УДК 550.341; 519.688; 550.34.013.4 doi 10.26089/NumMet.v18r326

ПРИМЕНЕНИЕ NVIDIA CUDA ДЛЯ УСКОРЕНИЯ ОБРАБОТКИ СЕЙСМИЧЕСКИХ ДАННЫХ ПРИ ПОМОЩИ РАЗЛОЖЕНИЯ ПО ВОЛНОВЫМ ПАКЕТАМ

В. В. Никитин[1], А. А. Дучков[2], Ф. Андерссон[3]

Сейсмические данные характеризуются своей нерегулярностью, многомерностью и большим объемом. В настоящей статье рассматривается разложение данных по одному из наиболее оптимальных базисов — гауссовым волновым пакетам. На базе графических процессоров реализован и оптимизирован быстрый алгоритм прямого и обратного преобразования по трехмерным гауссовым волновым пакетам. Оптимизированная версия программы для графических ускорителей демонстрирует рост производительности в 2–6 раз по сравнению с 20-ядерным процессором. Проведено успешное тестирование алгоритмов на синтетических сейсмических данных: восстановление изображения по коэффициентам гауссовых волновых пакетов, сжатие данных, подавление шумов данных, интерполяция данных в случае пропущенных трасс.

Ключевые слова: гауссовы волновые пакеты, преобразование Фурье, CUDA.

1. Введение. В настоящее время сейсморазведка является самым информативным методом изучения земных недр для целей разведки запасов нефти и газа. В ходе проведения сейсмических работ на земной поверхности проводится измерение волн, отраженных от геологических границ внутри Земли. Записанное волновое поле трансформируется в изображение разреза земной коры, который затем используется при поиске полезных ископаемых.

В настоящее время метод отраженных волн (МОВ) [1] является основным сейсморазведочным методом. Волны создаются с помощью контролируемого источника упругих колебаний (взрыв или вибратор при наземных работах или воздушная пушка при морских работах). Отраженные волны распространяются вверх и регистрируются на поверхности группой (косой) приемников, после чего эксперимент повторяется для нового положения источника. Таким образом, при проведении работ вдоль профиля (линии на поверхности земли) регистрируется трехмерный куб данных — амплитуда волн как функция от координаты источника, координаты приемника и времени. На рис. 1 приведены примеры получаемых сейсмических данных.

Перечислим основные характеристики сейсмических данных, получаемых при проведении сейсмических работ в настоящее время:

— многомерность (в самом простом случае данные являются трехмерной функцией, а в самом общем случае могут быть пятимерными);

— большой объем данных (данные для одного профиля будут занимать несколько Гб, а данные площадной съемки требуют несколько Тб памяти);

— нерегулярность данных (в полевых условиях возникают пропущенные и забракованные трассы, искривление профилей наблюдений и др.).

Отметим специфическую структуру сейсмических данных — они состоят из волн (фронтов), которые сосредоточены вдоль кусочно-гладких поверхностей. На рис. 1 хорошо видна структура полезных волн и большие области, содержащие нули (помехи в случае полевых данных).

В настоящее время интенсивно ведется поиск оптимального базиса, в котором удобно представлять данные и проводить их обработку. В последние годы было предложено разложение данных по специфическим базисным функциям — локализованным плоским волнам, или волновым пакетам [2, 3]. Для

[1] MAX IV Laboratory, Lund University, Fotongatan 2, 22592, Lund, Sweden; Postdoctoral Researcher, e-mail: viktor.nikitin@maxiv.lu.se

[2] Институт нефтегазовой геологии и геофизики им. А. А. Трофимука СО РАН, просп. Коптюга, 3, 630090, Новосибирск; зав. лабораторией, e-mail: duchkovaa@ipgg.sbras.ru

[3] Center for Mathematical Sciences, Lund University, Solvegatan 18, 22100, Lund, Sweden; Senior Lecturer, e-mail: fredrik.andersson@math.lth.se

© Научно-исследовательский вычислительный центр МГУ им. М. В. Ломоносова

двумерного случая существует пакет программ Curvelab [4, 5], который используется в большинстве приложений в настоящее время. Для трехмерного разложения требуется параллельная реализация на кластерных системах. В пакете Curvelab есть трехмерное разложение, основанное на использовании MPI. Однако использовать его для решения практических задач довольно затруднительно (см. обсуждение в [6]). Для трехмерного разложения недавно были предложены новые базисы волновых пакетов [7, 8], для которых базисные функции обладают лучшими свойствами (симметрией и изометричной формой) для обработки и фильтрации данных по сравнению с пакетом Curvelab. Эти новые типы разложения также допускают реализацию в форме алгоритмов на основе быстрых преобразований Фурье на нерегулярных сетках (USFFT — Unequally-Spaced Fast Fourier Transform) [9, 10]. Вычислительная сложность таких алгоритмов имеет такой же порядок, что и стандартное быстрое преобразование Фурье (FFT). Кроме применения при обработке сейсмических данных преимущество новых базисов волновых пакетов заключается в том, что для них явно определена аналитическая формула как пространственной области, так и в фурье-области. Это дает возможность аппроксимации распространяющихся волн при помощи переноса гауссовых волновых пакетов вдоль лучей [11].

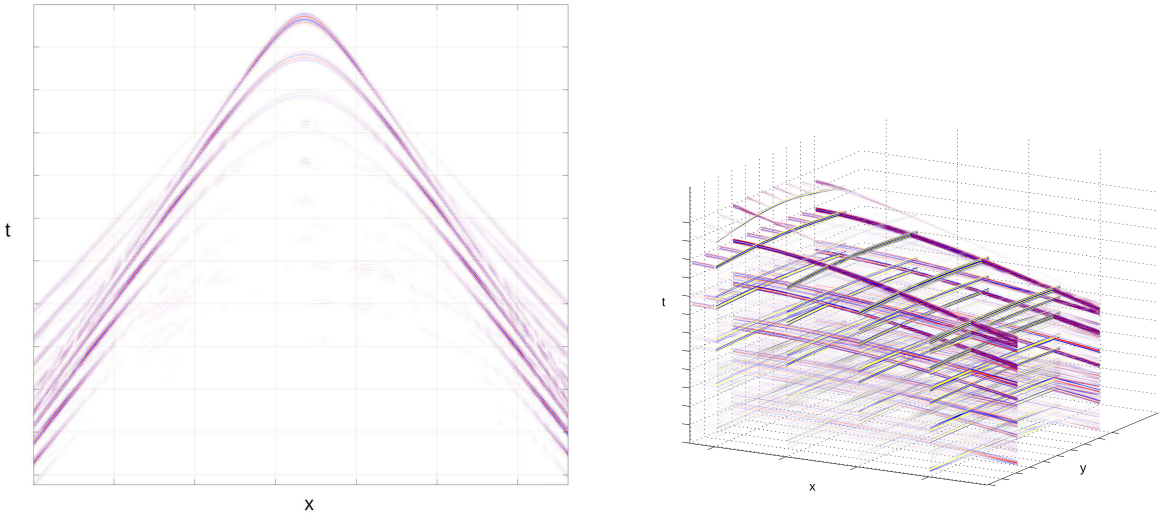

Рис. 1. Сейсмограмма и данные площадной съемки. Каждая трасса (по вертикали) — запись давления от времени t; приемник имеет координаты (x, y)

Для решения перечисленных задач необходима эффективная программная реализация процедуры прямого и обратного преобразования по гауссовым волновым пакетам, позволяющая обрабатывать большие объемы данных. Наиболее ресурсоемкой частью алгоритма является преобразование Фурье на нерегулярных сетках, которое состоит из большого количества мелких идентичных операций. Большинство из таких операций вычисляется независимо друг от друга, поэтому кажется естественным для ускорения вычислений использовать платформы GPU.

Технология NVidia CUDA для программирования на GPU в настоящее время широко используется при работе с сейсмическими данными. Значительные ускорения получены при моделировании распространения волн при помощи конечно-разностных схем [12, 13], когерентного суммирования [14], подавления кратных волн [15], гиперболического преобразования Радона для интерполяции данных [16] и др. Получаемый рост производительности достигает нескольких десятков раз по сравнению с последовательной реализацией алгоритмов на CPU. В большинстве случаев производительность GPU превышает производительность современных центральных процессоров, использующих все доступные вычислительные ядра.

2. Теория и алгоритмы.

2.1. Преобразование Фурье на нерегулярных сетках. Дискретное преобразование Фурье может быть представлено следующим образом:

$$\mathsf{F} = \mathcal{F}(\mathsf{f}): \quad \mathsf{F}_n = \sum_j \mathsf{f}_j e^{2\pi i n j/N},$$

где $\{\mathsf{F}\}_{n=-N/2}^{N/2-1}$ и $\{\mathsf{f}\}_{j=-N/2}^{N/2-1}$ — дискретно заданные значения функций. Алгоритм быстрого дискретного преобразования Фурье уменьшает вычислительную сложность с $\mathcal{O}(N^2)$ до $\mathcal{O}(N \log N)$ при помощи после-

довательности алгебраических операций [17].

Пусть теперь нам необходимо выполнить преобразование Фурье между регулярной и нерегулярной сетками размерами N и J соответственно. Для нормализованной сетки $\{x_j\}_{j=1}^{J}$, такой, что $|x_j| < 1/2$, дискретное преобразование Фурье с регулярной сетки на нерегулярную выглядит следующим образом:

$$\mathsf{F} = \mathcal{F}_{\mathbb{R}\to\mathbb{Z}}(\mathsf{f}): \quad \mathsf{F}_n = \sum_{j=0}^{J-1} \mathsf{f}_j e^{-2\pi i x_j n}. \tag{1}$$

Дискретное преобразование Фурье с нерегулярной сетки на регулярную, в свою очередь, задается следующим образом:

$$\mathsf{F} = \mathcal{F}_{\mathbb{Z}\to\mathbb{R}}(\mathsf{f}): \quad \mathsf{F}_j = \sum_{n=0}^{N-1} \mathsf{f}_n e^{-2\pi i x_j n}. \tag{2}$$

Алгоритм USFFT основан на применении операций свертки и FFT для аппроксимации сумм в (1) и (2) с заданной гарантированной точностью. В настоящей статье мы представим эвристическое описание работы алгоритма, подробная формулировка и оценки точности представлены в [9, 10].

Представим экспоненциальную функцию в виде распределения

$$e^{-2\pi i x_j \xi} = \int\limits_{-\infty}^{\infty} e^{-2\pi x \xi} \delta(x - x_j)\, dx.$$

Тогда преобразование Фурье с нерегулярной сетки на регулярную может быть записано в виде

$$\mathsf{F}_n = \sum_j \mathsf{f}_j e^{-2\pi i x_j n} = \int\limits_{-\infty}^{\infty} \sum_j \mathsf{f}_j \delta(x - x_j) e^{-2\pi i x n}\, dx = \int\limits_{-\infty}^{\infty} F(x) e^{-2\pi i x n}\, dx, \tag{3}$$

где $F(x) = \sum\limits_j \mathsf{f}_j \delta(x - x_j)$. Пусть φ — гладкая функция, имеющая ограниченный носитель. В нашей работе в качестве функции φ рассматривается гауссиан, другими примерами функций, тоже используемых для аппроксимации, являются B-сплайны и функции Кайзера–Бесселя [18, 19].

