УДК 519.6; 517.958:5

РАЗРАБОТКА ВЫСОКОЭФФЕКТИВНЫХ МАСШТАБИРУЕМЫХ ПРОГРАММ В ЗАДАЧЕ УЛЬТРАЗВУКОВОЙ ТОМОГРАФИИ

Вад. В. Воеводин[1], С. Л. Овчинников[1], С. Ю. Романов[1]

Статья посвящена разработке масштабируемых программ для суперкомпьютеров в обратной задаче ультразвуковой томографии в приложении к медицине. Задача рассмотрена в постановке нелинейной коэффициентной обратной задачи для волнового уравнения. Предложена структура программы, позволяющая эффективно распараллеливать вычисления на кластерных вычислительных системах. Проведена оптимизация кода, которая позволила ускорить работу программы на порядок, а также обеспечить высокую степень масштабируемости. Проведены многочисленные тестовые расчеты на вычислительных системах МГУ "Чебышев" и "Ломоносов". Работа выполнена при финансовой поддержке Министерства образования и науки РФ в рамках ФЦП "Исследования и разработки по приоритетным направлениям развития научно технологического комплекса России на 2007–2013 годы" (госконтракт № 07.514.12.4024).

Ключевые слова: коэффициентные обратные задачи, волновое уравнение, ультразвуковая томография, параллельные вычисления, масштабируемость, суперкомпьютер.

1. Введение. В настоящей статье рассмотрены методы повышения эффективности и масштабируемости программного обеспечения в задаче ультразвуковой томографии. Эта задача рассмотрена в приложении к медицине — диагностике заболеваний молочной железы. Дифференциальная диагностика доброкачественных и злокачественных новообразований требует высококачественных, с высоким разрешением изображений молочной железы в разных сечениях этого трехмерного объекта. Именно томографические методы исследования могут решить эту проблему. Современные тенденции в медицине связаны с сокращением лучевой нагрузки, что ставит ограничение на частое использование рентгеновских томографов. В этой связи наиболее приемлемыми являются УЗИ-томографические комплексы с высоким разрешением [1, 2].

Требование повышения разрешающей способности диагностики приводит к необходимости усложнения используемых математических моделей. В связи с этим обсуждаемую задачу приходится рассматривать в нелинейной волновой томографической постановке для диагностики трехмерных объектов. Решение задач в такой постановке связано с огромным объемом вычислений, что представляет серьезную проблему даже для современных суперкомпьютеров. Для эффективного решения этой проблемы можно использовать разные подходы. Один из них состоит в выборе математических методов решения, позволяющем уменьшить объем вычислений. В работах [3–7] обратная задача волновой томографии рассматривалась в интегральной постановке, полученной с использованием функции Грина. В работах [8–11] рассмотрен дифференциальный подход к решению. В дифференциальной постановке удается достаточно просто получить градиент минимизируемого функционала. Разумеется, отличие постановок никак не может сказаться на получаемом решении, однако дифференциальная постановка имеет принципиальное для данной задачи преимущество: вычислительная сложность задачи уменьшается на несколько порядков [11]. Это приводит к тому, что, если в интегральной постановке даже на современных суперкомпьютерах удавалось решать задачи на очень небольших сетках [12], то в дифференциальной постановке размерность сетки увеличивается в десятки и сотни раз [13, 14].

С последующим развитием и усложнением модели, а также с увеличением технических возможностей ультразвуковых томографических установок [1, 2] (уменьшением длины волны, увеличением числа трансдьюсеров и т.д.) объем необходимых вычислений будет только возрастать. Вместе с тем, развитие вычислительной техники идет по пути все большего распараллеливания вычислений (уже сейчас вычислительный кластер "Ломоносов" имеет несколько десятков тысяч вычислительных ядер [15]). Поэтому

[1] Научно-исследовательский вычислительный центр, Московский государственный университет им. М. В. Ломоносова, 119991, Москва; Вад. В. Воеводин, мл. науч. сотр., e-mail: vadim_voevodin@mail.ru; С. Л. Овчинников, электроник, e-mail: osl@starnet.ru; С. Ю. Романов, вед. науч. сотр., e-mail: romanov60@gmail.com

чрезвычайно важно разрабатывать именно такие методы и алгоритмы, которые бы позволяли эффективно распараллеливать решение задачи на большом числе вычислительных узлов. Настоящая работа посвящена этой проблеме.

