлений развития системного программного обеспечения становится создание систем, упрощающих для прикладного программиста написание программ, использующих вычислительные возможности таких суперкомпьютеров с новейшими вычислителями. Основная цель при создании подобных систем – обеспечить переносимость исходного текста программ на эти новые архитектуры. При этом исходный текст может снабжаться псевдокомментариями или прагмами, с помощью которых можно управлять работой специализированных компиляторов. Или же один и тот же исходный текст можно компилировать разными компиляторами и получать код для разных архитектур. Такие работы активно ведутся в том числе и в ИПМ им. М.В. Келдыша РАН. Например, язык НОРМА и система DVM. Среди других работ можно упомянуть высокоуровневый язык Liszt.

Второе направление связано с попыткой упростить написание сложных программ за счёт выноса часто повторяющегося кода в отдельные модули или классы и затем многократного их использования. При этом решаются две задачи: исходный текст становится, во-первых, короче, и, во-вторых, понятнее. Так как выносимый код обычно используется внутри

многократно повторяющегося тела цикла, важным становится вопрос эффективности. В частности, этот код плохо выносить в отдельные обычные функции, т.к. вызов функции – дорогая по времени операция. При программировании на языке C++ функции могут быть объявленными инлайновыми (inline). В этом случае код этих функций вставляется компилятором в точке вызова. При использовании шаблонов функций и классов (использование шаблонного метапрограммирования - Template Metaprogramming) компилятор такие функции и методы таких классов делает инлайновыми автоматически. При хорошо работающем оптимизаторе полученный машинный код по своей производительности может не уступать коду, полученному с помощью компилятора с языка Фортран. Одной из лучших библиотек, эффективно использующих шаблонное метапрограммирование, является библиотека Blitz++.

Есть системы, упрощающие запись вычислений за счёт реализации матрично-векторной алгебры, в которых оперирование векторами и матрицами производится как с едиными объектами. К таким системам можно отнести, например, систему Matlab и язык SIMIT. Запись сложных манипуляций с матрицами и векторами в этих системах очень компактна и не уступает по своей компактности записи в математической литературе.

Существуют также системы для решения задач математического моделирования с уже готовыми решателями различных задач. Как правило, пользователь может добавлять в них свои расширения. Среди таких систем можно упомянуть библиотеку PETSc и пакет OpenFOAM.

Вторая глава посвящена общему описанию сеточно-операторного подхода, его назначению и области применимости, типам обрабатываемых данных. Вводятся основные термины, описываются взаимосвязи основных объектов сеточно-операторного подхода.

Сеточно-операторный подход к программированию предназначен для упрощенной записи вычислений на произвольных сетках. Основными

объектами, с которыми позволяет работать библиотека, написанная в соответствие с данным подходом, являются: вычисляемый объект (evaluable object), сеточная функция (grid function), сеточный оператор (grid operator) и сеточный вычислитель (grid evaluator), который получается в результате «применения» сеточного оператора к вычисляемому объекту. К вычисляемым объектам относятся сеточные функции и сеточные вычислители. Для сеточных операторов реализована арифметика с другими сеточными операторами, при этом порождаются новые составные сеточные операторы. Для сеточных вычислителей также реализована своя арифметика с другими сеточными вычислителями и скалярными величинами, при этом порождаются новые составные сеточные вычислители.

Вычислители можно присваивать плотным сеточным функциям (сеточным функциям, хранящим все свои значения во всех узлах сетки). Запуск вычислений производится именно при таком присваивании, не раньше. До момента присваивания вычислителя плотной сеточной функции цепочка вычислений просто запоминается. Для запоминания цепочки вычислений используется механизм шаблонов выражений (expression templates). При присваивании происходит запуск вычислений для всех требуемых точек сеточной функции в левой части оператора присваивания. Для запуска вычислений используется специальный объект – исполнитель (executor). Он может быть указан пользователем явно или использоваться по умолчанию. При компиляции для CUDA (компилятор nvcc) и для обычного процессора по умолчанию подставляются разные исполнители. Для CUDA исполнитель вычисляет все точки параллельно путём вызова ядра (kernel) в графическом ускорителе. В последовательном случае исполнитель обходит все точки последовательно с помощью циклов.