После умножения (3) на $\mathcal{F}\varphi$ получим

$$\mathcal{F}\varphi(n)\mathsf{F}_n = \int\limits_{-\infty}^{\infty} \sum_j \mathsf{f}_j \varphi(x - x_j) e^{-2\pi i x n}\, dx \quad \text{или} \quad \mathsf{F}_n = \frac{1}{\mathcal{F}\varphi(n)} \int\limits_{-\infty}^{\infty} \sum_j \mathsf{f}_j \varphi(x - x_j) e^{-2\pi i x n}\, dx.$$

Полученный интеграл может быть аппроксимирован при помощи правила трапеции, так как функция под интегралом достаточно гладкая в силу свертки с функцией φ. Поэтому достаточно точные результаты могут быть гарантированы при использовании значения x на регулярной сетке с некоторым множителем повышения частоты дискретизации ν. Другим важным фактором выбора функции φ является то, что эта функция имеет малый численный носитель. Это факт позволяет уменьшить время вычисления сверток. Более того, численный носитель зависит от заданной точности вычислений ϵ, поэтому функция φ часто заменяется на ее компактную версию φ^ϵ. В итоге аппроксимация $\mathcal{F}_{\mathbb{R}\to\mathbb{Z}}$ может быть алгоритмически быстро вычислена за три основных шага:

1) свертка в области изображения;

2) преобразование Фурье на регулярной сетке;

3) деление в частотной области.

Эти шаги соответствуют следующей формуле аппроксимации с точностью ϵ:

$$\mathsf{F}_n = \sum_j \mathsf{f}_j e^{-2\pi i x_j n} \approx \frac{1}{\mathcal{F}\varphi(n)} \frac{1}{\nu N} \sum_{k=-\nu N/2}^{\nu N/2-1} \left(\sum_j \mathsf{f}_j \varphi^\epsilon\left(\frac{k}{\nu N} - x_j\right)\right) e^{-2\pi i k n/\nu N}, \tag{4}$$

где суммирование по переменной k производится только для $K \ll N$ ненулевых элементов. Константа K зависит от выбора функции φ и точности ϵ.

Первая операция (свертка) имеет вычислительную сложность $\mathcal{O}(KN)$, преобразование Фурье на регулярной сетке вычисляется при помощи FFT за $\mathcal{O}(\nu N \log N)$ операций и операция деления на функцию $\widehat{\varphi}$ занимает $\mathcal{O}(N)$ операций. Общая вычислительная сложность алгоритма в итоге $\mathcal{O}(\nu N \log N + KN)$; на практике наиболее ресурсоемкой является часть $\mathcal{O}(KN)$.

Алгоритм для $\mathcal{F}_{\mathbb{Z}\to\mathbb{R}}(\mathsf{F})$ может быть построен аналогичным образом, применяя изложенные выше операции в обратном порядке, так как после замены $x_j \to -x_j$ оператор $\mathcal{F}_{\mathbb{Z}\to\mathbb{R}}(\mathsf{F})$ становится сопряженным к $\mathcal{F}_{\mathbb{R}\to\mathbb{Z}}(\mathsf{f})$:

$$\mathsf{F}_j = \sum_n \mathsf{f}_n e^{-2\pi i x_j n} \approx \frac{1}{\nu N} \sum_{k=-\nu N/2}^{\nu N/2-1} \left(\sum_n \frac{\mathsf{f}_n}{\varphi(n)} e^{-2\pi i k n/\nu N} \right) \widehat{\varphi}^{\epsilon}(k - x_j \nu N). \tag{5}$$

2.2. Гауссовы волновые пакеты. Прямой оператор преобразования по гауссовым волновым пакетам должен преобразовывать дискретно заданную функцию $f(x)$ в набор коэффициентов $\{c_\gamma\}$, параметризованных индексом γ:

$$C : f(x) \to \{c_\gamma\}, \quad \text{таких, что} \quad f(x) = \sum_\gamma c_\gamma \phi_\gamma(x),$$

где x — координаты в пространстве и ϕ_γ — базисные функции волновых пакетов.

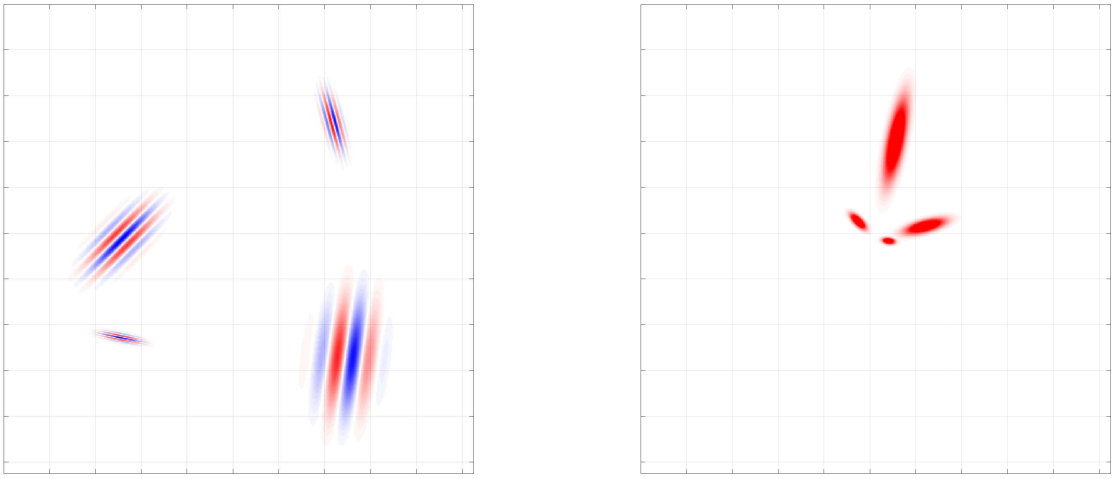

Рис. 2. Гауссовы волновые пакеты различной ориентации и размера в пространственной области (слева), в фурье-области (справа)

Для простоты будем рассматривать двумерный случай разложения. Трехмерный оператор является прямым аналогом двумерного. Базисные функции — гауссовы волновые пакеты определяются в пространстве следующим образом:

$$\phi_{k,\nu} = e^{2\pi i k^{\mathsf{T}}(x-y)}\, e^{-(x-y)^{\mathsf{T}} R_\theta^{\mathsf{T}} L(\alpha,\beta,k) R_\theta (x-y)}, \quad L(\alpha,\beta,k) = \ln(16) \begin{bmatrix} \dfrac{|k|^2}{\alpha^2} & 0 \\ 0 & \dfrac{|k|^2}{\alpha^2\beta^2} \end{bmatrix}.$$

Здесь (α, β, k) — параметры пакета, $L(\alpha, \beta, \gamma)$ — матрица коэффициентов затухания, y — центр пакета в пространстве, R_θ — матрица поворота, ассоциированная с ориентацией ν. На рис. 2 показаны примеры гауссовых волновых пакетов различной ориентации и размера.

Сами функции волновых пакетов задаются в фурье-области и имеют спектр $\widehat{\phi}_{k,\nu}(\omega)$, сосредоточенный в коробочке заданного размера, положения и ориентации. Здесь ω — координаты в фурье-области, дуальные к x. На рис. 3 (слева) показано покрытие фурье-области пересекающимися коробочками в двумерном случае: размеры коробочек и их расстояние от начала координат определяются параметром k,

ориентация — параметром ν. На каждой коробочке задается локальная регулярная сетка $\omega_j^{k,\nu}$ (точки на рис. 3, слева). Распределение пересекающихся коробочек в трехмерном случае показано на (рис. 3, справа). Для равномерного покрытия трехмерного спектра используются квадратуры Лебедева на сфере [20] (см. рис. 4).

Спектральные функции волновых пакетов подобраны таким образом, чтобы при возведении в квадрат задавать разбиение единицы фурье-области: $\sum\limits_{k,\nu} \widehat{\phi}_{k,\nu}(\omega) = 1$. Для того чтобы сопряженный оператор стал обратным, будем применять корректирующий фильтр, который является обратным к отклику на применение прямого и сопряженного преобразований на единичный импульс. Общая схема обратного оператора к C выглядит следующим образом:

$$f(x) \xleftarrow{\text{filtering}} f'(x) \xleftarrow{\text{adjoint USFFT}} \widehat{f}\big(w_j^{k,\nu}\big)\widehat{\phi}_{k,\nu}^2\big(w_j^{k,\nu}\big) \xleftarrow{\text{windowing}} \widehat{\phi}_{k,\nu}\big(w_j^{k,\nu}\big) \xleftarrow{\text{IFFT}} \{c_\gamma\}.$$

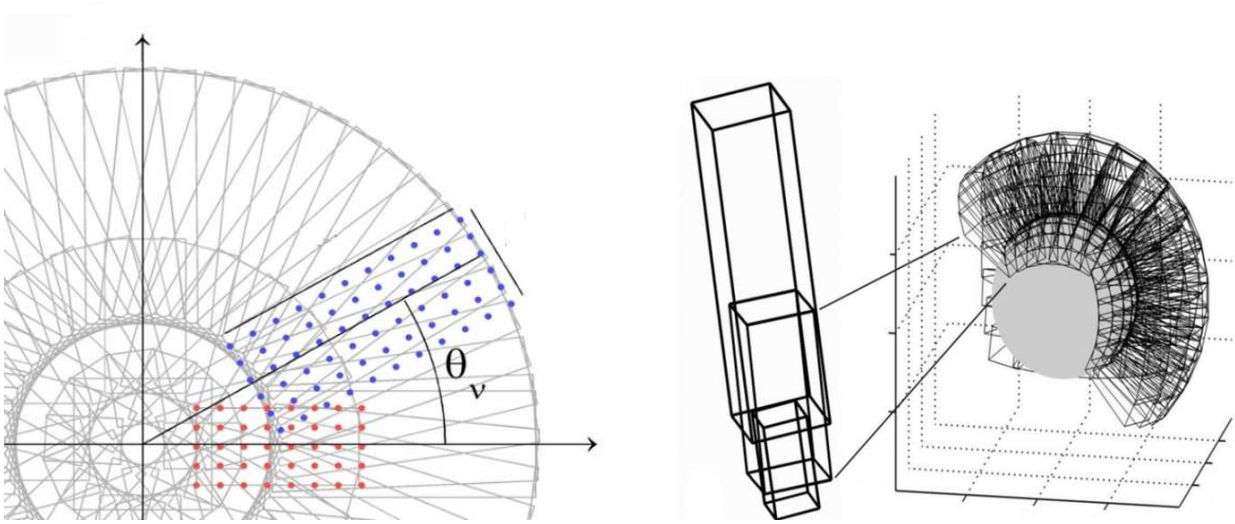

Рис. 3. Покрытие частотной области коробочками. Слева: локальные сетки коробочек различного размера и ориентации (иллюстрация для двумерного случая); справа: покрытие частотной области пересекающимися коробочками в трехмерном случае

