Примеры реализации алгоритмов и решения задач на GPU

Храмченков Э.М.
Казанский федеральный университет

Computer Science клуб, Казань, 21.10-23.10, 2015 г.

Параллельные алгоритмы

* Для **эффективного** использования GPU необходимы **оптимизированные** под них параллельные **алгоритмы**

Параллельные алгоритмы

* Для **эффективного** использования GPU необходимы **оптимизированные** под них параллельные **алгоритмы**
* Для **некоторых задач** параллельная реализация **очевидна,** для других или **невозможна** или **затруднительна**

Параллельные алгоритмы

* Для **эффективного** использования GPU необходимы **оптимизированные** под них параллельные **алгоритмы**
* Для **некоторых задач** параллельная реализация **очевидна,** для других или **невозможна** или **затруднительна**
* Пример – **вычисление суммы элементов** массива (редукция относительно сложения):
  * **Тривиальная** реализация на **CPU**
  * **Параллельная** редукция - **?**

Параллельные алгоритмы

* Для **эффективного** использования GPU необходимы **оптимизированные** под них параллельные **алгоритмы**
* Для **некоторых задач** параллельная реализация **очевидна,** для других или **невозможна** или **затруднительна**
* Пример – **вычисление суммы элементов** массива (редукция относительно сложения):

Параллельные алгоритмы

* Для **эффективного** использования GPU необходимы **оптимизированные** под них параллельные **алгоритмы**
* Для **некоторых задач** параллельная реализация **очевидна,** для других или **невозможна** или **затруднительна**
* Пример – **вычисление суммы элементов** массива (редукция относительно сложения):
  * **Тривиальная** реализация на **CPU**

Параллельные алгоритмы

* Для **эффективного** использования GPU необходимы **оптимизированные** под них параллельные **алгоритмы**
* Для **некоторых задач** параллельная реализация **очевидна,** для других или **невозможна** или **затруднительна**
* Пример – **вычисление суммы элементов** массива (редукция относительно сложения):
  * **Тривиальная** реализация на **CPU**
  * **Параллельная** редукция - **?**

Параллельная редукция

* **Параллелизация** методом **«разделяй и властвуй»:**

Параллельная редукция

* **Параллелизация** методом **«разделяй и властвуй»:**
  * **Массив** заносится в **разделяемую память**

Параллельная редукция

* **Параллелизация** методом **«разделяй и властвуй»:**
  * **Массив** заносится в **разделяемую память**
  * **Разделим** исходный массив на **части**

Параллельная редукция

* **Параллелизация** методом **«разделяй и властвуй»:**
  * **Массив** заносится в **разделяемую память**
  * **Разделим** исходный массив на **части**
  * Каждая **часть** соответствует **одному блоку** сетки блоков

Параллельная редукция

* **Параллелизация** методом **«разделяй и властвуй»:**
  * **Массив** заносится в **разделяемую память**
  * **Разделим** исходный массив на **части**
  * Каждая **часть** соответствует **одному блоку** сетки блоков
  * Внутри блока по отдельным **потокам** делим на **пары элементов,** потом на пары из их частичных сумм и т.д.

