УДК 004.021, 004.04

Н. Ю. Зятьков [1], **А. А. Айзенберг** [2], **А. М. Айзенберг** [3]

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

[2] *Бергенский университет*
Аллегатен, 41, 7803 №-5020, Берген, Норвегия

[3] *Институт нефтегазовой геологии и геофизики им. А. А. Трофимука СО РАН*
пр. Акад. Коптюга, 3, Новосибирск, 630090, Россия

nikolay.zyatkov@gmail.com

ВЫСОКООПТИМИЗИРОВАННАЯ РЕАЛИЗАЦИЯ ПРОЦЕДУРЫ РАСПРОСТРАНЕНИЯ ВОЛНОВОГО ПОЛЯ В ТРЕХМЕРНЫХ ГЕОЛОГИЧЕСКИХ СРЕДАХ С АДАПТАЦИЕЙ ДЛЯ GPU-КЛАСТЕРА [*]

Представлены алгоритмы оптимизации и адаптации на параллельные архитектуры процедуры, которая численно реализует распространение волнового поля в трехмерных геологических слоях. Эта процедура является частью разрабатываемого высокооптимизированного комплекса программ для дифракционного моделирования, реализующего метод наложения концевых волн (МНКВ). В работе кратко описывается идея МНКВ, показаны основные проблемы его реализации и предложены алгоритмы для наиболее эффективной обработки крупных массивов данных, используемые методом.

Ключевые слова: оптимизация программ, высокопроизводительные вычисления, GPU, сейсмическое моделирование

Введение

Сейсмический метод исследования недр Земли использует отраженные волны для послойного восстановления структуры и свойств реальной среды по наблюдаемым данным. Многие методы восстановления (методы обратных задач сейсмики) базируются на моделировании функции Грина (волновое поле точечного источника) для покрывающей среды, которая имитирует наблюдаемые волновые поля [1; 2]. Поиск функции Грина, которая дает разумный баланс между вычислительной скоростью и аналитической точностью, остается актуальной фундаментальной проблемой математической волновой теории [3]. Для случая покрывающей среды с сильной латеральной неоднородностью (соляные тела, базальтовые слои, рифовые структуры и т.д.) сейсмическое изображение – требующая затрат задача, которая привлекает огромное внимание при поисках нефтяных месторождений. Наличие больших скоростных контрастов, неоднородностей, анизотропии и затухания вкупе со сложными формами геологических границ понижает разрешающую способность сейсмики.

[*] Работа проводилась при частичной поддержке Шведского фонда по международному сотрудничеству в науке и высшем образовании и выполнена с использованием ресурсов информационно-вычислительного центра НГУ, а также ресурсов суперкомпьютерного комплекса МГУ им. М. В. Ломоносова.

Зятьков Н. Ю., Айзенберг А. А., Айзенберг А. М. Высокооптимизированная реализация процедуры распространения волнового поля в трехмерных геологических средах с адаптацией для GPU-кластера // Вестн. Новосиб. гос. ун-та. Серия: Информационные технологии. 2016. Т. 14, № 2. С. 38–51.

Строгая теория операторов прохождения-распространения, развитая в последнее время, дает точное аналитическое описание функции Грина для сложной среды [4]. Она основана на аналитическом решении прямой задачи для неоднородной среды с произвольно-гладкими границами в форме суперпозиции волновых сигналов многократно отраженных и преломленных волн согласно их волновому коду (последовательности проходимых слоев). Каждый отдельный сигнал описывается композицией операторов распространения в неоднородных слоях и операторов прохождения (отражения и преломления) конволюционного типа на гладких границах. Этот подход восстанавливает функцию Грина со структурой, подобной наблюдаемому волновому полю.

Для численной реализации теории операторов прохождения-распространения был предложен метод наложения концевых волн (МНКВ) [5; 6; 7]. Данный алгоритм использует аппроксимации операторов прохождения и распространения в диапазоне сейсмических частот и способен имитировать нерегулярности в волновом поле, например каустики, и порождать дифракции, головные и огибающие волны, которые не могут быть должным образом учтены при моделировании с помощью асимптотической лучевой теории или геометрической теории дифракции.

В данной статье авторы концентрируют свое внимание на оптимальной реализации и адаптации на параллельные архитектуры компонента МНКВ, численно реализующего распространение волнового поля внутри трехмерных геологических слоев. Работа состоит из введения, четырех разделов и заключения. В разделе 1 дано краткое описание метода наложения концевых волн. В разделе 2 описаны основные проблемы реализации данного алгоритма. В разделе 3 показана реализация и оптимизация процедуры действия матриц распространения на векторы волнового поля. В разделе 4 описан алгоритм адаптации данной процедуры для параллельных архитектур и GPU-кластера. В заключение приведены оценки полученных результатов и дальнейшие перспективы их развития.

1. **Метод наложения концевых волн**

МНКВ представляет собой численную реализацию теории операторов прохождения-распространения [4]. В данной теории для заданной слоистой модели среды, источника и приемников вводятся 2 оператора (рис. 1):

1) физически реализуемый оператор распространения P_F, распространяющий волновое поле между точками, предельными к криволинейным границам слоя, а также источником и приемниками;

2) оператор прохождения (преломления и отражения) T конволюционного типа, посредством которого производится преломление или отражение волнового поля на криволинейной границе – контакте двух сред.

При численной реализации данной теории все границы слоистой среды дискретизируются на N треугольников. Волновое поле на элементах каждой такой границы представляется в виде вектора a_F^s размерности N. Конволюционный оператор прохождения T и слоевой оператор распространения $P_F^{s_2 s_1}$ аппроксимируются в виде матриц размерности $N \times N$. Оператор распространения, переносящий волновое поле с границы в приемники P_F^{xs} – в виде матрицы размерности $N_x \times N$ (где N_x – количество приемников модели).