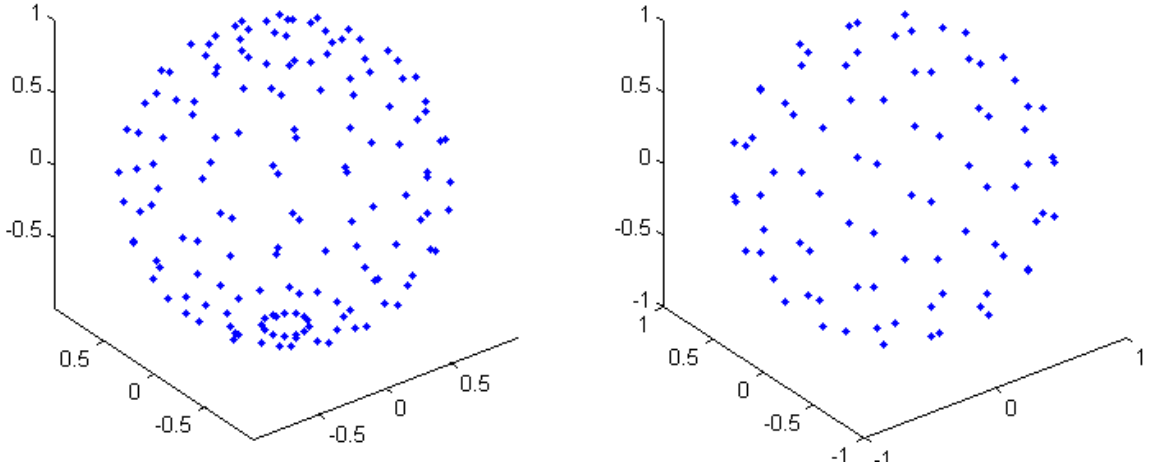

Рис. 4. Задание центров коробочек. Слева: сфера, построенная для одинаковых шагов по углам (неравномерное покрытие области). Справа: сфера, построенная при помощи квадратур Лебедева (равномерное покрытие)

3. Программная реализация и оптимизация.

3.1. Обоснование выбора GPU. В настоящее время графические ускорители широко используются для параллельной обработки данных в задачах с регулярным шаблоном доступа к памяти и с большим количеством арифметических операций. Одним классом таких задач является наложение фильтров на

изображения: размытие, выделение границ, подавление шума и др. В зависимости от размера (носителя функции) численное применение фильтра выполняется как в области изображения, так и в частотной области. В большинстве случаев фильтр представляет собой набор коэффициентов, который используется для дискретной свертки с матрицей изображения. Производится взвешенное суммирование значений части матрицы изображения с весами, заданными матрицей фильтра. Данный процесс выполняется независимо для каждой точки изображения, причем количество элементов при вычислении каждой суммы может быть достаточно большим. Его эффективность при выполнении на GPU была много раз продемонстрирована в литературе [21–24].

Для эффективной реализации USFFT необходимо быстро выполнять свертку с компактной версией гауссиана φ^{ϵ} (см. раздел 2.1 настоящей статьи). Свертка представляет собой взвешенное суммирование внутри заданной области (круга определенного радиуса), что при некоторой модификации фактически означает применение фильтра. Так как применение фильтров можно оптимально реализовать на графических процессорах, то для операции свертки с компактной версией гауссиана тоже представляется естественным использование GPU.

Кроме того, USFFT включает в себя выполнения преобразований Фурье на регулярных сетках. NVidia CUDA предоставляет специализированную библиотеку CUFFT (CUDA Fast Fourier Transform library), которая демонстрирует высокий рост производительности по сравнению с имеющимися известными аналогами FFTW, Intel MKL (см. NVidia Performance Report).

Одним из важных факторов, влияющих на скорость выполнения программы на GPU, является время копирования данных между оперативной памятью компьютера и памятью графического процессора. В некоторых случаях проблема медленного копирования данных может привести к существенному замедлению программы и в итоге к отказу от использования GPU. Однако в предлагаемом алгоритме такой проблемы можно избежать. Для разложения функций по гауссовым волновым пакетам необходимо много раз выполнять преобразование Фурье между сетками коробок и сеткой изображения. Вычисление значений на сетках коробок может проводиться независимо друг от друга. Поэтому при помощи функциональности CUDA Streams возможно совместить передачу данных с вычислениями на GPU.

3.2. USFFT на платформе GPU. Для алгоритмов разложения сейсмических данных по гауссовым волновым пакетам, а также во многих других приложениях преобразований Фурье на нерегулярных сетках необходимо осуществлять:

— прямое преобразование Фурье с регулярной на нерегулярную сетку;

— обратное преобразование Фурье с нерегулярной на регулярную сетку.

Прямое преобразование Фурье с регулярной сетки на нерегулярную сетку состоит из следующих программных шагов:

1) деление на гауссиан в области изображения;

2) удвоение изображения, дополнение нулями;

3) FFT;

4) задание функции на границах с использованием условия периодичности;

5) интерполяция и свертка в частотной области.

Для выполнения обратного преобразования с нерегулярной на регулярную сетку программные шаги выглядят следующим образом:

1) интерполяция и свертка в частотной области;

2) задание функции на границах с использованием условия периодичности;

3) обратное FFT;

4) взятие центральной части изображения;

5) деление на гауссиан в области изображения.

Наиболее ресурсозатратными частями алгоритмов USFFT являются преобразования Фурье и процессы интерполяции. Остановимся поподробнее на этих процессах и рассмотрим их адаптацию для GPU.

3.2.1. Реализация преобразований Фурье. Для выполнения быстрых преобразований Фурье NVidia предоставляет специализированную библиотеку cuFFT. Реализация этой библиотеки предлагает простой интерфейс для вычисления FFT с ускорением до нескольких десятков раз без необходимости разрабатывать собственную реализацию FFT, ориентированную на GPU.

Библиотека cuFFT с самых первых реализаций оптимизирована для выполнения преобразований Фурье для данных с размерами, являющимися степенью 2. Однако в последних версиях библиотеки добавлены реализации алгоритмов Кули–Тьюки [25] и Блуштайна [26], оптимизированные для размеров, представляемых в виде $\prod_i p_i^{n_i}$, где p_i — простые числа, а n_i — целые неотрицательные числа. Еще одной особенностью библиотеки cuFFT является наличие функциональности для выполнения преобразований для значений из вещественной области (без комплексной части). Такая функциональность позволяет ускорить вычисление преобразований вдвое и часто используется на практике при обработке измеренных данных.

Прямое и обратное преобразования Фурье могут производиться над большим количеством данных. Например, для разложения по гауссовым волновым входной функции, представляющей собой куб со стороной 128 точек, необходимо выполнить преобразование Фурье над 128 Мб (учитывая процедуру удвоения размеров, упомянутую в предыдущем разделе), со стороной 256 точек — 1 Гб, со стороной 512 точек — 8 Гб и т.д. Память графического процессора существенно меньше оперативной памяти компьютера, поэтому в большинстве случаев ее не хватит для обработки таких объемов данных. Например, глобальная память видеокарты Tesla C2050 составляет 2 Гб, видеокарта Quadro M4000 имеет 8 Гб и даже видеокарта GeForce Titan X, являющаяся на данный момент одной из самых мощных, имеет только 12 Гб памяти.

Трехмерное преобразование Фурье можно представить в виде комбинации двумерного и одномерного. Сначала выполняется преобразование по плоскостям, затем по столбцам. Такой способ требует значительно меньше памяти графического процессора. Однако большое количество копирований данных на видеокарту и обратно значительно увеличивает время работы. Количество копирований можно уменьшить, если копировать данные для одномерных преобразований Фурье по плоскостям. Библиотека cuFFT предоставляет интерфейс для выполнения нескольких одномерных преобразований Фурье за один вызов функции.

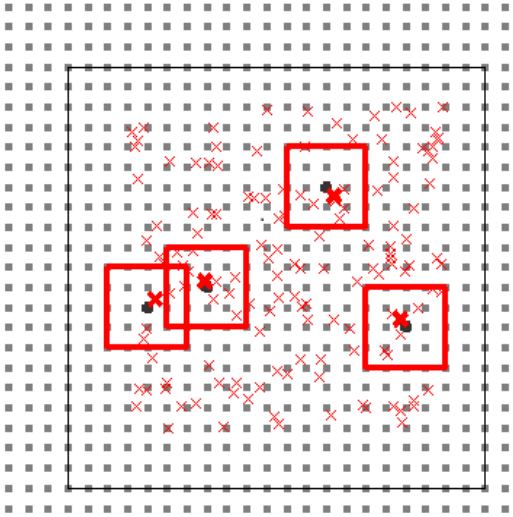

Рис. 5. Интерполяция в спектре. Вычисление значений на нерегулярной сетке (в крестиках) при помощи значений в точках регулярной сетки (квадратиках), находящихся внутри красного квадрата. Аналогично для вычисления значений на регулярной сетке

Интерполяции при прямом преобразовании. Интерполяция при прямом преобразовании называется “собирающей” интерполяцией; она проиллюстрирована на рис. 5. Серыми квадратами обозначена регулярная сетка, красными крестиками — нерегулярная. Для вычисления значения в крестике необходимо выполнить взвешенное суммирование значений в квадратах, находящихся в шаре радиуса M ($K = 2M + 1$ — количество используемых элементов в суммировании (4) по переменной k). Следует упомянуть, что формулы для одномерного, двумерного и трехмерного случаев аналогичны и отличаются только количеством вычисляемых сумм.

Таким образом, значения в крестиках вычисляются независимо друг от друга и поэтому могут быть вычислены параллельно. С каждым потоком на GPU ассоциируется вычисление значения в одном из крестиков, тогда общая размерность сетки потоков на GPU будет равна числу крестиков. Для запуска ядра (функции, выполняющейся на видеокарте одним потоком) необходимо задать размеры блока потоков `BS` и размеры сетки блоков `GS`. Существуют аппаратные ограничения для этих размеров. Так, например, для видеокарты Tesla C2050 (CUDA v 8.0) максимальные параметры таковы: `BS=(1024,1024,64)`, `GS = (65535,65535,65535)`. Если задавать одномерные `BS` и `GS`, то максимальное число потоков будет равно $1024 \times 65535 \approx 2^{20}$. Даже если точки нерегулярной сетки расположены равномерно в кубе со стороной 128, то их общее число равно $2^8 \times 2^8 \times 2^8 = 2^{24}$, что превышает допустимое ограничение. Размер блока потоков по первой координате для более современных GPU существенно увеличен, однако с увеличени-

ем объема обрабатываемых данных ограничение все же будет превышено. Поэтому размерность блоков и сеток блоков нужно задавать многомерной. В данной реализации общее число обрабатываемых точек равномерно распределяется на двумерную сетку потоков. Затем внутри ядра производится переход от двумерных индексов блоков и сеток блоков к одномерному индексу потока, который, как уже упоминалось, ассоциирован с точкой нерегулярной сетки.