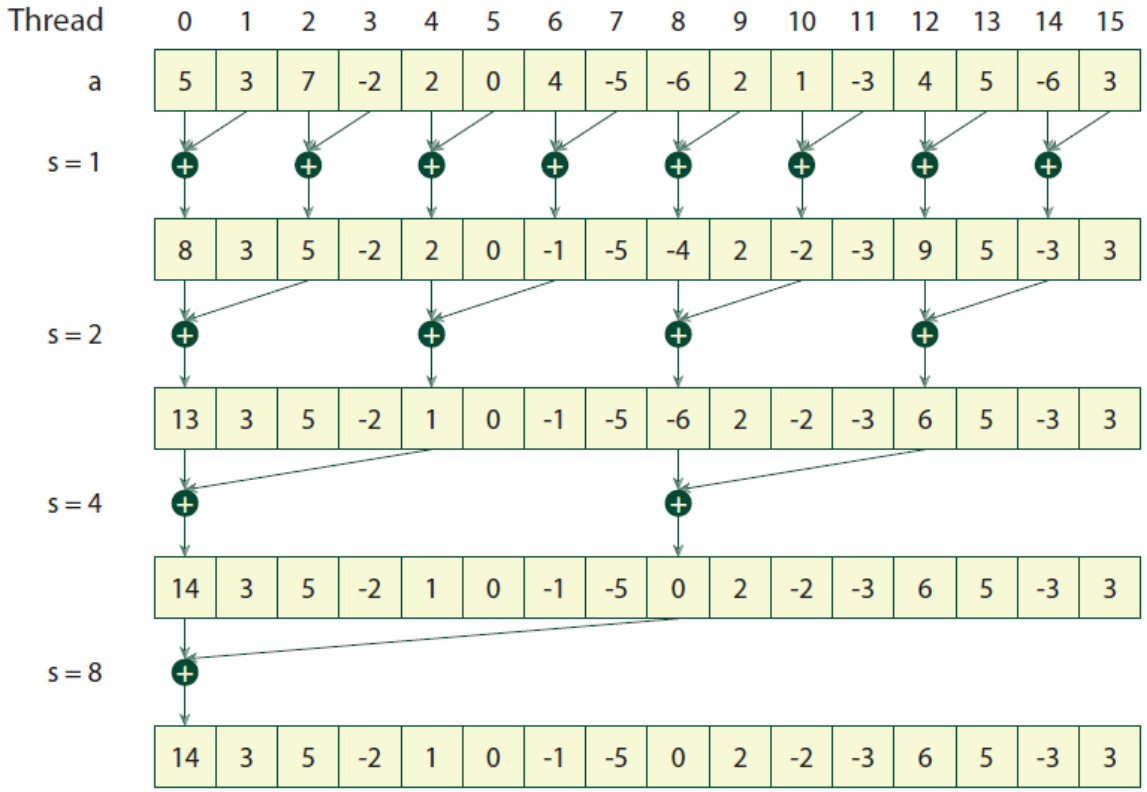
Параллельная редукция

* **Параллелизация** методом **«разделяй и властвуй»:**
  * **Массив** заносится в **разделяемую память**
  * **Разделим** исходный массив на **части**
  * Каждая **часть** соответствует **одному блоку** сетки блоков
  * Внутри блока по отдельным **потокам** делим на **пары элементов,** потом на пары из их частичных сумм и т.д.
  * На **каждом шаге** суммирования **число** элементов **уменьшается вдвое**

Параллельная редукция

reduce_1cpp

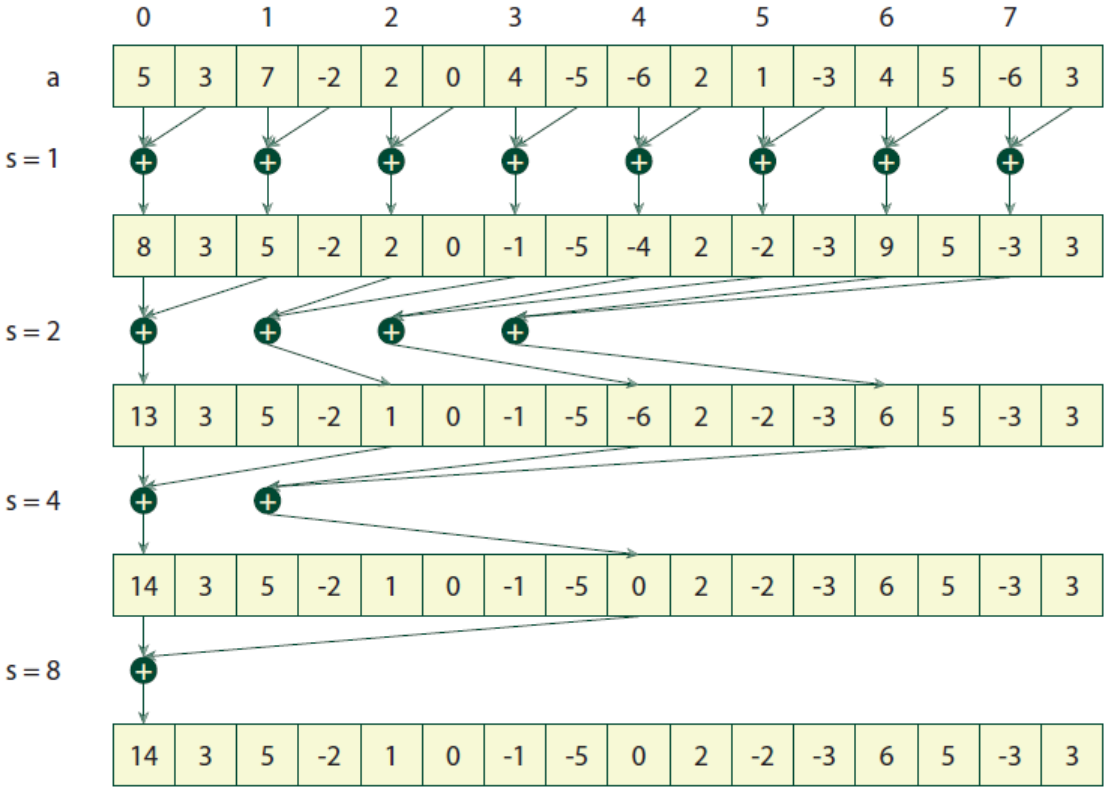
Схема редукции



Параллельная редукция

* **Недостаток** – **большое** количество **итераций** по s с **ветвлением**
* % – медленный оператор
* **Избыточные вычисления** можно **исключить, перераспределив** данные и операции

Схема редукции 2.0



Параллельная редукция

reduce_2.cpp

Параллельная редукция

* **Недостаток** – при такой реализации будут **конфликты доступа** к разделяемой памяти из за особенностей архитектуры

Параллельная редукция

* **Недостаток** – при такой реализации будут **конфликты доступа** к разделяемой памяти из за особенностей архитектуры
* Можно избавиться от конфликтов **изменив порядок**

