САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Кафедра компьютерного моделирования и многопроцессорных систем

Сахненко Александр Константинович

Выпускная квалификационная работа бакалавра

Оптимизация и распараллеливание численных алгоритмов для сильно нелинейных волновых процессов

Направление 010300

Фундаментальная информатика и информационные технологии

Научный руководитель,
доктор физ.-мат. наук,
профессор
Богданов А.В.

Санкт-Петербург

2016

Содержание

Введение

Нелинейные явления появляются в природе повсюду: от волн на водной поверхности до магнитных, от оптики до прогнозов погоды. Следовательно, их описание и понимание имеет принципиальное значение, как с теоретической, так и прикладной точки зрения.

Нелинейные явления, как правило, описываются дифференциальными уравнениями, решение которых часто является сложной проблемой. Тем не менее, существует специальный класс дифференциальных уравнений, которые разрешимы (в некотором смысле) - интегрируемые системы. Многие понятия современной математической физики, такие как солитоны, инстантоны и квантовые группы имеют свое происхождение в теории интегрируемых систем. Когда физическое явление описывается интегрируемой системой, ее поведение может быть понято во всем мире и его часто можно предсказать. Красота этой теории заключается в его универсальности: многие фундаментальные нелинейные уравнения оказываются как широко применимы, так и интегрируемы. Кроме того, в ряде случаев неинтегрируемые нелинейные уравнения могут быть приближены, при некоторых предположениях, нелинейными интегрируемыми уравнениями, что позволяет лучше понять явления, моделируемые ими.

Однако апроксимация неинтегрируемых нелинейных уравнений не всегда положительно сказывается на получаемых результатах, и найденное решение отличается от истинного довольно сильно. Решение же самих неинтегрируемых нелинейных уравнений очень трудно даже численно и практически невозможно с помощью стандартных аналитических методов. Особенно трудно получить надежные результаты в асимптотической области.

Множество полезных подходов, предложенных для векторных систем, вряд ли может быть перенесено на существующие кластерные системы. Разработка гетерогенных вычислительных систем, основанных на GPGPU, открывает новые возможности для анализа нелинейных эволюционных уравнений. Но GPGPU еще не векторный ускоритель, так что трудно контролировать и оптимизировать параллельные задачи, а «узкие места» встроенной памяти делают практически невозможным получение надежных результатов для трёхмерных проблем. Поэтому необходимо сделать предварительные испытания для простой проблемы, чтобы осветить все возможные трудности и найти оптимальные численные подходы для будущих оптимизаций алгоритмов.

Постановка задачи

Целью данной работы является оптимизация и распараллеливание алгоритма численного решения дифференциальных уравнений, описывающих сильно нелинейные волновые процессы в гидродинамике, на примере уравнения Бюргерса. Для достижения поставленной цели предполагается решить задачу распараллеливания алгоритма, осонованного на двухшаговом методе численного интегрирования «предиктор-корректор» Мак-Кормака.

Решение данной задачи требует реализации следующих этапов:

1. Анализ предметной области;
2. Обзор архитектуры Kepler графического ускорителя NVIDIA;
3. Построение разностной схемы;
4. Реализация параллельного алгоритма.

Глава 1. Описание предметной области

В данной работе рассматривается процесс непрерывно-корпускулярного моделирования волновых процессов в жидкостях.

Макроскопическое поведение любого материала определяется, в конечном счете, его микроструктурой, которая на молекулярном уровне носит принципиально дискретный характер. О микроструктуре и межмолекулярных взаимодействиях, связанных с определенными материалами, известно довольно много, и поэтому втаёт вопрос: может ли, и до какой степени, такая информация быть использована в описании непрерывных моделей. Первым шагом в этом направлении является обработка дискретных фундаментальных сущностей (например, молекул, атомов и ионов), как частиц и объединение теории непрерывного моделирования с корпускулярными соображениями. Подробнее это описано в книге И.Мёрдока [1].

Весь объём жидкости, при таком подходе, представляется набором крупных «частиц» воды, движение каждой из которых в каждый момент времени характеризуется набором внутренних состояний, инерцией и внешними силами, в том числе, со стороны других частиц (сила натяжения) [2].

Непрерывно-корпускулярный подход основан на численных схемах первого порядка с постоянным шагом в интегрировании законов движения. Каноническое представление о законах механики жидкости позволяет строго и однозначно связать численные объекты с арифметическими и логическими операциями и сложными геометрическими алгоритмами, в том числе с использованием быстрой интерполяции для нерегулярных сеточных пространств.

Основной вектор математики в этой области сейчас направлен на создание прямых вычислительных экспериментов при решении практических

задач механики жидкости. Разделение вычислительных этапов физическими процессами создаёт возможность непрерывного контролирования процесса эволюции моделируемой сплошной среды в соответствии с оценкой её текущего состояния, принимая во внимание интенсивность физического взаимодействия соседних частиц как виртуальных цифровых объектов.

Глава 2. Разностные схемы «предиктор-корректор»

2.1. Описание

Схемы «предиктор-корректор» - семейство методов, относящихся к классу алгоритмов, предназначенных для интегрирования обыкновенных дифференциальных уравнений. Все такие методы включают в себя два шага:

- На первом шаге (предиктор) вычисляется некоторая функция от значений, посчитанных на предыдущем шаге, чтобы получить апроксимированное значение искомой функции в следующей точке.
- На втором шаге (корректор) совершенствуется полученное приближение при помощи спрогнозированного значения функции и другого оператора, чтобы интерполировать значение искомой функции в той же самой точке.