Применение сеточного оператора к вычисляемому объекту реализовано с помощью функционального оператора `()`. Каким бы сложным ни было выражение в правой части оператора присваивания, процесс вычисления запускается один раз при присваивании. Это особенно важно при работе на графических ускорителях CUDA, когда вызов ядра является относительно дорогостоящей операцией. Независимо от сложности выражения в правой части при присваивании делается один вызов одного ядра, и все вычисления производятся в этом ядре.

Главное назначение вычислителей – «запомнить» последовательность и параметры вычислений. Таким образом, в подходе реализуется концепция отложенных вычислений, позволяющая избавиться от промежуточных переменных и за счёт этого сэкономить оперативную память (что становится важным при больших размерах сеток).

Библиотека *gridmath* поддерживает работу как со скалярными величинами, так и с векторными, причём и те, и другие могут быть как безразмерными, так и иметь размерность. Обычно при решении задач вычислительной математики работают с безразмерными типами данных, такими, как double, float или std::complex. Однако работа с типами данных, имеющими размерность, имеет свои преимущества. Во-первых, программа становится более наглядной, так как размерность той или иной переменной видна уже из её определения. Во-вторых, часть возможных программистских ошибок (таких, как сложение и вычитание величин с разными размерностями) обнаруживаются уже на стадии компиляции. Что касается эффективности вычислений (а это чрезвычайно важный аспект), то информация о размерности той или иной величины (переменной или вычисленного значения) является информацией времени компиляции (метаданными) и не приводит ни к увеличению объёма занимаемой оперативной памяти, ни к замедлению выполнения программы. Возможно,

использование размерных величин несколько замедлит время компиляции программы, но несущественно.

В **третьей главе** описываются принципы реализации библиотеки *gridmath*, реализующей сеточно-операторный подход, использованные технологии. Более подробно описываются интерфейсы основных объектов.

Библиотека *gridmath* является шаблонной (template). Большинство её классов и функций шаблонные. Таким образом, вся библиотека поставляется в исходных текстах в виде набора .h и .hpp файлов. Скомпилировать исходные тексты, использующие данную библиотеку, можно большинством современных компиляторов с языка C++.

Библиотека *gridmath*, помимо стандартной библиотеки языка C++ (STL) использует библиотеку *boost*, а при компиляции для CUDA – ещё и библиотеку thrust из состава CUDA SDK. Библиотека *thrust* содержит аналоги большинства необходимых компонентов как из STL, так и из *boost*.

С другой стороны, библиотека объектно-ориентированная и классы выстроены в определённое иерархическое дерево классов. Одно из свойств объектно-ориентированного программирования – полиморфизм, когда, имея ссылку на объект базового класса, можно вызвать метод из класса-наследника. В классическом объектно-ориентированном программировании на C++ полиморфизм реализуется с помощью механизма виртуальных функций. Однако, этот механизм является чрезвычайно затратным по времени исполнения. Если тело функции небольшое, то может оказаться, что временные затраты на вызов виртуальной функции сравнимы или даже превышают время выполнения самой функции. Гораздо эффективнее работают «inline» функции, когда код тела функции подставляется вместо вызова, но для виртуальных функций такое невозможно, т.к. компилятору тело функции в конечном классе неизвестно. С появлением в языке C++ шаблонов ситуация