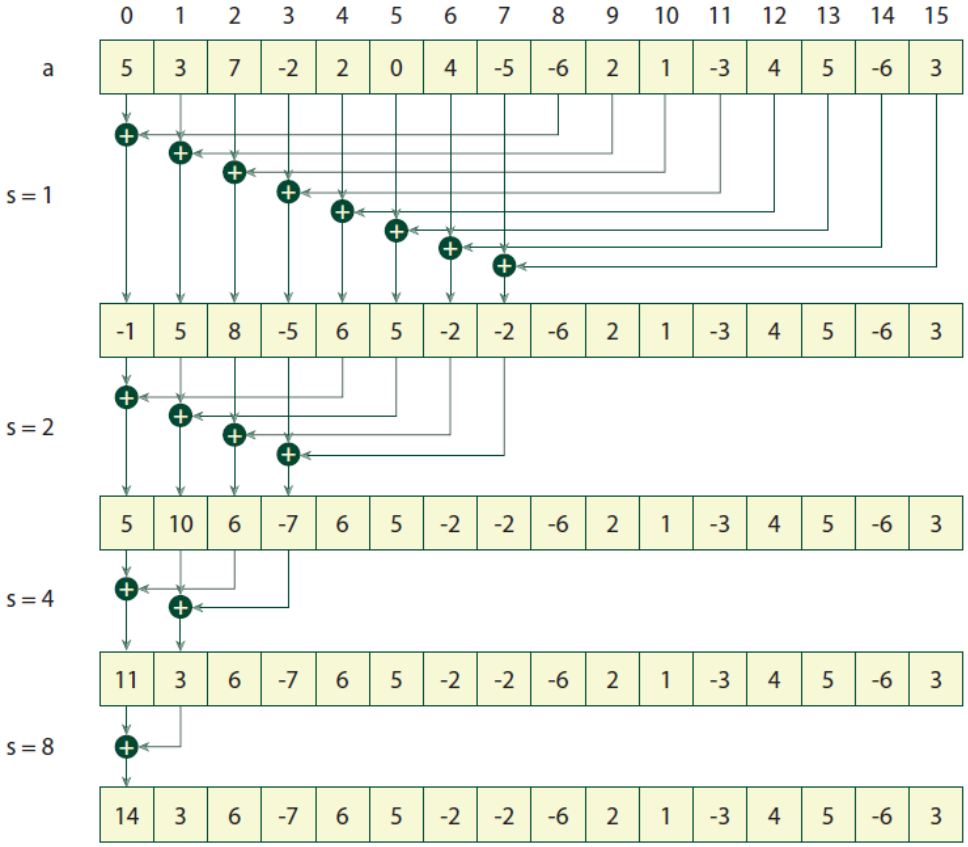
Параллельная редукция

* **Недостаток** – при такой реализации будут **конфликты доступа** к разделяемой памяти из за особенностей архитектуры
* Можно избавиться от конфликтов **изменив порядок**
* В предыдущих версиях суммирование **начиналось с соседних элементов** и **расстояние** увеличивалось **вдвое**

Параллельная редукция

* **Недостаток** – при такой реализации будут **конфликты доступа** к разделяемой памяти из за особенностей архитектуры
* Можно избавиться от конфликтов **изменив порядок**
* В предыдущих версиях суммирование **начиналось с соседних элементов** и **расстояние** увеличивалось **вдвое**
* Начнем **суммирование** с наиболее **удаленных** (на *dimBlock.x/2)* и будем **уменьшать расстояние**

Схема редукции 3.0



Параллельная редукция

reduce_3.cpp

Параллельная редукция

* **Недостаток** – на первом шаге цикла будет **загружена половина потоков** блока

Параллельная редукция

* **Недостаток** – на первом шаге цикла будет **загружена половина потоков** блока
* Потоки можно **загрузить полностью:**

Параллельная редукция

* **Недостаток** – на первом шаге цикла будет **загружена половина потоков** блока
* Потоки можно **загрузить полностью:**
  * **Уменьшим** вдвое **число блоков**

Параллельная редукция

* **Недостаток** – на первом шаге цикла будет **загружена половина потоков** блока
* Потоки можно **загрузить полностью:**
  * **Уменьшим** вдвое **число блоков**
  * В каждом **блоке** обрабатываем **вдвое больше элементов**

Параллельная редукция

* **Недостаток** – на первом шаге цикла будет **загружена половина потоков** блока
* Потоки можно **загрузить полностью:**
  * **Уменьшим** вдвое **число блоков**
  * В каждом **блоке** обрабатываем **вдвое больше элементов**
* **Суммирование** первых пар будем проводить **одновременно** с **записью** в разделяемую память

Параллельная редукция

reduce_4.cpp

Параллельная редукция

* В каждом **варпе 32** потока

Параллельная редукция

* В каждом **варпе 32** потока
* При **s≤32** в каждом **блоке** останется по **1 варпу**

Параллельная редукция

* В каждом **варпе 32** потока
* При **s≤32** в каждом **блоке** останется по **1 варпу**
* Все **потоки варпа синхронны** – для этой части цикла можно **убрать синхронизацию и проверки**

Параллельная редукция

* В каждом **варпе 32** потока
* При **s≤32** в каждом **блоке** останется по **1 варпу**
* Все **потоки варпа синхронны** – для этой части цикла можно **убрать синхронизацию и проверки**
* **Запись** нескольких **итераций** подряд – в **отсутствии синхронизации** компилятор полагает, что **между** 2 **последовательными чтениями** значение **элемента массива не изменяется**

Параллельная редукция

* В каждом **варпе 32** потока
* При **s≤32** в каждом **блоке** останется по **1 варпу**
* Все **потоки варпа синхронны** – для этой части цикла можно **убрать синхронизацию и проверки**
* **Запись** нескольких **итераций** подряд – в **отсутствии синхронизации** компилятор полагает, что **между** 2 **последовательными чтениями** значение **элемента массива не изменяется**
* В данном случае **это не так** – указываем **volatile**