Эту схему можно представить следующей системой:

$$\begin{cases} u^{n+\frac{1}{2}} = f^1(u^n, \dots) \\ u^{n+1} = f^2\left(u^{n+\frac{1}{2}}, \dots\right) \end{cases},$$

где $u^{n+\frac{1}{2}}$ – значение, полученное на шаге «предиктор», а u^{n+1} – уточнённое искомое значение.

Т.о. видно, что первый шаг реализуется с помощью явных методов, а второй шаг основан на применении формул неявных методов, в правую часть которых вместо неизвестного значения u^{n+1} подставляется результат предсказания $u^{n+\frac{1}{2}}$.

Методы, использующие схему предиктор-корректор:

- Метод Милна – для ОДУ;
- Метод Хойна (предиктор – Метод Эйлера, корректор – Метод трапеций);

При использовании схемы предиктор-корректор для решения ОДУ отмечают высокую точность расчета и отсутствие свойства самостартуемости (то есть для начала вычислений по схеме предиктор-корректор требуется предварительно воспользоваться другим, самостартующим методом).

- Метод Адамса-Башфорта – параллельный п.-к. для решения нежестких краевых задач (используется корректор Адамса-Башфорта-Мултона);
- Формулы Хемминга.
- Метод Мак-Кормака.

2.2. Метод Мак-Кормака

Метод Мак-Кормака является измененной двухшаговой схемой Лакса-Вендрофа, но он гораздо проще в применении.

Рассмотрим следующее гиперболическое уравнение первого порядка:

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = 0$$

Предиктор: На этом шаге прогнозируемое значение u в момент времени k + 1 ($\bar{u}_i^{k+1}$) оценивается следующим образом:

$$\bar{u}_i^{k+1} = u_i^k - a\frac{\Delta t}{\Delta x}(u_{i+1}^k - u_i^k)$$

Корректор: На этом шаге предсказанное значение $\bar{u}_i^{k+1}$ корректируется в соответствии с уравнением:

$$u_i^{k+1} = \frac{1}{2}\left[u_i^k + \bar{u}_i^{k+1} - a\frac{\Delta t}{\Delta x}(\bar{u}_i^{k+1} - \bar{u}_{i-1}^{k+1})\right]$$

Отметим, что промежуточное значение $\bar{u}_i^{k+1}$ никакого физического смысла не имеет.

Схема имеет второй порядок точности с погрешностью аппроксимации $O(\Delta t^2,\ \Delta x^2)$, при этом она устойчива при $|C| \leq 1$, а дифференциальное приближение схемы имеет вид:

$$u_t + au_x = -a\frac{\Delta t^2}{\Delta x}(1 - C^2)u_{xxx} - a\frac{\Delta x^3}{8}C(1 - C^2)u_{x^4} + \dots ,$$

где $C = a\frac{\Delta t}{\Delta x}$ – число Куранта.

Для линейного волнового уравнения схема Мак-Кормака эквивалентна схеме Лакса-Вендроффа.

Схема Мак-Кормака часто применяется в силу ряда своих достоинств [3]. В частности, она оперирует только величинами в основных узлах сетки и легко обобщается на многомерные задачи. Также, это схема второго порядка точности. Она обладает лишь малой диффузией, но имеет большую дисперсию и чувствительна к нелинейной неустойчивости. В областях больших градиентов могут возникать нефизические осцилляции. Эти численные колебания часто приводят к отрицательным значениям заведомо положительных величин.

Глава 3. Обзор архитектуры Kepler

Компания NVIDIA вот уже почти целое десятилетие выпускает видеокарты для общих вычислений. Нельзя сказать, что раньше было невозможно производить вычисления на GPU, но этот процесс был довольно неудобен и требовал большого количества времени и определённых навыков. В 2007 году была предложена универсальная архитектура параллельных вычислений (CUDA), позволяющая благодаря использованию GPU повысить производительность вычислений. Это дало возможность наращивать мощность устройств путём увеличения количества их составляющих, при этом не изменяя основные особенности архитектуры. Таким образом, объём памяти и количество процессоров постоянно увеличиваются, а вот разделение памяти на глобальну, разделяемую, текстурную и регистровую остаётся однообразным, неизменным в каком-то смысле (позже появились и другие структуры памяти, такие как surface или разные типы кэша, но это только расширило возможности CUDA).

Однако всё должно развиваться и архитектура и набор команд не стоят на месте и со временем модифицировались – иногда они изменялись значительно, иногда не очень. Для классификации семейств GPU с идентичной архитектурой было введено понятие «Compute capability» (CC). Для сравнения: процессоры с CC 1.0 не поддерживали атомарных операций в принципе, с CC 1.1 – могли производить их в глобальной памяти, а с CC 1.2 – ещё и в shared. Полный список возможностей разных CC традиционно приводится в конце CUDA C Programming Guide [4].

В 2012 году NVIDIA представила архитектуру Kepler (CC 3.0). Она рассчитана на энергоэффективность, программируемость и производительность, в отличае от предыдущей архитектуры, Fermi, основная направленность которой была сконцентрирована на чистой производительности.