Интерполяция при обратном преобразовании. Интерполяцию теперь необходимо провести в обратном направлении (рис. 5), т.е. с узлов нерегулярной сетки (крестиков) на узлы глобальной сетки (серые квадратики). Ее называют "рассеивающей" интерполяцией. Значение в точке нерегулярной сетки (крестике) нужно с разными весами записать в элементы глобальной сетки (серые квадратики), находящиеся в шаре радиуса M ($K = 2M + 1$ — количество используемых элементов в суммировании (5) по переменной k). Формулы для одномерного, двумерного и трехмерного случаев отличаются только количеством вычисляемых сумм.

Если аналогично прямому преобразованию задавать число потоков равным числу крестиков, то, как видно из рис. 5, возникает вероятность одновременного обращения к одной ячейке памяти для записи. Рассмотрим два крестика, находящиеся рядом, и для каждого выделим область влияния (красные квадратики); на рисунке видно, что они могут пересекаться. Следовательно, возможны случаи, когда два или более потоков будут пытаться одновременно писать в один и тот же участок памяти, что неминуемо приведет к неправильному результату (race condition — ошибки соревнования).

Для решения проблемы ошибок соревнований можно воспользоваться атомарными операциями. Атомарная операция происходит в два этапа: блокировка ресурса и выполнение самой операции. Здесь необходимо выполнить атомарное сложение. Возможны случаи, когда к одной ячейке памяти будут одновременно обращаться несколько операций, причем таких случаев довольно много. Результаты тестирования также показали, что использование атомарных операций в данном случае пагубно скажется на производительности программы. Для видеокарт это особенно заметно: при большом количестве потоков происходит большое количество блокировок для записи. Впрочем, сами производители NVidia не рекомендуют пользоваться атомарными операциями, потому что они настолько уменьшают скорость работы, что программа начинает работать медленнее последовательной. Остается вариант пересмотреть алгоритм интерполяции с нерегулярной сетки на регулярную. Рассмотрим два подхода.

Подход 1. Стандартный способ действия в ситуациях с ошибками соревнований — это изменение обхода данных таким образом, чтобы потоки отвечали за запись непересекающихся элементов. Для наглядности будем рассматривать двумерный случай, все рассуждения могут быть легко адаптированы для трехмерного случая.

Ассоциируем поток с точками глобальной сетки, находящимися на расстоянии $2M + 1$ друг от друга по каждой из координат (черные квадратики на рис. 6, слева). Значение $M = 6$ выбрано для алгоритма разложения данных по волновым пакетам с использованием одинарной точности вычислений.

Рассмотрим одну такую точку. Крестиками показаны узлы нерегулярной сетки, которые ассоциированы с этой точкой (т.е. после округления координат получаем текущую точку глобальной сетки). Значения в крестиках с различными весами "рассеиваются" по узлам глобальной сетки в шаре радиуса M (красный квадрат). Красные квадраты в данном случае не пересекаются.

Далее набор точек, с которыми ассоциированы потоки, изменяется (проводятся циклы $0 \dots 2M$ по каждой из координат). Таким образом, мы будем иметь $(2M + 1)^2$ запусков потоков для двумерного случая преобразования и $(2M + 1)^3$ — для трехмерного. Ошибок соревнований внутри каждого запуска потоков происходить не будет. В процессе вычисления возникает необходимость быстро извлекать набор точек нерегулярной сетки, используемых одним потоком. Поэтому проводится быстрая сортировка всех точек "по ключу". Ключом является индекс ближайшей точки в глобальной сетке. В процессе сортировки производятся перестановки не только значения сортируемого массива, но и массивов координат и значений точек нерегулярной сетки. Такая сортировка занимает незначительную часть от всего времени выполнения программы. При необходимости она может быть ускорена при помощи библиотеки Thrust (Cuda Toolkit v. 8), предоставляющей оптимизированные функции для сортировки данных и их переупорядочивания по заданному критерию.

Необходимо для каждой точки глобальной сетки определить начальный и конечный индексы в массиве необходимых точек нерегулярной сетки (точек, ближайших к рассматриваемой точке глобальной сетки). Возникает массив начальных индексов (конечные индексы могут быть найдены при помощи следующих элементов массива взятием разности). Значение в массиве точек нерегулярной сетки, расположенное по индексу k с различными весами, "рассеивается" по узлам глобальной сетки. При этом гарантировано отсутствие ошибок соревнований.

На примере трехмерного случая проведем оценку объемов вычислений. Количество запусков ядер равно $(2M+1)^3$. Количество потоков в рамках одного запуска ядра $P = \left\lceil \frac{N^3}{(2M+1)^3} \right\rceil$. Для $N = 128$ и $N = 256$ имеем $P = 955$ и $P = 7637$ соответственно. Во-первых, следует отметить, что на запуск ядра тратится некоторое время (например, для инициализации графического конвейера [27]). Во-вторых, количество потоков очень мало. Оптимальный размер блока потоков в нашем случае равен 1024. Мы будем иметь всего несколько блоков потоков. Во время выполнения будет очень мало переключений между этими блоками, и никаких достоинств конвейерной обработки данных не получится. Однако для данных большего размера $N = 512, 1024, \ldots$ этот подход может оказаться оптимальным.

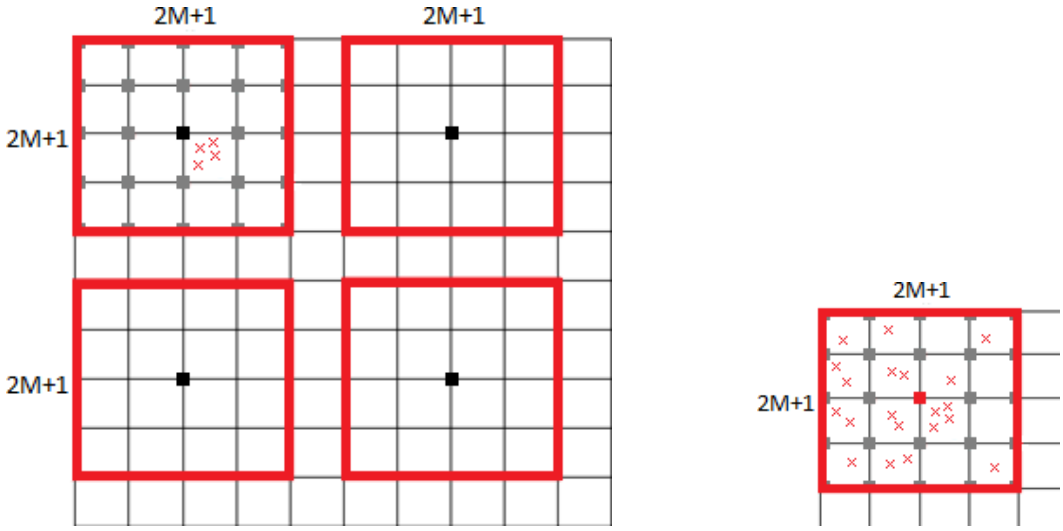

Рис. 6. Интерполяция в спектре. Подход 1 (слева): Вычисление значений на регулярной сетке (в серых квадратиках) при помощи значений в точках нерегулярной сетки (крестиках), ближайших к черному квадратику. Подход 2 (справа): Вычисление значения на регулярной сетке (в красном квадратике) при помощи значений в точках нерегулярной сетки (крестиках), ближайших к серым квадратикам

Подход 2. Рассмотрим другую модификацию алгоритма, которая является более оптимальной с точки зрения загруженности конвейера блока потоков. Аналогично, каждый поток ассоциирован с узлом глобальной сетки (рассматриваются только необходимые узлы). Внутри себя он исполняет цикл по узлам глобальной сетки, находящимся в шаре радиуса M. Значения в точках нерегулярной сетки, ассоциированные с этими узлами, с определенными весами прибавляются к значению в центральной точке красного квадрата (с ней ассоциирован поток) (рис. 6, справа). Теперь нет необходимости задавать расстояние между точками, с которыми ассоциированы потоки, чтобы избежать ошибок соревнований. Число потоков равно числу узлов регулярной сетки. Каждый поток производит запись в один элемент сетки, поэтому ошибок соревнования нет. Обращения к узлам нерегулярной сетки кэшируются вдоль одной размерности, так как лежат в отсортированном массиве. Общее число потоков для $N = 128$ и 256 равно соответственно 128^3 и 256^3, что при размере блока потоков 1024 дает огромное количество блоков потоков. Конвейерная обработка блоков потоков в этом случае приводит к существенному росту производительности.

3.3. Разложение по гауссовым волновым пакетам на основе USFFT. Для реализации непрерывного прямого преобразования по гауссовым волновым пакетам необходимо проделать следующие операции:

— прямое преобразование Фурье для всей функции;

— выделение частей спектра, соответствующих отдельным коробочкам;

— обратное преобразование Фурье на каждой из коробочек.

Заметим, что коробочки имеют прямоугольную форму, что важно для выполнения третьей операции — локального обратного преобразования Фурье на каждой коробочке (в этом случае на каждой коробочке можно задать локальную регулярную сетку). Однако они имеют разную ориентацию (рис. 3). Это повлияет на схему вычислений для дискретной реализации прямого преобразования:

— прямое преобразование Фурье с регулярной сетки (сетки изображения) на нерегулярную (сетку коробки);

— умножение спектра в коробочке на весовую функцию;

— обратное FFT на каждой из коробочек.

Объединение локальных сеток на всех коробочках можно рассматривать как нерегулярную сетку, покрывающую спектральную область (рис. 7 для двумерного случая). Преобразование Фурье с глобальной (регулярной) сетки на локальную (нерегулярную) создает наибольшую вычислительную трудность. Для эффективного выполнения этой операции принято использовать USFFT.

Восстановление изображения. Для того чтобы при помощи полученных коэффициентов разложения получить исходное изображение, необходимо выполнить схему вычислений преобразования по гауссовым волновым пакетам в обратном порядке:

— прямое FFT на каждой из коробочек;

— умножение спектра в коробочке на весовую функцию;

— обратное преобразование Фурье с нерегулярной сетки (сетки коробок) на регулярную (сетку изображения), т.е. применение USFFT.

