Т.Е. Романенко, А.В. Разгулин

ОБ ОДНОМ АЛГОРИТМЕ ТРЕХМЕРНОЙ ДЕКОНВОЛЮЦИИ С ИСПОЛЬЗОВАНИЕМ ГРАФИЧЕСКИХ ПРОЦЕССОРОВ*

Введение

Во многих прикладных задачах медицинской физики актуальна проблема трехмерной деконволюции полупрозрачных объектов на основе стека наблюдаемых изображений сечений объекта по глубине. Например, для возникающей в офтальмологии задачи исследования трехмерной структуры отделов глазного дна человека *in vivo* используется метод быстрой перефокусировки изображающей системы методами адаптивной оптики [1]. При этом наряду с истинными сечениями трёхмерного объекта в каждой фокальной плоскости полученные этим методом изображения содержат размытые изображения соседних по глубине сечений и различные искажения. Таким образом, возникает проблема устойчивого к помехам получения стека изображений искомого полупрозрачного объекта по глубине для его последующего использования в трёхмерной реконструкции. Аналогичные задачи возникают в биомикроскопии [2], [3].

Задача трехмерной деконволюции является существенно ресурсоемкой. В последнее время предпринимаются попытки увеличения быстродействия за счет использования вычислительного ресурса многопроцессорных графических плат (GPU), которыми комплектуется большинство современных персональных компьютеров начиная со среднего уровня. Достаточно полный анализ библиографии по использованию GPU для решения различных математических задач по состоянию на 2012 год приводится в обзоре [4]. Однако имеющийся здесь большой опыт решения аналогичных двумерных задач не может быть напрямую перенесен на трехмерный случай. Отметим некоторые из возникающих в этой связи проблем, нашедших отражение в современной литературе.

Во-первых, для получения выигрыша быстродействия количество вычислительных ядер GPU должно быть достаточно велико, поскольку, как правило, производительность одного ядра GPU существенно меньше производительности ядра центрального процессора (CPU). Например, как отмечалось в отчете [5], при использовании весьма популярного алгоритма

* Работа выполнена при поддержке РФФИ, грант №15-29-03896 офи_м.

Ричардсона-Люси трехмерной деконволюции использование графической карты со 128 блоками по 64 нити каждый не дало преимуществ перед реализацией алгоритма на симметричной мультипроцессорной (SMP) архитектуре с 16 процессорами.

Во-вторых, данные трехмерной задачи должны быть соответствующим образом подготовлены для наиболее эффективного использования возможностей GPU, изначально нацеленных для распараллеливания вычислительных алгоритмов, работающих с графическими данными. Также необходимо оптимизировать количество пересылок между памятью CPU и памятью GPU, весьма существенное при обработке медицинских данных высокого разрешения. Отмеченные проблемы могут быть решены несколькими способами.

Во многих случаях применяется векторизация [6], при которой трехмерная задача переформулируется в виде двумерной задачи решения системы линейных алгебраических уравнений (СЛАУ) с новой матрицей достаточно большой размерности, а также декомпозиция [7] для использования нескольких GPU с небольшой памятью.

Другой способ состоит в том, чтобы использовать специальный алгоритм решения трехмерной задачи, который естественным образом, не затрагивая структуры трехмерных данных, сводил бы ее к последовательности двумерных задач, допускающих естественное распараллеливание. Именно такой подход развивается в настоящей работе. Этот подход разбивает задачу на набор параллельно решаемых подзадач для определения набора по глубине плоских фурье-гармоник искомого трехмерного объекта. Нам неизвестны работы других авторов, которые использовали бы такой метод в применении к задачам трехмерной деконволюции офтальмологических изображений.

Для решения близких к рассматриваемой в настоящей работе постановках трехмерных задач биомикроскопии традиционно используется итерационный метод Ричардсона-Люси наибольшего правдоподобия ([8], [9], [10], [5], [11]); метод минимизации сглаживающего функционала невязки [6]; методы Винеровской деконволюции при задаваемом отношении сигнал/шум, итеративный Тихоновский метод с проекцией на множество неотрицательных функций и метод наилучшего правдоподобия в случае пуассоновского шума [7]; другие методы [12].

Необходимость обеспечить хорошую разрешающая способность при дискретизации функции рассеяния точки (PSF), которая является ядром свертки, приводит к тому, что расширенные изображения получаются практически в 2 раза больше, что дает четырехкратное увеличение числа

решаемых СЛАУ. Это приводит к существенному увеличению объема обрабатываемых данных и необходимости создания алгоритма, способного в реальном времени обрабатывать подобные объемы данных.

1. Постановка задачи

Задача восстановления сечений трехмерного объекта в оптической микроскопии сводится к системе уравнений

$$i_l(x,y) = \sum_{k=1}^{N} o_k(x,y) * h_{k-l}(x,y), \quad l = 1,2,\dots,N, \tag{1}$$

которая связывает наблюдаемые изображения $\{i_l = i_l(x,y) = i(x,y,z_l)\}$ с исходными данными $\{o_m = o_m(x,y) = o(x,y,z_m)\}$, расположенными на соответствующих N слоях объекта, трехмерной PSF

$$h_{k-l} = h_{k-l}(x,y) = h(x,y,z_k - z_l)\Delta z,$$

где "$*$" обозначает оператор двумерной свертки. Заметим, что для задач оптического секционирования вывод соответствующей PSF приводится, например, в [13].

