УДК 004.421

Н.Т. Данаев, Д.Ж. Ахмед-Заки, Б.С. Дарибаев *,
Т.С. Иманкулов, О.Н.Турар

Казахский национальный университет им. аль-Фараби, Республика Казахстан, г. Алматы
* E-mail: beimbet.daribayev@kaznu.kz

Проектирование и разработка высокопроизводительных приложений на мобильных платформах

В данной статье рассмотрена параллельная реализация метода Якоби для решения двумерного и трёхмерного уравнений Пуассона на мобильных платформах с использованием технологии CUDA. Использование мобильных технологий для расчетов обусловлено возрастанием их вычислительных возможностей и удобством использования для небольших задач в условиях отсутствия доступа к обычным вычислительным мощностям. Проанализированы методы распределения узлов сетки на каждом блоке для повышения эффективности алгоритма. Результаты были протестированы на мобильном устройстве Xiaomi MiPad с процессором Nvidia Tegra K1 и на стационарном устройстве с видеокартой NVIDIA GeForce GTX 550Ti. Nvidia Tegra K1 - первый мобильный процессор, который поддерживает технологию CUDA, основанный на архитектуре Kepler. Приведены сравнения результатов параллельного и последовательного кода программы на стационарных и соответственно на мобильных устройствах.
Ключевые слова: CUDA, разделяемая память, глобальная память, мобильное устройство, высокопроизводительные приложения, параллельные вычисления.

N.T. Danaev, D.Zh.Ahmed-Zaki, B.S. Daribaev, T.S. Imankulov, O.N. Turar

Design and development of high-performance applications on mobile platforms

This article describes a parallel implementation of Jacobi's method for solving the two-dimensional and three-dimensional Poisson equation on mobile platforms using the CUDA technology. The use of mobile technology for calculations motivated by increasing of computing capabilities and ease of use for small tasks in the absence of access to conventional computing capacities. Methods of grid nodes distribution on each block to improve efficiency of the algorithm were analyzed. The results were tested on a mobile device Xiaomi MiPad with Nvidia Tegra K1 processor and on a stationary device with a NVIDIA GeForce GTX 550Ti video card. Nvidia Tegra K1 - the first mobile processor that supports technology CUDA, - based on the Kepler architecture. Comparisons of results of parallel and serial program codes, on stationary and mobile devices, were provided.
Key words: CUDA, shared memory, global memory, mobile devices, high-performance applications, parallel computing.

Н.Т. Данаев, Д.Ж. Ахмед-Заки, Б.С. Дәрібаев, Т.С. Иманқұлов, О.Н. Тұрар

Жоғары өнімді қосымшаларды мобильді платформаларда жобалау және өңдеу

Бұл мақалада мобильді платформада CUDA технологиясын қолдану арқылы екі және үш өлшемді Пуассон теңдеуін Якоби әдісімен параллельді жүзеге асыру қарастырылды. Ғылыми есептеулерде мобильді технологияларды пайдалану олардың есептеу қуатылығының артуының мүмкіндігіне және қарапайым есептеуіш қуаттылығында шығара алмайтын шағын есептерді қолдану ыңғайлылығына байланысты. Алгоритмнің тиімділігін арттыру үшін тор түйіндерін әрбір блокта үлестіру әдістеріне талдау жасалынды. Есептеу нәтижелері NVIDIA Tegra K1 процессорі негізінде Xiaomi MiPad мобильді құрылғысында және NVIDIA GeForce GTX 550Ti видеокартасы бар стационарлық құрылғысында тестіленді. Nvidia Tegra K1 - Kepler архитектурасына негізделген, CUDA технологиясын қолданатын бірінші мобильді процессор. Стационарлық және мобильдік құрылғыларда параллель және тізбектелген программалау кодтарының нәтижелеріне салыстырулар келтірілді.
Түйін сөздер: CUDA, ортақ жады, глобальды жады, мобильді құрылғы, жоғары өнімді қосымшалар, параллельді есептеулер.

1. Введение

В современном мире развитие мобильных технологий является одним из самых востребованных направлений в индустрии развлечений и, как следствие, данные технологии развиваются с огромной скоростью. На данный момент мобильные процессоры являются полноценными вычислительными единицами, которые могут быть использованы как в качестве дополнительных потоков для вычислительных кластеров, так и для отдельного расчета задач, требующих относительно небольшие ресурсы. Часто в производстве возникает необходимость запуска задачи в оперативном режиме для внесения корректировок в работающую систему. Такая необходимость может возникнуть в самых разных областях.