Реализацию алгоритма МНКВ в общем случае можно разделить на три основных блока.

1. Реализация вектора падающего волнового поля источника y на произвольную криволинейную границу $\mathbb{S} - a_F^s$.

2. Реализация распространения волнового поля с предыдущей, в соответствии с заданным волновым кодом, границы слоя $\mathbb{S}_i$ на следующую границу $\mathbb{S}_{i+1} - a_F^{s_{i+1}} = P_F^{s_{i+1} s_i} a_F^{s_i}$.

3. Реализация распространения волнового поля с последней, в соответствии с заданным волновым кодом, границы слоя $\mathbb{S}$ в приемники $x - a_F^x = P_F^{xs} a_F^s$.

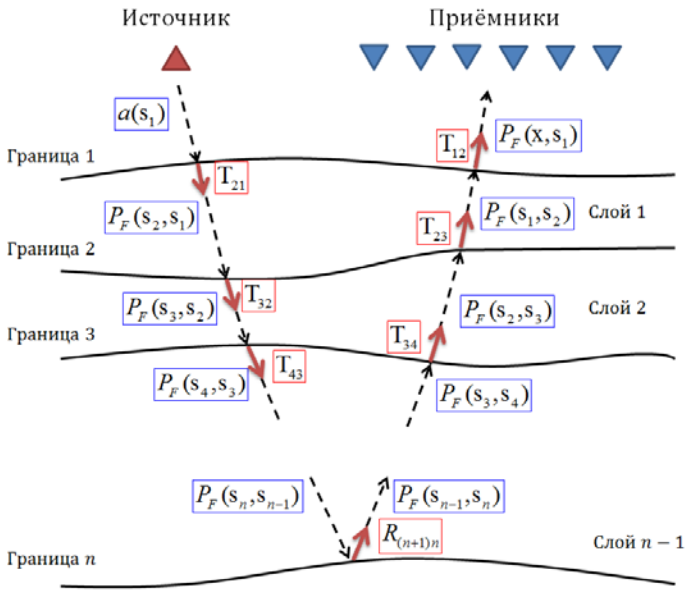

Рис. 1. Схема построения решения с помощью теории операторов прохождения-распространения

При задании волнового кода – траектории прохождения волновым полем слоев модели от источника в приемники, алгоритм, комбинируя данные реализованные блоки, выполняет расчет сейсмических данных в приемниках.

2. **Основные проблемы реализации МНКВ**

Алгоритм вычисления волновых полей посредством МНКВ сводится к многократным перемножениям матриц пучков концевых волн (ПКВ) [4] на векторы волнового поля. Основные проблемы реализации алгоритма МНКВ можно свести к следующим.

1. Теория операторов прохождения-распространения была построена в частотной области. Поэтому МНКВ для вычисления решения необходимо все матрично-векторные операции производить для каждой дискретной частоты ω_k из некоторого частотного набора ω_1 ... ω_K.

2. Из-за высокой дискретизации границ моделей среды размерность матриц МНКВ становится неприемлемо большой даже для современных вычислительных машин.

3. Все матрицы и векторы МНКВ являются плотными. Это свойство не позволяет использовать для их обработки готовые высокооптимизированные процедуры работы с разреженными матрицами сторонних библиотек.

Приведем следующий пример: объем необходимой оперативной памяти для хранения одной матрицы размерности $N \times N$ при $N = 150\ 000$, с учетом того, что все ее элементы являются комплексными и заполняются числами с плавающей запятой одинарной точности, составляет $N * N * 4 * 2 = 150\ 000 * 150\ 000 * 4 * 2 = 168$ Гб (путем экспериментальных исследований было показано, что использование одинарной точности при вычислении алгоритма МНКВ не ухудшает выходное сейсмическое изображение). Перед авторами стояла задача разработать алгоритм обработки массивов большой размерности при наличии ограниченных ресурсов оперативной памяти. При этом для стандартных операций линейной алгебры было решено использовать готовые высокооптимизированные решения сторонних библиотек. Обработка большого количества крупных массивов данных естественным образом отрицательно влияет на общее время вычислений. Для ускорения вычислений и возможности применения МНКВ в реалистических (прямых и обратных) задачах сейсмики алгоритм был

оптимизирован и адаптирован для параллельных архитектур и GPU-кластера. Разработанные алгоритмы и результаты работы процедур будут описаны в последующих разделах.

3. **Реализация и оптимизация процедуры распространения волнового поля МНКВ**

Как уже отмечено, теория операторов прохождения-распространения была построена в частотной области. Поэтому для получения решения все вычисления требуется проводить для каждой дискретной частоты ω_k из набора $\omega_1 ... \omega_K$. Ввиду этого волновое поле на элементах границы слоя представляется как набор векторов $a^{\omega_1} ... a^{\omega_K}$, каждый из которых имеет размерность N, где N – количество элементов дискретизированной границы. Опишем здесь процедуру МНКВ, которая перемножает матрицу распространения P^{ω_k} размерности $N \times N$ на соответствующий вектор волнового поля a^{ω_k} для каждой частоты ω_k из набора $\omega_1 ... \omega_K$. Процедура получает на входе K векторов волнового поля $a^{\omega_1} ... a^{\omega_K}$ и возвращает K преобразованных векторов $a'^{\omega_1} ... a'^{\omega_K}$ (рис. 2).