радикальным образом изменилась. Хотя, возможно, первоначальной целью введения шаблонов в язык было обеспечить работу с типизированными коллекциями и другие относительно простые примеры использования, выяснилось, что шаблоны являются мощным механизмом для реализации таких вещей, как, например, метавычисления (вычисления времени компиляции) и нового типа полиморфизма – шаблонного. Шаблонный полиморфизм реализуется с помощью шаблона проектирования (design pattern), известного как CRTP - Curiously recurring template pattern. Идея CRTP следующая: в базовый класс в качестве шаблонного параметра передаётся имя конечного класса. При этом, если имеется ссылка на объект базового класса, то её всегда можно привести к ссылке на конечный класс и вызвать тот или иной метод конечного класса непосредственно. При этом компилятор знает всё про этот конечный класс и может вместо вызова метода подставить его тело, что и требуется. Фактически такой полиморфизм является статическим – для каждого конечного класса компилятор генерирует свой экземпляр шаблонной функции. Покажем на простом примере, как это реализуется. Пусть Base – это базовый класс, а Derived – конечный класс и мы хотим, имея ссылку на объект базового класса, вызвать метод в конечном классе. Сделать это можно, например, так:

```
template<class D>
struct Base {
     D& self(){ return static_cast<D&>(*this); }
};
struct Derived : Base<Derived> {
     void m(){ … }  // Метод, который хотим вызвать
};
template<class D>   // Полиморфная функция
void f(Base<D> &obj){ // Ссылка на объект базового класса
     obj.self().m(); // Вызов метода конечного класса
```

```
}
```

В библиотеке gridmath активно используются метавычисления. Метавычисления – это вычисления, производимые во время компиляции. В языке C++ во время компиляции можно вычислять типы или целочисленные константы. Для метавычислений можно писать т.н. метафункции.

Приведём пример. Метафункция is_numeric принимает произвольный тип и возвращает в переменной value булевскую константу (true или false) в зависимости от того, является ли переданный тип числовым:

```
template<bool B>    // Вспомогательный класс
struct bool_ { static const bool value = B; };
template<typename T>
struct is_numeric : bool_<false>{}; // Общий случай, по
умолчанию тип не числовой
// Специализации для числовых типов данных
template<> struct is_numeric<int> : bool_<true>{};
template<> struct is_numeric<long> : bool_<true>{};
template<> struct is_numeric<short> : bool_<true>{};
template<> struct is_numeric<float> : bool_<true>{};
template<> struct is_numeric<double> : bool_<true>{};
template<> struct is_numeric<long double> : bool_<true>{};
template<typename T> struct is_numeric<std::complex<T> > :
bool_<true>{};
```

Теперь к этой метафункции можно обращаться как из других метафункций, так и из исполняемого кода. Например:

```
bool for_double = is_numeric<double>::value;      // true
bool for_string = is_numeric<std::string>::value; // false
```

Использование размерных величин повышает читаемость программы, т.к. уже из размерности переменных видно, что же они хранят (например, скорость, плотность или давление). При этом использование размерных величин вместо безразмерных не меняет объём памяти, занимаемой

переменными и не влияет на скорость работы программы, т.к. вся информация о размерности хранится в метаданных. Библиотека gridmath поддерживает работу с размерными величинами. Для работы со скалярными размерными величинами (плотность, давление) предназначен класс quantity, а с векторными (скорость, ускорение, сила) – vector_quantity. Оба класса принимают два шаблонных параметра – тип хранимого значения (например, float или double) и собственно размерность:

```
template<typename T, class Dimensions>
struct quantity
{
… // Реализация класса
private:
     data_type m_value;
};
```

В качестве второго параметра следует передавать специальный тип quantity_dim, хранящий в метаданных (параметрах шаблона) семь целых чисел, соответствующий семи основным физическим величинам: масса, длина, время, сила тока, температура, сила света и количество вещества. Единицы физических величин (например, граммы или килограммы для массы, метры или сантиметры для длины) в библиотеке не определены, но подразумеваются одинаковыми:

```
template<int N1,int N2,int N3,int N4,int N5,int N6,int N7>
struct quantity_dim {
     static const int value1 = N1;
     static const int value2 = N2;
     static const int value3 = N3;
     static const int value4 = N4;
     static const int value5 = N5;
     static const int value6 = N6;
     static const int value7 = N7;
```

```
};
```