2. Математическая постановка задачи. Рассмотрим следующее волновое уравнение, которое в скалярном приближении описывает акустическое или электромагнитное поле $u(r,t)$ в трехмерной области $\Omega \subset \mathbb{R}^3$, ограниченной поверхностью S, на временно́м интервале $[0,T]$ с точечным источником, располагающимся в точке r_0 [9–11]:

$$c(r)u_{tt}(r,t) - \Delta u(r,t) = \delta(r-r_0)\,f(t). \tag{1}$$

Начальные и граничные условия запишем в виде

$$u(r,t=0) = u_t(r,t=0) = 0, \quad \partial_n u\big|_{ST} = p(r,t). \tag{2}$$

Здесь $c^{-1/2}(r)$ — скорость волны в среде; $r \in R^3$ — положение точки в пространстве; Δ — оператор Лапласа по переменной r; генерируемый источником импульс описывается функцией $f(t)$; $\partial_n u\big|_{ST}$ — производная вдоль нормали к поверхности S в области $ST = S \times (0,T)$; $p(r,t)$ — некоторая известная функция. Будем предполагать, что неоднородность среды вызвана только изменениями скорости, а вне области неоднородности, для простоты, скорость постоянна: $c(r) \equiv c_0 = \mathrm{const}$, где скорость c_0 известна.

Обратная задача состоит в нахождении функции $c(r)$, описывающей неоднородность, по экспериментальным данным измерения волны $U(s,t)$ на границе S области за время $(0,T)$ при различных положениях источника.

3. Метод решения. Как известно, задача (1), (2) задает $u(r,t)$ как неявную функцию от $c(r)$. Поставим обратную задачу как задачу минимизации квадратичного функционала

$$\Phi\big(u(c),c\big) = \frac{1}{2}\left\|u\big|_{ST} - U\right\|^2 = \frac{1}{2}\int\limits_0^T \int\limits_S \big(u(s,t) - U(s,t)\big)^2\,ds\,dt. \tag{3}$$

Здесь $\|\cdot\|^2$ — квадрат нормы в пространстве $L_2\big(S\times(0,T)\big)$, $U(s,t)$ — экспериментальные данные измерения волны на границе S области за время $(0,T)$. Для минимизации функционала будем использовать градиентные итерационные методы. В работе [11] показано, что градиент функционала (3) можно выписать в форме

$$\Phi'_C\big(u(c),c\big) = \int\limits_0^T w_t(r,t)u_t(r,t)\,dt. \tag{4}$$

Здесь $u(r,t)$ — решение основной задачи (1), (2), а $w(r,t)$ — решение следующей "сопряженной задачи" при заданной скорости $c^{-1/2}(r)$:

$$c(r)w_{tt}(r,t) - \Delta w(r,t) = 0, \tag{5}$$

$$w(r,t=T) = w_t(r,t=T) = 0, \quad \partial_n w\big|_{ST} = u\big|_{ST} - U. \tag{6}$$

Здесь u — решение основной задачи (1), (2). Таким образом, для вычисления градиента функционала необходимо решить последовательно основную и "сопряженную" задачи.

4. Использование параллельных вычислений при решении обратной задачи. Как отмечалось выше, одна из проблем решения рассматриваемой задачи состоит в необходимости выполнения огромного объема вычислений. Эффективным методом решения этой проблемы является использование параллельных вычислений на многопроцессорных системах. Возможность эффективного использования параллельных вычислений при решении обратной задачи определяется структурой алгоритма, возможностью выделения большого числа максимально вычислительно независимых блоков.

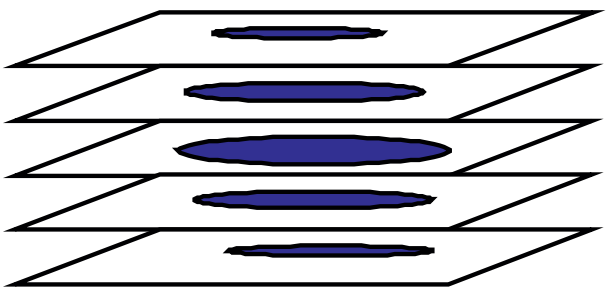

Рис. 1. Представление структуры объекта в томографии по слоям

Следуя классическим традициям томографических исследований, задача диагностики 3D объекта рассматривается как набор двумерных задач, каждая из которых представляет собой коэффициентную обратную задачу для уравнения гиперболического типа.