Уменьшение энергопотребления происходит благодаря тому, что шейдерные блоки работают на одной частоте с ядром. Это позволило снизить энергопотребление даже при том, что необходимо использовать большее количество шейдерных ядер для получения такой же производительности, которая достигалась с помощью предыдущих разработок. Энергоэффективность достигается не только тем, что новая архитектурапотребляет меньше энергии, чем архитектура предыдущего поколения (двум шейдерным ядрам Kepler требуется приблизительно 90% питания, потребляемого одним ядром Fermi), но и благодаря тому, что унификация тактовой частоты приводит к снижению частоты шейдерных блоков, что в свою очередь серьёзно снижает энергопотребление[5].

Улучшенная программируемость была достигнута за счёт введения новой модели обработки текстур, которая не требует привязки к CPU.

Улучшение же производительности было достигнуто за счёт внедрения абсолютно новых контроллера памяти и шины. В свою очередь это позволило поднять тактовую частоту памяти до 6 ГГц, что всё ещё ниже, чем теоретически максимальные для GDDR5 7 ГГц, но значительно больше, чем частота памяти в 4 ГГц при архитектуре предыдущего поколения [6].

Также, архитектуру Kepler отличает от предшественников новый огромный мультипроцессор. Если раньше мультипроцессоры имели 8 (CC 1.x), 32 (CC 2.0) или 48 (CC 2.1) потоковых процессоров, то в Kepler используется новый чип на 192 процессора. Другие характеристики в сравнении с архитектурой Fermi представлены ниже (Таблица.1).

Таблица 1. Сравнение характеристик процессоров архитектуры Kepler и Fermi

	FERMI GF100	**FERMI GF104**	**KEPLER GK104**	**KEPLER GK110**
Версия CC	2.0	2.1	3.0	3.5
Потоков в Warp'е	32	32	32	32
Число warp'ов на мультипроцессор	48	48	64	64
Потоков на мультипроцессор	1536	1536	2048	2048
Блоков на мультипроцессор	8	8	16	16
32-битные регистры на мультипроцессор	32768	32768	65536	65536
Максимальное количество регистров на поток	63	63	63	255
Конфигурации количества shared-памяти	16K 48K	16K 48K	16K 32K 48K	16K 32K 48K
Максимальный размер grid по оси X	2^16-1	2^16-1	2^16-1	2^32-1
Hyper-Q	Нет	Нет	Нет	Есть
Динамический параллелизм	Нет	Нет	Нет	Есть

Чтобы извлечь наибольшую выгоду из использования потокового мультипроцессора (SMX), т.е. максимизировать количество одновременно загруженных блоков/потоков, нужно использовать гораздо меньше регистров, чем при использовании архитектуры предыдущего поколения. С другой стороны, Kepler позволяет хранить до 255 регистров на поток, и если раньше было справедливо утверждение «лучше заново вычислить, чем хранить», то сейчас оно подвергается сомнению. Таким образом, имеет место некая «развилка» на пути оптимизации – можно записывать результаты

вычислений в регистры, но тогда придётся пожертвовать количеством блоков, которые могут поместиться на SMX, что, вполне вероятно, приведёт к простоям на чтение/запись в RAM.

Ещё несколько заметных нововведений:

1. Новый Warp Scheduler. На каждый мультипроцессор приходятся 4 планировщика, и каждый из них умеет исполнять 2 инструкции варпа за такт;
2. Dynamic Parallelism – позволяет ядру CUDA создавать и синхронизировать вложенные задачи, используя CUDA во время выполнения API для запуска других ядер, управлять памятью устройства, а также создавать и использовать потоки и события, и все это без участия центрального процессора;
3. GPUDirect – данная технология повышает скорость передачи данных между GPU и другими устройствами, обходя использование CPU, в частности – обеспечивает между графическими процессорами, находящимися в одной системе, peer-to-peer (P2P) соединение;
4. Shuffle Instruction – благодаря этим инструкциям параллельные потоки могут считывать значения регистров друг друга;
5. Read-only кэш данных – Kepler предоставляет возможность отмечать данные как read-only, чтобы компилятор помещал их в соответствующий кэш;
6. Hyper-Q – блок, позволяющий сразу нескольким ядрам процессора работать с GPU.
7. Grid Management Unit – модуль, призванный утилизировать ресурсы GPU, что приводит к увеличению одновременно выполняемых работ.

Подробнее это описано в [4].

Глава 4. Построение разностной схемы

4.1. Конечно-разностная схема для уравнения КдВБ

Рассмотрим уравнение Кортевега – де Вриза – Бюргерса (КдВБ), которое является универсальным для описания одномерных нелинейных волн в средах с дисперсией и диссипацией:

$$u_t + uu_x + \alpha u_{xx} + \beta u_{xxx} = \gamma I(u), \quad (1)$$

где α – мера диссипации, β – мера дисперсии, γ – мера межфазных взаимодействий, а $I(u)$ – интегральный оператор.

При интегрировании уравнения КдВБ, вместо исходного уравнения мы используем его эквивалент, написанный в дивергентной форме:

$$u_t + 0.5(u^2)_x + \alpha u_{xx} + \beta u_{xxx} = \gamma I(u), \quad (2)$$

Решение данного уравнения в полуплоскости $t \geq 0$ ищется для начального распределения $u(x,0) = f(x),\ x \in (-\infty; +\infty)$. Граничные условия – условия Дирихле: $u(-\infty, t) =$ a, $u(+\infty, t) = b$.

В основном, численное моделирование уравнения (2) проводилось с использованием двухступенчатой явной разностной схемы Мак-Кормака [7][8] вместе с методом коррекции потоков [9]. Также есть и другие численные подходы, в частности обобщение Залесака для гиперболических систем [10].