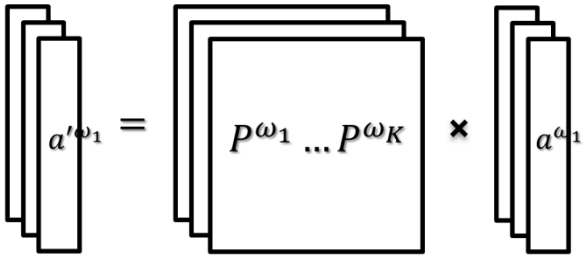

Рис. 2. Общая схема преобразования векторов волнового поля МНКВ

Пусть мы имеем в распоряжении оперативную память для хранения набора векторов $a^{\omega_1} ... a^{\omega_K}$, каждый размерности N, и блок памяти P размера $N * N * 4 * 2$ байт, где можно хранить одну комплексную матрицу P^{ω_k} размерности $N \times N$, используя одинарную точность. Тогда алгоритм реализации данной процедуры может выглядеть следующим образом:

```
// Цикл по номерам дискретных частот
for ( k = 1 ... K )
{
// Вычисляем частоту ω_k
ω_k = k * dω;
// Заполняем каждый элемент блока памяти P по формуле ПКВ,
// зависящий от частоты ω_k . Блок P на каждой итерации
// по частоте k переиспользуется.
[P_ij] = [TWB_ij](ω_k);
// Вычисляем преобразованный вектор a'^{ω_k}
a'^{ω_k} = P × F^{ω_k};
}
```

Для хранения результирующих векторов $a'^{\omega_1} ... a'^{\omega_K}$ можно переиспользовать оперативную память, в которой хранятся векторы $a^{\omega_1} ... a^{\omega_K}$. Но даже при такой экономии памяти для хранения одной матрицы P^{ω_k} потребуется $N * N * 4 * 2$ байт памяти. Только при $N \simeq 10^5$

требуется $\simeq 100$ Гб оперативной памяти. Даже для современных вычислительных систем этот объем является очень большим.

Проблема хранения матриц МНКВ целиком в оперативной памяти была решена с помощью разбиения исходной задачи на подзадачи и последовательного их решения. В данном случае каждая матрица P^{ω_k} разбивалась на виртуальные полосы размерности $M \times N$, где $M \le N$ (рис. 3). Размерность M пользователь выбирает таким образом, что полосу размерности $M \times N$ есть возможность хранить в оперативной памяти вычислительной машины, которую он использует. Например, если $M \simeq 10^3$, а $N \simeq 10^5$, то для хранения полосы размерности $M \times N$ потребуется $10^3 \times 10^5 \times 4 \times 2 \simeq 1$ Гб оперативной памяти, что уже приемлемо для современных вычислительных систем. Если пользователь не обладает подобными ресурсами, то он может принять M за 10^2. Тогда объем требуемой памяти для хранения такой полосы составит $10^2 \times 10^5 \times 4 \times 2 \simeq 100$ Мб.

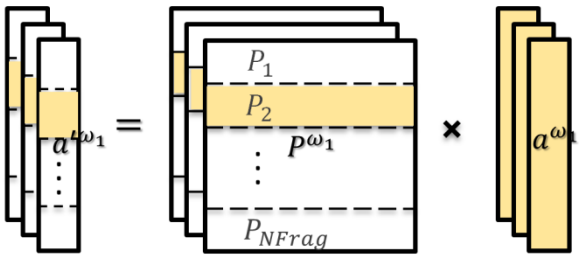

Рис. 3. Схема преобразования векторов волнового поля
с разбиением матриц распространения на виртуальные полосы

Теперь задачу перемножения K матриц большой размерности на K векторов можно сформулировать следующим образом. Пусть мы имеем в распоряжении оперативную память для хранения K комплексных векторов размерности N и оперативную память для хранения прямоугольной матрицы размерности $M \times N$. Необходимо реализовать процедуру, принимающую на входе K векторов размерности N и возвращающую K преобразованных векторов размерности N путем перемножения K матриц размерности $N \times N$ на K векторов размерности N, соответственно. Существует два способа реализации подобной задачи. Опишем два алгоритма и затем, путем анализа, выберем наилучший вариант для нашей задачи.

Алгоритм 1

// Вычисляем количество полос разбиения матрицы
$NFrag = N / M$;
// Цикл по номерам дискретных частот
for ($k = 1 \ldots K$)
{
// Вычисляем частоту ω_k
$\omega_k = k * d\omega$;
// Цикл по полосам матрицы P^{ω_k}
for ($frag = 1 \ldots NFrag$)
{
// Заполняем $frag$-ю часть матрицы P^{ω_k}
$[P_{frag\_ij}] = [TWB_{ij}](\omega_k)$;
// Вычисляем $frag$-ю часть вектора a'^{ω_k}

$a_{frag}{}'^{\omega_k} = P_{frag} \times a^{\omega_k}$;
}
}

Алгоритм 2
// Вычисляем количество полос разбиения матрицы
$NFrag = N / M$;
// Цикл по полосам матрицы P^{ω_k}
for ($frag = 1 \ldots NFrag$)
{
// Цикл по номерам дискретных частот
for ($k = 1 \ldots K$)
{
// Вычисляем частоту ω_k
$\omega_k = k * d\omega$;
// Заполняем $frag$-ю часть матрицы P^{ω_k}
$[P_{frag\_ij}] = [TWB_{ij}](\omega_k)$;
// Вычисляем $frag$-ю часть вектора a'^{ω_k}
$a_{frag}{}'^{\omega_k} = P_{frag} \times a^{\omega_k}$;
}
}

В описанных алгоритмах на каждой итерации внутреннего цикла используется полоса памяти размером $M \times N$ для хранения части матрицы P^{ω_k}, зависящей от частоты ω_k. С первого взгляда время исполнения Алгоритма 1 и Алгоритма 2 будет одинаковым: на каждой итерации цикла происходят заполнения полосы памяти P_{frag} размерности $M \times N$ и умножение ее на вектор волнового поля a^{ω_k} размерности N. Но, зная формулу ПКВ, вычисления можно организовать таким образом, что Алгоритм 2 окажется эффективнее Алгоритма 1.

