УДК 519.72, 550.34.06

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

[1] Новосибирский государственный университет
ул. Пирогова, 2, Новосибирск, 630090, Россия

[2] Институт нефтегазовой геологии и геофизики СО РАН
пр. Акад. Коптюга, 3, Новосибирск, 630090, Россия

[3] Lund University
Paradisgatan, 2, Lund, SE-221 00, Sweden

E-mail: vnikitin90@gmail.com

ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ РАЗЛОЖЕНИЯ ФУНКЦИЙ ПО ВОЛНОВЫМ ПАКЕТАМ ДЛЯ GPU И ЕГО ПРИМЕНЕНИЕ В ГЕОФИЗИКЕ *

Данные, получаемые при проведении сейсмических работ методом отраженных волн, характеризуются многомерностью, большим объемом, а также своей нерегулярностью. Возникает необходимость их оптимального представления, а именно разложения по базису, наиболее подходящему для их дальнейшей обработки. В качестве такого представления в данной работе используется (переопределенный) базис волновых пакетов. При помощи технологии NVIDIA CUDA на базе GPU был реализован быстрый алгоритм прямого и обратного преобразования по трехмерным волновым пакетам. Проведен целый ряд оптимизаций, не только связанных с физическим устройством графического процессора, но и со структурой исходного алгоритма. Достигнуто ускорение до 45 раз на одной карте, выполнен анализ масштабируемости для нескольких видеокарт. Программа тестировалась на синтетических сейсмических данных для реализации процедур сжатия, подавления шума и регуляризации трехмерных данных в случае пропущенных трасс.

Ключевые слова: GPU, волновые пакеты, быстрое преобразование Фурье, сейсмика.

Введение

В ходе сейсмических работ на земной поверхности проводится измерение волн, отраженных от геологических границ внутри земли. Записанное волновое поле используется для построения сейсмического разреза земной коры, который затем применяется при поиске полезных ископаемых.

Основным сейсморазведочным методом в настоящее время является метод отраженных волн (МОВ) [1]. Схема наблюдений при проведении сейсмических работ приведена на рис. 1, *а*. Технология работ состоит в использовании контролируемого источника упругих колебаний (взрыв, вибратор или воздушная пушка). Волны (давление, вектор смещения), созданные источником, распространяются вглубь среды и отражаются от геологических границ, разде-

* В работе использовались ресурсы Сибирского суперкомпьютерного центра. Работа проводилась при частичной поддержке Шведского фонда по международному сотрудничеству в науке и высшем образовании (Swedish Foundation for International Cooperation in Research and Higher Education) и Сибирского отделения РАН.

Никитин В. В., Дучков А. А., Романенко А. А., Андерссон Ф. Параллельный алгоритм разложения функций по волновым пакетам для GPU и его применение в геофизике // Вестн. Новосиб. гос. ун-та. Серия: Информационные технологии. 2013. Т. 11, вып. 1. С. 93–104.

ISSN 1818-7900. Вестник НГУ. Серия: Информационные технологии. 2013. Том 11, выпуск 1

ляющих геологические породы с разными упругими свойствами. Отраженные волны распространяются вверх и регистрируются на поверхности группой (косой) приемников. Белыми линиями показаны лучи падающих и отраженных волн (рис. 1, *б*). В процессе движения судна эксперимент последовательно повторяется для разных положений источника и группы приемников.

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

- многомерность (в случае профильных наблюдений данные являются трехмерной функцией, а в самом общем случае могут быть пятимерными, когда источники и приемники плотно покрывают поверхность наблюдений);
- большой объем (данные для одного профиля занимают несколько гигабайт, а данные площадной съемки требуют несколько терабайт памяти);
- нерегулярность (при измерениях в полевых условиях возникают пропущенные и забракованные трассы и т. д.).

В последние годы в сейсмике начали применяться методы, разработанные для анализа и оптимального представления изображений (фото, видео и т. д.). Сейсмические данные рассматриваются в качестве многомерной функции, которая раскладывается по специфическим базисным функциям – волновым пакетам (или «кёрвлетам») [2]. Оптимальность разложения заключается в том, что функция может быть представлена в виде линейной комбинации небольшого количества базисных функций, так как их форма хорошо подходит под структуру данных (рис. 2). Возможность сжатия данных интуитивно понятна из формы волновых пакетов, поскольку они могут служить «кирпичиками», из которых можно «выстраивать» волны на сейсмограммах (рис. 1, *б*). Математическое доказательство оптимальности данного базиса приведено в [3].

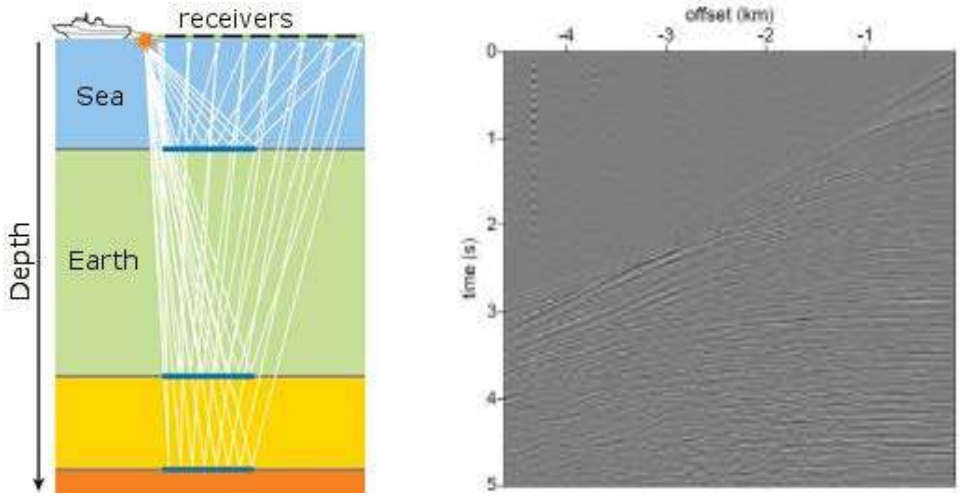


Рис. 1. Схема морской сейсморазведки: *а* – схема наблюдений при сейсмических работах; *б* – амплитуда колебаний (давления) от времени для поверхностной косы приемников