Вычисления выполняются в области $[x_{min}, x_{max}] \times [0, T]$. Сетка конечных разностей по времени t и пространству координаты x строится следующим образом:

$$u_j^n = u(x_j, t^n),$$
$$x_j = j\Delta x, \qquad j \in [1, M],\ x_1 = x_{min},\ x_M = x_{max},$$
$$t^n = n\Delta t, \qquad n = 0,1, \ldots,$$

где Δx является шагом пространственной координаты; Δt – шаг по времени. Граничные условия: $u(x_{min}, t) =$ a, $u(x_{max}, t) = b$.

Разностная схема Мак-Кормака для уравнения (2) имеетследующий вид ($r = \frac{\alpha \Delta t}{\Delta x 2}$, $s = \frac{\beta \Delta t}{2 \Delta x 3}$):

Шаг 1. Предиктор.

$$u_j^* = u_j^n - \frac{\Delta t}{2\Delta x}\left[\left(u_{j+1}^n\right)^2 - \left(u_j^n\right)^2\right] - r\left(u_{j+1}^n - 2u_j^n + u_{j-1}^n\right) -$$
$$-s\left(u_{j+2}^n - 2u_{j+1}^n + 2u_{j-1}^n + u_{j-2}^n\right) + \gamma \Delta t I_j^n$$

Шаг 2. Корректор.

$$\overline{u} = \frac{u_j^n + u_j^*}{2} - \frac{\Delta t}{4\Delta x}\left[\left(u_j^*\right)^2 - \left(u_{j-1}^*\right)^2\right] - \frac{r}{2}\left(u_{j+1}^* - 2u_j^* + u_{j-1}^*\right) -$$
$$-\frac{s}{2}\left(u_{j+2}^* - 2u_{j+1}^* + 2u_{j-1}^* + u_{j-2}^*\right) + \gamma \frac{\Delta \mathrm{t}}{2} I_j^n$$

Приближение переносного члена в шаге «предиктор» для нечетных шагов по времени выполняется разностями вперед, а для четных шагов – разностями назад. Таким образом, ошибка фазы уменьшается в расчетах, что приводит к движению волновых фронтов в правильном направлении.

Известно, что схема Мак-Кормака, повышая при этом порядок аппроксимации по сравнению с монотонными схемами, порождает в окрестности сильных скачков колебаня, которые носят нефизический смысл.

Метод коррекции потоков, согласно [8], включает в себя следующие этапы:

1. Вычисление диффузионных потоков:

$$u_{j+\frac{1}{2}}^d = \nu_{j+\frac{1}{2}}\left(u_{j+1}^n - u_j^n\right)$$

2. Вычисление антидиффузионных потоков:

$$u_{j+\frac{1}{2}}^{ad} = \mu_{j+\frac{1}{2}}\left(\overline{u}_{j+1} - \overline{u}_j\right)$$

Коэффициенты $\nu_{j+\frac{1}{2}}$ и $\mu_{j+\frac{1}{2}}$ вычисляются следующим образом:

$$\nu_{j+\frac{1}{2}} = \frac{1}{6} + \frac{1}{3}C^2_{j+\frac{1}{2}}, \qquad \mu_{j+\frac{1}{2}} = \frac{1}{6} - \frac{1}{3}C^2_{j+\frac{1}{2}}, \qquad C_{j+\frac{1}{2}} = \frac{u_j^n + u_{j+1}^n}{2}\frac{\Delta t}{\Delta x}.$$

После корректирующего шага мы получаем отрегулированное значение u_j^{corr}. Окончательное решение на $n+1$ шаге имеет вид:

$$u_j^{n+1} = \overline{u}_j + u^d_{j+\frac{1}{2}} - u^d_{j-\frac{1}{2}} - u_j^{corr} + u_{j-1}^{corr}.$$

4.2. Начальные распределения

При расчетах в качестве начальных профилей используются три распределения:

1. "Ударная волна" (прыжок): $u(x \geq x_0, 0) = u_1$; $u(x < x_0, 0) = u_2$. Скорость сдвига скачка: $U = (u_1 + u_2)/2 = Const$.

2. Солитон. Известное стационарное решение уравнения КдВ:

$$u(x,0) = f(x) = \varepsilon \cosh^{-2} x/\delta \ , \qquad \delta = \sqrt{12\beta/\varepsilon}\,.$$

3. Прямоугольный профиль: $u(x,0) = h \times w$, при $x \in \left[-\frac{w}{2}, \frac{w}{2}\right]$. Выбор между h и w продиктован законом сохранения «массы» для исходного профиля:

$$\int_{-\infty}^{\infty} f(x)dx = \int_{-\infty}^{\infty} \varepsilon \cosh^{-2} x/\delta \ = 4\sqrt{3\varepsilon\delta} = h \times w \qquad (3)$$

При рассмотрении уравнения КдВ, выражение (3) имеет вид: $h \times w = 12$. Для поддержания этого равенства в расчете, конкретное значение w также должно быть связано с шагом по координате x.

4.3. Чистый перенос

Рассмотрим частные случаи исходного уравнения (1): чистый перенос, уравнение Бюргерса, уравнение КдВ. Базовые свойства разностной схемы (устойчивость, порядок апроксимации и т.п.) описаны в работах [7], [8]

В случае чистого переноса уравнение (2) записывается в виде:

$$u_t + 0.5(u^2)_x = 0. \qquad (4)$$

Конечно-разностная схема для чистого переноса традиционно тестируется «ударной волной». Пусть скачок имеет единичную амплитуду: $u_1 = 1, u_2 = 0, x_0 = 0$. Тогда скорость сдвига скачка $U = 1/2$.