В данной статье рассматриваются краевые задачи для уравнения Пуассона, поскольку данное уравнение чаще всего встречается в виде подзадач в задачах эллиптического типа и именно на данной задаче изначально отрабатывается методология решений [1]. В статье рассматривается решение задачи с использованием параллелизации на графическом процессоре мобильного устройства.

2. Постановка задачи и численный алгоритм

Для тестирования высокопроизводительных приложений на мобильных платформах были численно решены задачи в области $D = \{0 \le x_\alpha \le 1, \alpha = 1, 2\}$ для двумерных уравнений и $D = \{0 \le x_\alpha \le 1, \alpha = 1, 2, 3\}$ для трехмерных уравнений Пуассона следующего вида:

$$\Delta u = -f \tag{1}$$

с краевыми условиями Дирихле в двумерном случае:

$$\begin{aligned} &u(0, x_2) = u(1, x_2) = 0, 0 \le x_2 \le 1, \\ &u(x_1, 0) = u(x_1, 1) = 0, 0 \le x_1 \le 1. \end{aligned} \tag{2}$$

в трехмерном случае:

$$\begin{aligned} &u(0, x_2, x_3) = u(1, x_2, x_3) = 0, 0 \le x_2 \le 1; 0 \le x_3 \le 1, \\ &u(x_1, 0, x_3) = u(x_1, 1, x_3) = 0, 0 \le x_1 \le 1; 0 \le x_3 \le 1, \\ &u(x_1, x_2, 0) = u(x_1, x_2, 1) = 0, 0 \le x_1 \le 1; 0 \le x_2 \le 1. \end{aligned} \tag{3}$$

где Δ– оператор Лапласа.

Рассматривается итерационный метод Якоби для численного решения уравнения Пуассона.

В двумерном случае:

$$u_{i,j}^{n+1} = \frac{u_{i-1,j}^n + u_{i+1,j}^n + u_{i,j-1}^n + u_{i,j+1}^n + h^2 * f_{i,j}}{4} \tag{4}$$

В трехмерном случае:

$$u_{i,j,k}^{n+1} = \frac{u_{i-1,j,k}^n + u_{i+1,j,k}^n + u_{i,j-1,k}^n + u_{i,j+1,k}^n + u_{i,j,k-1}^n + u_{i,j,k+1}^n + h^2 * f_{i,j,k}}{6} \tag{5}$$

3. Параллельный алгоритм с использованием технологии CUDA

Для оптимизации программы использовалась технология CUDA. Тестировались три кода программы: последовательный код на языке C/C++, параллельный не оптимизированный (работа только с глобальной памятью) и параллельный оптимизированный (работа с разделяемой памятью) код с применением технологии CUDA. Для сравнения результатов, расчеты тестировались на планшете Xiaomi MiPad с процессором NVIDIA Tegra K1 и на стационарном компьютере с видеокартой NVIDIA GeForce GTX 550Ti.

Подробно рассмотрим устройства и технологию. Мобильный планшет Xiaomi MiPad - разработан китайской компанией Xiaomi. Основное преимущество этого мобильного устройства то, что он оснащен процессором от компании Nvidia - Tegra K1. Tegra K1 создан на базе той же архитектуры NVIDIA Kepler и обладает 192 CUDA-ядрами в конфигурации 192:8:4 на частоте до 950 МГц и производительностью 360 GFLOPS. Стационарный компьютер оснащен процессором Intel® Core™ i7-3770 с частотой 3.40GHz и восьми гигабайтной оперативной памятью. Видео карта этого компьютера - NVIDIA GeForce GTX550 Ti с памятью два гигабайт. Здесь каждый мультипроцессор состоит из нескольких CUDA ядер, из модулей для вычисления математических функций, из разделяемой и кэш памяти.