Рис. 2. Примеры двумерных базисных функций, волновых пакетов, различных ориентаций и размеров

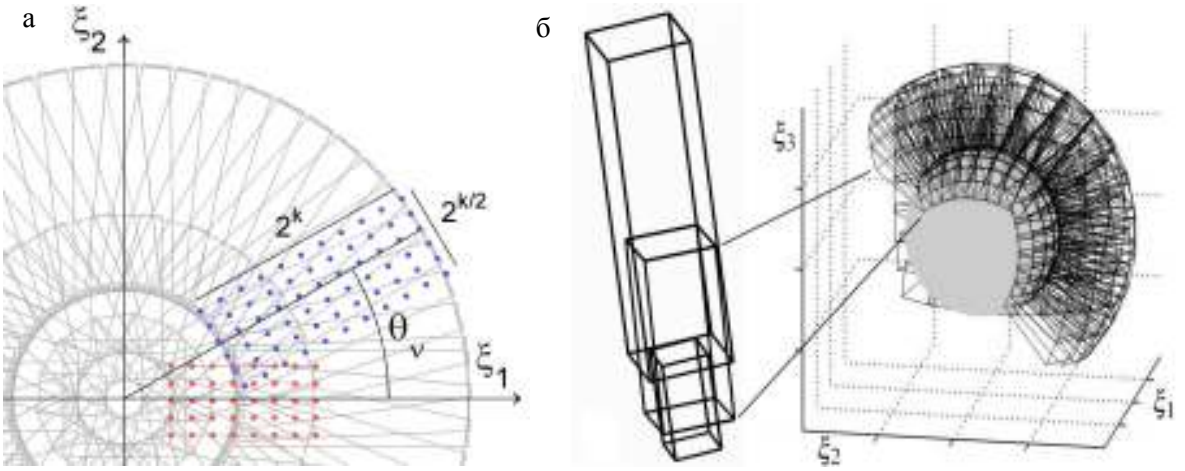


Рис. 3. Покрытие частотной области коробочками: *а* – параметры, задающие размер и положение коробочек и локальные сетки на них (иллюстрация для двумерного случая); *б* – покрытие частотной области пересекающимися коробочками в трехмерном случае

Метод разложения по волновым пакетам позволяет решать целый ряд задач обработки сейсмических данных [4–6]: их сжатие, подавление шума, интерполяция, регуляризация (пересчет с нерегулярной сетки на регулярную) и т. д. Для решения всех перечисленных задач для трехмерных сейсмических кубов данных был предложен новый базис трехмерных волновых пакетов, специально созданный для сейсмических приложений [7]. В статье решается задача эффективной программной реализации процедуры прямого и обратного преобразования по трехмерным волновым пакетам, позволяющая обрабатывать большие объемы сейсмических данных.

Алгоритм разложения по волновым пакетам

Алгоритм прямого и обратного преобразования по волновым пакетам (ПВП) подробно изложен в [7]. Здесь приведем краткое изложение алгоритма, описание самой трудоемкой вычислительной процедуры и обоснование целесообразности ее переноса на GPU.

Прямой оператор ПВП преобразует дискретно заданную трехмерную функцию $f(x)$ в набор коэффициентов $\{c_\gamma\}$, параметризованных индексом γ:

$$C: f(x) \to \{c_\gamma\},$$

таких что

$$f(x) = \sum_\gamma c_\gamma \varphi_\gamma(x),$$

где $x = (x_1, x_2, x_3)$ – координаты в трехмерном пространстве; $\varphi_\gamma(x)$ – базисные функции волновых пакетов; γ – мультииндекс.

Сами функции волновых пакетов задаются в Фурье-области и имеют спектр $\hat{\varphi}_{k,\nu}(\xi)$, сосредоточенный в коробочке заданного размера, положения и ориентации. Здесь ξ – координаты в Фурье-области, дуальные к x. На рис. 3, *а* 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 $\xi_j^{k,\nu}$ (см. точки на рис. 3, *а*). Распределение пересекающихся коробочек в трехмерном случае показано на (рис. 3, *б*).