Рассмотрение слоев в совокупности описывает трехмерную структуру объекта (рис. 1). При таком подходе вычисления в каждом слое производятся независимо. Следовательно, распараллеливание вычислений по слоям является максимально эффективным.

Дальнейшее распараллеливание вычислений выполняется по источникам излучения. Как видно из постановки (1), (2), задача решается для различных положений источника. Количество таких положений в реальном эксперименте может достигать нескольких десятков. Заметим, что основной объем вычислений, т.е. решение основной (1), (2) и "сопряженной" (5), (6) задач для разных положений источника выполняется независимо. Это говорит о том, что структура рассматриваемого алгоритма такова, что имеется возможность эффективного распараллеливания вычислений для каждого слоя задачи по источникам.

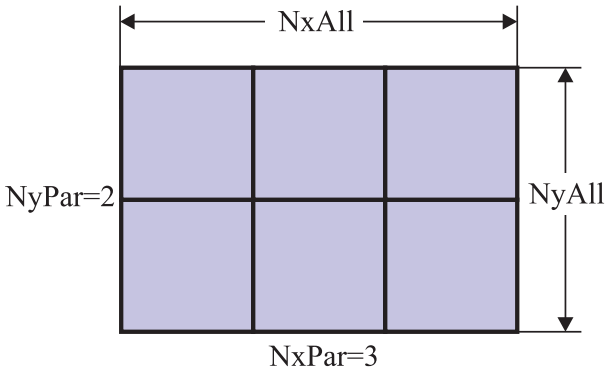

Рис. 2. Разбиение поля вычислений на независимые части

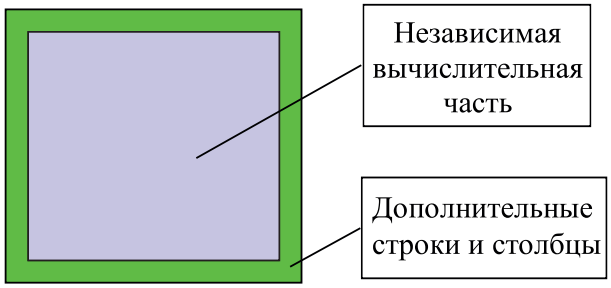

Рис. 3. Структура области вычислений каждого процесса

Рассмотрим возможность дальнейшего распараллеливания при проведении вычислений для выбранного слоя и заданного источника. Необходимо найти градиент $\Phi'_C\big(u(c),c\big)$ по формуле (4) функционала невязки. Для этого находим поля $u(r,t)$ и $w(r,t)$ основной и "сопряженной" задач соответственно. Воспользуемся явной разностной схемой для уравнения (1) в области, не содержащей источников

$$c_{ij}\,\frac{u_{i,j,k+1}-2u_{i,j,k}+u_{i,j,k-1}}{\tau^2}-\frac{u_{i+1,j,k}-2u_{i,j,k}+u_{i-1,j,k}}{h^2}-\frac{u_{i,j+1,k}-2u_{i,j,k}+u_{i,j-1,k}}{h^2}=0. \qquad (7)$$

Аналогичная схема используется для уравнения (5).

Использование явных схем очень эффективно с точки зрения распараллеливания. Воспользуемся известным методом распараллеливания по пространству [16]. Как видно из схем, вычисление значения в точке на новом слое по времени зависит только от ближайших соседних точек на текущем и предыдущем слоях. Предлагаемый способ распараллеливания по пространству состоит в том, что общее поле вычислений размером NyAll×NxAll точек разбивается на NyPar×NxPar одинаковых частей, вычисления в которых производятся различными вычислительными ядрами (рис. 2).

Для вычисления точки на каждой границе части необходимо знать значения на соответствующей границе соседней части с предыдущего слоя по времени. Поэтому добавим с каждой стороны части строку или столбец значений соответствующей границы соседней части (рис. 3).

Таким образом, каждый процесс выполняет вычисления для нового временно́го слоя во внутренней области своей выделенной части общего поля независимо от других, после чего происходит обмен граничными значениями с "соседями".