Параллельная редукция

reduce_5.cpp

Тест Nvidia Tesla C2070
16 млн. элементов

Алгоритм	Время работы, мс
Reduce serial	6,92
Reduce_1	5,28
Reduce_2	2,52
Reduce_3	1,88
Reduce_4	0,99
Reduce_5	0,65
OpenMP reduction (Xeon 5620)	2,34

Прикладные библиотеки CUDA

* **Редукция – элементарный алгоритм,** но много **подводных камней** при **параллелизации**

Прикладные библиотеки CUDA

* **Редукция – элементарный алгоритм,** но много **подводных камней** при **параллелизации**
* Неочевидные **нюансы** могут существенно **влиять** на **производительность**

Прикладные библиотеки CUDA

* **Редукция – элементарный алгоритм,** но много **подводных камней** при **параллелизации**
* Неочевидные **нюансы** могут существенно **влиять** на **производительность**
* **Важно – не** стоит **изобретать велосипед**

Прикладные библиотеки CUDA

* **Редукция – элементарный алгоритм,** но много **подводных камней** при **параллелизации**
* Неочевидные **нюансы** могут существенно **влиять** на **производительность**
* **Важно – не** стоит **изобретать велосипед**
* Многие **эффективные алгоритмы** для **GPU** уже **созданы –** use **Google**, Luke

Прикладные библиотеки CUDA

* **Редукция – элементарный алгоритм,** но много **подводных камней** при **параллелизации**
* Неочевидные **нюансы** могут существенно **влиять** на **производительность**
* **Важно – не** стоит **изобретать велосипед**
* Многие **эффективные алгоритмы** для **GPU** уже **созданы –** use **Google**, Luke
* Для многих **прикладных задач** разработаны очень **быстрые библиотеки** для расчетов на **GPU**

CUBLAS

* Реализует программный интерфейс **BLAS** на **CUDA** для **одного GPU**
* Используется для **плотных матриц**
* **Алгоритм** использования:
  * **Инициализировать дескриптор** CUBLAS функцией cublasCreate
  * **Выделить** необходимую **память** на **GPU,** загрузить данные
  * **Вызвать** необходимые **функции CUBLAS**
  * **Выгрузить** результаты вычислений из GPU **на хост**
  * **Освободить дескриптор** CUBLAS функцией cublasDestroy

CUBLAS

gemm.cpp

CUSPARSE

* Реализует основные **операции** линейной алгебры для **разреженных векторов и матриц**
* Поддерживаемые **форматы хранения:**
  * Плотный
  * Координатный
  * Строчный разреженный (CSR)
  * Столбцовый разреженный (CSC)
* Есть операции csrsv_analysis и csrsv_solve – для **анализа и решения треугольной** разреженной системы **СЛАУ** (матрица в **CSR формате!**)

CUFFT

* **Прямое** и **обратное** быстрое преобразование **Фурье**
* **одно-, двух- и трехмерные вещественные** и **комплексные** преобразования
* **одинарная и двойная** точность
* одномерное преобразование до **128 млн.** элементов **одинарной** точности и до **64 млн.** элементов **двойной** точности (ограничено **памятью GPU**)

CURAND

* Библиотека генераторов **квази-** и **псевдослучайных чисел**
* Библиотека CURAND имеет **два интерфейса:** для **хоста** (curand.h) и для **GPU**-ядер (curand_kernel.h)
* В **первом** интерфейсе **вызов функций** идет с **хоста, числа** могут генерироваться как на **хосте,** так и на **GPU**
* **Второй** интерфейс позволяет генерить случайные **числа** непосредственно в **CUDA ядрах**

Thrust

* **Библиотека параллельных** GPU **алгоритмов** обработки данных представимых в виде вектора
* Интерфейс **аналогичен STL**
* **Быстрая реализация** вычислительных алгоритмов в **простой** и **читаемой** форме (в отличии от чистой CUDA)
* **Эффективная реализацию** некоторых **алгоритмов** (scan, sort, reduce)

Thrust

* **Высокий** уровень **абстракции** скрывает **низкоуровневую реализацию**
* **Не дает** возможности управлять **памятью**, **синхронизацией** потоков и т.д.
* Дает **скорость** и **надежность разработки** приложений с **поддержкой GPU**
* **Входит** в состав **CUDA Toolkit** с версии 4.0, может устанавливаться отдельно
* **Библиотека** в виде **заголовочных файлов**

Thrust: сложение векторов

thrust_sum.cpp

Thrust

* **device_vector** – аналог **vector** из **STL**
* **Выделение памяти** на GPU происходит при **объявлении** векторов
* **thrust::plus<float>** – библиотечный **функтор** операции **сложения** с параметром типа
* **Память освобождается** автоматически при **выходе** за **область видимости**
* Собирается как **обычное CUDA приложение**

Трансформации в Thrust

Унарная	$X[i] = f(A[i])$
Бинарная	$X[i] = f(A[i], B[i])$
Тернарная	$X[i] = f(A[i], B[i], C[i])$
Общего вида	$X[i] = f(A[i], B[i], C[i], \ldots)$

Трансформации в Thrust

* **Трансформацию** можно **обобщить** для различного числа последовательностей с помощью **кортежей** (tuples)
* ***zip_iterator*** объединяет **n** входных **последовательностей** в n-местные **кортежи**
* За счет этого **thrust::transform** можно представить как **унарное** преобразование над **кортежем**