Теперь перейдем к общему алгоритму решения задачи. В одной сетке GPU размещено несколько блоков. У каждого блока есть три направления, то есть отдельные блоки располагаются в трехмерном виде. А в каждом блоке находится несколько нитей. Нити тоже отображаются в трехмерном виде. Каждый расчетный узел располагается на одной нити. Далее, при расчете данные копируются с глобальной памяти. В этом случае, нельзя повторно использовать уже принятые данные. Чтобы рассчитать данные на одном внутреннем узле необходимо четыре соседних узла, приходится каждый узел использовать четыре раза. Учитывая, что глобальная память считается самой медленной, копировать внутренние узлы четыре раза считается не эффективным.

Если разбить сетку на блоки, где сами блоки имеют несколько нитей, и каждая нить вычисляет один узел, то каждый блок может копировать данные в разделяемую память, после чего каждый узел отдельного блока выполняет расчет и сохраняет рассчитанные данные в глобальную память. Для расчета в каждой подобласти, нужно использовать данные из соседней подобласти, т.е. нужно скопировать граничные данные из глобальной памяти[2]. После этого размер подобласти увеличивается.

Задачу можно оптимизировать несколькими способами.

1. Потоки данных внутренней ячейки подобласти копируются в разделяемую память из глобальной памяти, затем из глобальной памяти копируются граничные узлы. В данном случае размер подобласти не меняется. Из-за того, что разделяемая память считается самой быстрой, этот алгоритм является эффективным.

2. Нельзя избежать повторного копирования данных на границе подобласти из глобальной памяти. В вышеуказанных случаях копируются столбцы и строки на границах подобласти. Поэтому, нужно изменить схему двумерной декомпозиции на одномерную. Тем самым мы исключаем повторное копирование столбцов.

4. Составление параллельного не оптимизированного и оптимизированного алгоритма с помощью технологии CUDA на мобильных и стационарных устройствах

В случае не оптимизированного параллельного алгоритма в GPU все данные хранятся в глобальной памяти (global memory). В следующем листинге показан псевдо код функции ядра для решения двумерного уравнения Пуассона методом Якоби (1),(2),(4):

```
KernelFunction (output [], input [], f [], size, step) {
set index by x in one dimension - indexByX
set index by y in one dimension - indexByY
...
set input array's index by the thread index
...
set output array values with input array (Jacobi method)
}
```

В этом коде видно, чтобы рассчитать выходной массив функции ядра каждый раз вызывается входной массив из глобальной памяти, которая замедляет расчет, из-за низкой пропускной способности глобальной памяти (Рисунок 1а). Трехмерный случай метода Якоби для решения уравнения Пуассона аналогичен. В обоих случаях нужно преобразовать многомерный массив в одномерный, чтобы каждый раз не считать по двум или трем направлениям. Многие ошибки являются следствием неправильного преобразования и установки индексов.

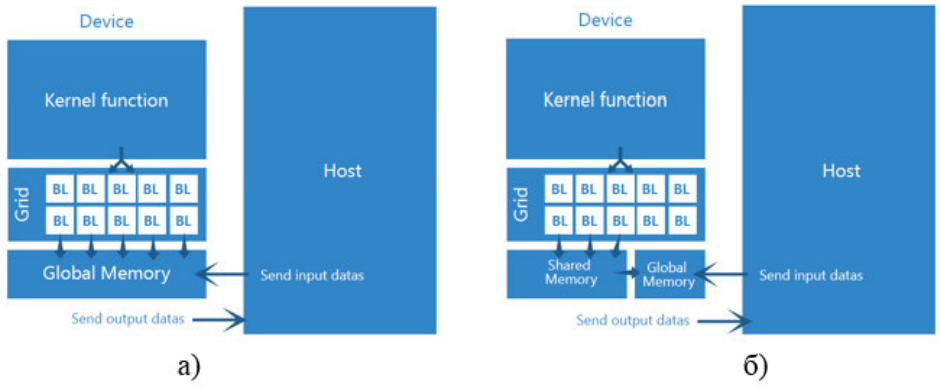

Рисунок 1 – Схематичное представление работы: а) не оптимизированный параллельный алгоритм на CUDA; б) оптимизированный параллельный алгоритм на CUDA

При оптимизированном параллельном алгоритме для решения двумерного или трехмерного уравнения Пуассона методом Якоби в функции ядра объявляется массив соответственно с размерностью задачи в разделяемой памяти. В одном блоке каждая нить будет работать только с разделяемой памятью, известно, что у разделяемой памяти (shared memory) очень высокая пропускная способность, и здесь не надо каждый раз