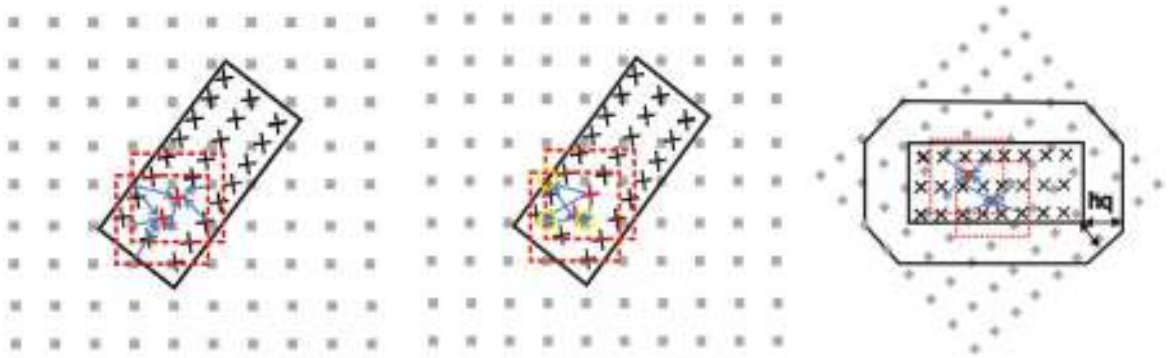


Рис. 4. Схема интерполяции для одной коробки: *а* – собирающее усреднение при прямом преобразовании; *б* – рассеивающее усреднение при обратном преобразовании; *в* – модификация рассеивающего усреднения в собирающее при обратном преобразовании

Алгоритм ПВП. Приведем вычислительную схему дискретной реализации оператора C прямого ПВП:

$$f(x) \xrightarrow{\text{USFFT}} \hat{f}(\xi_j^{k,\nu}) \xrightarrow{\text{windowing}} \hat{f}(\xi_j^{k,\nu})\hat{\varphi}_{k,\nu}(\xi_j^{k,\nu}) \xrightarrow{\text{IFFT}} \{c_\gamma\},$$

где $\gamma = (k,\nu,j)$, $\hat{f}(\xi)$ обозначает спектр Фурье от $f(x)$.

Таким образом, дискретное преобразование состоит из трех основных этапов.

1. Прямое преобразование Фурье $f(x)$ на нерегулярную сетку, состоящую из объединения сеток $\xi_j^{k,\nu}$ на всех коробочках, параметризованных (k,ν).

2. Умножение спектра в коробочке (k,ν) на соответствующую спектральную функцию волнового пакета.

3. Обратное преобразование Фурье на каждой из коробочек.

Спектральные функции волновых пакетов подобраны таким образом, чтобы при возведении в квадрат задавать разбиение единицы Фурье-области:

$$\sum_{k,\nu} \hat{\varphi}_{k,\nu}^2(\xi) = 1.$$

В этом случае сопряженный оператор является обратным (см. [7]):

$$f(x) \xleftarrow{\text{сопр. USFFT}} \hat{f}(\xi_j^{k,\nu})\hat{\varphi}_{k,\nu}^2(\xi_j^{k,\nu}) \xleftarrow{\text{windowing}} \hat{f}(\xi_j^{k,\nu})\hat{\varphi}_{k,\nu}(\xi_j^{k,\nu}) \xleftarrow{\text{FFT}} \{c_\gamma\}.$$

Заметим, что вычислительная сложность алгоритма оказывается ограничена реализацией первого этапа. Стандартные алгоритмы быстрого преобразования Фурье (БПФ, FFT) могут быть использованы для реализации третьего этапа, так как локальные сетки на каждой коробочке являются регулярными. Но они не могут применяться на первом этапе, поскольку выходная сетка является нерегулярной. Для эффективного выполнения этой операции мы будем использовать алгоритм быстрого преобразования Фурье на нерегулярные сетки (USFFT) (см. [8]), который имеет тот же порядок сложности, что и стандартные алгоритмы БПФ. Таким образом, вычислительная скорость дискретного ПВП оказывается близкой к скорости быстрого преобразования Фурье.

Алгоритм USFFT. Процедура быстрого преобразования Фурье с регулярной сетки x_i на нерегулярную сетку $\xi_j^{k,\nu}$ сама состоит из двух основных этапов:

а) прямое (обратное) БПФ с регулярной сетки x_i на регулярную сетку ξ_i;

б) «интерполяция», пересчет значений спектра с сетки ξ_i в точки нерегулярной сетки $\xi_j^{k,\nu}$ с помощью собирающего или рассеивающего усреднения.

Собирающее усреднение (для прямого ПВП). На рис. 4, *а* приведен фрагмент глобальной регулярной сетки (квадратики), на котором задан спектр $\hat{f}(\xi_i)$ после прямого БПФ функции

$f(x_i)$. Прямоугольником обозначена одна коробочка, локальная сетка $\xi_j^{k,\nu}$ на которой обозначена крестиками.

Основной вычислительной процедурой является суммирование: для каждой точки локальной сетки $\xi_j^{k,\nu}$ (крестики) нужно провести суммирование значений с регулярной сетки (квадратики) в шаре с радиусом *hq* (пунктирная область на рис. 4, *а*) с гауссовскими весами $w(r)$:

$$\hat{f}(\xi_j^{k,\nu})\hat{\varphi}_{k,\nu}^2(\xi_j^{k,\nu}) \xleftarrow{\text{windowing}} \hat{f}(\xi_j^{k,\nu})\hat{\varphi}_{k,\nu}(\xi_j^{k,\nu}) \xleftarrow{\text{FFT}} \{c_\gamma\},$$

$$\hat{f}(\xi_{j1}^{k,\nu},\xi_{j2}^{k,\nu},\xi_{j3}^{k,\nu}) = \sum_{k=-hq}^{hq}\sum_{m=-hq}^{hq}\sum_{n=-hq}^{hq} w(k+d_1)w(m+d_2)w(n+d_3)F(i_1+k,i_2+m,i_3+n),$$

$$w(r) = \exp(-\lambda r^2), \qquad (1)$$

где $\xi_j^{k,\nu} = (\xi_{j1}^{k,\nu},\xi_{j2}^{k,\nu},\xi_{j3}^{k,\nu})$ – точка локальной сетки (крестики); $i = (i_1,i_2,i_3)$ – ближайшая к ней точка глобальной сетки (серые квадратики); (d_1,d_2,d_3) – расстояние между этими точками, (k,m,n) – индексы, которые пробегают точки глобальной сетки в шаре радиуса *hq*; $w(r) = \exp(-\lambda r^2)$ – весовая функция; λ – коэффициент, зависящий от требуемой точности ε вычисления USFFT, $hq = \lceil -2\pi \ln \varepsilon \rceil$, $\lambda = -.5\pi^2 / \ln \varepsilon$ (см. [8]).