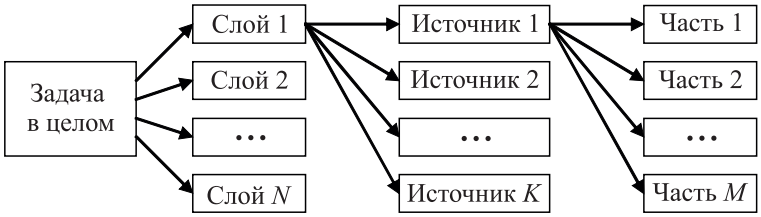

Рис. 4. Схема распараллеливания

В целом, описанные выше подходы в задаче томографии теоретически обеспечивают высокую эффективность и масштабируемость решения поставленной задачи. Первичное распараллеливание производится по слоям трехмерной томографии. Вычисления в каждом таком слое могут быть распараллелены по различным источникам сигнала, которые, в свою очередь, могут быть разбиты на независимые вычислительные части (рис. 4).

При решении реальных задач ультразвуковой томографии размер вычислительной сетки может достигать нескольких тысяч точек по каждой координате, количество источников — нескольких десятков, количество слоев — нескольких десятков. Вычислительные ресурсы (память, вычислительная мощность), необходимые для решения такой задачи в разумные сроки, многократно превышают возможности отдельно взятого вычислительного ядра. Программа с предлагаемой архитектурой хорошо масштабируема и позволяет значительно наращивать ресурсы вычислительного кластера. Например, если требуется произвести расчеты на сетке 2048×2048 точек в 50 слоях при наличии 8 источников излучения, используемый подход к распараллеливанию вычислений позволяет легко масштабировать вычисления, меняя количество вычислительных частей для расчета каждого слоя. При разбиении слоя на 16 частей по координатам X и Y получаем распараллеливание задачи на $50 \times 8 \times 16 \times 16 = 102\,400$ процессов. Однако с тем же успехом можно разбить слой на 8 частей по координатам X и Y. В таком случае мы получаем число задействованных процессоров $50 \times 8 \times 8 \times 8 = 25\,600$ соответственно. Таким образом, предложенная схема распараллеливания позволяет свободно маневрировать, меняя соотношение время расчета/используемые ресурсы для достижения максимальной эффективности даже для очень большого числа процессов.

5. Оптимизация процедуры параллельных расчетов на слое. Как видно из архитектуры программы, распараллеливание по слоям и источникам достаточно очевидно и его высокая эффективность не вызывает сомнения. Поэтому более подробно рассмотрим распараллеливания процесса вычислений в отдельно взятом томографическом слое при фиксированном положении источника. Для нахождения градиента согласно формуле (4) необходимо знание полей $u(r,t)$ и $w(r,t)$ в любой точке слоя r в каждый момент времени из заданного интервала $(0,T)$. Из структуры алгоритма следует, что для этого необходимо провести вычисление $u(r,t)$ в "прямом" времени, а затем, используя полученные граничные значения, провести вычисление $w(r,t)$ в "обратном" времени. На первый взгляд кажется естественным сохранять значения $u(r,t)$ при расчете в "прямом" времени, чтобы далее использовать их при расчете градиента. Однако оценим необходимый объем памяти для сохранения $u(r,t)$. Если отдельный процесс проводит вычисления на прямоугольной области размера $N_x \times N_y$ точек с количеством слоев по времени N_t, то необходимо хранить $N_x \times N_y \times N_t$ значений типа double. Значение N_t обычно раза в три больше NxAll — количества точек вдоль одной оси на всей области расчетов задачи. Кроме того, для областей, прилегающих к границе всего томографического слоя, необходимо хранить экспериментальные данные задачи на границе. С учетом всех накладных расходов это приводило к тому, что для вычислительных узлов с памятью ~ 1 Гб и размером томографического слоя 1000×1000 точек требовалось разбивать весь слой минимум на 7×7 вычислительных частей, для того чтобы данные одной части поместились в память одного вычислительного узла кластера. Таким образом, возникает парадоксальная ситуация, когда задача может решаться только на большом числе процессов, что не всегда приемлемо.

Для преодоления этой проблемы был предложен алгоритм, при котором поле $u(r,t)$ не сохраняется при прямом проходе, а сохраняются только граничные значения. При расчетах в "обратном" времени поля $w(r,t)$ одновременно вычисляем $u(r,t)$ в "обратном" времени, используя сохраненные граничные значения. Такая схема расчетов приводит к увеличению количества вычислений на $\sim$ 30–50%, однако позволяет свободно маневрировать с имеющимися вычислительными узлами.