Рассмотрим структуру формулы ПКВ. Эта формула содержит в себе тяжелые с вычислительной точки зрения операции взятия квадратного корня, синусов и косинусов. Особенность этой формулы в том, что ПКВ можно представить в виде произведения:

$$TWB_{ij}(\omega_k) = Amp_{ij}(d\omega_k) * k * Pha_{ij}(d\omega_k)^k. \qquad (1)$$

Формулы $Amp_{ij}(d\omega_k)$ и $Pha_{ij}(d\omega_k)$ содержат тяжелые вычислительные операции взятия синусов, косинусов и извлечения квадратного корня. $d\omega_k$ – это шаг разбиения интервала частот от ω_1 до ω_K, т. е. является константой, k – номер частоты. Получается, что выражения $Amp_{ij}(d\omega_k)$ и $Pha_{ij}(d\omega_k)$ являются константами для каждой частоты $\omega_k = k * d\omega$. Из формулы (1) следует ее матричное выражение:

$$[TWB_{ij}](\omega_k) = [TWB_{ij}](\omega_{k-1}) \cdot [Pha_{ij}](d\omega_k) * \frac{k}{k-1}. \qquad (2)$$

За знак $\cdot$ принята операция, осуществляющая поэлементное умножение двух матриц одинаковых размерностей.

Таким образом, используя Алгоритм 2, для каждой $frag$-й части матрицы P^{ω_k} перед циклом по частотам мы можем заполнить массив P_{frag} значениями $Amp(d\omega_k)$ и, предварительно заведя дополнительный массив Pha размерности $M \times N$, заполнить его значениями $Pha(d\omega_k)$. Затем в цикле по частотам модифицировать значения массива P_{frag} для каждой следующей k-й частоты (начиная со 2-й), используя полученную формулу (2) таким образом:

$$[P_{frag\_ij}] = [P_{frag\_ij}] \cdot [Pha_{ij}] * \frac{k}{k-1}.$$

Данной модификацей операция заполнения массивов больших размерностей по формулам, содержащим тяжелые для вычисления синусы, косинусы и операции взятия квадратного корня, по сути, была сведена к операции поэлементного перемножения двух массивов размерности $M \times N$. Далее приведен *Модифицированный алгоритм 2*.

Модифицированный алгоритм 2
// Вычисляем количество полос разбиения матрицы
$NFrag = N / M$;
// Цикл по полосам матрицы P^{ω_k}
for ($frag = 1 \ldots NFrag$)
{
$[P_{frag\_ij}] = [Amp_{ij}(d\omega_k)]$;
$[Pha_{ij}] = [Pha_{ij}(d\omega_k)]$;
// Заполняем полосу P_{frag} в соответствии с номером частоты $k = 1$
$[P_{frag\_ij}] = [P_{frag\_ij}] \cdot [Pha_{ij}]$;
// Вычисляем $frag$-ю часть вектора a'^{ω_1}
$F_{frag}{}'^{\omega_1} = P_{frag} \times F^{\omega_1}$;
// Цикл по номерам дискретных частот
for ($k = 2 \ldots K$)
{
// Вычисляем частоту ω_k
$\omega_k = k * d\omega$;
// Перезаполняем полосу P_{frag} в соответствии с номером
// частоты k
$[P_{frag\_ij}] = [P_{frag\_ij}] \cdot [Pha_{ij}] * \frac{k}{k-1}$;
// Вычисляем *frag*-ю часть вектора a'^{ω_k}
$F_{frag}{}'^{\omega_k} = P_{frag} \times F^{\omega_k}$;
}
}

Алгоритм 1, Алгоритм 2 и Модифицированный алгоритм 2 были протестированы на 1 ядре процессора Intel(R) Xeon(R) CPU E5630 @2.53GHz для случая $N = 112\ 000$ элементов дискретизированной границы, ширины полос разбиения матриц $M = 512$ элементов и $K = 128$ дискретных частот. Время исполнения Алгоритма 1 и Алгоритма 2 составило примерно 27 ч. Время исполнения Модифицированного алгоритма 2 составило примерно 16 ч. Другими словами, только за счет изменения алгоритма реализации процедуры перемножения

K матриц на K векторов мы получили ускорение в 1,7 раз при однопроцессорном исполнении. Это оказалось возможным за счет специфики формулы ПКВ и использования двух дополнительных массивов памяти размерности $M \times N$, размер которых контролируется путем изменения числа M в меньшую или большую сторону. Далее будет описан алгоритм адаптации данной процедуры для параллельных архитектур и GPU-кластера.

4. Адаптация процедуры распространения волнового поля МНКВ для параллельных архитектур и GPU-кластера

В этом разделе будут описаны алгоритмы адаптации процедуры перемножения матриц распространения $P^{\omega_1}...P^{\omega_K}$ на набор векторов волнового поля $a^{\omega_1}...a^{\omega_K}$ для параллельных архитектур и, в частности, для GPU-кластера. Напомним, что за счет алгоритма реализации этой процедуры уже удалось ускорить ее в 1,7 раз для однопроцессорной машины. Также, несмотря на огромные размеры матриц, процедуру (здесь и далее под процедурой понимается *Модифицированный алгоритм 2* из раздела 3) удалось реализовать, при наличии ограниченных ресурсов оперативной памяти машины.