Рассеивающее усреднение (для обратного ПВП). При обратном преобразовании необходимо выполнить операцию усреднения в обратном порядке, т. е. произвести пересчет с узлов нерегулярной сетки (крестиков) на узлы глобальной сетки (серые квадратики). Значения $\hat{f}(\xi_{j1}^{k,\nu},\xi_{j2}^{k,\nu},\xi_{j3}^{k,\nu})$ из точки $\xi_j^{k,\nu}$ (крестики) нужно с разными весами $w(r)$ прибавить к значениям в элементах глобальной сетки (серые квадратики) в шаре радиуса *hq* вокруг этой точки (см. пунктирные области на рис. 4, *б*).

Перенос и оптимизация программы на GPU

Подытожим основные особенности вычислений. Во-первых, для решения наших задач достаточно использовать операции с одинарной точностью. Во-вторых, алгоритм ПВП содержит большое число простых однотипных операций (например, при интерполяции). В-третьих, в процессе вычислений требуется организовать доступ к большим объемам памяти. В силу перечисленных требований для параллельной реализации алгоритма кажется естественным использовать вычислительную платформу GPU [9].

Сейсмический куб данных в типичном случае будет иметь несколько сотен отсчетов по координатам источников и приемников и порядка 1 000 отсчетов по координате времени. В качестве типичных размеров данных мы возьмем кубические области с размерами $N = 128$, 256 и 512 точек по каждому из направлений.

Количество подобластей (коробочек) исчисляется сотнями. Общее количество точек в локальных сетках на всех коробочках (нерегулярная сетка) может варьироваться от 100 до 500 тысяч точек. Для интерполяции необходимо провести взвешенное суммирование вокруг каждой точки подобласти, что в среднем составляет несколько тысяч элементарных операций.

Основными ресурсно-затратными частями алгоритма ПВП являются процессы интерполяции (собирающее и рассеивающее усреднения) и преобразования Фурье. Эти части и были перенесены на графический процессор (GPU).

Преобразования Фурье на GPU

Как упоминалось ранее, БПФ в алгоритме используются в двух местах: «большое» преобразование на глобальной сетке и «малые» преобразования на локальных сетках на каждой коробочке. Библиотека CUFFT технологии CUDA осуществляет быстрые преобразования Фурье на видеокарте. В последней версии CUDA Toolkit 4.1 при помощи этой библиотеки можно получить ускорение до 25 раз по сравнению с использованием стандартной библиоте-

ки FFTW для выполнения БПФ на центральном процессоре. Причем использования библиотек CUFFT и FFTW аналогичны, только в CUFFT дополнительно нужно копировать данные на / с видеокарты. Ниже приведены примеры кода, осуществляющие трехмерное ПФ на центральном процессоре и на графическом (рис. 5).

```
/*FFTW*/
fftw_complex* f;
/*init f*/
fftwnd_plan plan_forward =
fftw3d_create_plan(512, 512, 512,
FFTW_FORWARD,FFTW_ESTIMATE|FFTW
_IN_PLACE);
fftwnd_one(plan_forward,&f,NULL);
fftwnd_destroy_plan(plan_forward);
```

```
/*CUFFT*/
cufftComplex *f_dev;
/*copy data to device memory*/
cufftHandle plan;
cufftPlan3d(&plan, 512, 512, 512,
CUFFT_C2C);
cr = cufftExecC2C(plan, f_dev,
f_dev,CUFFT_FORWARD);
cufftDestroy(plan);
/*copy data from device memory*/
```

Рис. 5. Реализация преобразования Фурье на CPU (*слева*) и на GPU (*справа*)

Ускорение в 25 раз получается для больших объемов данных, в частности для данных размером 512^3 (1 Гб), при выполнении «большого» БПФ. При выполнении «малого» преобразования Фурье размер входных данных варьируется от 200 Кб до 2 Мб в зависимости от уровня, на котором находится коробка. В таком случае ускорение будет меньше (10–15 раз), так как время копирования данных на / с видеокарты будет занимать значительное время. Однако следует отметить, что в нашем случае коробочка уже будет находиться в памяти видеокарты. Ведь перед этим шагом на GPU выполняется интерполяция с сетки спектра Фурье на эту коробочку.

Интерполяция на GPU

Процесс интерполяции может проводиться независимо для каждой коробки, если обеспечить доступ к спектру, заданному на глобальной сетке. В случае, когда весь спектр не помещается в память графического процессора, необходимо подгружать в память часть спектра, содержащую интересующую коробку или группу коробок. Эта область схематически обозначена пунктирным квадратом на рис. 4, *а*.

При выполнении интерполяции в прямом преобразовании на графическом процессоре запускается число потоков, равное количеству узлов в коробочке (100 000–300 000 потоков). Каждый поток независимо рассчитывает свой элемент в коробке, используя соответствующие точки спектра Фурье и соответствующую весовую функцию.