обращаться к глобальной памяти (Рисунок 1б). Это дает ощутимую производительность. Псевдо код функции ядра рассчитывающего оптимизированный параллельный алгоритм для решения двумерного уравнения Пуассона методом Якоби показан внизу:

```
KernelFunction(output[], input[], f[], size, step) {
set index by x in one dimension - indexByX
set index by y in one dimension – indexByY
set thread index
...
set previous and next index of input array
...
set input array's index by the thread index
declare temp array in shared memory
get boundary values from neighboring blocks and set to temp array
...
set output array values with input array (Jacobi method)
}
```

Здесь видно, чтобы рассчитать выходной массив функция ядра объявляет временной массив в разделяемой памяти, и устанавливает его значение со значением входного массива и, в дополнение к этому, устанавливает граничные значения из соседних блоков. И тогда при вычислении уравнения каждый раз вызывает входной массив не из глобальной памяти, а из разделяемой памяти. За счет высокой пропускной способности разделяемой памяти, ускоряется время расчета программы. При получении граничных значений из соседних блоков, нужно правильно подобрать индексы для того, чтобы работать с правильными данными (Рисунок 2).

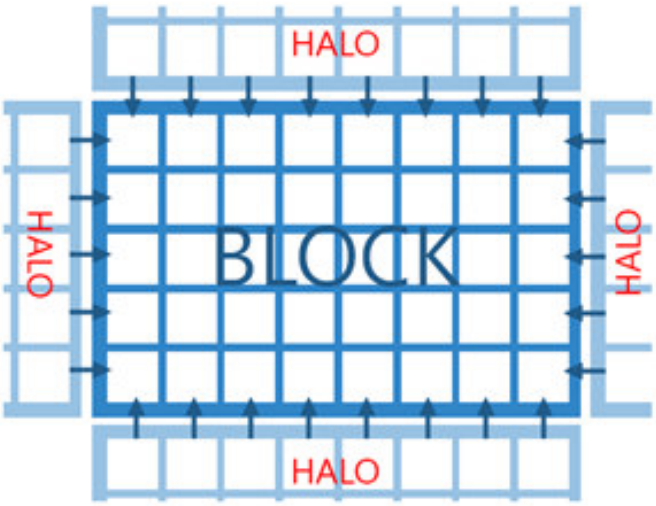

Рисунок 2 – Расчетные блоки и соседние элементы в оптимизированном параллельном алгоритме на CUDA

До появления процессора от NVIDIA Tegra K1 на мобильных устройствах почти невозможно было запустить программу на больших расчетных сетках. Идея создания параллельных программ на мобильных устройствах - показать гидродинамический расчет всегда и везде, при этом за малое количество времени. Эту идею можно развивать до распределения задач на нескольких мобильных устройствах по беспроводной или иной сети. Во время тестирования двумерного и трехмерного уравнений, произведены сравнения результаты расчетов последовательной и параллельной программы на мобильных (стационарных) устройствах. А также будут показаны сравнения результатов расчетов параллельных (не оптимизированной и оптимизированной) программ соответственно на мобильных и стационарных устройствах.

5. Результаты и сравнения вычислительных экспериментов

Двумерный случай. При тестировании расчетов двумерного уравнения Пуассона методом Якоби были использованы аналитические решения для сравнения и проверки правильности результатов. Результаты тестирования показали, что, на стационарных устройствах для решения задачи параллельный не оптимизированный метод по сравнению с оптимизированным методом на CUDA в среднем в полтора раза медленнее, если размер блока по двум направлениям равен 8, если размер блока по двум направлениям равен 16, то тогда первый метод приблизительно в пять раз медленнее работает чем второй метод (Таблица 1, Таблица 2). Оптимальный выбор размера блока не должен превышать показанного в характеристиках видеокарты количества нитей в одном блоке. Обычно для быстрого расчета число нитей в одном блоке должно быть в интервале 128-512 [3]. Эти числа при тестировании были верны, потому что при выборе размера блоков 8x8 программа считала медленнее чем 16x16.