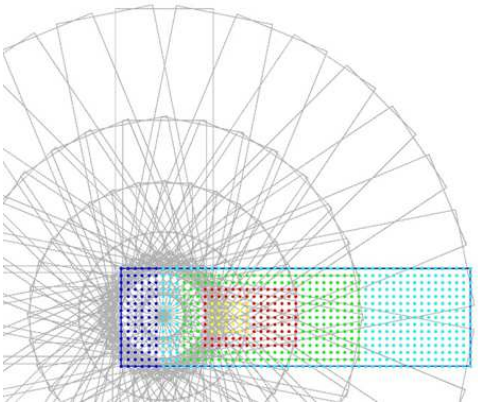

Рис. 7. Нерегулярная сетка в двумерной спектральной области (объединение повернутых сеток на всех коробочках)

Полученный сопряженный оператор C^* не является обратным к C, поэтому для получения исходной функции производится свертка с корректирующим фильтром. Корректирующий фильтр находится следующим образом:

1) создание нулевого сигнала g с единичным импульсом в центре;

2) применение к g прямого преобразования по гауссовым волновым пакетам, получение набора коэффициентов c_γ;

3) применение обратного преобразования по гауссовым волновым пакетам к c_γ, получение сигнала h;

4) взятие обратного преобразования к h.

Действие корректирующего фильтра представлено на рис. 8.

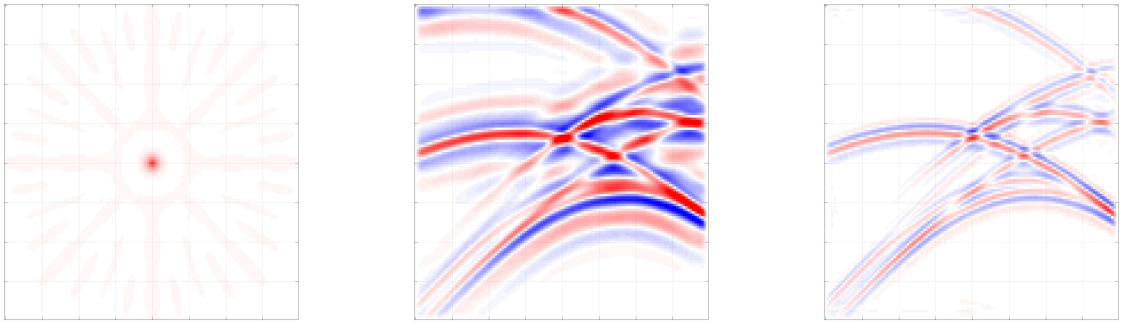

Рис. 8. Корректирующая фильтрация: отклик преобразования (C^*C) на единичный импульс (слева), применение оператора C^*C к сейсмическим данным (в центре), результат фильтрации (справа)

3.4. Аппаратная оптимизация кода. Оптимизация производительности для GPU была выполнена по следующим основным направлениям [27, 28]:

— оптимизация инструкций;

— оптимизация работы с памятью;

— оптимизация загруженности GPU.

Профилирование программы проводилось при помощи NVidia Visual Profiler (рис. 9). Данный инструмент позволяет узнать время работы отдельных частей программы, количество непоследовательных обращений к памяти, размеры блоков потоков и сетки блоков, количество промахов по кэшу и многое другое. Все это помогает при дальнейшей оптимизации.

Оптимизация инструкций. Во-первых, нужно оптимизировать операции деления. На GPU операция деления занимает 36 тактов, а на взятие обратного и на умножение требуется 16+4=20 тактов. Поэтому выражения типа `x/y` надо заменить на `1/y*x`. Во-вторых, тригонометрические операции одинарной точности `sinf`, `cosf`, `expf` нужно заменить на их быстрые аналоги. При небольшой потере точности функции `__sinf`, `__cosf`, `__expf` выполняются намного быстрее. Кроме того, важной оптимизацией является использование побитовых операций. Например, в нашем коде много раз приходится удваивать значения либо уменьшать в два раза. Такие вычисления можно ускорить при помощи операций побитового сдвига.

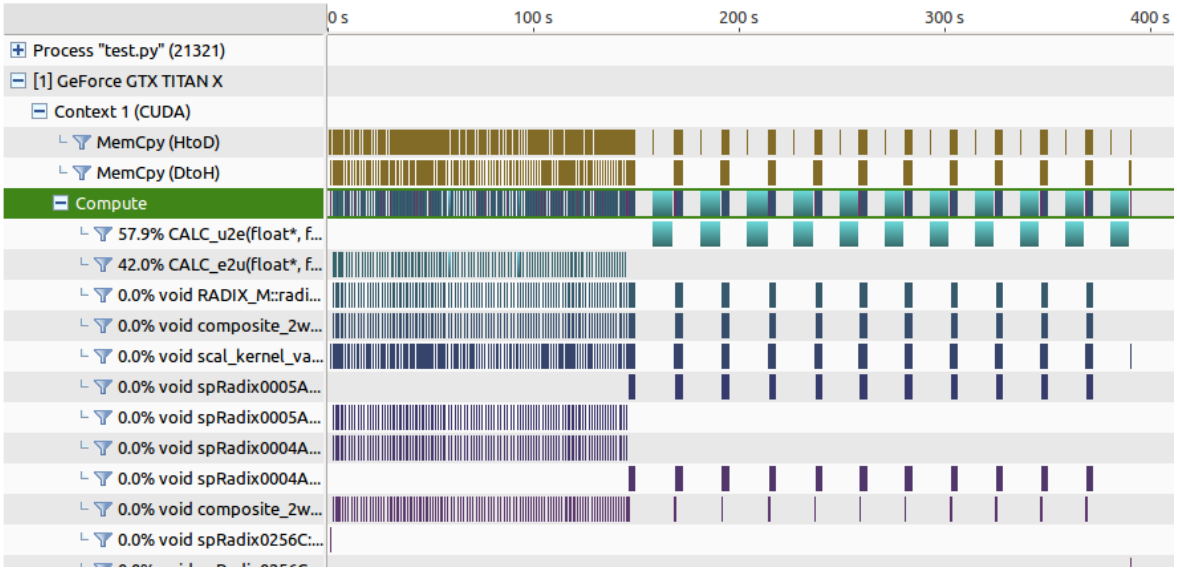

Рис. 9. NVidia Visual Profiler. Профилирование ядер для выполнения прямого и сопряженного операторов разложения по гауссовым волновым пакетам. При прямом преобразовании операция USFFT осуществляется отдельно для каждой коробки (11×3 запусков ядра `CALC_e2u`). При сопряженном — нерегулярные сетки коробок объединяются (11 запусков ядра `CALC_u2e`)

Более интересной для оптимизации является операция округления `round`, которая в программе используется для того, чтобы найти ближайшую точку нерегулярной сетки к регулярной. Число тактов процессора можно сократить, если использовать структуру представления вещественных чисел на компьютере. Так как мантисса занимает 23 бита, то операцию округления можно выполнить следующим образом: `round(x)=x+(2<<23)-(2<<23)`. Аналогичным образом может быть реализована операция округления в меньшую сторону.

Оптимизация работы с памятью. Оптимизация работы с памятью является самой важной оптимизацией при программировании на графических процессорах. Пропускная способность памяти очень сильно влияет на производительность. Необходимо добиваться того, чтобы реальная пропускная способность была сравнима с теоретической. Внутри цикла для того, чтобы взять значение из спектра, производится обращение к глобальной памяти устройства. В последних версиях устройств с compute capability $\geqslant 2.0$ на каждом мультипроцессоре устройства имеется кэш L1. Он в данном случае будет использоваться неэффективно, так как паттерн доступа к данным хаотичный, а мультипроцессоры не могут обращаться к кэшам L1 других мультипроцессоров, что приводит к большому количеству промахов ($> 90\%$). Лучше всего в этом случае отключить кэш, добавив опцию `-Xptxas -dlcm=cg`. Паттерн доступа хаотичный и данные не влезают в разделяемую память устройства — это две причины того, что нужно воспользоваться текстурами CUDA. Обращения к глобальной памяти видеокарты через текстурную ссылку кэшируются, текстурный кэш является общим для всех мультипроцессоров (SM). При попадании по кэшу GPU затрачивает 4 такта вместо 400 при чтении элемента из глобальной памяти. В коде программы мы воспользовались трехмерной текстурной ссылкой.

Следующим важным шагом оптимизации работы с памятью является использование регистров. Например, при расчете взвешенной суммы (интерполяции) каждый раз проводилась запись в глобальную

память устройства внутри цикла, что на каждой итерации требовало 400 тактов. После оптимизации суммирование проводилось для выделенной переменной-регистра, которая записывалась в глобальную память по окончании выполнения цикла. В рассматриваемом цикле используется большое количество переменных, созданных непосредственно в функции ядра. Попытка хранить все эти переменные в регистрах приводила к спиллингу регистров, т.е. часть переменных выгружалась в медленную локальную память. Для уменьшения количества регистров мы воспользовались константной и разделяемой памятью, обращение к которым занимает 4 такта. Для хранения переменных, не изменяющихся в процессе выполнения ядра, мы создали специальную структуру и поместили ее в константную память видеокарты. Для хранения изменяющихся переменных кроме регистров использовалась разделяемая память. После проведения оптимизаций по работе с памятью удалось добиться увеличения реальной пропускной способности памяти с 20% (до оптимизации) до 70% от теоретически возможной.

Оптимизация загруженности GPU. Размеры регистровой и разделяемой памяти, а также выбор размера блока потоков влияют на эффективность использования видеокарты.

При оптимизации был использован инструментарий CUDA Occupancy Calculator, позволяющий подобрать оптимальные размеры блока потоков для заданного количества регистров на поток и разделяемой памяти на блок. Визуализация зависимости загруженности видеокарты от различных параметров представляется в виде соответствующих графиков.

Вычислительные ядра для операций взвешенного суммирования и рассеивания используют большое количество регистров, значения в некоторых регистрах в процессе выполнения ядра копируются в локальную память GPU (происходит спиллинг регистров). Регистровая память находится на каждом мультипроцессоре. Для того чтобы уменьшить спиллинг регистров и увеличить загруженность GPU вычислениями, было произведено уменьшение размера блока потоков.