Трансформации в Thrust

tuple.cpp

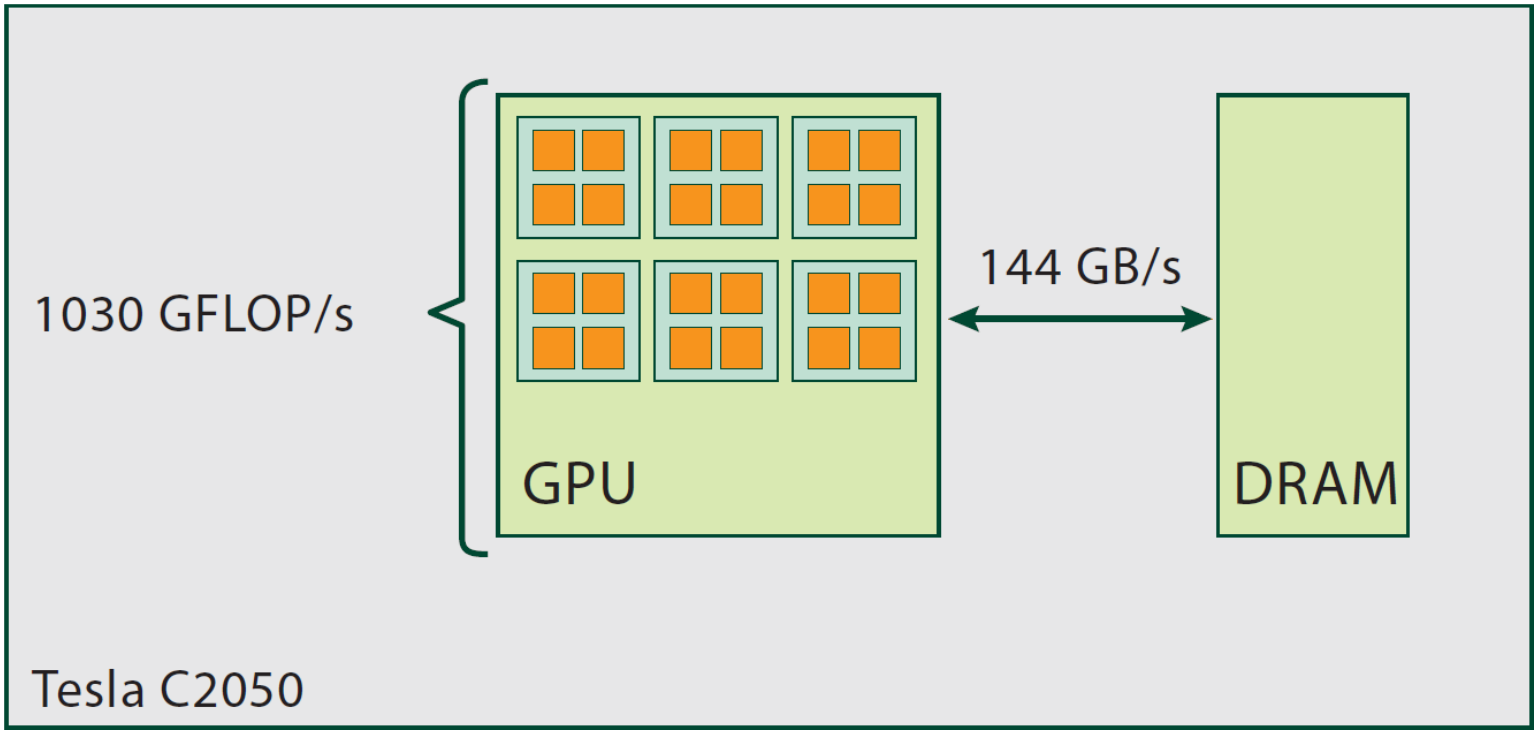
Редукция в Thrust

reduce_t.cpp

Производительность Thrust

* **Обратная сторона** удобства – **невозможность** использования элементов **программирования CUDA-ядер**
* **Производительность** вычислений зависит от **параметров** самой **библиотеки**
* Следует верно оценивать **соотношение производительности** и **скорости обмена** данными - **computational intensity**

Computational intensity



Решение уравнений Навье-Стокса

$$\nabla \cdot \vec{\mathbf{u}} = 0,$$

$$\rho \left[\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} \right] = -\nabla T + \frac{1}{Re} \nabla^2 \mathbf{u},$$

$$\rho \left[\frac{\partial T}{\partial t} + \mathbf{u} \cdot \nabla T \right] = \frac{1}{Re \cdot Pr} \Delta T + \frac{\gamma - 1}{\gamma \cdot Re} \Phi,$$

$$p = \rho R T,$$

Решение уравнений Навье-Стокса

$$\Phi = \Phi_x + \Phi_y + \Phi_z$$

$$\Phi_x = 2(\frac{\partial u}{\partial x})^2 + (\frac{\partial v}{\partial x})^2 + (\frac{\partial w}{\partial x})^2 + \frac{\partial u}{\partial y}\frac{\partial v}{\partial x} + \frac{\partial u}{\partial z}\frac{\partial w}{\partial x},$$

$$\Phi_y = (\frac{\partial u}{\partial y})^2 + 2(\frac{\partial v}{\partial y})^2 + (\frac{\partial w}{\partial y})^2 + \frac{\partial v}{\partial x}\frac{\partial u}{\partial y} + \frac{\partial v}{\partial z}\frac{\partial w}{\partial y},$$

$$\Phi_z = (\frac{\partial u}{\partial z})^2 + (\frac{\partial v}{\partial z})^2 + 2(\frac{\partial w}{\partial z})^2 + \frac{\partial w}{\partial x}\frac{\partial u}{\partial z} + \frac{\partial w}{\partial y}\frac{\partial v}{\partial z}.$$