Таблица 1 – Время расчета (сек.) двумерной задачи Пуассона на разных сетках и методах (при расчете с технологией CUDA с размером 8x8)

Протестированные алгоритмы	128x128	256x256	512x512	1024x1024
Параллельный не оптимизированный метод (стационарное устройство)	0,30	1,21	4,85	21,69
Параллельный оптимизированный метод (стационарное устройство)	0,18	0,79	3,12	12,95
Параллельный не оптимизированный метод (мобильное устройство)	1,27	5,77	25,19	104,50
Параллельный оптимизированный метод (мобильное устройство)	0,84	2,14	8,10	32,24

Если размер блока по двум направлениям равен 8, тогда по увеличению сетки соотношение их времени работы возрастает, потому что при каждом обращении в глобальную память время расчета замедляется, из-за того что глобальная память является самой медленной памятью [4]. Это означает, что при увеличении размера расчетной сетки в оптимизированном варианте изменение времени уменьшается. В оптимизированном варианте использовалась разделяемая память, после этого расчеты выполнялись быстрее.

Потому что пропускная способность разделяемой памяти по сравнению с глобальной памятью намного быстрей. При расчете в функции ядра нить для использования соседних элементов массива каждый раз будет ссылаться в разделяемую память, а не в глобальную. Соответствующие варианты параллельного вычисления на мобильном устройстве в 3-5 раз уступают стационарному устройству в зависимости от размера сетки (Таблица 1).

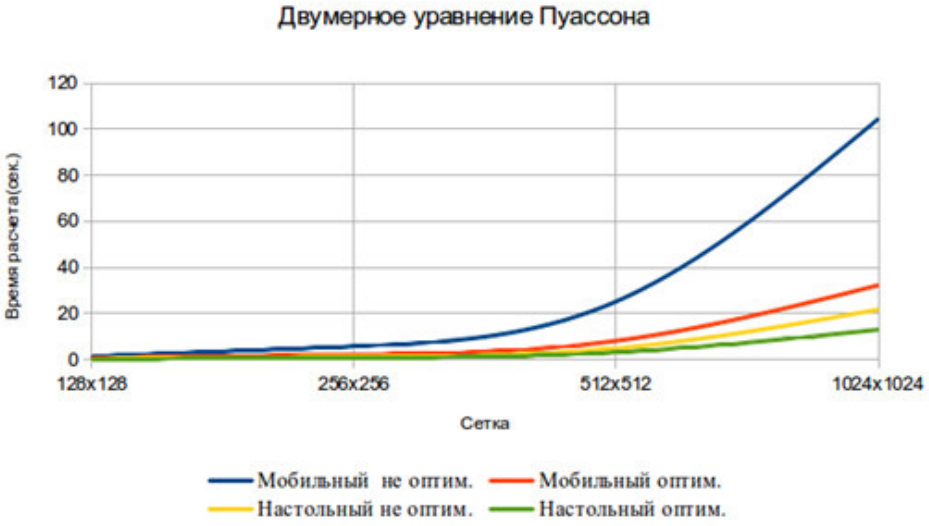

Рисунок 3 – Сравнительный график времени расчетов (размер блока 8x8)

Таблица 2–Время расчета (сек.) двумерной задачи Пуассона на разных расчетных сетках и методах (при расчете с технологией CUDA с размером 16x16)

Протестированные алгоритмы	128x128	256x256	512x512	1024x1024
Параллельный не оптимизированный метод (стационарное устройство)	0,37	1,48	6,04	27,88
Параллельный оптимизированный метод (стационарное устройство)	0,08	0,33	1,28	5,49
Параллельный не оптимизированный метод (мобильное устройство)	2,63	12,29	52,42	204,78
Параллельный оптимизированный метод (мобильное устройство)	0,53	1,53	5,71	23,69

Теперь сравним параллельные варианты оптимизированного алгоритма с не оптимизированным алгоритмом в блоках размером 16 по двум направлениям в разных устройствах для решения двумерного уравнения Пуассона. Сравнивая оптимизированные алгоритмы (Таблица 1, Таблица 2) можно увидеть, что на второй таблице в обеих устройствах, учитывая размерности сетки, время расчета в несколько раз быстрее. Это можно объяснить тем что, если количество нитей в одном блоке будет меньше чем 128 и больше чем 512 то время расчета увеличивается [5]. В сетке с размером 1024x1024 время расчета оптимизированного алгоритма в стационарном устройстве, в первом случае равно