В библиотеке реализована концепция отложенных вычислений: до тех пор, пока не будет исполнен оператор присваивания вычислителя плотной сеточной функции, вычисления не производятся. Вместо этого последовательность вычислений запоминается. При этом запоминаются сеточные функции, операторы, вычислители и операции с ними и со скалярами. Но ссылки на объекты сохранять нельзя, и приходится создавать и сохранять копии объектов. Ссылки на объекты нельзя сохранять по двум причинам. Первая – это то, что объекты (операторы и вычислители) часто создаются «на лету» как результат арифметических операций и нет явных переменных, соответствующих этим объектам. Ссылки на такие объекты (созданные «на лету») сохранять нельзя и необходимо делать их копии. Вторая причина состоит в том, что при компиляции для CUDA объекты создаются в памяти host-процессора, а исполняются в памяти графического вычислителя CUDA. Чтобы вычисления можно было выполнить, нужно вычисляемый объект скопировать из памяти host-процессора в память графического вычислителя, копировать же ссылки (или указатели) на объекты из памяти host-процессора в память CUDA не имеет смысла, т.к. это разные адресные пространства. Фактически в случае плотной сеточной функции нужно скопировать указатель на данные.

Для того чтобы решить указанную проблему, в библиотеке вводится понятие «заместителя» объекта (proxy). В качестве замещаемого объекта может быть практически любой объект библиотеки: сеточная функция, оператор или вычислитель. Тот или иной объект может быть «лёгким» или «тяжёлым» для копирования. «Лёгкими» для копирования будем называть объекты, не содержащие векторы данных, а «тяжёлыми» – содержащие их. Если объект «лёгкий», то при копировании будет сохраняться копия самого объекта, а если «тяжёлый» – то его «лёгкий» заместитель.

Заместитель объекта похож на сам объект за тем исключением, что в заместителе объекта все вектора данных заменяются на указатели на эти данные.

По умолчанию все объекты (сеточные функции, операторы и вычислители) считаются «лёгкими», т.е. в качестве класса-заместителя по умолчанию передаётся сам класс объекта, а в качестве объекта-заместителя создаётся копия самого объекта. Если сеточный оператор или сеточная функция, написанные программистом, является «тяжёлым» (содержит один или несколько векторов данных), то следует создать класс заместителя и в базовый класс передать в качестве второго шаблонного параметра этот класс заместителя.

Четвёртая глава содержит описание приложений, написанных с использованием сеточно-операторного подхода к программированию.

В общем случае перенос программы на графические ускорители CUDA является непростой задачей, подразумевающей, в том числе, знакомство с архитектурой CUDA и методами программирования на ней. Библиотека gridmath позволяет «автоматически» переносить на CUDA написанные с её использованием программы. Программа может быть отлажена в последовательном режиме на «обычном» процессоре, а затем перенесена на CUDA практически без изменений путём перекомпиляции компилятором nvcc.

Сеточно-операторный подход был задуман при решении задачи по переносу программы, реализующей многосеточный метод (multigrid) на графические ускорители NVidia CUDA в рамках проекта для Института проблем безопасного развития атомной энергетики РАН. С использованием библиотеки gridmath, реализующий данный подход, задача была успешно решена.

Кроме того, автором была переписана с использованием библиотеки gridmath последовательная версия теста MG из набора тестов NAS Parallel

Benchmarks, которая затем была перенесена на NVIDIA CUDA, а также известная задача о каверне с подвижной крышкой с помощью системы квазигазодинамических уравнений по явной схеме.

Первоначальная версия библиотеки была написана для регулярных прямоугольных сеток, позже она была перенесена на произвольные трёхмерные сетки. С помощью библиотеки для тетраэдральных сеток был реализован разрывный метод Галёркина.

Сотрудниками ИПМ им. Келдыша РАН Жуковым Виктором Тимофеевичем, Феодоритовой Ольгой Борисовной и Новиковой Натальей Дмитриевной был разработан параллельный многосеточный метод для разностных эллиптических уравнений и написана программа, реализующая этот метод. В рамках проекта для Института проблем безопасного развития атомной энергетики РАН (ИБРАЭ РАН) была поставлена задача о переносе этой программы на графические ускорители NVidia CUDA.