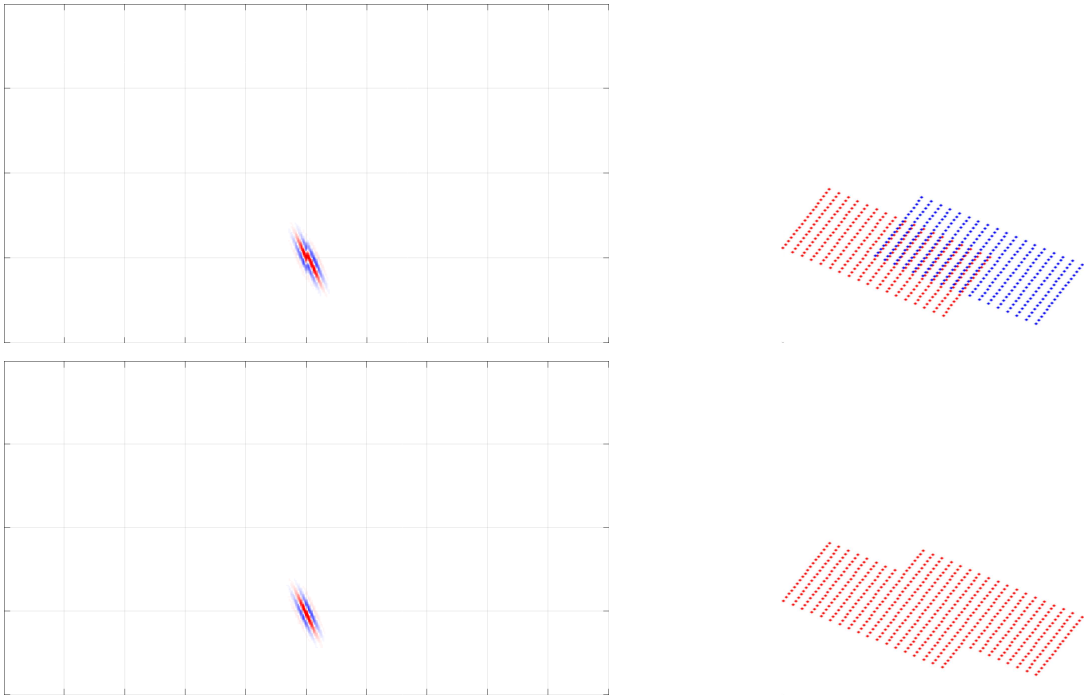

Рис. 10. Совмещение сеток при обработке изображения по частям: изображение при несовпадающих сетках коробок в спектре (сверху), изображение после сдвига сеток коробок в спектре (снизу)

3.5. Работа с большими объемами данных. Как уже упоминалось, сейсмические данные имеют большой объем, а также они могут иметь не кубическую структуру. Наш алгоритм рассчитан для обработки кубических данных максимального размера 256^3 из-за ограничений по памяти. Возникает необходимость разбивать данные на части и обрабатывать их отдельно, после чего объединять коэффициенты. При этом необходимо выполнять дополнительные работы по совмещению сеток на коробках, расположенных вдоль одного направления в спектре (рис. 10). Для того чтобы сдвинуть точки в спектре, воспользуемся правилом модуляции: $\mathcal{F}h(\omega) = \mathcal{F}f(\omega - \omega_0)$, тогда $h(x) = e^{2\pi i\omega_0 x} f(x)$.

Предположим теперь, что данные разбиваются на много частей. В этом случае для каждого из углов строится одна большая сетка, к которой сдвигаются сетки коробок одного направления. Для трехмерного

случая сдвиги выполняются аналогично. Для примера рассмотрим параллелепипед размера $1024 \times 1024 \times 400$. Разложение по пакетам выполняется только для кубических данных; пусть мы работаем с кубами со стороной 256. Тогда мы дополняем данные нулями до размера $1024 \times 1024 \times 512$ и получаем параллелепипед, содержащий 32 куба со стороной 256. Затем выполняется 32 разложения по гауссовым волновым пакетам, внутри которых осуществляются сдвиги сеток к одной большой общей сетке. Для уменьшения объемов используемой памяти данные в общей сетке имеют разреженное представление.

Отметим, что обработка всех кубов, составляющих параллелепипед, выполняется независимо друг от друга. Поэтому для обработки каждого такого куба можно выделить отдельный вычислительный узел, например отдельную видеокарту. Операция сдвига сеток занимает менее 1% от всего времени выполнения программы; отсюда можно сделать вывод, что при увеличении числа вычислительных модулей рост производительности будет линейным.

4. Тестирование производительности и апробация на сейсмических данных.

4.1. Анализ производительности. Тестирование производительности производилось на следующих вычислительных платформах:

— Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz, 10 physical cores (20 virtual cores);

— Quadro M4000 (1664 cores, clock rate 0.77GHz, memory 8Gb, CUDA Capability 5.2);

— Tesla K80 (2496 cores, clock rate 0.82GHz, memory 12Gb, CUDA Capability 3.7);

— GeForce TITAN X (3072 cores, clock rate 1.08GHz, memory 12Gb, CUDA Capability 5.2).

Все GPU присоединены посредством PCI Express x16 шины. Компиляция кода производилась отдельно для каждой платформы с использованием CUDA Toolkit 8.0. Точность вычислений одинарная.

Для CPU распараллеливание происходит следующим образом. Поток при преобразовании с регулярной сетки на нерегулярную ассоциируется с узлом нерегулярной сетки (рис. 5). При преобразовании с нерегулярной сетки на регулярную для каждого потока выделяется своя отдельная память для хранения куба данных с результатами операции взвешенного рассеивания. Затем производится суммирование данных этих кубов (reduction). Получившийся куб данных в этом случае будет построен без каких-либо ошибок соревнований между потоками. Следует отметить, что этот способ не является допустимым для GPU, так как количество потоков на GPU исчисляется тысячами и выделить столь большое количество памяти для кубов не получится.

В табл. 1 представлено время выполнения прямого и сопряженного трехмерного USFFT для трех случаев: количество точек нерегулярной сетки равно количеству точек нерегулярной ($T = N^3$), в два раза меньше ($T = N^3/2$) и в 4 раза меньше ($T = N^3/4$). Во всех случаях нерегулярная сетка равномерно покрывает трехмерный куб данных.

Таблица 1

Время (в секундах) выполнения 3D USFFT преобразования с регулярной сетки на нерегулярную (e2u) и с нерегулярной сетки на регулярную (u2e) для кубов размера $N \times N \times N$ и различного количества точек нерегулярной сетки (K)

	Xeon E5-2650		Quadro M4000		Tesla K80		GeForce TITAN X	
	e2u	u2e	e2u	u2e	e2u	u2e	e2u	u2e
$N = 64, \quad T = N^3/4$	0.7	0.8	0.3	0.6	0.3	0.5	0.2	0.4
$N = 64, \quad T = N^3/2$	1.3	1.5	0.4	0.7	0.3	0.6	0.3	0.6
$N = 64, \quad T = N^3$	2.5	2.6	0.6	0.8	0.5	0.8	0.5	0.7
$N = 128, \quad T = N^3/4$	6.1	6.3	1.3	3.8	1.2	3.3	1.3	2.5
$N = 128, \quad T = N^3/2$	9.9	10.4	2.2	4.9	1.8	4.0	1.9	2.9
$N = 128, \quad T = N^3$	15.2	16.0	3.9	5.4	3.0	5.2	3.0	3.6
$N = 256, \quad T = N^3/4$	47.8	52.2	16.5	30.1	14.3	26.8	14.0	20.0
$N = 256, \quad T = N^3/2$	79.7	82.1	29.1	35.3	24.4	33.3	22.0	23.5
$N = 256, \quad T = N^3$	126.9	133.8	57.7	43.1	44.5	42.9	37.9	28.9

Таблица 1 демонстрирует несколько важных результатов. Во-первых, графические процессоры по производительности опережают в 2–5 раз центральный процессор, использующий 20 ядер. Во-вторых, таблица подтверждает вычислительную сложность алгоритма USFFT. Соотношение времени вычислений при переходе от размера N к размеру $2N$ примерно равно 8–9 для центрального процессора. Для графических процессоров имеются случаи, когда это соотношение меньше 8. Это происходит в силу того, что обработка наиболее больших объемов данных загружает GPU-конвейер сильнее, чем данных меньшего размера, и тем самым приводит к большей эффективности загрузки GPU.

И наконец, рассмотрим результат, необходимый для дальнейшего использования USFFT в разложении сейсмических данных по гауссовым волновым пакетам. Сравним время выполнения преобразований для различных значений параметра T (количество узлов нерегулярной сетки). При увеличении этого параметра в два раза время выполнения программы на CPU увеличивается примерно в два раза. Однако такая масштабируемость не выполняется для программы на GPU. Наиболее этот факт заметен для преобразования с нерегулярной сетки на регулярную. Для $N = 256$ измерим соотношения времен для $T = \dfrac{N^3}{4}$ и $T = \dfrac{N^3}{2}$, а также для $T = \dfrac{N^3}{2}$ и $T = N^3$:

— Quadro M4000: $\dfrac{35.3}{30.1} \approx 1.17$, $\dfrac{43.1}{35.3} \approx 1.22$;

— Tesla K80: $\dfrac{33.3}{26.8} \approx 1.24$, $\dfrac{42.9}{33.3} \approx 1.28$;

— GeForce TITAN X: $\dfrac{23.5}{20.0} \approx 1.17$, $\dfrac{42.9}{33.3} \approx 1.23$.

На основе этих данных можно сделать вывод, что при использовании высокопроизводительных средств лучше всего выполнять обратное USFFT для большего количества точек нерегулярной сетки. В частности, в разложении по гауссовым волновым пакетам будет более оптимальным объединять сетки коробок, ограничиваясь лишь объемами памяти. На рис. 9 представлен результат профилирования программы разложения данных по гауссовым волновым пакетам при помощи NVidia Visual Profiler. На графике видно, что ядро для вычисления преобразования Фурье с регулярной сетки на нерегулярную (`CALC_e2u`) запускается большое количество раз, каждый запуск при этом занимает малое количество времени. Ядро для вычисления преобразования с объединенных нерегулярных сеток на регулярную (`CALC_u2e`) запускается малое количество раз, каждый из которых выполняется по времени гораздо дольше.

Таблица 2

Время (в секундах) выполнения прямого (C) и сопряженного (C^*) операторов разложения данных размером $N \times N \times N$ по гауссовым волновым пакетам

	Xeon E5-2650		Quadro M4000		Tesla K80		GeForce TITAN X	
	C	C^*	C	C^*	C	C^*	C	C^*
$N = 64$	220	232	75	120	59	106	26	57
$N = 128$	1992	2153	748	988	469	812	313	556
$N = 256$	20313	23332	4053	6573	3871	5663	3234	4956

Полученные ускоренные версии USFFT были использованы в программе разложения сейсмических данных по гауссовым волновым пакетам. Проведенное тестирование производительности показало существенные ускорения для графических процессоров. В табл. 2 представлено время выполнения прямого и сопряженного преобразований для входных данных размерностью 64^3, 128^3, 256^3. Параметры разложения по гауссовым волновым пакетам выбраны таким образом, чтобы качество восстановленных данных было возможным для анализа. Количество различных углов для покрытия спектра при помощи квадратур Лебедева (рис. 4) равно 350. Количество уровней, на которых находятся коробочки для покрытия спектра (рис. 7), определяется размером входных данных. Куб размером 64^3 покрывается одним уровнем коробочек, размером 128^3 — тремя уровнями и размером 256^3 — пятью уровнями. Размеры коробочек выбраны исходя из равномерного покрытия спектра, количество точек в коробочках определяется на основе теоремы Котельникова.

Таблица 3

Ускорение для GPU по сравнению с 20-ядерным процессором Xeon E5 при выполнении прямого (C) и сопряженного (C^*) операторов для данных размера $N \times N \times N$

	Xeon E5-2650		Quadro M4000		Tesla K80		GeForce TITAN X	
	C	C^*	C	C^*	C	C^*	C	C^*
$N = 64$	1	1	2.9	1.9	3.7	2.2	8.5	4.0
$N = 128$	1	1	2.6	2.2	4.2	2.7	6.4	3.9
$N = 256$	1	1	5.0	3.5	5.2	4.1	6.3	4.7

По данным таблицы можно рассчитать получившиеся ускорения для GPU по сравнению с 20-ядерным процессором Xeon E5 (табл. 3).