Уравнение (4) запишется в линейной форме как: $u_t + Uu_x = 0$.

На Рисунке 1 показаны результаты численных вычислений для 125 шагов, C=0.25 и $x = Ut = 62.5$. Результаты численных вычислений и аналитическое решение для «ударной волны» доволно похожи.

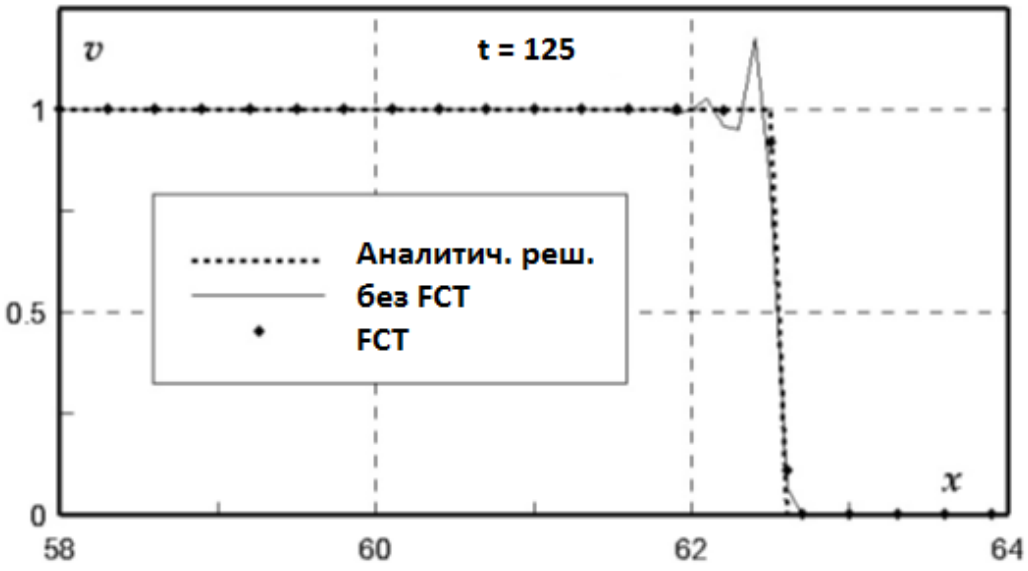

Рисунок 1. Сравнение результатов для распределения "ударной волны"

Для линейного случая (4), с использованием метода скорректированных потоков, существует известное условие устойчивости: $C \leq 1/2$, где $C = U\Delta t/\Delta X$ – это число Куранта, а Δt и Δx – шаги по времени и координате. Для нелинейного случая условие устойчивости преобразуется в следующее:

$$\left|\max_{\forall j,n}\frac{u_j^n\,(\Delta t)}{\Delta x}\right| \le 1/2.$$

Из формулы (3) хорошо видно влияние ошибки дисперсии в зависимости от числа Куранта. Без алгоритма коррекции потоков, мы не можем получить правильное численное решение.

Вычисления показывают, что при шаге $\Delta x = 0.1$ предельная величина шага по времени $\Delta t = 0.071$ ($C = 0.355$).

4.4. Уравнение Бюргерса

При $\beta = \gamma = 0$ уравнение (2) записывается в виде уравнения Бюргерса:

$$u_t + uu_x + \alpha u_{xx} = 0 \qquad (5)$$

Для уравнения (5) существует множество точных решений [11],[12]. В данном случае для тестирования схемы будет использоваться стационарное аналитическое решение:

$$u = \frac{u_2 - u_1}{1 + \exp\left(-\frac{u_2 - u_1}{2\alpha}\xi\right)}, \qquad \xi = x - Ut.$$

Как и в случае чистого переноса для начального распределения, мы возьмем единичный скачок: $u_1 = 1,\ u_2 = 0, x_0 = 0$. На Рисунке 2 аналитическое решение сравнивается с эволюцией единичного скачка и со скачком без влияния вязкости.

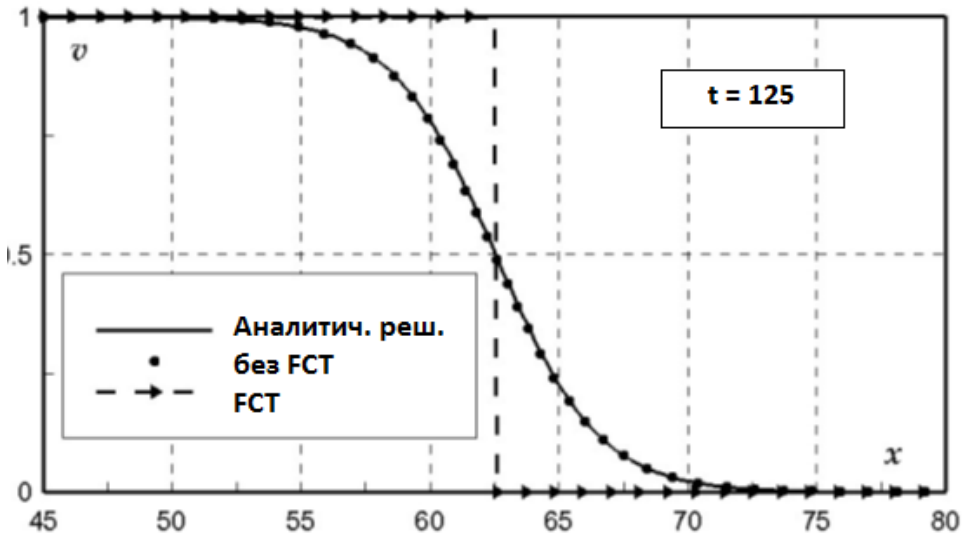

Рисунок 2. Сравнение числовых и аналитических решений для уравнения Бюргерса

Доминирующий диссипативный эффект – физическая диффузия, которая подавляет числовую ошибку дисперсии.