Поскольку задача прямой деконволюции трехмерного объекта является достаточно ресурсоемкой, был осуществлен переход к системе уравнений относительно Фурье-образов I_l, O_l, H_l рассматриваемых функций i_l, o_l, h_l в плоскости спектральных переменных(u,v):

$$I_l\ (u,v) = \sum_{k=1}^{N} O_k(u,v) \cdot H_{k-l}\,(u,v)\ \ , \quad l = 1,2,\dots,N. \tag{2}$$

Система (2) в каждой точке (u,v) представляет собой СЛАУ относительно вектора коэффициентов Фурье $\vec{O}(u,v) = (O_1(u,v), O_2(u,v), \dots, O_N(u,v))$ исходных данных с матрицей S размера $N \times N$, составленной из коэффициентов $\vec{H}(u,v) = (H_1(u,v), H_2(u,v), \dots, H_N(u,v))$, и может быть записана в виде

$$S\vec{O} = \vec{I}. \tag{3}$$

При решении используется неявный итерационный метод, зависящий от параметра $\mu > 0$ и задающий регуляризирующий алгоритм для нахождения решений системы линейных алгебраических уравнений (3):

$$\vec{O}^{(k+1)} = (E + \mu\, S^*S)^{-1}\vec{O}^{(k)} + \mu(E + \mu\, S^*S)^{-1}S^*\vec{I}, \tag{4}$$

$$k = 1,2,\dots, \quad \vec{O}^{(0)} = (0,0,\dots 0).$$

Особенности выбора параметров метода, его реализация на скалярных и многопроцессорных CPU, а также результаты вычислительного эксперимента, в том числе с зашумленными данными, обсуждаются в [14], [15], [13]. Далее мы основное внимание уделим особенностям эффективной реализации метода с использованием возможностей современных GPU.

2. Специфика входных данных

Рассмотрим особенности входных данных алгоритма. В исходной постановке на вход поступает N наблюдаемых изображений размера $M \times M$, $(2N-1)$ предрассчитанных импульсных функций размера $(M-1) \times (M-1)$, учитывающих специфику различного вклада слоев, находящихся на равном расстоянии перед и за слоем, на который идет фокусировка. Характерные размеры рассматриваемых наблюдаемых изображений $M = 512, 1024, 2048,\quad N = 5, 10, 15.$ Отметим, что при переходе от системы уравнений (1), включающих в себя операцию свертки, к системе (2) относительно Фурье-образов I_m, O_n, H_n, учитывается, что в случае дискретных изображений при переходе от уравнения для свертки исходных массивов (1) к уравнению для Фурье-образов (2) массивы изображений i_l, o_l, h_l предварительно должны соответствующим образом расширяться нулями до размера $(2M) \times (2M)$. Помимо этого, для каждой точки Фурье-пространства необходимо работать с матрицей размера $N \times N$ и соответствующим вектором правых частей. Таким образом, объем памяти, необходимый для работы с текущей задачей, задается по формуле:

$$\Sigma = 4M^2N(N+2) * sizeof(\text{complex})$$

Это приводит к объемам данных, представленным на рисунке 1. Как видно из рисунка, даже для серверных компьютеров с объемами оперативной памяти до 128ГБ подобная задача не может быть решена с использованием напрямую только оперативной памяти. При росте же числа слоев трехмерного объекта и, соответственно, числа входных изображений, объем памяти будет увеличиваться пропорционально квадрату числа изображений.

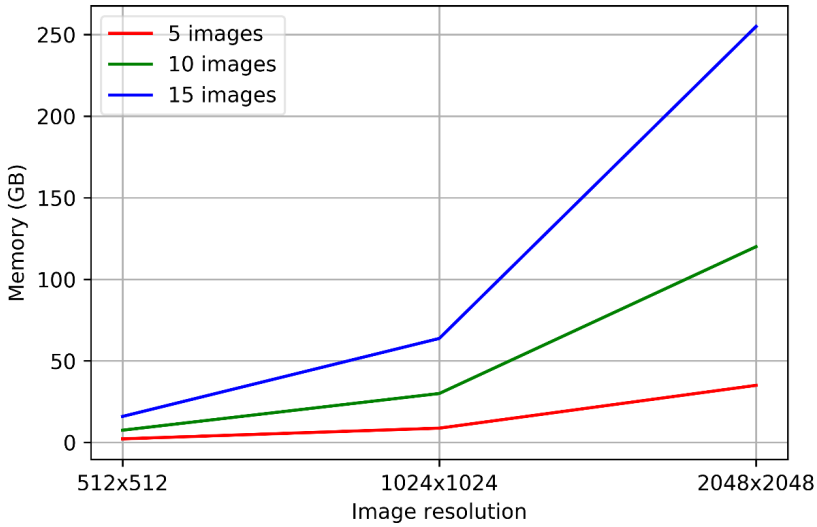

Рис. 1. *Объем необходимой памяти в зависимости от входных данных.*

3. Описание алгоритма

С учетом того, что итерационный алгоритм (4) для нахождения решения (3) в каждой точке (u, v) Фурье-пространства выполняется независимо от других точек алгоритмическая реализация на C++ с использованием мощностей только центрального процессора может быть логически разделена на три основных этапа:

1. Прямое БДПФ (быстрое дискретное преобразование Фурье) предварительно расширенных наблюдаемых изображений и передаточных функций.
2. Решение СЛАУ во всех точках (u, v).
3. Обратное БДПФ восстановленных изображений.

Отметим, что при расчете прямого и обратного БДПФ параллельность по данным является незначительной, что позволяет в достаточной степени использовать эффективные мощности современных многоядерных CPU, тогда как эффективный параллельный расчет решений значительно большего числа СЛАУ на них все еще недоступен. В этой связи было предложено следующее решение, позволяющее использовать мощности современных многоядерных дискретных графических процессоров NVIDIA с использованием программно-аппаратной архитектуры CUDA наряду с использованием стандарта OpenMP для параллельного выполнения на многоядерном центральном процессоре.