Как сказано в разделе 2, реализация алгоритма МНКВ, по сути, представляет собой большое количество матрично-векторных операций. Известно, что эти операции хорошо перекладываются на параллельные архитектуры. Попытаемся разработать алгоритмы распараллеливания для процедуры перемножения матриц распространения на векторы волнового поля. Заметим, что процедура имеет 2 уровня параллелизации:

1) возможность обработки полос матрицы в параллельном режиме – каждой полосе назначается одно вычислительное устройство;

2) возможность параллельного заполнения элементов полосы матрицы по формулам ПКВ и параллельного умножение матрицы на вектор – полосы обрабатываются последовательно, каждой полосе назначается несколько вычислительных устройств.

Авторами работы были проведены тесты для случая распараллеливания процедуры перемножения матриц распространения на векторы волнового поля в соответствии с уровнем 2 на 8 ядрах процессора Intel(R) Xeon(R) CPU E5630 @2.53GHz. Были взяты те же самые параметры дискретизации границы, размер полос матриц и количество дискретных частот, что и в разделе 3: $N = 112000$, $M = 512$ и $K = 128$ (заметим, что только при однократном вызове процедуры с такими параметрами машине приходится обрабатывать $N * N * 4 * 2 * K \simeq 12$ Тб данных!). В результате работы удалось ускорить процедуру в 8 раз – время ее исполнения сократилось с 16 ч на одноядерной машине до 2 ч на восьмиядерной. Такое линейное увеличение производительности связано с поддающимися эффективному распараллеливанию большими данными, которые обрабатывает процедура, и относительно небольшим числом ядер процессора, используемых для тестирования ее параллельной версии.

Все большей популярностью на сегодняшний день пользуются графические процессоры (GPUs) для параллельной обработки больших данных. Особенность архитектуры GPU состоит в том, что, в отличие от центрального процессора (CPU), он может содержать сотни вычислительных блоков, способных в параллельном режиме решать небольшие задачи (рис. 4). Исторически GPU предназначался для обработки 3D-графики, когда необходимо одновременно производить несложные операции над большим количеством треугольников, на которые разбита 3D-сцена (поворот, параллельный перенос и т. д.). Такая работа под силу и центральному процессору, но GPU со своей массивно-параллельной архитектурой делает это намного эффективнее. В 2007 г. компания Nvidia представила свою программно-аппаратную архитектуру параллельных вычислений CUDA (Compute Unified Device Architecture). Эта архитектура изначально являлась надстройкой над языком C и позволяет программистам реализовывать их алгоритмы с передачей управления исполнения программы графическому процессору. Если алгоритм имеет высокую степень параллелизмами, т. е. если есть возможность его разбить на большое количество маленьких независимых подзадач, которые можно исполнить одновременно, то использование GPU позволит существенно увеличить производительность программы.

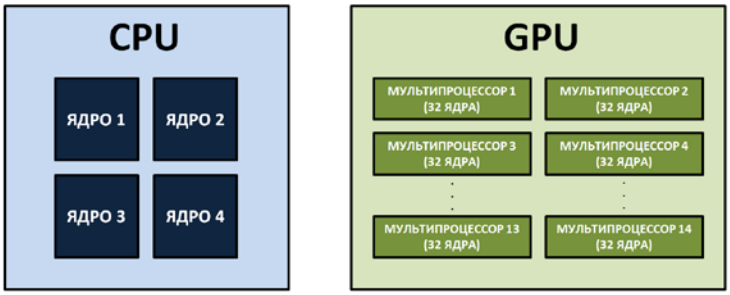

Рис. 4. Схематичное сравнение архитектуры CPU и GPU

Для задачи параллельного умножения набора крупных матриц на набор векторов можно было бы исполнить в параллельном режиме заполнение матриц по формулам ПКВ (каждый GPU-поток (нить) заполнял бы соответствующий элемент матрицы) и затем параллельно умножать матрицу на вектор. Для последней процедуры в CUDA имеется специальная высокооптимизированная библиотека линейной алгебры для GPU cuBLAS.

Для реализации заполнения матрицы в параллельном режиме на GPU авторами была реализована функция (ядро), при вызове которой исполнение программы передавалось графическому процессору, создавалось столько нитей, сколько элементов содержит матрица, каждая их которых заполняла соответствующий элемент матрицы по формуле ПКВ в параллельном режиме (листинг 1).

Листинг 1. Ядро, заполняющее в параллельном режиме элементы матрицы распространения, соответствующей дискретной частоте ω_1.

```
__global__ void PropagationMatrixPart_kernel(int lnum, int istrnsm, Triangle *tr_g,
int N, int M, float *v_g, float *rho_g, cuComplex *P_g, cuComplex *PhaP_g)
{
int i = threadIdx.y + blockIdx.y * blockDim.y;
int j = threadIdx.x + blockIdx.x * blockDim.x;

if (i < M && j < N)
{
// заполняем мнимую и действительную часть матрицы P_g
P_g[i + M * j].x = TWB_amp_x;
P_g[i + M * j].y = TWB_amp_y;

// заполняем мнимую и действительную часть матрицы PhaP_g
PhaP_g[i + M * j].x = TWB_pha_x;
PhaP_g[i + M * j].y = TWB_pha_x;

// поэлементно умножаем матрицу P_g на PhaP_g и сохраняем в P_g
// в P_g будет храниться теперь формула TWB для частоты с номером 1
P_g[i + M * j].x = -P_g[i + M * j].y * PhaP_g[i + M * j].y;
P_g[i + M * j].y = P_g[i + M * j].y * PhaP_g[i + M * j].x;
}
}
```

В листинге 1 переменные M и N хранят в себе размерность полос матриц, переменная `tr_g` содержит адрес массива треугольных элементов, на которые разбита граница, для которой исполняется процедура. TWB_amp_x, TWB_amp_y, TWB_pha_x, TWB_pha_y – формулы ПКВ, по которым происходят преобразования. Данное ядро реализует заполнение полосы `P_g` матрицы распространения для первой частоты ω_1. В следующем листинге 2 показано ядро, реализующее перезаполнение матрицы `P_g` для последующих частот по формуле (2), полученной в разделе 3.