12.95 секунд, а на второй таблице показывает 5.49 секунд. А для мобильных устройств время расчета оптимизированного алгоритма в первом случае показывает 32.24 секунд, а во втором случае 23.69 секунд. И здесь можно увидеть, что варианты параллельного вычисления в мобильном устройстве в зависимости от размера сетки в 3-5 раз уступает стационарному устройству. Это нормально, потому что до появления мобильных устройств с поддержкой технологии CUDA нельзя было бы получить выше показанные сравнительные результаты. Скорость расчета по оптимизированному алгоритму на мобильном устройстве не уступает, а даже превышает скорость параллельной программы, вычисляющей по не оптимизированному алгоритму на стационарном компьютере.

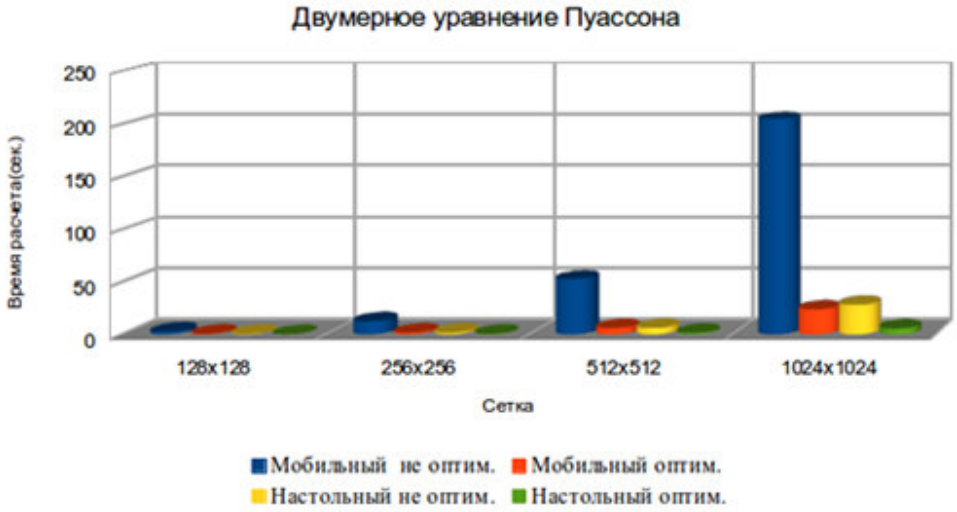

Рисунок 4 – Сравнительный график времени расчетов (размер блока 16x16)

Таблица 3– Время расчета (сек.) двумерной задачи Пуассона на разных сетках и методах (при расчете с технологией CUDA с размером 8x8 и 16x16)

Протестированные алгоритмы	128x128	256x256	512x512	1024x1024
Параллельный не оптимизированный метод (стационарное устройство) 8x8	0,30	1,21	4,85	21,69
Параллельный оптимизированный метод (стационарное устройство) 8x8	0,18	0,79	3,12	12,95
Параллельный не оптимизированный метод (стационарное устройство) 16x16	0,37	1,48	6,04	27,88
Параллельный оптимизированный метод (стационарное устройство) 16x16	0,08	0,33	1,28	5,49
Последовательный метод (стационарное устройство)	1,57	7,13	28,72	115,85

Следующий этап тестирования это сравнение времени расчета параллельной и последовательной программ двумерного уравнения Пуассона на разных расчетных сетках.

Если сравнить оптимизированный алгоритм с размером блока 8x8 с последовательным алгоритмом, то он работает быстрее в зависимости от размера сетки приблизительно в 3-9 раз (Таблица 3). А если сравнить этот же алгоритм, но с размером блока 16x16, с последовательным алгоритмом, то параллельный код работает в 20-22 раз быстрее. Из рисунка 5 можно заметить, что с увеличением размера расчетной сетки в последовательном алгоритме время расчета быстро увеличивается, а в оптимизированном параллельном алгоритме время расчета увеличивается медленно.