4.5. Уравнение Кортевега – де Вриза

При $\alpha = \gamma = 0$ уравнение (2) записывается в виде уравнения КдВ:

$$u_t + 0.5(u^2)_x + \beta u_{xxx} = 0 \qquad (6)$$

Уравнение (6) не имеет диссипативных членов, и поэтому численные результаты очень чувствительны к ошибкам аппроксимации, особенно фазовым ошибкам. И не только члены с производными первого порядка имеют эффект ошибки аппроксимации, но и с производными высших порядков.

Непосредственное применение описанного выше метода конечных разностей приводит к ошибочным результатам. Это связано с тем, что процедура коррекции сама имеет диффузию сетки. Результаты, полученные при применении процедуры коррекции, как и ожидалось, являются вполне удовлетворительными.

Таким образом, простое уменьшение шага по времени может существенно повлиять на получаемое решение. Скорость солитона распространения U = ε / 3 для стационарных решений (ε = 12) равна 4 ($r = \beta\Delta t/(2\Delta x^3)$). Скорость движения солитона в случае 2 на рис. 3 составляет около 3,97, относительная погрешность равна 0,64%, что является вполне приличным результатом. Относительная погрешность расчетной амплитуды солитона составляет 0,58%.

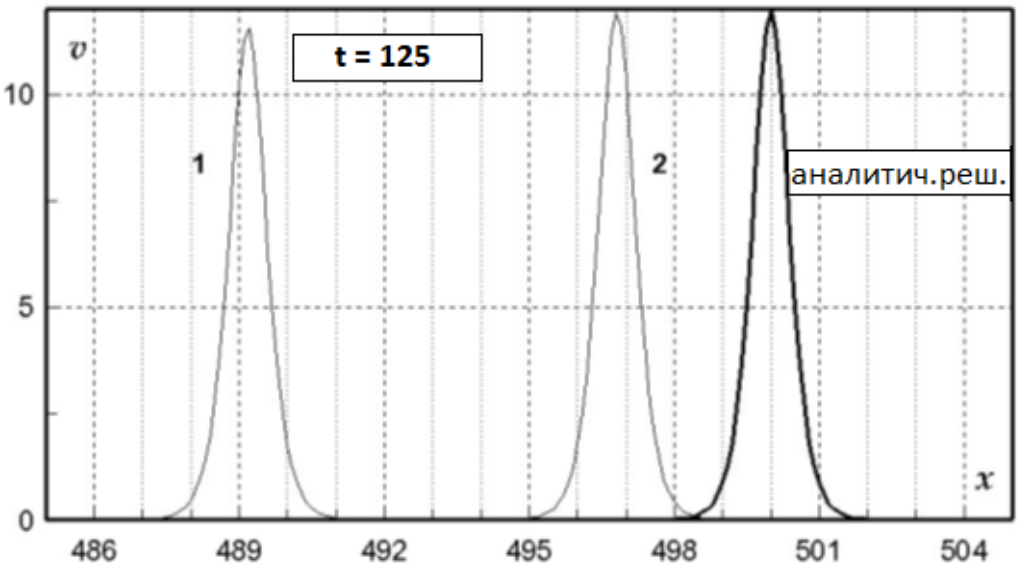

Рисунок 3.Сравнение решений для различных временных шагов: $1 - r = 6.25 \cdot 10^{-3};\ 2 - r = 6.25 \cdot 10^{-4}$

Глава 5. Реализация алгоритма

В данной главе мы рассмотрим реализацию алгоритма решения уравнения Бюргерса:

$$u_t + uu_x + \alpha u_{xx} = 0$$

Для реализации алгоритма была выбрана технология CUDA и язык программирования C++. Также, при работе с этой технологией полезными могут оказаться такие работы, как [4], [13], [14], [15].

5.1. Работа с памятью

Константы, необходимые для нахождения решения, помещаются в конмтантную память (*constant*) – кэшируемую область DRAM, доступ к которой у GPU есть только на чтение, а у хоста (CPU) – на чтене и запись при помощи функции CUDA API:

```
cudaError_t cudaMemcpuToSymbol(
      const char * symbol, const void * src,
      size_t count, size_t offset,
      enum cudaMemcpuKind kind);
```

Параметр *kind* принимает одно из следующих значений, задающих направление копирования:

- *cudaMemcpuHostToHost;*
- *cudaMemcpuHostToDevice;*
- *cudaMemcpuDeviceToHost;*
- *cudaMemcpuDeviceToDevice*

Несмотря на то, что константная память находится в DRAM, она имеет высокую скорость доступа благодаря наличию независимого кэша.