4.2. Апробация на сейсмических данных. Разложение сейсмических данных по волновым пакетам является оптимальным в том смысле, что целевые волны могут быть представлены малым количеством больших коэффициентов, а шум представлен большим количеством малых коэффициентов.

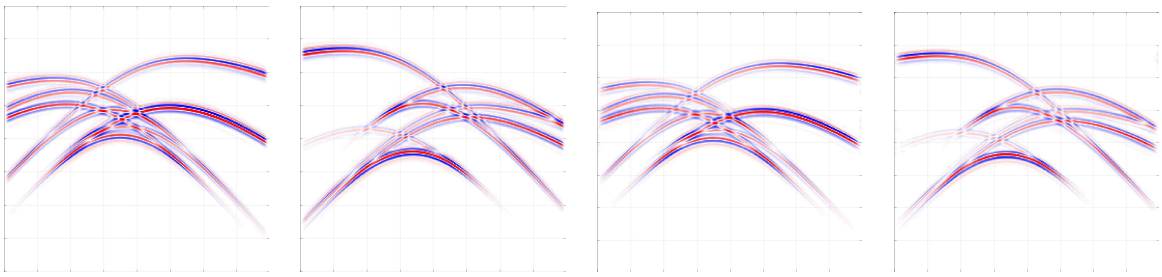

Рис. 11. Сжатие данных: два вертикальных среза через куб оригинальных данных (слева) и данных, восстановленных после сжатия при помощи гауссовых волновых пакетов (справа)

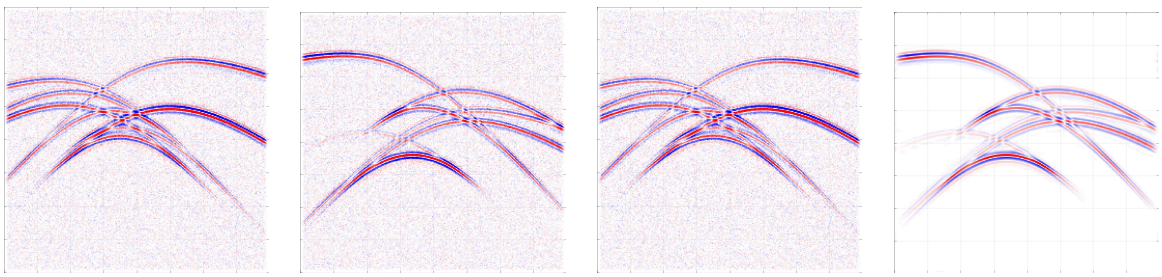

Рис. 12. Подавление шумов: два вертикальных среза через куб оригинальных данных, содержащих шум (слева), и восстановленных данных при помощи гауссовых волновых пакетов (справа)

Пусть имеется куб сейсмических данных со стороной $N = 256$ (два вертикальных среза через куб показаны на рис. 11, слева). Процедуры сжатия данных, подавления шумов и интерполяции данных можно реализовать, применив прямое преобразование по гауссовым волновым пакетам, сохранение коэффициентов, абсолютное значение которых больше заданного уровня, и сопряженное преобразование по гауссовым волновым пакетам:

$$f(x) \xrightarrow{C} \{c_\gamma\} \xrightarrow{\text{thresholding}} \{c_{\widehat{\gamma}}\} \xrightarrow{C^*} f'(x).$$

Пусть после применения прямого преобразования по гауссовым волновым пакетам берется только S наибольших коэффициентов, остальные коэффициенты отбрасываются. Определим коэффициент сжатия как $\chi = \dfrac{S}{N^3}$, где N^3 — количество точек в оригинальном кубе данных. Восстановленные данные показаны на рис. 11 (справа) для $\chi = 0.3$. На рисунке заметны лишь малые ухудшения качества изображения. Несмотря на ослабевающие амплитуды, по ним все еще можно узнать необходимую информацию о кинематике волн.

Подавление случайного шума и интерполяция данных производятся аналогично сжатию. Производим прямое преобразование по волновым пакетам, отсекаем коэффициенты, меньшие заданного порога, и выполняем обратное преобразование. На рис. 12 представлено подавление случайного шума с отношением сигнал/шум, равным 30.

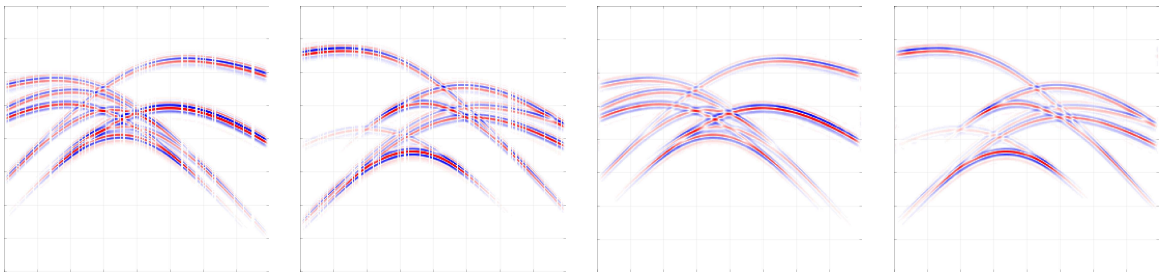

Рис. 13. Подавление шумов: два вертикальных среза через куб оригинальных данных, содержащих шум (слева), и восстановленных данных при помощи гауссовых волновых пакетов (справа)

Рассмотрим теперь интерполяцию данных. На рис. 13 исходные нормированные данные содержали по 50 пропущенных трасс в каждой сейсмограмме для фиксированного источника (всего пропущено 12 800). При восстановлении данных происходит интерполяция на пропущенные трассы.

5. Заключение. В данной работе были программно реализованы алгоритмы быстрого преобразования Фурье на нерегулярных сетках с дальнейшим применением к реализации разложении трехмерных сейсмических данных по гауссовым волновым пакетам.

Для ускорения использовались высокопроизводительные средства NVidia CUDA. Оптимизированная версия программы для графических ускорителей демонстрирует рост производительности в 2–6 раз по сравнению с 20-ядерным процессором. Было проведено успешное тестирование алгоритмов на синтетических сейсмических данных: восстановление изображения по коэффициентам гауссовых волновых пакетов, сжатие данных, подавление шумов данных, интерполяция данных в случае пропущенных трасс.

С другой стороны, разложение по волновым пакетам является таким же универсальным преобразованием, как и преобразование Фурье, вейвлет-преобразование и др. Тем самым, круг возможных приложений может оказаться достаточно широким. В перспективе разложение по волновым пакетам может оказаться полезным при анализе трехмерных изображений и выделении границ, например в медицинской томографии [29, 30].

Следует отметить, что в реализации алгоритмов на GPU можно провести дополнительные оптимизации. Загрузка графического процессора во время исполнения программы не превышает 70%. Связано это с затратами на копирование данных на видеокарту и с видеокарты, чтение и запись данных в файл, а также с проведением некоторых вспомогательных вычислений на CPU. Следует заметить, что само развитие аппаратурных решений для графических ускорителей позволяет оптимизировать выполнение программ за счет совмещения передачи данных с вычислениями и выполнения двустороннего копирования между CPU и GPU. Новые GPU-архитектуры (например, Maxwell и Pascal) добавляют новые возможности оптимизации кода при помощи увеличенной выделенной общей памяти, быстрых атомарных операций в разделяемой памяти и динамического параллелизма. Возможности динамического параллелизма востребованы, например, при вычислении преобразования Фурье на нерегулярных сетках для задач, в которых точки нерегулярной сетки распределены неравномерно по области.

СПИСОК ЛИТЕРАТУРЫ

1. *Гурвич И.И., Боганик Г.Н.* Сейсмическая разведка. М.: Недра, 1980.
2. *Andersson F., de Hoop M.V., Smith H.F., Uhlmann G.* A multi-scale approach to hyperbolic evolution equations with limited smoothness // Communications in Partial Differential Equations. 2008. **33**, N 6. 988–1017.
3. *Duchkov A.A., Andersson F., de Hoop M.V.* Discrete almost-symmetric wave packets and multiscale geometrical representation of (seismic) waves // IEEE Transactions on Geoscience and Remote Sensing. 2010. **48**, N 9. 3408–3423.
4. *Candès E.J., Donoho D.L.* Continuous curvelet transform: I. Resolution of the wavefront set // Applied and Computational Harmonic Analysis. 2005. **19**, N 2. 162–197.
5. *Candès E., Demanet L., Donoho D., Ying L.* Fast discrete curvelet transforms // Multiscale Modeling & Simulation. 2006. **5**, N 3. 861–899.
6. *Neelamani R., Baumstein A.I., Gillard D.G., Hadidi M.T., Soroka W.L.* Coherent and random noise attenuation using the curvelet transform // The Leading Edge. 2008. **27**, N 2. 240–248.