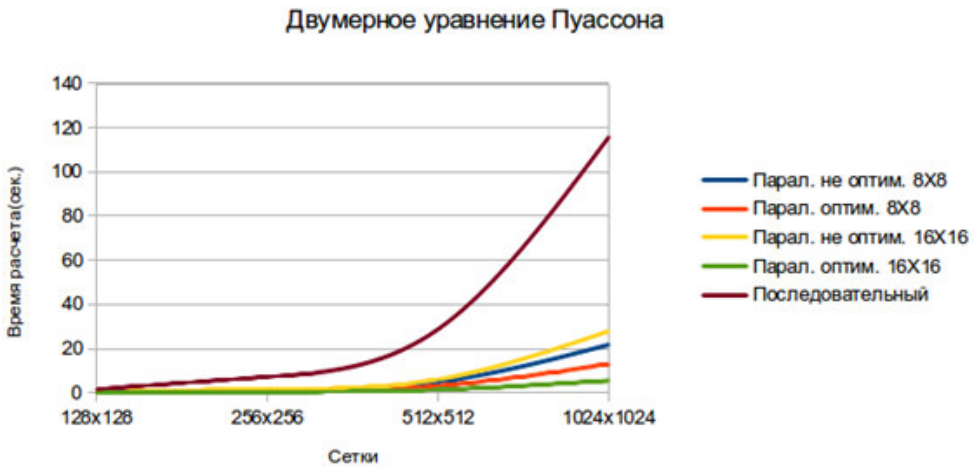

Рисунок 5 – Сравнительный график времени расчетов

Таблица 4– Время расчета (сек.) трехмерной задачи Пуассона на разных сетках и методах (при расчете с технологией CUDA с размером блока 8x8x8)

Протестированные алгоритмы	32x32x32	64x64x64	128x128x128	256x256x256
Параллельный не оптимизированный метод (мобильное устройство)	1,57	11,93	107,84	787,7
Параллельный оптимизированный метод (мобильное устройство)	0,23	0,96	14,75	62,09
Параллельный не оптимизированный метод (стационарное устройство)	0,28	2,87	27,8	296,87
Параллельный оптимизированный метод (стационарное устройство)	0,03	0,25	2,14	20,59

Трехмерный случай. Рассмотрим параллельные варианты оптимизированного и не оптимизированного алгоритма в разных устройствах для решения трехмерного уравнения Пуассона. Размер одного блока необходимо взять равным 8 в трех измерениях. Потому что количество нитей в одном блоке равно 8x8x8 = 512, что обеспечивает быстроту. Время расчета оптимизированного и не оптимизированного алгоритмов на разных сетках и устройствах показано в таблице 4. Сравнивая оптимизированные алгоритмы с учетом размерности сетки можно увидеть, что на стационарном компьютере время расчета в несколько раз быстрее чем на мобильном устройстве. Это можно объяснить тем,

что размерность уравнения больше, и это влияет на производительность при получении граничных элементов одного блока с соседних блоков. В сетке с размером 256x265x256 время расчета оптимизированного алгоритма в настольном компьютере равно 20.59 секунд, а для мобильных устройств время расчета оптимизированного алгоритма показывает 62.09 секунд. В зависимости от размера сетки здесь можно увидеть, что варианты параллельного вычисления в мобильном устройстве в 3-7 раз уступает стационарному компьютеру. Если сравнить время расчета оптимизированного варианта с не оптимизированным вариантом для обеих устройств по отдельности, их соотношение составляет от 7 до 15 (Рисунок 6).