В рассматриваемом алгоритме используются следующие константы (здесь *dt* – шаг по времени, *dx* – шаг по координате *x)*:

- tx = dt/(2*dx) – отношение шага по времени к шагу по координате, делённое на 2 (используется на этапе «предиктор»);

- tx2 = dt/(4*dx) – отношение шага по времени к шагу по координате, делённое на 4 (используется на этапе «корректор»);
- atx=alf*ta/pow(dx,2) – отношение шага по времени к шагу по координате в квадрате, умноженное на параметр α (используется на этапе «предиктор»)
- atx2=alf*ta/(2*pow(dx,2)) отношение шага по времени к шагу по координате в квадрате, делённое на 2 и умноженное на параметр α (используется на этапе «корректор»);
- istep – номер шага по времени.

Константная память выделяется через статическое объявление в коде с добавлением атрибута *__constnat__:*

```
__constant__ float    tx;
__constant__ float    tx2;
__constant__ float    atx;
__constant__ float    atx2;

float host_tx         = dt / (2*dx);
float host_tx2   = dt / (4*dx);
float host_atx   = alf * dt / pow(dx,2);
float host_atx2  = alf * dt / (2*pow(dx,2));

cudaMemcpuToSymbol(tx, host_tx,
           sizeof(data), 0, cudaMemcpuHostToDevice);
cudaMemcpuToSymbol(tx2, host_tx2,
           sizeof(data), 0, cudaMemcpuHostToDevice);
cudaMemcpuToSymbol(atx, host_atx,
           sizeof(data), 0, cudaMemcpuHostToDevice);
cudaMemcpuToSymbol(atx2, host_atx2,
           sizeof(data), 0, cudaMemcpuHostToDevice);
```

В ходе выполнения алгоритма для хранения полученных промежуточных и окончательных значений используются следующие массивы:

- *u0* – массив значений функции u на входе итерации;
- *us* – массив значений u^* послевыполнения шага «предиктор»;

- *u1* – массив промежуточных значений функции *u* и на выходе из итерации;
- *cu_d* – массив значений диффузионных коэффициентов на этапе коррекции потоков;
- *cu_ad* – массив значений антидиффузионных коэффициентов на этапе коррекции потоков;
- *u_d* – массив значений диффузионого потока на этапе коррекции потоков;
- *u_ad* – массив значений антидиффузионого потока на этапе коррекции потоков;
- *d_u* – массив первых разностей.

Все данные массивы хранятся в глобальной памяти GPU, выделение памяти и копирование в неё данных происходит следующим образом (здесь и далее *n* – количество точек):

```
int nb = n * sizeof(float);
float* u0 = NULL;
cudaError_t cuerr = cudaMalloc( (void**)&u0, nb);
cuerr = cudaMemcpy(u0, host_u0, cudaMemcpuHostToDevice);
```

При этом значение переменной *cuerr* будет содержать информацию, об ошибках при выполнении функции (значение *cudaSuccess* соответствует успешному выполнению операции).

5.2. Функции ядра

Фунции ядра описываются с атрибутом *__global__*, ниже описаны некоторые из них:

1. Предиктор:

```
__global__ void predictor_kernel (float* u0, float* us)
{
      int j = threadIdx.x + blockIdx.x * blockDim.x + 1;
      int jl = j-1;
      int jr = j+1;

      us[j] = u0[j] - tx * (pow(u0[j],2) - pow(u0[jl],2))
                  - atx*(u0[jr]-2*u0[j]+u0[jl]);
}
```

2. Корректор:

```
__global__ void corrector_kernel (float* us, float* u1)
{
      int j = threadIdx.x + blockIdx.x * blockDim.x + 1;
      int jl = j-1;
      int jr = j+1;

      u1[j] = 0.5 * (u0[j]+us[j]) -
            tx2 * (pow(us[jr], 2) - pow(us[j], 2)) -
            atx2 * (us[jr] - 2*us[j] + us[jl]);
}
```

3. Функция вычисления коэффициентов для метода коррекции потоков:

```
__global__ void coef_fct_kernel (float* u0, float* cu_d, float*
cu_ad)
{
      int j = threadIdx.x + blockIdx.x * blockDim.x + 1;
      int jr = j+1;

      float ur   = u0[j] + u0[jr];
      float txx  = pow(tx*ur, 2)/2;

      cu_d[j]     = 1.0/6.0 + 1.0/3.0 * txx;
      cu_ad[j] = 1.0/6.0 + 1.0/3.0 * txx;
}
```

4. Вычисление диффузионного и антидиффузионного потоков:

```
__global__ void fct_kernel(float* u0, float* u1,
                                float* cu_ad, float* cu_d,
                                float* u_ad, float* u_d)
{
      int j = threadIdx.x + blockIdx.x * blockDim.x + 1;
      int jr = j+1;

      u_d[j]     = cu_d[j] * (u0[jr] - u0[j]);
      u_ad[j]    = cu_ad[j] * (u1[jr] - u1[j]);
}
```

5. Вычисление первых разностей для метода коррекции потоков:

```
__global__ void d_fct_kernel(float* u1, float* u_d)
{
      int j = threadIdx.x + blockIdx.x * blockDim.x + 1;
      int jl = j-1;

      u1[j] = u1[j] + u_d[j]-u_d[jl];
}
```