7. *Duchkov A.A., de Hoop M.V., Andersson F.* Migration velocity analysis using wave packets-geometric approach // SEG Technical Program Expanded Abstracts. 2009. doi 10.1190/1.3255704.
8. *Andersson F., Carlsson M., Tenorio L.* On the representation of functions with Gaussian wave packets // Journal of Fourier Analysis and Applications. 2012. **18**, N 1. 146–181.
9. *Beylkin G.* On applications of unequally spaced fast Fourier transform // Mathematical Geophysics Summer School. Stanford: Stanford Univ., 1998.
10. *Dutt A., Rokhlin V.* Fast Fourier transforms for nonequispaced data // SIAM Journal on Scientific computing. 1993. **14**, N 6. 1368–1393.
11. *Duchkov A.A., Andersson F., Ojala R.* Prestack shot-gather depth migration by a rigid flow of Gaussian wave packets // Studia Geophysica et Geodaetica. 2012. **56**, N 1. 83–106.
12. *Глинский Б.М., Мартынов В.Н., Сапетина А.Ф.* Технология суперкомпьютерного 3D моделирования сейсмических волновых полей в сложно построенных средах // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. 2015. **4**, № 4. 101–116.
13. *Траченко А.В., Прокопенко А.С.* Решение сверхбольших трехмерных задач сейсмики в слоистых средах на multiGPU системе с помощью CAE Fidesys // Вестник Нижегородского университета им. Н.И. Лобачевского. 2011. № 4-2. 541–542.
14. *Городничев М.А., Дучков А.А., Сарычев В.Г.* Эффективная реализация алгоритма когерентного суммирования для исполнения на GPU с использованием программной модели CUDA // Тр. Междунар. Науч. Конф. “Параллельные вычислительные технологии” (ПаВТ-2016). Челябинск: Издат. центр ЮУрГУ, 2016. 118–130.
15. *Курин Е.А., Денисов М.С.* Применение высокопроизводительных вычислительных систем в задаче подавления многократно отраженных волн-помех // Технологии сейсморазведки. 2011. № 4. 35–40.
16. *Nikitin V.V., Andersson F., Carlsson M., Duchkov A.A.* Fast hyperbolic Radon transform represented as convolutions in log-polar coordinates // Computers and Geosciences. 2017. **105**. 21–33.
17. *Бриллинджер Д.* Временные ряды. Обработка данных и теория. М.: Мир, 1980.
18. *Fessler J.A., Sutton B.P.* Nonuniform fast Fourier transforms using min-max interpolation // IEEE Transactions on Signal Processing. 2003. **51**, N 2. 560–574.
19. *Greengard L., Lee J.-Y.* Accelerating the nonuniform fast Fourier transform // SIAM Review. 2004. **46**, N 3. 443–454.
20. *Лебедев В.И.* О квадратурах на сфере // Журнал вычислительной математики и математической физики. 1976. **16**, № 2. 293–306.
21. *Chen X., Qiu Y., Yi H.* Implementation and performance of image filtering on GPU // Proc. Fourth Int. Conf. on Intelligent Control and Information Processing (ICICIP). New York: IEEE Press, 2013. 514–517.
22. *Fialka O., Cadik M.* FFT and convolution performance in image filtering on GPU // Proc. Tenth Int. Conf. on Information Visualization. New York: IEEE Press, 2006. 609–614.
23. *Sigg C., Hadwiger M.* Fast third-order texture filtering // GPU Gems. Vol. 2. Reading: Addison-Wesley, 2005. 313–329.
24. *Zhang N., Chen Y.-S., Wang J.L.* Image parallel processing based on GPU // Proc. 2nd Int. Conf. on Advanced Computer Control (ICACC). Vol. 3. New York: IEEE Press, 2010. 367–370.
25. *Cooley J.W., Tukey J.W.* An algorithm for the machine calculation of complex Fourier series // Mathematics of Computation. 1965. **19**, N 90. 297–301.
26. *Bluestein L.* A linear filtering approach to the computation of discrete Fourier transform // IEEE Transactions on Audio and Electroacoustics. 1970. **18**, N 4. 451–455.
27. *Kirk D.B., Hwu W.W.* Programming massively parallel processors: a hands-on approach. Boston: Morgan Kaufmann, 2016.
28. *Боресков А.В., Харламов А.А.* Основы работы с технологией CUDA. М.: Litres, 2015.
29. *Andersson F.* Fast inversion of the Radon transform using log-polar coordinates and partial back-projections // SIAM Journal on Applied Mathematics. 2005. **65**, N 3. 818–837.
30. *Andersson F., Carlsson M., Nikitin V.V.* Fast algorithms and efficient GPU implementations for the Radon transform and the back-projection operator represented as convolution operators // SIAM Journal on Imaging Sciences. 2016. **9**, N 2. 637–664.

Поступила в редакцию
09.07.2017

Acceleration of Seismic Data Processing with Wave-Packet Decomposition Using NVidia CUDA

V. V. Nikitin[1], A. A. Duchkov[2], and F. Andersson[3]

[1] *MAX IV Laboratory, Lund University; Fotongatan 2, 22592, Lund, Sweden; Ph.D., Postdoctoral Researcher, e-mail: viktor.nikitin@maxiv.lu.se*

[2] *Trofimuk Institute of Petroleum Geology and Geophysics, Siberian Branch of Russian Academy of Sciences; prospekt Koptyuga 3, Novosibirsk, 630090, Russia; Ph.D., Head of Laboratory, e-mail: duchkovaa@ipgg.sbras.ru*

[3] *Center for Mathematical Sciences, Lund University; Solvegatan 18, 22100, Lund, Sweden; Ph.D., Associate Professor, e-mail: fredrik.andersson@math.lth.se*

Received July 9, 2017

Abstract: Seismic data are characterized by multidimensionality, large data sizes, and irregular structures. In this paper we consider an optimal decomposition of seismic data using the basis of Gaussian wave packets. We implemented and optimized a number of fast algorithms for forward and inverse transforms for three-dimensional seismic data decomposition. The algorithms implemented on GPU demonstrate 2-6x speedup compared to 20-core CPUs. The programs were tested on synthetic seismic data sets: data reconstruction by Gaussian wave-packet coefficients, data compression, denoising, and interpolation in the case of missing traces.

Keywords: Gaussian wave packets, Fourier transform, CUDA.

References

1. I. I. Gurvich and G. N. Boganik, *Seismic Prospecting* (Nedra, Moscow, 1980) [in Russian].

2. F. Andersson, M. V. de Hoop, H. F. Smith, and G. Uhlmann, "A Multi-Scale Approach to Hyperbolic Evolution Equations with Limited Smoothness," Commun. Part. Diff. Eq. **33** (6), 988–1017 (2008).

3. A. A. Duchkov, F. Andersson, and M. V. de Hoop, "Discrete Almost-Symmetric Wave Packets and Multiscale Geometrical Representation of (Seismic) Waves," IEEE Trans. Geosci. Remote Sens. **48** (9), 3408–3423 (2010).

4. E. J. Candès and D. L. Donoho, "Continuous Curvelet Transform: I. Resolution of the Wavefront Set," Appl. Comput. Harmon. Anal. **19** (2), 162–197 (2005).

5. E. Candès, L. Demanet, D. Donoho, and L. Ying, "Fast Discrete Curvelet Transforms," Multiscale Model. Simul. **5** (3), 861–899 (2006).

6. R. Neelamani, A. I. Baumstein, D. G. Gillard, et al., "Coherent and Random Noise Attenuation Using the Curvelet Transform," The Leading Edge **27** (2), 240–248 (2008).

7. A. A. Duchkov, M. V. de Hoop, and F. Andersson, "Migration Velocity Analysis Using Wave Packets — Geometric Approach," SEG Technical Program Expanded Abstracts 2009. doi 10.1190/1.3255704

8. F. Andersson, M. Carlsson, and L. Tenorio, "On the Representation of Functions with Gaussian Wave Packets," J. Fourier Anal. Appl. **18** (1), 146–181 (2012).

9. G. Beylkin, "On Applications of Unequally Spaced Fast Fourier Transform," in *Mathematical Geophysics Summer School* (Stanford Univ., Stanford, 1998).

10. A. Dutt and V. Rokhlin, "Fast Fourier Transforms for Nonequispaced Data," SIAM J. Sci. Comput. **14** (6), 1368–1393 (1993).

11. A. A. Duchkov, F. Andersson, and R. Ojala, "Prestack Shot-Gather Depth Migration by a Rigid Flow of Gaussian Wave Packets," Stud. Geophys. Geod. **56** (1), 83–106 (2012).

12. B. M. Glinskii, V. N. Martynov, and A. F. Sapetina, "Technology of Supercomputer Simulation of Seismic Wave Fields in Complicated Media," Vestn. South Ural State Univ. Ser. Vychisl. Mat. Inf. **4** 4, 101–116 (2015).

13. A. V. Trachenko and A. S. Prokopenko, "Solution of Large Three-Dimensional Problems of Seismicity in Layered Media on a Multi-GPU System Using CAE Fidesys," Vestn. Lobachevskii Univ. Nizhni Novgorod, No. 4–2, 541–542 (2011).

14. M. A. Gorodnichev, A. A. Duchkiv, and V. G. Sarychev, "Efficient GPU-Implementation of Coherent Stacking with CUDA," in *Proc. Int. Conf. on Parallel Computational Technologies, Arkhangelsk, Russia, March 28–April 1, 2016* (South Ural State Univ., Chelyabinsk, 2016), p. 118–130.

15. E. A. Kurin and M. S. Denisov, "Application of High-Performance Computing to Seismic Multiple Attenuation," Tekhnol. Seismorazved., No. 4, 35–40 (2011).

16. V. V. Nikitin, F. Andersson, M. Carlsson, and A. A. Duchkov, "Fast Hyperbolic Radon Transform Represented as Convolutions in Log-polar Coordinates," Comput. Geosci. **105**, 21–33 (2017).

17. D. R. Brillinger, *Time Series. Data Analysis and Theory* (Holt, Rinehart and Winston, New York, 1975; Mir, Moscow, 1980).

18. J. A. Fessler and B. P. Sutton, “Nonuniform Fast Fourier Transforms Using Min-Max Interpolation,” IEEE Trans. Signal Process. **51** (2), 560–574 (2003).

19. L. Greengard and J.-Y. Lee, “Accelerating the Nonuniform Fast Fourier Transform,” SIAM Rev. **46** (3), 443–454 (2004).

20. V. I. Lebedev, “Quadratures on a Sphere,” Zh. Vychisl. Mat. Mat. Fiz. **16** (2), 293–306 (1976) [USSR Comput. Math. Math. Phys. **16** (2), 10–24 (1976)].

21. X. Chen, Y. Qiu, and H. Yi, “Implementation and Performance of Image Filtering on GPU,” in *Proc. Fourth Int. Conf. on Intelligent Control and Information Processing, Beijing, China, June 9–11, 2013* (IEEE Press, New York, 2013), pp. 514–517.

22. O. Fialka and M. Cadik, “FFT and Convolution Performance in Image Filtering on GPU,” in *Proc. Tenth Int. Conf. on Information Visualization, London, England, July 5–7, 2006* (IEEE Press, New York, 2006), pp. 609–614.

23. C. Sigg and M. Hadwiger, “Fast Third-Order Texture Filtering,” in *GPU Gems* (Addison-Wesley, Reading, 2005), Vol. 2, pp. 313–329.

24. N. Zhang, Y.-S. Chen, and J. L. Wang, “Image Parallel Processing Based on GPU,” in *Proc. 2nd Int. Conf. on Advanced Computer Control, Shenyang, China, March 27–29, 2010* (IEEE Press, New York, 2010), Vol. 3, 367–370.

25. J. W. Cooley and J. W. Tukey, “An Algorithm for the Machine Calculation of Complex Fourier Series,” Math. Comput. **19** (90), 297–301 (1965).

26. L. Bluestein, “A Linear Filtering Approach to the Computation of Discrete Fourier Transform,” IEEE Trans. Audio and Electroacoust. **18** (4), 451–455 (1970).

27. D. B. Kirk and W. W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach* (Morgan Kaufman, Boston, 2016).

28. A. V. Boreskov and A. A. Kharlamov, *The Basics of Working with CUDA Technology* (Litres, Moscow, 2015) [in Russian].

29. F. Andersson, “Fast Inversion of the Radon Transform Using Log-polar Coordinates and Partial Back-Projections,” SIAM J. Appl. Math. **65** (3), 818–837 (2005).

30. F. Andersson, M. Carlsson, and V. V. Nikitin, “Fast Algorithms and Efficient GPU Implementations for the Radon Transform and the Back-Projection Operator Represented as Convolution Operators,” SIAM J. Imaging Sci., **9** (2), 637–664 (2016).