Исходная задача считалась на кластере, при этом вся трёхмерная сетка была распределена между узлами, обмен граничными условиями вёлся с помощью MPI, а на каждом узле счёт вёлся последовательно. Теперь предстояло счёт на каждом узле также распараллелить на графических ускорителях. Это потребовало весьма существенного переписывания алгоритмической части программы, т.к. в исходном коде цикл в операторе присваивания вёлся по сеточной функции не в левой части оператора, а в правой. Это не давало возможности распараллелить существующий цикл. Были написаны три сеточных оператора: интерполяции, проектирования и сглаживания. Особенностью данной задачи было то, что, хотя сетка и прямоугольная, но не равномерная, шаг сетки в каждом направлении для каждого индекса свой (хотя от индексов в других направлениях он не зависит). Оператор сглаживания реализовывал семиточечную схему со своим коэффициентом в каждом узле сетки и с переменным шагом сетки в каждом направлении.

Ещё одной особенностью задачи было то, что нужно было обмениваться по MPI данными, лежащими в памяти GPU. Это значит, что эти данные нужно было вначале копировать в память основной (host) системы, пересылать по MPI, а затем копировать обратно в память GPU. Для копирования данных использовалась возможность получать срезы (грани) плотных сеточных функций.

Также следует обратить внимание, что при работе на GPU библиотека хранит данные в памяти GPU, а её не так уж и много по сравнению с объёмом памяти в основной системе. Так, на кластере К-100 на основной системе имеется 96 Гбайт памяти, что в пересчёте на один процесс (при полной загрузке узла – 12 процессов) даёт по 8 Гбайт на процесс, в то время как на каждой графической плате имеется только 2,8 Гбайт памяти. Это накладывает ограничение на размер сетки. Максимальный размер сетки на один процесс при работе на GPU составляет 256 точек в каждом направлении.

Тест MG из набора тестов NAS Parallel Benchmarks является реализацией стандартного варианта многосеточного метода для решения трехмерного уравнения Пуассона с периодическими краевыми условиями на равномерной декартовой сетке. Используются: линейный оператор интерполирования, точечный метод Якоби в качестве сглаживающей процедуры и набор вложенных сеток с коэффициентом 2 прореживания узлов по каждому направлению. Операторы на каждой сетке строятся по простейшим формулам 27-точечной разностной аппроксимации.

Для задачи определены классы задачи, определяющие размер сетки и число сглаживающих итераций. Есть классы S, W, A, B, C, D, E. класс S – самый маленький, а класс E – самый большой. Для класса B размер сетки – 256 точек в каждом направлении, а число итераций равно 20. Для класса C размер сетки – 512 точек в каждом направлении. При таком количестве точек требуемый размер оперативной памяти около 3,5 Гбайт и на К-100

на GPU памяти уже не хватает. Кстати, исходный текст на Фортране также на классе C не работает, но не из-за нехватки памяти, а из-за того, что в нём память выделяется в статической области одним куском, а для этого есть ограничение в 2 Гбайта. Максимальный класс, на котором можно провести сравнительное тестирование – это класс B.

Задача о решении системы квазигидродинамических уравнений QGD3D интересна в двух отношениях. Во-первых, в ней использовались не только скалярные трёхмерные поля (такие, как поля плотностей, давлений и температур), но и векторные (такие, как поля скоростей, ускорений и сил). Вторая особенность – использование размерных величин, т.е. для, например, давления, использовалась не переменная типа double, а переменная специального типа, хранящая в метаданных (данные времени компиляции) ещё и размерность. Работа с такими размерными величинами (как скалярными, так и векторными) также поддерживается библиотекой gridmath. Использование размерных типов данных позволяет решить две задачи. Во-первых, исходный текст программы становится более наглядным, т.к. из размерности той или иной переменной сразу становится ясно, что в ней хранится (например, скорость или давление). Во-вторых, часть возможных ошибок в программе, связанных с неправильным использованием переменных (присваивать, складывать и вычитать можно только величины одной размерности) обнаруживаются уже на этапе компиляции. Использование размерных типов данных никак не увеличивает объём памяти, занимаемый переменными, и не замедляет скорость выполнения программы.