4. Описание предобработки данных

В силу специфики задачи импульсные функции остаются неизменными и могут быть вычислены заранее. Более того, учитывая, что

размерности итоговых расширенных наблюдаемых изображений также заранее известны, заранее может быть вычислен и ряд матриц размерности $N \times N$ для каждой точки Фурье-пространства (u, v), что позволит провести максимальную вычислительную работу до запуска метода уже на конкретных наблюдаемых входных данных. Дополнительная предобработка входных данных, параллельно с их последовательным получением, также позволит вынести ряд вспомогательных операций на этап предобработки.

Для эффективной работы с данными в оперативной памяти или памяти графической карты предлагается разбить данные на «чанки» (chunk - порция данных), размер и количество которых зависит от числа изображений, их размерности и доступной оперативной памяти и памяти графического процессора. Деление на чанки предлагается проводить для точек Фурье-пространства, поскольку вычисление для каждой точки Фурье пространства может произведено независимо от других и подобное разделение не повлечет дополнительных накладных расходов, связанных с синхронизацией обработки отдельных частей данных. Для входных данных предлагается провести аналогичную предобработку с разбиением на чанки для тех же точек Фурье-пространства.

С учетом деления на чанки общий объем памяти, включающий в себя набор матриц для каждой точки Фурье-пространства, соответствующие вектора правых частей и вспомогательную память, необходимую для работы используемых библиотек линейной алгебры и БДПФ, составит

$$\Sigma_{\mathrm{CPU}} = 4M^2N(N+2) * \mathrm{sizeof(complex)}/P$$

и

$$\Sigma_{\mathrm{GPU}} = 4M^2N(N+3) * \mathrm{sizeof(complex)}/P$$

байт для CPU и GPU соответственно, где P – число чанков.

Общая схема предобработки входных наблюдаемых данных представлена на рисунке 2. Сначала изображения расширяются до увеличенного размера для применения дискретной теоремы о свертки, затем к каждому из изображений применяется быстрое дискретное преобразование Фурье. Далее в каждый чанк записывается по части данных из Фурье-образа каждого наблюдаемого изображения.

Общая схема предобработки импульсных функций представлена на рисунке 3. Сначала импульсные функции, вычисленные с соотнесенной с физическим разрешением исходных данных точностью, расширяются до увеличенного размера для применения дискретной теоремы о свертки. После этого к каждой импульсной функции применяется БДПФ по поперечным переменным. Далее для каждой точки Фурье-плоскости вычисляются матрицы $C = \mu(E + \mu S^*S)^{-1}S^*$ и $B = (E + \mu S^*S)^{-1}$. Затем матрицы B и C

делятся на чанки по соответствующим точкам Фурье-плоскости и сохраняются в предрассчитанные файлы.