Листинг 2. Ядро, заполняющее в параллельном режиме элементы матрицы распространения, соответствующей дискретной частоте ω_k, $k = 2 \ldots K$.

```
__global__ void PropagationMatrixPart_JF_kernel(cuComplex *P_g, cuComplex *PhaP_g,
float JF, int N, int M)
{
int j = threadIdx.x + blockIdx.x * blockDim.x;

float a, b, c, d;
while (j < M * N)
{
a = P_g[j].x;
b = P_g[j].y;
c = PhaP_g[j].x;
d = PhaP_g[j].y;

P_g[j].x = (a * c - b * d) * JF / (JF - 1);
P_g[j].y = (a * d + c * b) * JF / (JF - 1);

j += blockDim.x * gridDim.x;
}
}
```

Здесь переменная `JF` выступает в роли номера k дискретной частоты, $k = 2 \ldots K$.

Для перемножения матрицы на вектор была использована высокооптимизированная функция `cublasCgemv` из библиотеки cuBLAS (NVIDIA CUDA Basic Linear Algebra Subroutines), реализующая умножение матрицы на вектор в параллельном режиме с использованием GPU:

```
cublasCgemv(handle, CUBLAS_OP_N, M, N, &alpha, P_g, M, u_g, 1, &beta, y_g + k*M11, 1).
```

Здесь, полоса `P_g` размерности $M \times N$ умножается на вектор волнового поля `u_g` размерности N на GPU. Результат перемножения (блок памяти размерности M) сохраняется в вектор `y_g` размерности N по адресу `y_g + k*M11`, где `k` в программе – номер обрабатываемой полосы матрицы распространения.

На рис. 5 показан результат профилирования описанной процедуры, реализованной на GPU.

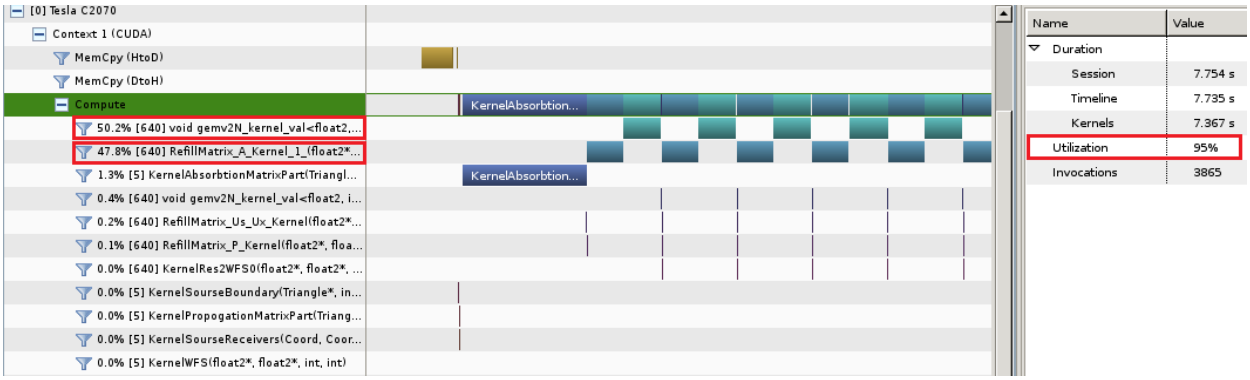

Рис. 5. Профилирование процедуры *Модифицированный алгоритм 2*, реализованной на GPU

Из рис. 5 видно, что ядро, реализующее перезаполнение матриц, занимает 48 % времени работы всего алгоритма, а ядро, реализующее перемножение матрицы на вектор, – 50 %. Также из рис. 5 видно, что загруженность графических процессоров составляет 95 %, что говорит о высокой эффективности использования GPU алгоритмом. Это связано с отсутствием передачи данных между GPU и CPU во время работы алгоритма: перед запуском программы

все данные копируются в память GPU, затем после вычислений результат копируется обратно на CPU. При $N = 112\ 000$ и $K = 128$ время исполнения процедуры перемножения K матриц размерности $N \times N$ на K векторов размерности N на видеокарте NVIDIA Tesla C2070 составило 10 мин, что почти в 162 раза быстрее времени работы первоначальной версии последовательной процедуры.

Далее опишем алгоритм реализации данной процедуры для GPU-кластера. При реализации процедуры на GPU, как и в случае с многоядерным процессором, использовался лишь второй уровень распараллеливания, описанный в начале подраздела: перезаполнение и умножение матриц на GPU происходит в параллельном режиме, но не использовался первый уровень распараллеливания – алгоритм по-прежнему обрабатывает последовательно полосы матрицы. Для того чтобы максимально воспользоваться высокой степенью параллелизма процедуры, была произведена ее реализация с использованием GPU-кластера. Под GPU-кластером понимается набор GPUs, способных производить вычисления одновременно и связанных между собой с целью обмена данными. Теперь каждый GPU может заниматься обработкой назначенных ему полос матрицы распространения. На рис. 6 показана схема реализации процедуры перемножения K матриц распространения на K векторов волнового поля для GPU-кластера.