Для выполнения интерполяции при обратном преобразовании можно также задать число потоков равным числу узлов в коробочке. Но, как видно из рис. 4, *б*, при таком подходе возникает вероятность одновременного обращения к одной ячейке памяти для записи (в отличие от прямого преобразования, где доступ к ячейкам осуществлялся для чтения). Рассмотрим два крестика, находящихся рядом, и для каждого выделим область влияния (пунктирные квадратики); видно, что они пересекаются. 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).

Одним из вариантов является использование атомарных операций, которые реализуются в два этапа: блокировка ресурса и выполнение самой операции. При выполнении рассеивающего усреднения часто возникает ситуация, когда к одной ячейке памяти одновременно обращаются примерно 700–800 потоков. Очевидно, что использование атомарных операций в данном случае пагубно сказывается на производительности программы. Поэтому необходимо было пересмотреть алгоритм интерполяции с нерегулярной сетки на регулярную.

Стандартный способ действия в таких ситуациях – это изменение обхода данных таким образом, чтобы каждый поток отвечал за запись одного элемента данных. Эта проблема решается, если переформулировать «рассеивающую» интерполяцию в форме «собирающей». В этом случае цикл пробегает не по сетке $F(\varpi)$ (крестики), а по узлам глобальной сетки (серые квадратики) (см. рис. 4, *в*). Тогда запись в каждую точку сетки (серые квадратики) производится один раз, а вычисления проводятся по формуле «собирающей» интерполяции (1).

Таким образом, интерполяция при прямом и обратном преобразовании выполняется с использованием формул «собирающего» усреднения. Каждый поток в этом случае выполняет тройной цикл ($0 \leftarrow k \leftarrow 10{,}0 \leftarrow m \leftarrow 10{,}0 \leftarrow n \leftarrow 10$ – всего 1 331 итерация). Внутри цикла производится проверка на то, что точка лежит близко к коробочке (расстояние меньше *hq*); вычисление весовой функции $w(r)$; считывание соответствующего элемента спектра из глобальной памяти видеокарты. И лишь затем вычисляется значение для точки локальной сетки. Именно этот цикл в большей степени влияет на производительность и нуждается в оптимизации.

Оптимизация процесса интерполяции

Оптимизация программы была выполнена по следующим основным направлениям [9].

1. *Оптимизация операций во вложенных циклах.* Вычисление всех экспонент необязательно проводить во внутреннем цикле. Например, $w(k + d_1) = \exp(-\lambda(k + d_1)^2)$ не зависит от индексов *m* и *n*, поэтому ее можно вынести, чтобы вычисление проводилось только во внешнем цикле. Выражение $w(m + d_2) = \exp(-\lambda(m + d_2)^2)$ также не зависит от *n*. Аналогичные соображения можно привести к проверкам условий на то, что точка глобальной сетки находится внутри выделенной области.

Следует отметить, что в предложенной реализации все же много раз производится вычисление одних и тех же весовых функций. Например, $w(n + d_3) = \exp(-\lambda(n + d_3)^2)$ выполняется 121 раз. Можно было вычислять эти функции отдельно, т. е. создать ядро, вычисляющее массив весовых функций, а потом просто брать оттуда элементы. Такой способ был проверен авторами, однако он работал медленнее вследствие долгих операций чтения из памяти.

2. *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 sinf, cosf, expf нужно заменить на их быстрые аналоги. При небольшой потере точности функции __sinf, __cosf, __expf выполняются намного быстрее. И наконец, также важной оптимизацией является использование побитовых операций. Например, в нашем коде много раз приходится удваивать значения либо же уменьшать в два раза. *N*/2 и 2**N* можно заменить на операции побитового сдвига: $N >> 1$ и $N << 1$ соответственно.

3. *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 = 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, данные не вмещаются в разделяемую память устройства по причине 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 (рис. 6) мы воспользовались трехмерной текстурной ссылкой. Функция tex3D возвращает элемент трехмерного массива, к которому привязана текстура tex.

Следующим важным шагом оптимизации работы с памятью является использование регистров. Например, при расчете взвешенной суммы (1) 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 структуру gridparm 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 % теоретической.

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

Был проведен анализ программы при помощи программы NVIDIA Compute Visual Profiler. При помощи этого инструментария можно измерять время отдельных частей программы, количество промахов по кэшам, количество конфликтов по банкам памяти и др. Также был

```
__device__ float gauss_bell(float r,float lam){return __expf(-lam*r*r);}
__constant__ gridparm gp;
__global__ void ker (/*parameters*/,float* F)
{
 int tx = threadIdx.x;
 int w_local = blockIdx.x*blockDim.x+threadIdx.x
 __shared__ float2 FS[BS]; /*BS – block size*/
/*precalculating*/
 for(int k=0; k<=10; k++)
 {
 if(wk_global out of area) continue;
 wk = gauss_bell(k+dz,lam);
 for(int j=0; j<=10; j++)
 {
 if(wj_global out of area) continue;
 wkj=wk*gauss_bell(j+dy,lam);
 for(int i=0; i<=10; i++)
 {
 if(wi_global out of area) continue;
 wkji=wkj*gauss_bell(i+dx,lam);
 FS[tx]+=wkji*tex3D(tex, wk_global, wj_global, wi_global);
 }
 }
 F[w_local] = FS[tx];
 }
}
```