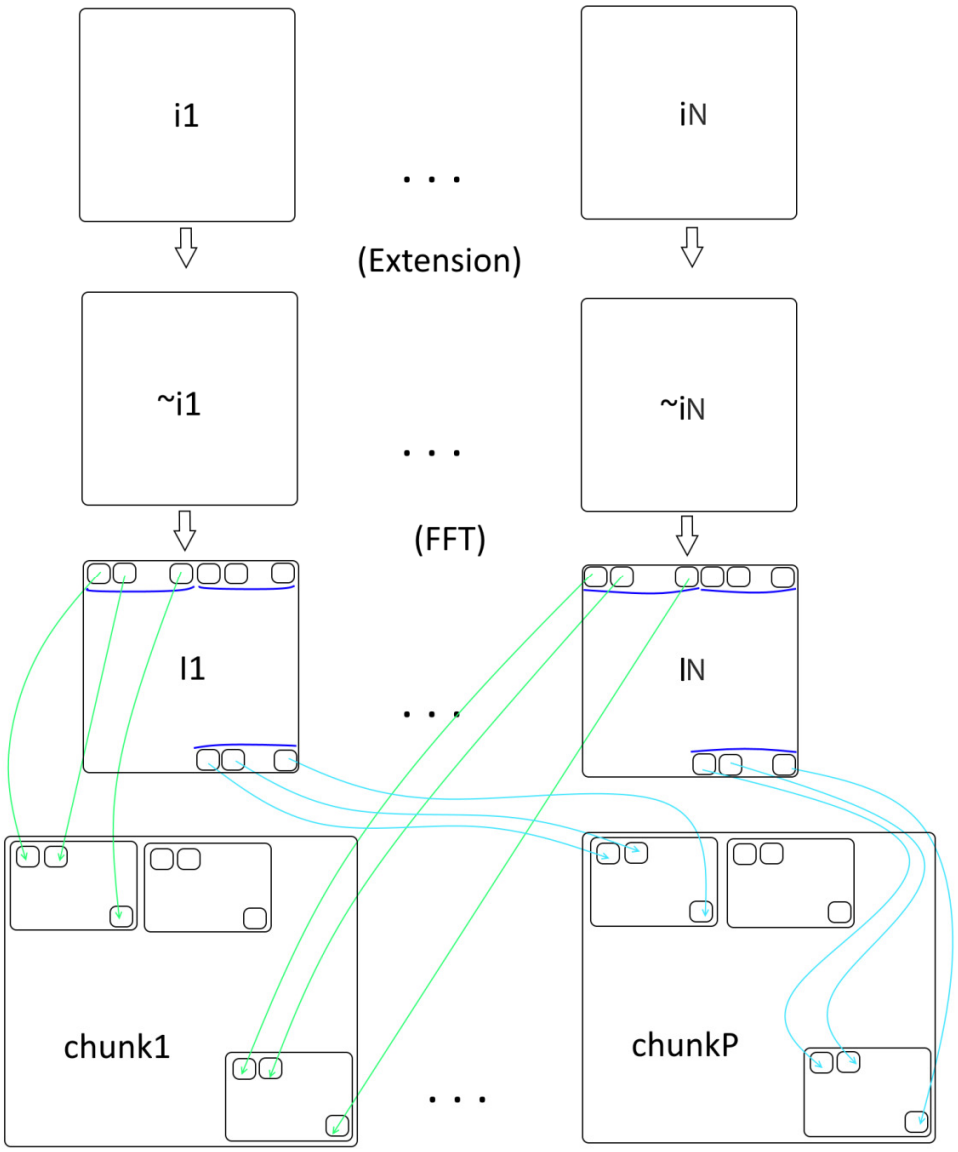

Рис. 2. *Предобработка наблюдаемых данных.*

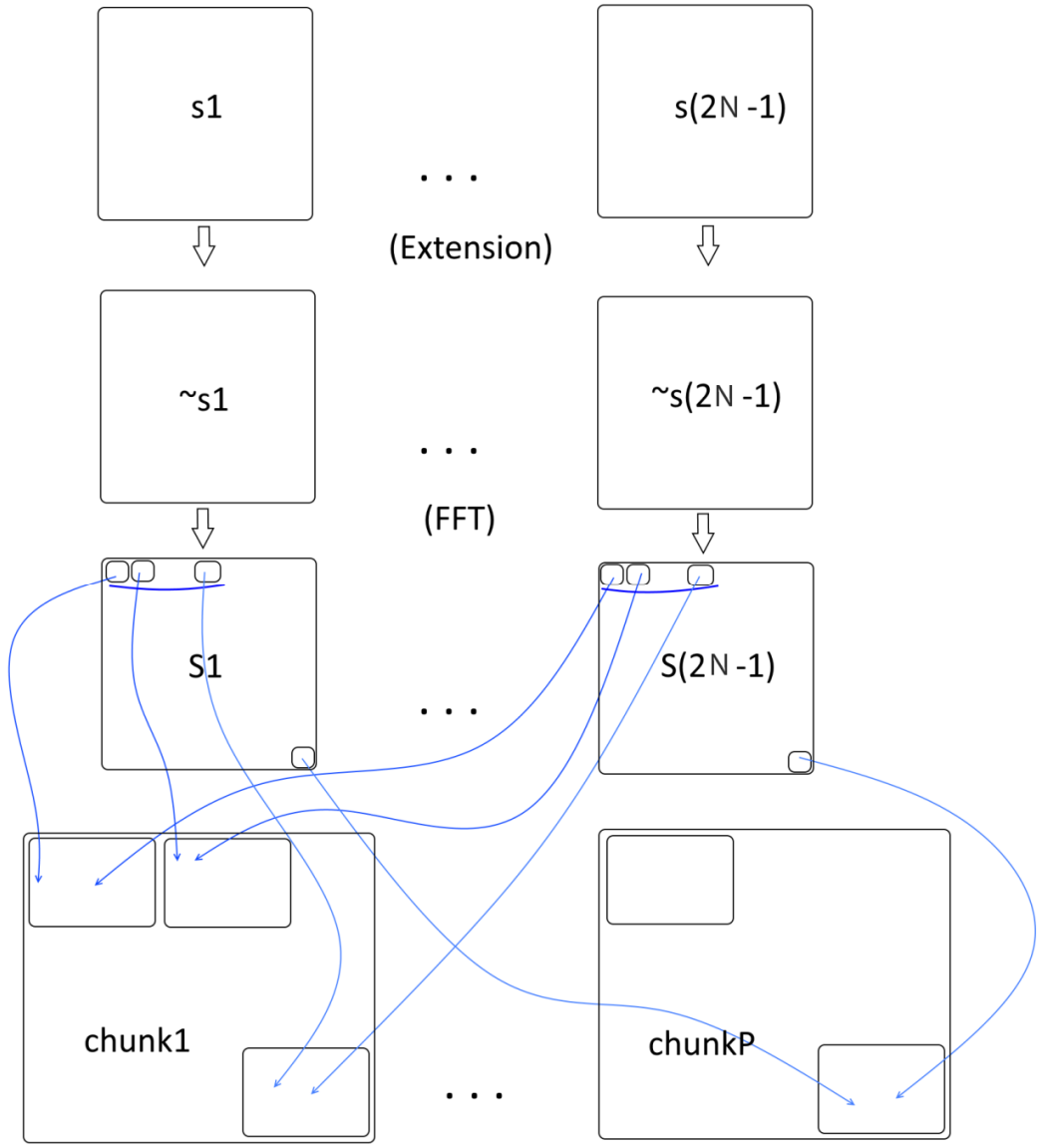

Рис. 3. *Предобработка PSF.*

Для каждого чанка алгоритм загружает в оперативную память соответствующий чанк матриц и преобразованных входных данных. Далее, если расчет производится на CPU, запускается итерационный метод решения соответствующей СЛАУ с учетом предрассчитанных матриц B и C для нескольких точек Фурье-пространства параллельно на всех доступных ядрах центрального процессора. При вычислении же на GPU запускается

параллельное решение всех соответствующих СЛАУ с использованием соответствующих параллельных методов линейной алгебры. После выполнения обработки всех чанков осуществляется обратное преобразование Фурье и срезка до соответствующих размеров.