Возможность корректного вычисления $u(r,t)$ в "обратном" времени по сохраненным граничным значениям легко следует из следующих соображений.

Пусть задан вектор u_{ijk}, где $i = 0,\ldots,N_x$, $j = 0,\ldots,N_y$ и $k = 0,1,2$, причем для всех внутренних точек выполняется соотношение (7). Зададим вектор v_{ijk}, где $i = 0,\ldots,N_x$, $j = 0,\ldots,N_y$ и $k = 0,1,2$; пусть $v_{ijk} = u_{ijk}$ при $k = 1,2$ и при выполнении хотя бы одного из условий $i = 0$, $i = N_x$, $j = 0$ и $j = N_y$. Пусть для всех внутренних точек v_{ijk} выполняется соотношение, аналогичное (7). Тогда $v_{ijk} = u_{ijk}$ во всех точках.

6. Исследование масштабируемости программы на слое и поиск узких мест. Исследуем масштабируемость программы, после чего определим и реализуем способы оптимизации MPI-коммуникаций для улучшения масштабируемости. Основными параметрами этой программы, которые могут влиять на ее масштабируемость, являются следующие параметры (рис. 2.): NX_ALL и NY_ALL — размеры двумерной сетки, на которой происходят вычисления всей задачи в слое; NX_PAR и NY_PAR — число областей разбиения сетки слоя по каждому измерению; NITER_SOU — число различных положений источника в слое. Вычисления в каждой области выполняются в рамках отдельного процесса. Общее число процессов, используемое при выполнении программы в слое, должно быть равно NITER_SOU×NX_PAR×NY_PAR.

Без ограничения общности предположим, что число положений источника далее всегда равно 4: NITER_SOU=4. Все дальнейшие исследования и замеры времени выполняются для 10 итераций итерационного процесса решения обратной задачи в одном слое. Значения NX_ALL и NY_ALL совпадают

и равны 1002. Значения NX_PAR и NY_PAR также совпадают и могут варьироваться от 1 до 11. При таких значениях параметров число процессов может изменяться от 4 ($4 \times 1 \times 1$) до 484 ($4 \times 11 \times 11$). Такие значения параметров позволяют получить результаты, близкие к требуемым на практике.

Для первоначального кода программы было проведено исследование масштабируемости при выбранных значениях входных параметров. Результаты экспериментов приведены в табл. 1.

По результатам, приведенным в этой таблице, видно, что при увеличении числа процессов задача сначала хорошо масштабируется, однако достаточно быстро время не только перестает уменьшаться, но и начинает увеличиваться. Это означает, что накладные расходы на коммуникации начинают расти быстрее, нежели уменьшается объем вычислений, который приходится на каждый процесс. Более подробное исследование MPI-вызовов, используемых в программе, поможет определить, почему возникают такие накладные расходы, а также найти возможные пути оптимизации.

Таблица 1
Время выполнения программы, NX_ALL=NY_ALL=1002

Число процессов	16	36	64	100	144	196	256
Время, с	665	343	267	192	187	201	220

Для исследования времени выполнения MPI-вызовов был использован профилировщик mpiP [17, 18]. Данный программный инструмент позволяет получить общее представление о структуре коммуникаций, в частности: определить процент времени по каждому процессу, который был потрачен на все MPI-вызовы; процент времени, потраченного на выполнение отдельных MPI-вызовов; объем передаваемой информации и т.д.

Результаты, полученные с помощью этого профилировщика, приведены в табл. 2. В строках указано: число процессов, которое было использовано в программе; время выполнения всей программы в секундах; доля времени выполнения всей программы, которое пришлось на обработку MPI-вызовов; доля времени обработки MPI-вызовов, которое пришлось на выполнение только вызовов MPI_Bcast.