Рис. 6. Ядро ker, выполняющее собирающее усреднение

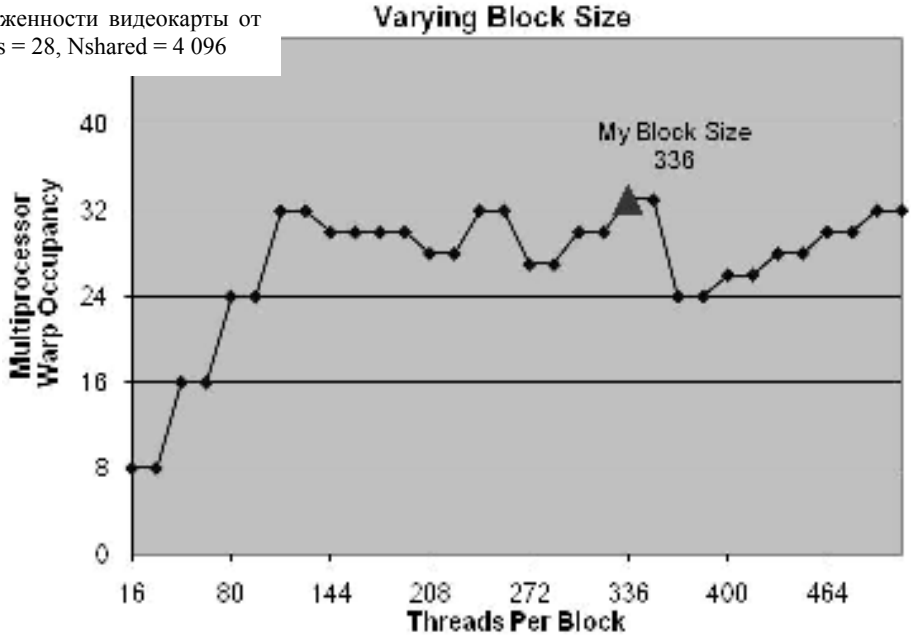


Рис. 7. Зависимость загруженности видеокарты от BS при заданных Nregisters = 28, Nshared = 4 096

использован инструментарий CUDA Occupancy Calculator, позволяющий подобрать оптимальные размеры блока (BS) для заданного количества регистров на поток (Nregisters) и разделяемой памяти на блок (Nshared). В нем строятся графики зависимости occupancy от BS, Nregisters и Nshared, где параметр occupancy показывает, насколько видеокарта загружена. Максимальная загруженность видеокарты будет при BS = 336 (рис. 7).

На рис. 6 представлено оптимизированное ядро (функция, исполняющаяся на видеокарте), которое осуществляет собирающее усреднение на видеокарте.

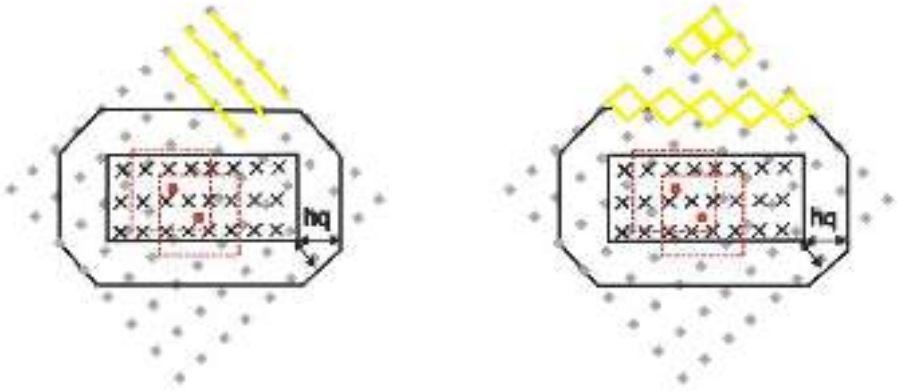
5. *Оптимизация процесса интерполяции при обратном преобразовании.* Узлы глобальной сетки, не вносящие вклад в вычисление элементов коробки, можно отсечь. На рис. 8 для заданного радиуса *hq* построен восьмиугольник, внутри которого расположены необходимые узлы глобальной сетки. Тогда, ассоциировав поток с серым квадратиком, на краях мы получаем «лишние» потоки.

Проверка на то, что точка находится внутри восьмиугольника, происходит вне циклов. Если вычисляется узел, находящийся за пределами области, то внутри потока произойдут только предварительные вычисления (~ 20 операций), т. е. менее процента от общего числа операций.

На GPU каждый потоковый блок исполняется на одном потоковом процессоре (SM), а на одном SM может исполняться несколько потоковых блоков. Рассмотрим одномерный блок, который соответствует линейному порядку обхода сетки, что схематически показано на рис. 8, *а*. В этом случае мы получим большое количество варпов (варп – набор потоков внутри блока, исполняющихся физически одновременно), потоки которых будут выполнять несопоставимое число операций (дивергентные варпы). Минимальное число операций для потока будет равно 20, а максимальное – 6 000. Для оптимизации вычислений сделаем сетку потоков внутри блока трехмерной, уменьшив также размер блока (обход сетки по областям, схематически изображенным квадратиками на рис. 8, *б*).

Анализируя статистики, полученные на основании Compute Visual Profiler и CUDA Occupancy Calculator, а также проведя экспериментальные исследования, удалось подобрать оптимальные для производительности параметры: сохранить максимальную загруженность видеокарты, равную 67 %, и при этом уменьшить число дивергентных варпов с 80 до 15 %.

В итоге, после проведения перечисленных оптимизаций, был достигнут значительный рост производительности: для NVIDIA Tesla C2050 ускорение составило 45 раз (для прямого преобразования) и 35 раз (для обратного преобразования) в сравнении с последовательным кодом.