Для реализации разрывного метода Галёркина сеточно-операторный подход был перенесён на трёхмерные нерегулярные сетки. У этого метода относительно сложная математика, которая хорошо формулируется в терминах операторов (лимитирования, вычисления объёмных интегралов и потоков через грани), и на нём сеточно-операторный подход может

показать все свои преимущества. Как известно, разрывный метод Галёркина характеризуется высоким порядком точности решения, что влечет за собой и высокую вычислительную сложность, что является большим недостатком метода Галёркина. В данной работе активно применяется механизм метапрограммирования языке C++, позволяющий вынести часть вычислений на стадию компиляции.

Для метода Галёкрина было реализовано несколько операторов, в том числе операторы лимитирования, объёмного интегрирования и вычисления потоков через грани. Вычисления проводились на кластере К-100 с использованием MPI, вычисления на одном узле распараллеливались с помощью OpenMP или CUDA.

Метод ENO был реализован для одномерного случая. Данная реализация ещё раз показала широкую применимость сеточно-операторного подхода для различных задач и типов сеток.

В **заключении** сформулированы основные результаты работы.

Основные результаты работы:

1. Предложен новый подход к разработке ПО для решения задач математической физики с использованием сеточных операторов, аналогичных математическим операторам.
2. Подход реализован в виде программных библиотек для разных типов сеток, что обеспечивает наглядную математическую нотацию, улучшающую структурированность исходных текстов программ и упрощает их перенос на параллельные архитектуры.
3. С помощью данного подхода был решён ряд задач на различных типах сеток: регулярных трёхмерных, трёхмерных локально-адаптивных, нерегулярных тетраэдральных.

В целом данный подход упрощает написание программ, решающих задачи на различных сетках. Текст программ становится более структурированным и понятным. Благодаря этому уменьшается

количество возможных ошибок в программах и время на разработку и отладку. При этом программы работают достаточно эффективно и не требуют дополнительной памяти для промежуточных вычислений.

Публикации по теме диссертации

1. Краснов М.М., Феодоритова О.Б. Операторная библиотека для решения трёхмерных сеточных задач математической физики с использованием графических плат с архитектурой CUDA // Препринты ИПМ им. М.В. Келдыша. 2013. № 9. 32 с. URL: http://library.keldysh.ru/preprint.asp?id=2013-09
2. Жуков В.Т., Краснов М.М., Новикова Н.Д., Феодоритова О.Б. Параллельный многосеточный метод: сравнение эффективности на современных вычислительных архитектурах // Препринты ИПМ им. М.В. Келдыша. 2014. № 31. 22 с. URL: http://library.keldysh.ru/preprint.asp?id=2014-31
3. Краснов М.М. Оптимальный параллельный алгоритм обхода точек гиперплоскости фронта вычислений и его сравнение с другими итерационными методами решения сеточных уравнений // Препринты ИПМ им. М.В.Келдыша. 2015. № 20. 20 с. URL: http://library.keldysh.ru/preprint.asp?id=2015-20
4. Жуков В.Т., Краснов М.М., Новикова Н.Д., Феодоритова О.Б. Численное решение параболических уравнений на локально-адаптивных сетках чебышевским методом // Препринты ИПМ им. М.В. Келдыша. 2015. № 87. 26 с. URL: http://library.keldysh.ru/preprint.asp?id=2015-87
5. Краснов М.М., Ладонкина М.Е., Тишкин В.Ф. Разрывный метод Галёркина на трёхмерных тетраэдральных сетках. Использование операторного метода программирования // Препринты ИПМ им. М.В.Келдыша. 2016. № 23. 27 с. URL: http://library.keldysh.ru/preprint.asp?id=2016-23