Решение уравнений Навье-Стокса

Решение уравнений Навье-Стокса

* Идеальный газ

Решение уравнений Навье-Стокса

* Идеальный газ
* Метод покоординатного расщепления:

Решение уравнений Навье-Стокса

* Идеальный газ
* Метод покоординатного расщепления:
  * Уравнения расщепляются – по уравнению вдоль каждой из координат

Решение уравнений Навье-Стокса

* Идеальный газ
* Метод покоординатного расщепления:
  * Уравнения расщепляются – по уравнению вдоль каждой из координат
  * Производные по соответствующему направлению – неявно, остальные считаются постоянными

Решение уравнений Навье-Стокса

* Идеальный газ
* Метод покоординатного расщепления:
  * Уравнения расщепляются – по уравнению вдоль каждой из координат
  * Производные по соответствующему направлению – неявно, остальные считаются постоянными
  * Конечно-разностная схема

Решение уравнений Навье-Стокса

* Идеальный газ
* Метод покоординатного расщепления:
  * Уравнения расщепляются – по уравнению вдоль каждой из координат
  * Производные по соответствующему направлению – неявно, остальные считаются постоянными
  * Конечно-разностная схема
* Нелинейная система – нужны итерации

Решение уравнений Навье-Стокса

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z} = -\frac{\partial T}{\partial x} + \frac{1}{Re}(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2})$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = -\frac{\partial T}{\partial x} + \frac{1}{Re}\frac{\partial^2 u}{\partial x^2},$$
$$\frac{\partial u}{\partial t} + v\frac{\partial u}{\partial y} = \frac{1}{Re}\frac{\partial^2 u}{\partial y^2},$$
$$\frac{\partial u}{\partial t} + w\frac{\partial u}{\partial z} = \frac{1}{Re}\frac{\partial^2 u}{\partial z^2}.$$

Решение уравнений Навье-Стокса

* Глобальные итерации – для целого временного шага всей системы

Решение уравнений Навье-Стокса

* Глобальные итерации – для целого временного шага всей системы
* Невязка в уравнении неразрывности не должна превышать заданную

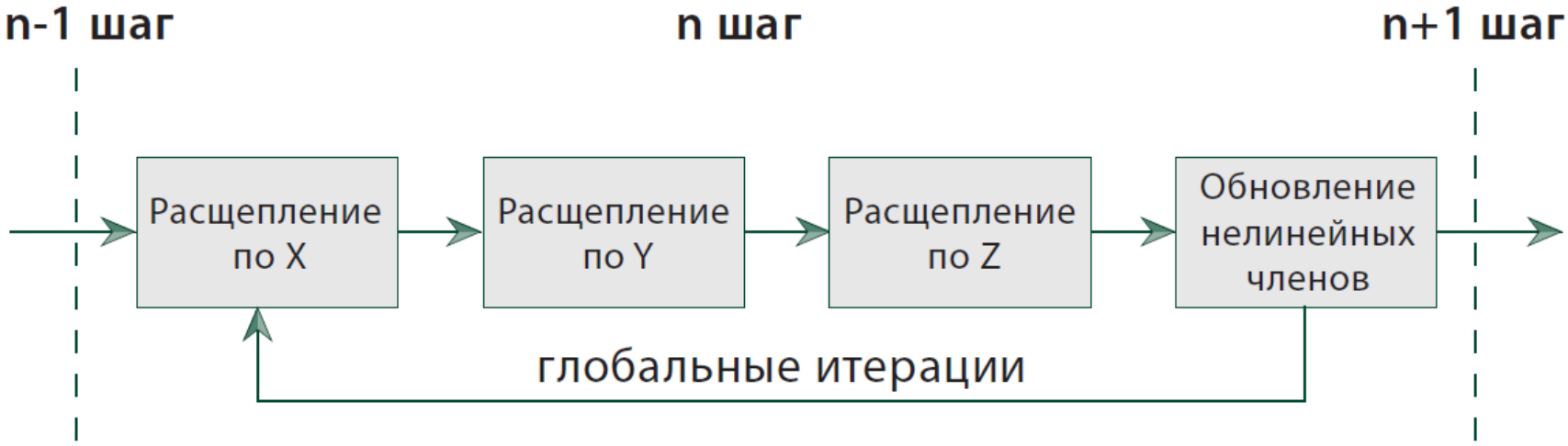
Решение уравнений Навье-Стокса

* Глобальные итерации – для целого временного шага всей системы
* Невязка в уравнении неразрывности не должна превышать заданную
* Локальные итерации – для каждого дробного шага по направлению