Таблица 2
Данные профилировки программы: NX_ALL=NY_ALL=1002

Число процессов	16	64	144	256
Время, с	665	267	187	220
% времени на MPI	10.4	47.6	93	92.7
% MPI_Bcast от MPI	79	97.1	98.1	95.7

Как видно из табл. 2, при достаточно большом числе процессов на обработку MPI-вызовов тратится почти все время выполнения программы; более того, при увеличении числа процессов практически все это время затрачивается на выполнение только операций MPI_Bcast (здесь учитываются все операции MPI_Bcast в программе). Следующей MPI-операцией после MPI_Bcast, которая использует больше всего времени в данной программе, является MPI_Allreduce, однако на нее затрачивается всего несколько процентов, поэтому в первую очередь необходимо провести оптимизацию именно вызовов MPI_Bcast. Анализ исходного кода показывает, что MPI_Bcast используется только в трех небольших внутренних циклах программы. Таким образом, узкое место программы локализовано. Все три цикла устроены практически одинаково:

```
for (int i = 0; i < prNumber; i++) {
     MPI_Bcast(sides[i].u, Tx * sizeof(double), MPI_BYTE, i, MPI_COMM_WORLD);
     <несколько аналогичных коллективных операций MPI_Bcast>
}
```

Здесь `prNumber` — общее число процессов и `Tx =(NX_ALL-2)/NX_PAR+2`. Каждый процесс рассылает некоторый набор данных по всем остальным процессам. Отличие трех циклов заключается только в том, что в одном из них рассылается чуть больший объем данных, однако это отличие несущественно. Поэтому в дальнейшем мы будем рассматривать только один цикл, и все описанные преобразования будут аналогичным образом применяться к другим циклам.

Кроме того, с помощью mpiP удалось определить, что вычисления и основные коммуникации между процессами распределены достаточно равномерно: время, затрачиваемое на выполнение MPI_Bcast и на сами вычисления, практически одинаково у всех процессов. Это облегчает оптимизацию, поскольку в противном случае отдельное внимание необходимо было бы сосредоточить на оптимизацию нагрузок между процессами для того, чтобы каждый процесс выполнял одинаковый объем работ.

7. Оптимизация узких мест программы. После определения узкого места в программе необходимо попытаться определить, можно ли и каким образом модифицировать этот фрагмент программы для того, чтобы уменьшить время его выполнения.

Как было сказано ранее, в программе происходят вычисления для нескольких различных положений

источника, при этом на каждое положение выделяется одинаковое число процессов, которые предназначены для вычислений только в рамках этого положения. В нашем исследовании использовано 4 различных положения источника. Для вычислений по каждому источнику выделяется по `prNumber/4` процессов. В тех циклах, которые являются узким местом, в первоначальной версии программы для простоты кода проводилась рассылка граничных данных области по всем остальным процессам, включая и процессы с другим положением источника, с помощью коммуникатора MPI_COMM_WORLD. Однако алгоритм программы устроен таким образом, что вычисления для разных источников независимы, и пересылать данные между ними не нужно.

Это приводит к первой оптимизации: нет необходимости делать рассылку по всем процессам, нужно передавать данные только процессам с тем же положением источника. Для реализации этого следует создать 4 коммуникатора, по одному на каждое положение источника, и делать рассылку только по соответствующему коммуникатору. В данном случае достаточно с помощью команды MPI_Comm_split() разбить коммуникатор MPI_COMM_WORLD, после чего заменить его в циклах на новый созданный коммуникатор. Такая оптимизация реализуется просто, но позволяет сократить объем передаваемых данных в 4 раза.

Далее, из структуры алгоритма ясно, что в циклах, которые являются узким местом программы, на самом деле необходима передача данных только процессам-соседям по сетке, в то время как в действительности происходит передача всем процессам с тем же положением источника. Соответственно, для дальнейшей второй оптимизации программы необходимо заменить коллективную рассылку данных на рассылку только по соседям. В этом случае использование вместо MPI_COMM_WORLD коммуникаторов меньшего размера, что применялось при первой оптимизации, становится уже не актуальным.

Рассмотренная оптимизация немного сложнее и требует замену вызовов MPI_Bcast на коммуникации типа “точка–точка”. Кроме того, необходимо учитывать, что число соседей у процессов может быть разное. После внесенных изменений оптимизированный цикл выглядит следующим образом:

```
for (int i = 0; i <= 8; i++)
{ if (neighs[i] != -1) { // then send and resv
     MPI_Irecv(sides[neighs[i]].u,Tx,MPI_DOUBLE,neighs[i],0,MPI_COMM_WORLD,reqs[0]);
     <несколько аналогичных операций MPI_Irecv>
     MPI_Send(sides[nProc].u,Tx,MPI_DOUBLE,neighs[i],0,MPI_COMM_WORLD);
     <несколько аналогичных коллективных операций MPI_Send>

     int ret = MPI_Waitall(5,reqs,stats);
     }
}
```

Массив neighs содержит ранги процессов-соседей, с которыми текущему процессу необходимо обменяться данными. Каждый процесс инициирует асинхронный прием сообщений от всех соседей, а затем в синхронном режиме отправляет свои данные соседям. После получения и отправки всех сообщений одному процессу-соседу происходит ожидание завершения асинхронных вызовов с помощью вызова MPI_Waitall. В данном случае использование асинхронного приема сообщений используется для того, чтобы избегать тупиковых ситуаций [19].