5. Описание используемых программных средств

В качестве реализации БДПФ были использованы библиотеки FFTW [16] и cuFFT [17] от NVIDIA, включающие модули параллельной обработки БДПФ на центральных и графических процессорах соответственно. Для использования возможности параллельного БДПФ для ряда изображений с помощью стандарта OPENMP на центральном процессоре были использованы методы fftwf_plan_many_dft и fftwf_execute для данных с одинарной точностью и fftw_plan_many_dft и fftw_execute для данных с двойной точностью соответственно. Для того, чтобы максимально задействовать параллельные мощности графических ускорителей, были использованы методы cufftPlanMany и cufftExecC2C и cufftExecZ2Z для данных с одинарной и двойной точностью соответственно. Все перечисленные выше методы позволяют использовать организацию памяти для хранения массивов расширенных наблюдаемых изображений последовательно единым блоком, что дает возможность параллельно выполнять БДПФ сразу для ряда изображений, переданных на обработку едиными массивами.

В качестве реализации численных методов линейной алгебры использовались официальные реализации единого стандарта BLAS (Basic Linear Algebra Subprograms) от Netlib для Windows в рамках библиотеки LAPACK и cuBLAS [18] от NVIDIA. При выполнении расчетов на центральном процессоре на языке C++ это методы cgemv_, cgemm_, cgetrf_, cgetri_ для работы с данными одинарной точности и zgemv_, zgemm_, zgetrf_, zgetri_ для работы с данными двойной точностью соответственно. При выполнении расчетов на графическом процессоре cublasCgemmStridedBatched, cublasCgetrfBatched, cublasCgetriBatched и cublasZgemmStridedBatched, cublasZgetrfBatched, cublasZgetriBatched для данных с одинарной и двойной точностью соответственно.

Отметим, что параллельный вызов методов линейной алгебры при работе только с центральным процессором ограничен числом его ядер, тогда как основная особенность используемых методов графического процессора cublas в том, что они, в отличие от центрального процессора, позволяют

одновременно использовать все ядра GPU, число которых на несколько порядков превышает число ядер центрального процессора и работа с которыми оптимизирована именно для выполнения методов линейной алгебры.

Исходный код алгоритма доступен по ссылке [19].

6. Результаты

Тестирование производительности работы алгоритма проводилось на компьютере с центральном процессором Intel Xeon e5-2623 v4 2.6 ГГц с 30ГБ оперативной памяти и 500ГБ SSD диском, профессиональной графической картой NVIDIA Quadro P6000 с 3840 ядрами и 24ГБ памяти GDDR5X под управлением операционной системы Windows Server 2016 Datacenter. В качестве временного результата бралось среднее значение по 10 прогонам метода для случая данных, задаваемых с одинарной точностью.

На рисунке 4 приведено общее время работы метода на CPU и GPU соответственно для различных размерностей и различного числа изображений для 200 итераций. Как видно из рисунка, GPU-реализация алгоритма работает быстрее и позволяет обрабатывать 255ГБ данных, соответствующих 15 изображениям размерности 2048x2048, за время чуть более 3 минут, что позволяет решать подобные задачи практически в реальном времени.

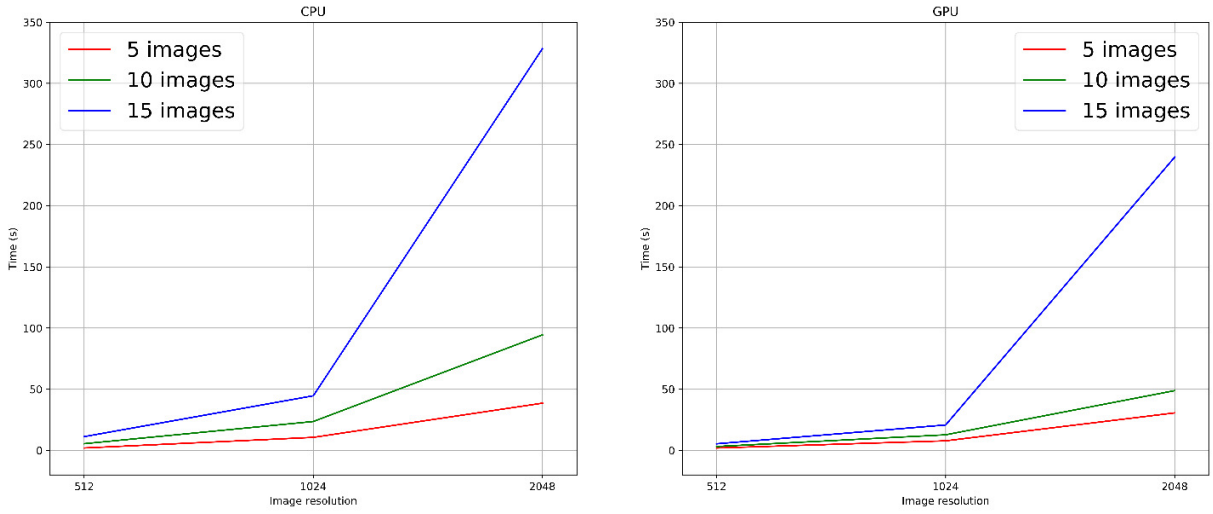

Рис. 4. *Время работы метода на 200 итерациях для CPU и GPU соответственно.*

На рисунках 5-7 представлены графики коэффициента масштабируемости по данным для обоих реализаций алгоритма для

различного числа слоев в зависимости от роста объема данных для случая 100 итераций. Из рисунков видно, что на сравнительно небольших объемах данных GPU-реализация алгоритма показывает коэффициент масштабируемости, близкий к единице, что фактически является идеальным коэффициентом, а для части экспериментов показывает даже меньшие значения коэффициента, что говорит более чем о линейном ускорении. CPU-реализация показывает менее эффективный, но также неплохой результат.