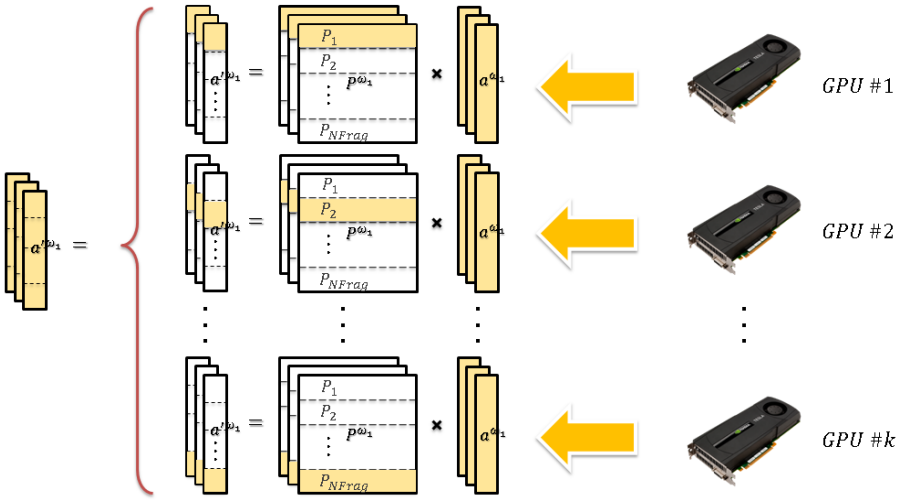

Рис. 6. Схема реализации процедуры *Модифицированный алгоритм 2* для GPU-кластера

Каждый GPU вычисляет K частей результирующих векторов. Затем все GPUs обмениваются друг с другом посчитанными результатами и формируют K преобразованных процедурой векторов волнового поля. Ввиду того что перезаполнения полос матриц и умножения их на вектор длятся много больше, чем обмен результирующих частей преобразованных векторов, то это не сильно повлияло на масштабируемость программы, которая оказалась достаточно высокой. На рис. 7 показан график зависимости производительности программы от числа задействованных GPUs в сравнении с идеальным случаем (линейная масштабируемость).

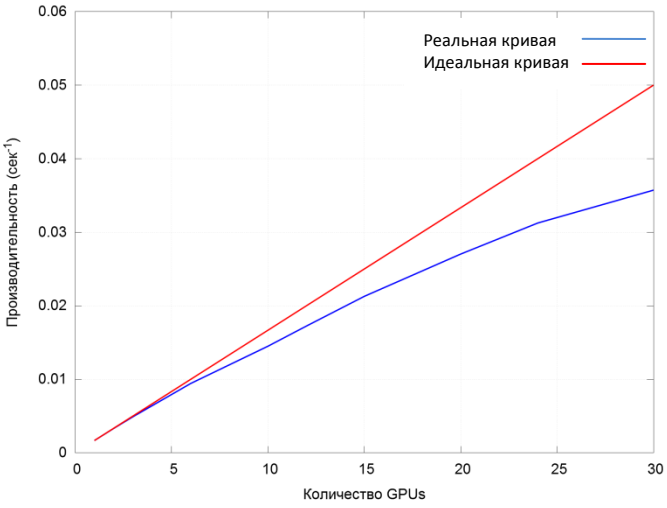

Рис. 7. График масштабируемости процедуры *Модифицированный алгоритм 2* при ее реализации для GPU-кластера

В таблице показано время исполнения процедуры (реальный и идеальный случаи) в зависимости от числа задействованных GPUs для случая перемножения K матриц распространения размерности $N \times N$ на K векторов волнового поля размерности N, где $N = 112\ 000$ и $K = 128$.

Анализ масштабируемости процедуры *Модифицированный алгоритм 2*

Количество задействованных GPU	Время работы программы (сек.)	Время работы программы (сек.) (идеальный случай)
1	600,00	600,00
2	302,00	300,00
3	204,00	200,00
6	106,00	100,00
10	69,00	60,00
12	58,00	50,00
15	47,00	40,00
20	37,00	30,00
24	32,00	25,00
30	28,00	20,00

Отличие реального времени счета от идеала возникает в результате того, что после вычисления каждым GPU назначенных ему K частей векторов волнового поля (см. рис. 6) видеокарты должны обменяться друг с другом посчитанными результатами. Это необходимо, поскольку в памяти каждого GPU должен быть собран результирующий набор векторов волнового поля размерности N. Обмен данными между GPUs порождают временные издержки, но, тем не менее, видно, что при использовании 30 GPU время исполнения процедуры стало составлять 28 сек., вместо 27 ч первоначальной последовательной версии. Получено ускорение процедуры в 3 471 раз. Важное свойство алгоритма состоит в том, что оно способно захватить любое доступное количество GPUs, меньшее числа N – количества треугольников дискретизованной границы. Как уже было сказано, в алгоритме МНКВ число N имеет поря-

док $10^5 - 10^6$. Это значит, что приложение способно максимально и эффективно (ввиду достаточно высокой масштабируемости) использовать доступные ресурсы GPU-кластера.

МНКВ неоднократно тестировался для моделей сред со сложными границами [5–7]. В подобных тестах авторами было показано, что полученные результаты подтверждают математическую теорию волн, а также точность и устойчивость алгоритма.