5.3. Запуск алгоритма

Далее следует задать конфигурацию запуска *n* нитей и вызывать функции ядра для обработки данных согласно алгоритму, описанному в Главе 4:

```
dim3 threads = dim3(BLOCK_SIZE,1);
dim3 blocks  = dim3((n-2) / BLOCK_SIZE, 1);

predictor_kernel<<<blocks, threads>>> (u0, us);
cuerr = cudaDeviceSynchonize();

corrector_kernel<<<blocks, threads>>> (us, u1);
cuerr = cudaDeviceSynchonize();

coef_fct_kernel<<<blocks, threads>>> (u0, cu_d, cu_ad);
cuerr = cudaDeviceSynchonize();

coef_fct_kernel<<<blocks, threads>>> (us, u1);
cuerr = cudaDeviceSynchonize();
```

```
fct_kernel<<<blocks, threads>>> (u0, u1, cu_ad, cu_d, u_ad, u_d);
cuerr = cudaDeviceSynchonize();

d_fct_kernel<<<blocks, threads>>> (u1, u_d);
cuerr = cudaDeviceSynchonize();

du_fct_kernel<<<blocks, threads>>> (u1, d_u);
cuerr = cudaDeviceSynchonize();

correct_ad_fct_kernel<<<blocks, threads>>> (u_ad, d_u);
cuerr = cudaDeviceSynchonize();

ad_fct_kernel<<<blocks, threads>>> (u1, u_ad);
cuerr = cudaDeviceSynchonize();
```

После окончания алгоритма результаты выгружаются из памяти GPU в память CPU.

Выводы

Основной целью данной работы была реализация параллельного алгоритма для сильно нелинейных волновых процессов.

Для достижения поставленной цели был изучен непрерывно-корпускулярный подход к моделированию сильно-нелинейных волновых процессов в гидродинамике, проанализирована архитектура Kepler графического ускоритля NVIDIA и построена разнстная схема решения дифференциального уравнения в частных производных.

Применение различных разностных схем в задачах решения дифференциальных уравнений в частных производных требует огромного количества операций над данными одинаковой структуры, а значит, эти задачи обладают высокой степенью параллелилизма, что побуждает использовать GPGPU для их решения.

В ходе проделанной работы был реализован алгоритм решения дифференциального уравнения в частных производных Бюргерса, которое моделирует нелинейные волновые процессы в средах с диффузией и диссипацией. Алгоритм основан на разностной схеме Мак-Кормака и использует технологию CUDA для высокопроизводительных вычислений на GPU с архитектурой Kepler.

Реализованный алгоритм является примером использования параллельных вычислений для задач моделирования нелинейных волн и может быть использован для построения алгоритмов решения более сложных уравнений.

Заключение

В настоящее время графические ускорители приобретают всё большую популярность, причём они используются не только для решения специализированных задач компьютерной графики или обработки видео, но и для общих вычислений.

Одной из задач, где возможно использование вычислений на GPU является решение дифференциальных уравнений в частных производных. Отдельно стоит подчеркнуть, что это очень трудоёмкий процесс, а производительности CPU недостаточно для решения задач моделирования нелинейных волновых процессов в реальном времени, что делает более предпочтительным использование высокопроизводительных систем и параллельных алгоритмов.

В данной работе был показан процесс построения такого алгоритма для решения дифференциального уравнения в частных производных, описывающего нелинейные волновые процессы в гидродинамике.

Список литературы

1. Murdoch A. I. A corpuscular approach to continuum mechanics: basic considerations //Analysis and Thermomechanics. – Springer Berlin Heidelberg, 1987. – С. 81-111.
2. Bogdanov A., Khramushin V. Tensor Arithmetic, Geometric and Mathematic Principles of Fluid Mechanics in Implementation of Direct Computational Experiments //EPJ Web of Conferences. – EDP Sciences, 2016. – Т. 108. – С. 02013.
3. Pulliam T. H., Zingg D. W. Fundamental algorithms in computational fluid dynamics. – Switzerland : Springer, 2014. – С. 84-85.
4. Cuda C. Programming guide. – 2012.
5. Круглов В. Н., Папуловская Н. В., Чирышев А. В. Преимущества совместного использования CPU И CUDA-устройства //Фундаментальные исследования. – 2014. – №. 8-2.
6. Kepler (microarchitecture) на Wikipedia https://en.wikipedia.org/wiki/Kepler_(microarchitecture)
7. Pletcher R. H., Tannehill J. C., Anderson D. Computational fluid mechanics and heat transfer. – CRC Press, 2013, 774 p.
8. Fletcher C. Computational techniques for fluid dynamics 2: Specific techniques for different flow categories. – Springer Science & Business Media, 2012, 496 p.
9. Boris J. P., Book D. L. Flux-corrected transport. I. SHASTA, A fluid transport algorithm that works //Journal of computational physics. – 1973. – Т. 11. – №. 1. – С. 38-69.
10. Zalesak S. T. Fully multidimensional flux-corrected transport algorithms for fluids //Journal of computational physics. – 1979. – Т. 31. – №. 3. – С. 335-362.

11. Benton E. R., Platzman G. W. A table of solutions of the one-dimensional Burgers equation //Quarterly of Applied Mathematics. – 1972. – С. 195-212.
12. Bogdanov A., Stankova E., Mareev V. High performance algorithms for multiphase and multicomponent media //14th Ship stability workshop, UTMSPACE, Malaysia. – 2014. – С. 242-245.
13. Боресков А. В. и др. Параллельные вычисления на GPU //Архитектура и программная модель CUDA. М.: Изд-во Московского университета. – 2012.
14. Боресков А., Харламов А. Основы работы с технологией CUDA. – Litres, 2015.
15. Сандерс Д., Кэндрот Э. Технология CUDA в примерах: введение в программирование графических процессоров //М.: ДМК Пресс. – 2011. – Т. 232.