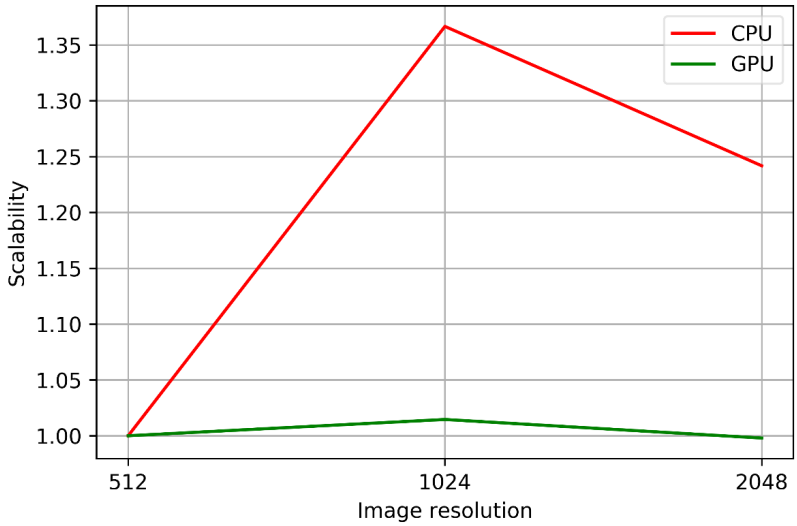

Рис. 5. *Коэффициент масштабируемости для 5 изображений для реализации метода на CPU и GPU для 200 итераций.*

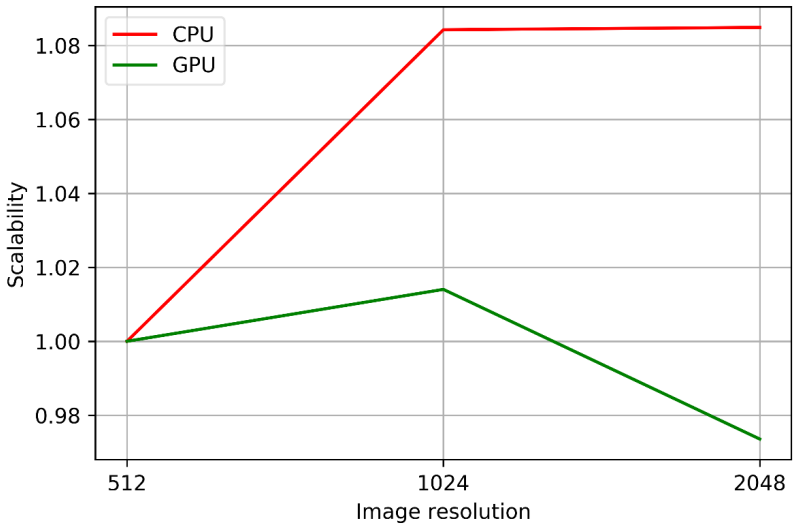

Рис. 6. *Коэффициент масштабируемости для 10 изображений для реализации метода на CPU и GPU для 200 итераций*

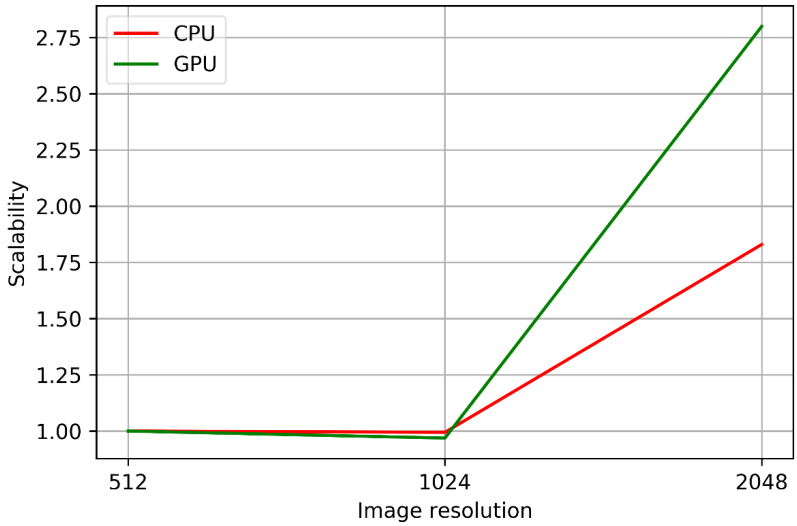

Рис. 7. *Коэффициент масштабируемости для 15 изображений для реализации метода на CPU и GPU для 200 итераций.*