Заключение

В данной работе были приведены алгоритмы реализации, оптимизации и адаптации для кластера из графических ускорителей компонента метода наложения концевых волн, вычисляющие распространение волнового поля в трехмерных геологических слоях со сложными криволинейными границами. За счет специфики формул алгоритма и современных вычислительных технологий авторам удалось реализовать данную процедуру таким образом, что время ее вычисления сократилось с десятков часов на однопроцессорной вычислительной машине до десятков секунд на GPU-кластере. Кроме того, достигнута приемлемая величина оперативной памяти ЭВМ, использующаяся алгоритмом. Данные результаты дают перспективу использования МНКВ, как моделирующего ядра для решения обратной задачи геофизики – послойного восстановления параметров и свойств геологических сред.

* * *

Авторы выражают благодарность Ф. Андерссону (Университет Лунда, г. Лунд, Швеция), М. А. Айзенберг (Исследовательский Центр фирмы Статойл, г. Берген, Норвегия), А. А. Дучкову (Институт нефтегазовой геологии и геофизики им. А. А. Трофимука СО РАН, г. Новосибирск, Россия) и А. А. Романенко (Новосибирский государственный университет, г. Новосибирск, Россия) за полезные обсуждения и замечания в ходе работы.

Список литературы

1. *Гольдин С. В.* Оценка коэффициента отражения при миграции обменных и монотипных волн // Геология и геофизика. 1992. Вып. 4. С. 90–105.

2. *Gray S. H.* Seismic imaging // Geophysics. 2001. Vol. 66. P. 15–17.

3. *Carcione J. M., Herman G. C., A. P. E. ten Kroode.* Seismic modeling // Geophysics. 2002. Vol. 67. 4, P. 1304–1325.

4. *Aizenberg A. M., Ayzenberg A. A.* Feasible fundamental solution of the multi-physics wave equation in inhomogeneous domain of complex shape // Wave Motion. 2015. Vol. 53. P. 66–79.

5. *Aizenberg A. M., Ayzenberg M. A., Klem-Musatov K. D.* Seismic diffraction modeling with the tip-wave superposition method. Extended Abstracts of the 73th EAGE Conference & Exhibition, Austria, Vienna, 23–26 May 2011, B018.

6. *Zyatkov N., Ayzenberg A., Aizenberg A. M., Romanenko A.* Highly-optimized TWSM Algorithm for Modeling Cascade Diffraction in Terms of Propagation-absorption Matrices // Extended Abstracts, 75th Conference and Exhibition, European Association of Geoscientists & Engineers, London, England, 10–13 June 2013, Th-P02-11.

7. *Ayzenberg A. A., Zyatkov N. Y., Stovas A., Aizenberg A. M.* The Feasible Near-front Wavefield Below Salt Overhang in Terms of Cascade Diffraction // Extended Abstracts, 76th EAGE Conference & Exhibition, Amsterdam, Netherlands, 16–19 June 2014, We P06 06.

Материал поступил в редколлегию 07.04.2016

N. Yu. Zyatkov [1], **A. A. Ayzenberg** [2], **A. M. Aizenberg** [3]

[1] *Novosibirsk State University*
2 Pirogov Str., Novosibirsk, 630090, Russian Federation

[2] *University of Bergen*
41 Allegaten, Bergen, 78 N-5020, Norway

[3] *Trofimuk Institute of Petroleum Geology and Geophysics of SB RAN*
3 Acad. Koptug Ave., Novosibirsk, 630090, Russian Federation

nikolay.zyatkov@gmail.com, Alena.Ayzenberg@uib.no, AizenbergAM@ipgg.sbras.ru

HIGHLY-OPTIMIZED PROCEDURE OF WAVEFIELD PROPAGATION IN 3-DEMENTIONAL GEOLOGY MEDIA WITH ADAPTATION FOR GPU-CLUSTER

This paper presents algorithms for optimization and adaptation to the parallel architecture of the procedure which is numerically implementing wavefield propagation in the 3-dimensional geological layers. This procedure is part of the developed highly-optimized software package for diffraction modeling which implements the tip-wave superposition method (TWSM). The paper briefly describes the idea of TWSM, shows the main problems of its implementation and proposes the most efficient algorithms for processing large arrays used by this method.

Keywords: software optimization, high performance computing, GPU, seismic modeling.

References

1. Goldin, S. V. Estimation of reflection coefficient under migration of converted and monotype waves. *Russian Geology and Geophysics*, 1992, vol. 33, 4, p. 76–90. (In Russ.)

2. Gray, S. H. Seismic imaging. *Geophysics*, 2001, vol. 66, p. 15–17.

3. Carcione, J. M., Herman G.C., A.P.E. ten Kroode. Seismic modeling. Geophysics, 2002, vol. 67, 4, p. 1304–1325.

4. Aizenberg A. M., Ayzenberg A. A. Feasible fundamental solution of the multiphysics wave equation in inhomogeneous domain of complex shape. Wave Motion, 2015, vol. 53, p. 66–79.

5. Aizenberg A. M., Ayzenberg M. A., Klem-Musatov K. D. Seismic diffraction modeling with the tip-wave superposition method. Extended Abstracts of the 73th EAGE Conference & Exhibition, Austria, Vienna, 23–26 May 2011, B018.

6. Zyatkov N., Ayzenberg A., Aizenberg A. M., Romanenko A. Highly-optimized TWSM Algorithm for Modeling Cascade Diffraction in Terms of Propagation-absorption Matrices. Extended Abstracts, 75th Conference and Exhibition, European Association of Geoscientists & Engineers, London, England, 10–13 June 2013, Th-P02-11.

7. Ayzenberg A. A., Zyatkov N. Y., Stovas A., Aizenberg A. M. The Feasible Near-front Wavefield Below Salt Overhang in Terms of Cascade Diffraction. Extended Abstracts, 76th EAGE Conference & Exhibition, Amsterdam, Netherlands, 16–19 June 2014, We P06 06.