а б

Рис. 8. Собирающая интерполяция при обратном преобразовании:
а – одномерная сетка потоков; *б* – трехмерная сетка потоков

	1GPU	2GPU	4GPU	8GPU
1 v.	1.00	1.69	2.59	3.52
2 v.	1.00	1.99	3.93	7.70

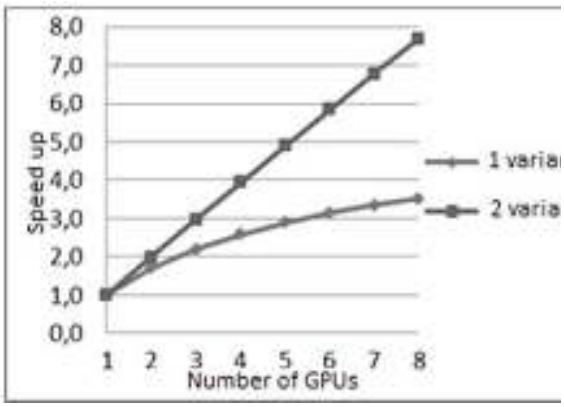
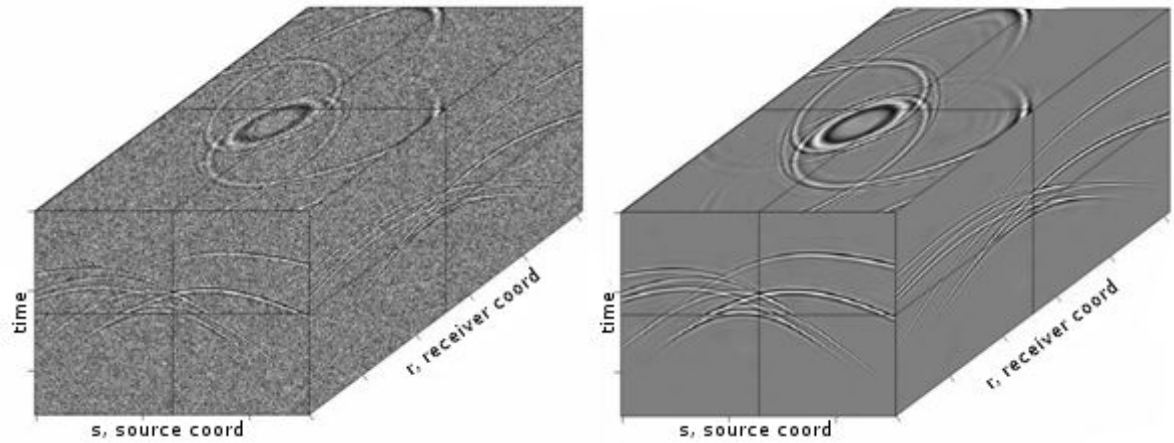
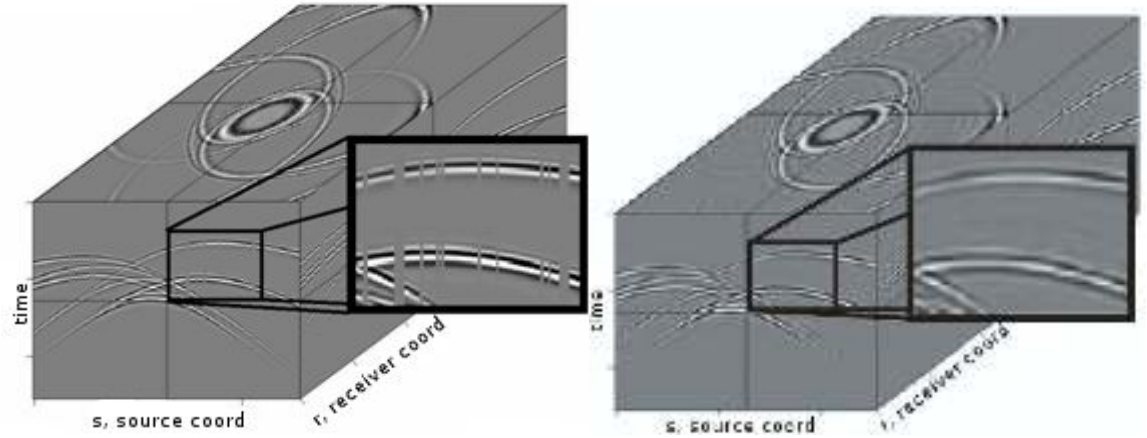


Рис. 9. Анализ масштабируемости по закону Амдала



а б

Рис. 10. Сжатие сейсмических данных и подавление шумов:
а – исходные данные со случайным шумом; *б* – результат сжатия в 50 раз (*CR* = .02) и подавления шумов.



а б

Рис. 11. Интерполяция сейсмических данных:
а – исходные данные с пропущенными трассами; *б* – результат сжатия в 50 раз (*CR* = .02) и интерполяции данных

Использование нескольких GPU

Проведено тестирование разных вариантов реализации алгоритма на нескольких графических картах в рамках одного узла (OpenMP); выполнен анализ масштабируемости для большого количества GPU.

Вариант 1. Первый вариант состоит в обработке всего объема входных данных следующим образом: «большое» преобразование Фурье производится на 1 GPU, а далее вычисления на коробках (интерполяция и «малое» преобразование Фурье) распределяются между несколькими GPU. Поскольку время на обработку одной коробки не является фиксированным, то требуется дополнительная работа по сборке выходных данных в правильном порядке. Были проведены вычисления на двух картах и построен график масштабируемости по закону Амдала (рис. 9). Из графиков видно, что для обработки одного куба в данном варианте нет смысла использовать больше количество карт. Сам процесс интерполяции масштабируется почти линейно, но есть большое количество операций, которые выполняются последовательно на CPU или одном GPU («большое» преобразование Фурье, операции копирования между CPU и GPU и др.).