6. Краснов М.М., Ладонкина М.Е. Разрывный метод Галёркина на трёхмерных тетраэдральных сетках. Применение шаблонного метапрограммирования языка C++
// Препринты ИПМ им. М.В.Келдыша. 2016. № 24. 23 с.
URL: http://library.keldysh.ru/preprint.asp?id=2016-24
7. Краснов М.М. Операторная библиотека для решения трёхмерных сеточных задач математической физики с использованием графических плат с архитектурой CUDA. // Математическое моделирование, 2015, т.27, № 3, с. 109-120.
8. Жуков В.Т., Краснов М.М., Новикова Н.Д., Феодоритова О.Б. Сравнение эффективности многосеточного метода на современных вычислительных архитектурах
// Программирование, 2015, № 1, с. 21-31.
9. Краснов М.М. Параллельный алгоритм вычисления точек гиперплоскости фронта вычислений. // Журнал вычислительной математики и математической физики, 2015, том 55, № 1, с. 145-152.
10. Краснов М.М., Кучугов П.А., Ладонкина М.Е., Тишкин В.Ф. Разрывный метод Галёркина на трёхмерных тетраэдральных сетках. Использование операторного метода программирования.
// Математическое моделирование, 2017 г., т. 29, № 2, с. 3-22 (принято в печать).
11. Краснов М.М., Ладонкина М.Е. Разрывный метод Галёркина на трёхмерных тетраэдральных сетках. Применение шаблонного метапрограммирования языка C++. // Программирование, 2017 г., № 3 (принято в печать).
12. Краснов М.М. Операторная библиотека для решения трёхмерных сеточных задач математический физики с использованием графических плат с архитектурой CUDA. // Национальный суперкомпьютерный форум (НСКФ-2013), Переславль-Залесский,

ИПС им. А.К. Айламазяна РАН, 26-29 ноября 2013 г. URL: http://2013.nscf.ru/TesisAll/Section%206/04_1000_KrasnovMM_S6.pdf

13. Краснов М.М. Несколько примеров переноса программ на графические ускорители CUDA с использованием библиотеки gridmath. // Тезисы докладов XX Всероссийской конференции и Молодёжной школы-конференции «Теоретические основы и конструирование численных алгоритмов решения задач математической физики», посвященная памяти К.И. Бабенко (Дюрсо, 15-20 сентября 2014). – М: Институт прикладной математики им. М.В. Келдыша, 2014, с. 72-73.
14. Краснов М.М. Операторная библиотека для решения задач математической физики на трёхмерных локально-адаптивных сетках с использованием графических ускорителей CUDA. // Международная конференция «Суперкомпьютерные дни в России», 28 - 29 сентября 2015 г., г. Москва.
15. Краснов М.М., Кучугов П.А., Ладонкина М.Е., Тишкин В.Ф. Разрывный метод Галёркина на трёхмерных тетраэдральных сетках. Использование операторного метода программирования. // XIV Международный семинар «Математические модели и моделирование в лазерно-плазменных процессах и передовых научных технологиях», 4 - 9 июля 2016 г., г. Москва. URL: http://lppm3.ru/files/histofprog/LPpM3-2016-1-Programme.pdf
16. Краснов М.М., Кучугов П.А., Ладонкина М.Е., Тишкин В.Ф. Разрывный метод Галёркина на трёхмерных тетраэдральных сетках. Использование операторного метода программирования. // VII всероссийская научная молодежная школа-семинар «Математическое моделирование, численные методы и комплексы программ» имени Е. В. Воскресенского с международным участием, 12-15 июля 2016 г., г. Саранск.

17. Краснов М.М., Кучугов П.А., Ладонкина М.Е., Тишкин В.Ф. Метод Галеркина с разрывными базисными функциями в многомерном случае. // XXI Всероссийская конференция «Теоретические основы и конструирование численных алгоритмов для решения задач математической физики», посвященная памяти К.И.Бабенко, 5-11 сентября 2016 г. г. Новороссийск.
18. Krasnov M.M., Kuchugov P.A., Ladonkina M.E., Tishkin V.F. Application of discontinuous Galerkin method for modeling of three-dimensional problems of flow past solid bodies. // XV International Seminar «Mathematical models & modeling in laser plasma processes & advanced science technologies», 26-30 September, 2016, Montenegro, Petrovac. URL: http://lppm3.ru/files/histofprog/lppm3-2016-2-programme.pdf
19. Краснов М.М., Кучугов П.А., Ладонкина М.Е., Тишкин В.Ф. Обобщение метода Годунова, использующее кусочно-полиномиальные аппроксимации. // XVI Международная конференция «Супервычисления и математическое моделирование», г. Саров, 3-7 октября 2016 (Тезисы докладов)