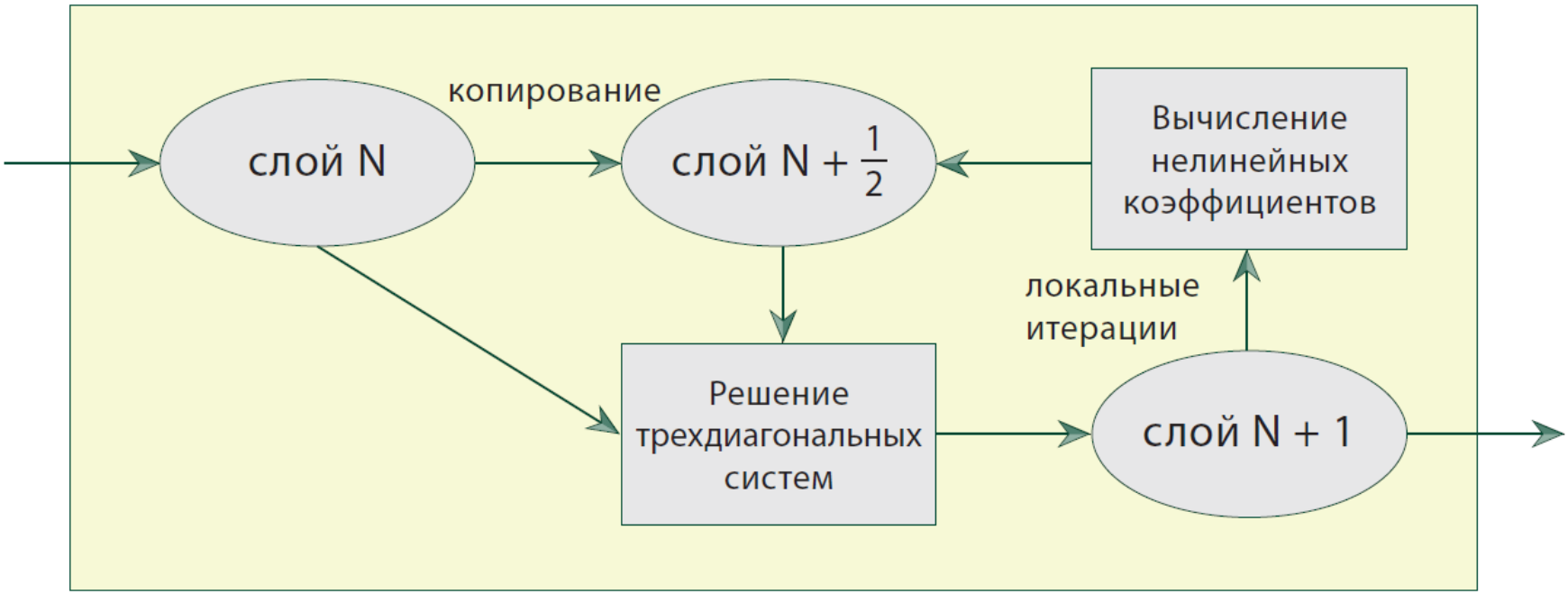
Решение уравнений Навье-Стокса

* Глобальные итерации – для целого временного шага всей системы
* Невязка в уравнении неразрывности не должна превышать заданную
* Локальные итерации – для каждого дробного шага по направлению
* Основное вычислительное ядро – решатель 3-х диагональных систем