Вариант 2. Второй вариант основан на использовании особенностей структуры сейсмических данных. В ходе обработки большого блока входных данных можно проводить его сегментацию на фрагменты и обрабатывать их практически независимо (в этом случае необходимо только правильно объединить коэффициенты разложения, количество которых мало для всех фрагментов, и, следовательно, эта процедура выполняется очень быстро). Данный вариант разложения не требует организации обмена данными при обработке разных фрагментов, так что можно задействовать как несколько GPU на одном узле, так и GPU на разных узлах. Наши эксперименты показали, что на одной видеокарте удобно работать с фрагментом размера 256^3.

Тестирование второго варианта программы на двух картах и анализ масштабируемости по закону Амдала (см. рис. 9) показали почти линейное ускорение: процент последовательных частей программы составил менее единицы, а теоретическое ускорение для 4-х и 8-ми графических карт составило соответственно 3.93 и 7.70. Это позволяет использовать много графических карт одновременно при обработке больших объемов данных.

Заметим, что второй вариант также позволяет обрабатывать данные, заданные на «прямоугольной» сетке $N1 \times N2 \times N3$, т. е. когда имеется неравное количество узлов в разных направлениях (изначально алгоритм был рассчитан на входные данные, заданные на кубической сетке размером N^3). Такую сетку достаточно разбить на кубические подобласти оптимального размера и обрабатывать их независимо.

Тестирование на синтетических данных

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

Пусть имеется куб сейсмических данных $f(\mathbf{x})$ размером N (количество пикселей), с шумом (рис. 10, *а*) или пропущенными трассами (рис. 11, *а*); на гранях показаны сечения через куб в местах, отмеченных тонкими линиями. Процедуру подавления помех и интерполяции данных можно реализовать, применив прямое ПВП, сохранение M коэффициентов, абсолютное значение которых больше заданного уровня, обратное ПВП:

$$f(\mathbf{x}) \xrightarrow{C} \{c_\gamma\} \xrightarrow{\text{отсев малых коэффициентов}} \{c_{\tilde{\gamma}}\} \xrightarrow{C^{-1}} \tilde{f}(\mathbf{x}).$$

Здесь $\tilde{f}(\mathbf{x})$ соответствует данным с подавленной помехой (рис. 10, *б*) и проведенной интерполяцией (рис. 11, *б*); C – оператор прямого ПВП (см. (1)). При $M < N$ происходит также сжатие данных. Степень сжатия будем измерять коэффициентом $CR = M / N$. В примерах на рис. 10–11 показано сжатие данных в 50 раз (CR = .02) без заметной потери качества изображения целевых волн.

Список литературы

1. *Гурвич И. И., Боганик Г. Н.* Сейсмическая разведка. М.: Недра, 1980. 551 с.

2. *Candes E., Demanet L., Donoho D., Ying L.* Fast discrete curvelet transforms // SIAM Multiscale Model. Simul. 2006. Vol. 5–3. P. 861–899.

3. *Candes E. J., Donoho D. L.* New Tight Frames of Curvelets and Optimal Representations of Objects with Piecewise-C^2 Singularities // Comm. Pure Appl. Math. 2002. Vol. 57. P. 219–266.

4. *Hennenfent G., Herrmann F.* Seismic Denoising with Non-Uniformly Sampled Curvelets // Computing in Science and Engineering. 2006. Vol. 8 (3). P. 16–25.

5. *Naghizadeh M., Sacchi M. D.* Beyond Alias Hierarchical Scale Curvelet Interpolation of Regularly and Irregularly Sampled Seismic Data // Geophysics. 2010. Vol. 75. P. 189–202.

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. Vol. 27. No. 2. P. 240–246.

7. *Duchkov A. A., Andersson F. A., Hoop M. V.* Discrete Almost-Symmetric Wave Packets and Multiscale Geometrical Representation of (Seismic) Waves // IEEE Transactions on Geoscience and Remote Sensing. 2010. Vol. 48. No. 9. P. 3408–3423.

8. *Dutt A. F., Rokhlin V. I.* Fast Fourier Transforms for Nonequispaced Data // SIAM Journal on Scientific Computing. 1993. Vol. 14. P. 1368–1393.

9. *Kirk D.* Programming Massively Parallel Processors: A Hands-on Approach. Morgan Kaufmann, 2010. 280 p.

Материал поступил в редколлегию 02.07.2012

V. V. Nikitin, A. A. Duchkov, A. A. Romanenko, F. Andersson

IMPLEMENTING ALGORITHM OF WAVE-PACKET DECOMPOSING ON GPUS AND ITS APPLICATIONS IN GEOPHYSICS

Seismic data is characterized by multidimensionality, large size and irregular structure. There is a need for optimal representation of this data by decomposing it using appropriate basis. In this paper we consider (redundant) basis of wave packets. With NVIDIA CUDA technology for programming on GPU we implemented a fast algorithm of forward and inverse 3D wave-packet transform. The code was optimized based on physical device characteristics and structure of the algorithm. We obtained speed-up ~45 for one GPU, and analyzed scalability for several GPUs. The program was tested on synthetic seismic data for their compression, de-noising and regularization.

Keywords: GPU, wave packets, fast Fourier transform, seismic.