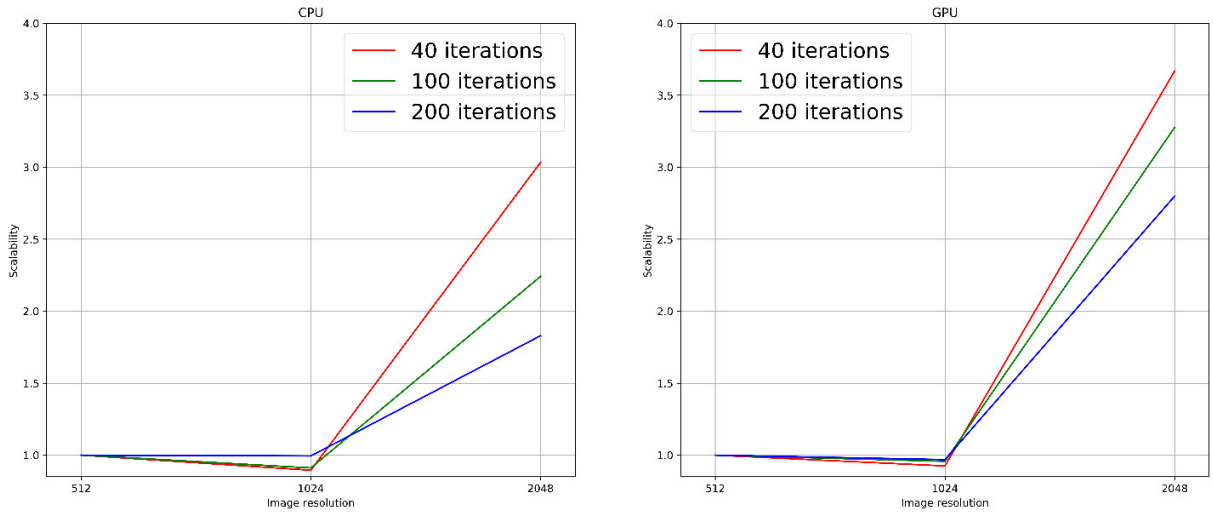

Рис. 8. *Изменение коэффициента масштабируемости для 15 изображений для реализации метода на CPU и GPU для различного числа итераций.*

Для существенных же объемов данных, как для CPU, так и для GPU-реализации можно отметить рост коэффициента масштабируемости, что объясняется существенным увеличением обрабатываемых данных и временными расходами на их считывание как в оперативную память, так и в память GPU. Отметим, что по сравнению с CPU-реализацией, для GPU-реализации копирование обрабатываемых данных в память GPU является дополнительной операцией с соответствующими дополнительными

временными затратами на ее выполнение. Это подтверждается и рисунком 8, на котором представлены графики масштабируемости в зависимости от числа итераций для различных реализаций алгоритма. Из графиков видно, что с ростом числа итераций, затраты на копирование данных в память начинают вносить все меньший вклад во время работы метода, что отражается в уменьшении коэффициента масштабируемости.

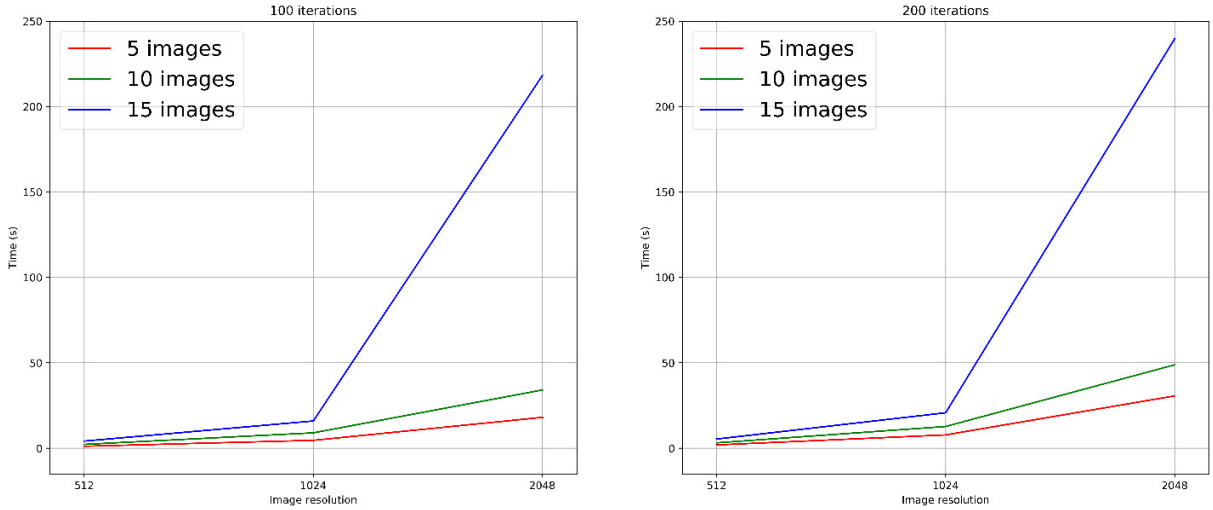

Рис. 9. *Время работы GPU-реализации метода для 100(слева) и 200(справа) итераций.*

Рис. 10. *Время работы CPU-реализации метода для 100(слева) и 200(справа) итераций.*

На рисунках 9 и 10 представлено время работы GPU и CPU-реализаций метода соответственно для случаев 100 и 200 итераций. Как видно из рисунка 9, с увеличением числа итераций в 2 раза время работы

метода на GPU увеличивается незначительно, что говорит о хорошей масштабируемости метода на ядрах графического процессора за счет запуска параллельного решения целого чанка СЛАУ и показывает, что основной вклад в подобные временные значения вносит время считывание данных и их копирование в память GPU. Для CPU-реализации увеличение времени с увеличением числа итераций в 2 раза более существенно и составляет порядка 30 секунд на размерности 1024×1024 и порядка минуты на размерности 2048×2048. Это объясняется тем, что несмотря на то, что метод выполняется параллельно, пусть и на более мощных ядрах центрального процессора, количество ядер практически на 3 порядка меньше числа более слабых ядер графического процессора. Отметим, что использование графической карты с большей пропускной способностью, например, NVIDIA V100 архитектуры VOLTA, способно еще больше сократить время работы метода.