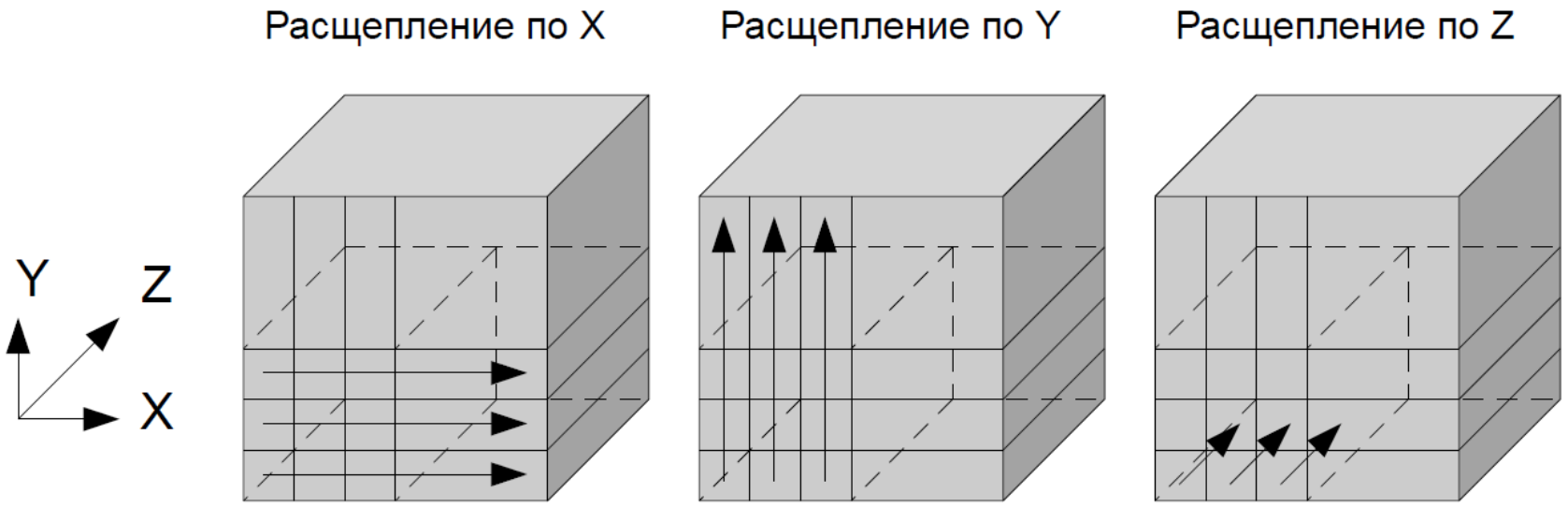
Решение уравнений Навье-Стокса



Решение уравнений Навье-Стокса



Решение уравнений Навье-Стокса



Решение уравнений Навье-Стокса

* Для решения используется CUDA-прогонка

Решение уравнений Навье-Стокса

* Для решения используется CUDA-прогонка
* Задействован один GPU

Решение уравнений Навье-Стокса

* Для решения используется CUDA-прогонка
* Задействован один GPU
* Каждый поток решает ровно одну трехдиагональную систему

Решение уравнений Навье-Стокса

* Для решения используется CUDA-прогонка
* Задействован один GPU
* Каждый поток решает ровно одну трехдиагональную систему
* Количество систем должно быть велико, чтобы нагрузить систему

Решение уравнений Навье-Стокса

* Для решения используется CUDA-прогонка
* Задействован один GPU
* Каждый поток решает ровно одну трехдиагональную систему
* Количество систем должно быть велико, чтобы нагрузить систему
* «Наивная» реализации прогонки – как на CPU

Решение уравнений Навье-Стокса

```
__device__ void solve_tridiagonal( FTYPE *a, FTYPE *b, FTYPE *c, FTYPE *d, FTYPE *x,
int num, int id, int num_seg, int max_n )
{
    get(c,num-1) = 0.0;
    get(c,0) = get(c,0) / get(b,0);
    get(d,0) = get(d,0) / get(b,0);
    // прямой ход прогонки
    for (int i = 1; i < num; i++)
    {
        get(c,i) = get(c,i) / (get(b,i) - get(a,i) * get(c,i-1));
        get(d,i) = (get(d,i) - get(d,i-1) * get(a,i)) / (get(b,i) - get(a,i) * get(c,i-1));
    }
    get(x,num-1) = get(d,num-1);
    // обратный ход прогонки
    for (int i = num-2; i >= 0; i--)
        get(x,i) = get(d,i) - get(c,i) * get(x,i+1);
}
```

Решение уравнений Навье-Стокса

* Прогонки вдоль Z приводят к необъединенным запросам в память

Решение уравнений Навье-Стокса

* Прогонки вдоль Z приводят к необъединенным запросам в память
* Прогонки по X и Y соседние нити читают последовательные данные – coalesced

Решение уравнений Навье-Стокса

* Прогонки вдоль Z приводят к необъединенным запросам в память
* Прогонки по X и Y соседние нити читают последовательные данные – coalesced
* Оптимизация прогонок по Z – входной массив транспонируется, а потом запускается прогонка по Y

Решение уравнений Навье-Стокса

* Прогонки вдоль Z приводят к необъединенным запросам в память
* Прогонки по X и Y соседние нити читают последовательные данные – coalesced
* Оптимизация прогонок по Z – входной массив транспонируется, а потом запускается прогонка по Y
* Нужен эффективный алгоритм транспонирования

Решение уравнений Навье-Стокса

точность	одинарная			двойная		
ядро	оригинал	с трансп.	ускорение	оригинал	с трансп.	ускорение
X dir	523	528	**1.0x**	717	709	**1.0x**
Y dir	525	531	**1.0x**	694	686	**1.0x**
Z dir	1681	533	**2.4x**	1901	693	**2.7x**
трансп.		164			190	
всего	2729	1756	**1.6x**	3312	2278	**1.5x**

Решение уравнений Навье-Стокса

направление	CPU (4 ядра)	Tesla C1060	Tesla C2050	Tesla C2050 vs CPU
X	1.55	5.61	17.73	**11.4x**
Y	2.17	5.36	18.11	**8.3x**
Z	2.34	5.64	18.12	**7.8x**
все	1.96	5.30	16.09	**8.2x**

Решение уравнений Навье-Стокса

* Тестовая задача – течение внутри прямоугольного канала
* Показатель – количество решенных систем в секунду
* Реализация на CPU – OpenMP, задействованы 4 ядра, Intel Core i7 2.8 GHz
* Размер сетки – 448×448×448
* Размер подобран для того, чтобы задействовать все CUDA-ядра одного GPU класса Tesla C2050/2070

Перспективы

* Профилирование своего приложения – определение узких мест и потенциала оптимизаций

Перспективы

* Профилирование своего приложения – определение узких мест и потенциала оптимизаций
* Чем дальше в лес тем злее дятлы – более глубокая оптимизация требует лучшего знания и понимания

Перспективы

* Профилирование своего приложения – определение узких мест и потенциала оптимизаций
* Чем дальше в лес тем злее дятлы – более глубокая оптимизация требует лучшего знания и понимания
* Использование нескольких GPU на одном узле

Перспективы

* Профилирование своего приложения – определение узких мест и потенциала оптимизаций
* Чем дальше в лес тем злее дятлы – более глубокая оптимизация требует лучшего знания и понимания
* Использование нескольких GPU на одном узле
* Использование гетерогенных кластеров – несколько узлов, в каждом по несколько GPU

Спасибо