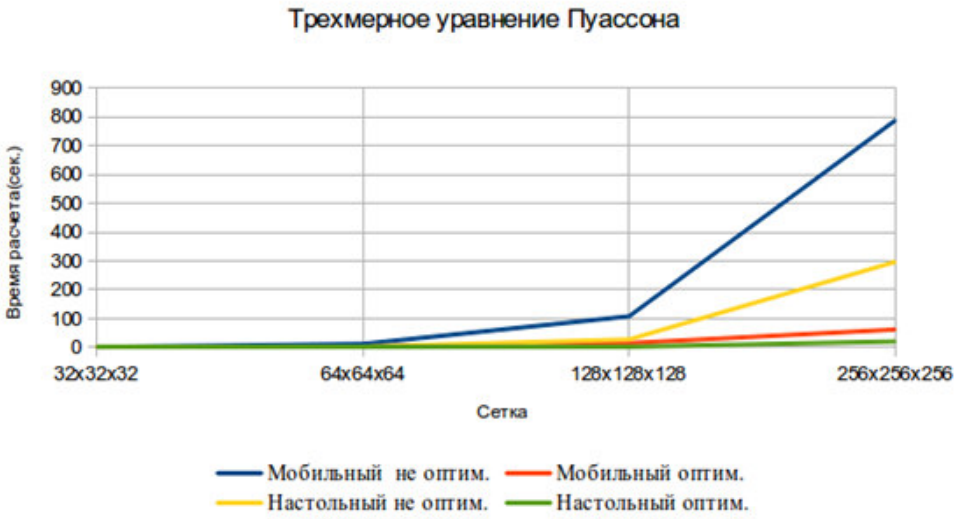

Рисунок 6 – Сравнительный график времени расчетов (размер блока 8x8x8)

Таблица 5– Время расчета (сек.) трехмерной задачи Пуассона на разных сетках и методах (при расчете с технологией CUDA с размером блока 8x8x8)

Протестированные алгоритмы	32x32x32	64x64x64	128x128x128	256x256x256
Параллельный не оптимизированный метод (стационарное устройство)	0,28	2,87	27,8	296,87
Параллельный оптимизированный метод (стационарное устройство)	0,03	0,25	2,14	20,59
Последовательный метод (стационарное устройство)	0,41	3,53	29,65	241,61

Сравнивая время расчета параллельной и последовательной программы трехмерного уравнения Пуассона на разных расчетных сетках, на рисунке 7 можно заметить, что в параллельном не оптимизированном алгоритме с увеличением размера сетки время расчета сравнительно одинаково с последовательным алгоритмом. А оптимизированный параллельный алгоритм работает в 10-15 раз быстрее чем последовательный алгоритм (Таблица 5).

Рисунок 7 – Сравнительный график времени расчетов (размер блока 8x8x8)

6. Заключение

В статье описан подход разработки и тестирования численного параллельного решения задачи для уравнения Пуассона с использованием технологии CUDA на мобильных устройствах. Результаты проведенных тестов показывают явное преимущество вычислении на мобильных платформах по сравнению с последовательной программой, даже при условии, что та выполняется на стационарном компьютере. В некоторых случаях этот выигрыш сравним с эффективностью аналогичной программы на видеокарте компьютера, что в свою очередь демонстрирует, что сейчас мобильные технологии представляют собой достаточную вычислительную мощность. Представленная работа является первым шагом по осуществлению функционала предоставляющего различным специалистам возможность проведения расчетов в любых условиях на мобильных платформах.

В будущем планируется добавить в приложение модуль визуализации для прорисовки больших данных, реализующий алгоритм трассировки лучей с использованием технологии CUDA.

Литература

[1] А.А. Самарский Теория разностных схем. – М.: Наука, 1977. – 653 с.

[2] Jason Sanders, Edward Kandrot CUDA by Example: An Introduction to General-Purpose GPU Programming. – 2010. – 312 p.

[3] Иноземцева Н.Г., Перепёлкин Е.Е., Садовников Б.И. Оптимизация алгоритмов задач математической физики для графических процессоров. – 2012. – 240 с.

[4] Shane Cook CUDA Programming: A Developer's Guide to Parallel Computing with GPUs // Applications of Gpu Computing. – 2012. – 600 p.

[5] Nicholas Wilt CUDA Handbook: A Comprehensive Guide to GPU Programming // The Paperback. – 2013. – 528 p.

References

[1] Samarskij A.A. Teoria rasnostnyh shem. - Moscow: Nauka, 1977. – 653 c.

[2] Jason Sanders, Edward Kandrot CUDA by Example: An Introduction to General-Purpose GPU Programming. – 2010. – 312 p.

[3] Inozemtseva N.G., Perepolkin Ye.Ye., Sadovnikov B.I. Optimizatsiya algoritmov zadach matematicheskoy fiziki dlya graficheskikh protsessorov. – 2012. – 240 s.

[4] Shane Cook CUDA Programming: A Developer's Guide to Parallel Computing with GPUs // Applications of Gpu Computing. – 2012. – 600 p.

[5] Nicholas Wilt CUDA Handbook: A Comprehensive Guide to GPU Programming // The Paperback. – 2013. – 528 p.