Заключение

Представленный в работе итерационный алгоритм трехмерной деконволюции в Фурье-плоскости с использованием параллельных вычислений на CPU и GPU показал хорошую масштабируемость и позволяет обрабатывать произвольное число входных изображений произвольной размерности и ограничивается только объемом локального хранилища.

Дальнейшие способы ускорения работы алгоритма могут быть направлены на одновременную распределенную работу метода на CPU и GPU соответственно, что позволит сократить время работы до нескольких раз. Также в ряде случаев можно использовать априорную информацию о радиальной симметрии изображающей системы, что приводит к экономии одной размерности на этапе предобработки и хранения PSF, а также использование специальных асимптотических формул для быстрого вычисления интегралов Фурье от PSF специального вида. Эти вопросам будут посвящены наши дальнейшие публикации.

Литература

1. *Larichev A.V., Ivanov P.V., Iroshnikov N.G., Shmalgauzen V.I., Otten L.J.* Adaptive system for eye-fundus imaging // Quantum Electronics, 2002, vol. 32, no 10, pp. 902-908, doi:10.1070/QE2002v032n10ABEH002314.
2. *Wu Q, Merchant F, Castleman K.* Microscope image processing. Academic Press; 2008.
3. *Sage D., Donati L., Soulez F., Fortun D., Schmit G., Seitz A., Guiet R., Vonesch C., Unser M.* DeconvolutionLab2: An open-source software for

deconvolution microscopy // Methods - Image Processing for Biologists, 2017, vol. 115, pp. 28-41, doi: 10.1016/j.ymeth.2016.12.015.
4. *Eklund A., Dufort P., Forsberg D., LaConte S.M.* Medical image processing on the GPU – Past, present and future // Medical Image Analysis, 2013, vol. 17, issue 8, pp. 1073-1094, doi:10.1016/j.media.2013.05.008.
5. *Quammen C.W., Feng D., Taylor R.M.* Performance of 3D deconvolution algorithms on multi-core and many-core architectures // University of North Carolina at Chapel Hill Department of Computer Science. 2009. Technical Report TR09-001.
6. *Kromwijk S., Lefkimmiatis S., Unser M.* High-performance 3D deconvolution of fluorescence micrographs // Proceedings of the 2014 IEEE International Conference on Image Processing (ICIP'14), Paris, 2014, pp. 1718-1722, doi:10.1109/ICIP.2014.7025344.
7. *Karas P., Kuderjavý M., Svoboda D.* Deconvolution of huge 3-D images: parallelization strategies on a multi-GPU system // In: Kołodziej J., Di Martino B., Talia D., Xiong K. (eds) Algorithms and Architectures for Parallel Processing. ICA3PP 2013. Lecture Notes in Computer Science, vol. 8285, pp. 279-290. Springer, Cham. doi:10.1007/978-3-319-03859-9_24
8. *Bruce M.A., Manish J.B.* Real-Time GPU-Based 3D Deconvolution // Optics Express, 2013, vol. 21, no 4, pp. 4766–4773, doi: 10.1364/OE.21.004766.
9. *Domanski L., Bednarz T., Vallotton P., Taylor J.* Heterogeneous parallel 3D image deconvolution on a cluster of GPUs and CPUs// 19th International Congress on Modelling and Simulation, Perth, Australia, 12-16 December 2011, http://mssanz.org.au/modsim2011/A8/domanski.pdf.
10. *D'Amore L., Marcellino L., Mele V., Romano D.* Deconvolution of 3D fluorescence microscopy images using graphics processing units // In: Wyrzykowski R., Dongarra J., Karczewski K., Waśniewski J. (eds) Parallel processing and applied mathematics. PPAM 2011. Lecture Notes in Computer Science, 2012, vol. 7203, Part 1, pp. 690-699. Springer, Berlin, Heidelberg. doi:10.1007/978-3-642-31464-3_70.
11. *Cao L., Juan P., Zhang Y.* Real-time Deconvolution with GPU and Spark for big imaging data analysis // In: Wang G., Zomaya A., Martinez G., Li K. (eds) Algorithms and architectures for parallel processing. Lecture Notes in Computer Science, 2015, vol. 9530, pp. 240-250. Springer, Cham. doi:10.1007/978-3-319-27137-8_19.
12. *Zanella R., Zanghirati G., Cavicchioli R., Zanni L., Boccacci P., Bertero M., Vicidomini G.* Towards real-time image deconvolution:

application to confocal and STED microscopy // Scientific Reports 3, Article number: 2523 (2013), doi:10.1038/srep02523.
13. *Razgulin A.V., Iroshnikov N.G., Larichev A.V., Romanenko T.E., Goncharov A.S.* Fourier domain iterative approach to optical sectioning of 3d translucent objects for ophthalmology purposes // Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci., 2017, vol. 42-2, issue W4, pp. 173-177, doi:10.5194/isprs-archives-XLII-2-W4-173-2017.
14. *Razgulin A.V., Iroshnikov N.G., Larichev A.V., Pavlov S.D., Romanenko T.E.* On a problem of numerical sectioning in ophthalmology // Computer Optics, 2015, vol. 39, no 5, pp. 777-786, doi: 10.18287/0134-2452-2015-39-5-777-786.
15. *Krylov A., Nasonov A., Razgulin A., Romanenko T.* A post-processing method for 3D fundus image enhancement // 2016 IEEE 13th International Conference on Signal Processing (ICSP), Chengdu, 2016, pp. 49-52, doi: 10.1109/ICSP.2016.7877794.
16. http://www.fftw.org/
17. https://developer.nvidia.com/cufft
18. https://developer.nvidia.com/cublas
19. https://github.com/TatianaRomanenko/deconvolution