Таблица 3
Сравнение первого и второго вариантов оптимизации, NX_ALL=NY_ALL=1002

Число процессов	144	196	256	324	400	484
Время, с, opt_v1	118	114	120	134	148	168
Время, с, opt_v2	82	67	62	58	59	60

Использование такой оптимизации позволяет значительно ускорить работу программы за счет существенного уменьшения объема передаваемых данных. Если при использовании первой оптимизации объем передаваемых данных при увеличении числа процессов продолжал расти с той же скоростью, что и в изначальной, неоптимизированной версии программы, то в данном случае объем передаваемых данных увеличивается гораздо медленнее. Сравнение первой и второй (opt_v2) оптимизаций программы приведено в табл. 3.

Более подробные данные по выполнению MPI-вызовов приведены в табл. 4.

Анализируя результаты в табл. 4, можно увидеть, что при увеличении числа процессов объем передаваемых данных в варианте opt_v2 действительно увеличивается медленнее, чем в варианте opt_v1: на 196 процессах объем данных меньше в 7.7 раз, а на 484 процессах — уже в 17.5 раз.

Дальнейшее исследование показало, что использование синхронной передачи сообщений с помощью MPI_Send не является оптимальным вариантом, поскольку на синхронизацию в MPI_Send (т.е. на ожи-

Таблица 4
Данные по MPI, первый и второй варианты оптимизации, NX_ALL=NY_ALL=1002

Число процессов	196		484	
Версия программы	opt_v1	opt_v2	opt_v1	opt_v2
Время, с	114	67	168	60
% времени на MPI	77.15	63.8	93.5	88.8
Объем передаваемых данных, байт	4.26×10^{11}	5.54×10^{10}	1.67×10^{12}	9.57×10^{10}

дание начала передачи) тратится достаточно много времени. Это происходит из-за того, что приходится ожидать вызов MPI_Irecv на другом процессе. Более того, после обмена сообщениями с каждым соседом из-за MPI_Waitall приходится ожидать завершения всех приемов сообщений от этого соседа, что также требует значительного времени. Поэтому было решено провести третью оптимизацию (opt_v3): использовать только асинхронный прием и передачу, а также вынести MPI_Waitall из тела цикла. В таком случае будет использован только один вызов MPI_Waitall после запуска всех MPI_Isend и MPI_Irecv, для ожидания завершения всех операций. Измененный фрагмент выглядит следующим образом:

```
int MPI_comms = 10;
int neighs_num = 9;
int neigh_count = 0;
for (int i = 0; i < neighs_num; i++) {
         if (neighs[i] != -1) { // then send and resv
         MPI_Irecv(sides[neighs[i]].u,Tx,MPI_DOUBLE,neighs[i],0,MPI_COMM_WORLD,
              &reqs[MPI_comms*neigh_count+0]);
         <несколько аналогичных операций MPI_Irecv>
         MPI_Isend(sides[nProc].u,Tx,MPI_DOUBLE,neighs[i],0,MPI_COMM_WORLD,
              &reqs[MPI_comms*neigh_count+5]);
         <несколько аналогичных операций MPI_Isend>
         neigh_count++;
         }
}
int ret = MPI_Waitall(neigh_count*MPI_comms,reqs,stats)
```

Здесь `MPI_comms` — число коммуникаций между двумя процессами (всего 5 вызовов MPI_Isend и 5 вызовов MPI_Irecv) и `neigh_count` — число соседей, с которыми уже были инициированы все операции приема и передачи. Эти переменные нужны для корректной работы вызова MPI_Waitall. Использование данного приема позволяет получить выигрыш во времени, сопоставимый с выигрышем от предыдущей оптимизации (табл. 5).

Таблица 5
Сравнение второго и третьего вариантов оптимизации, NX_ALL=NY_ALL=1002

Число процессов	144	196	256	324	400	484
Время, с, opt_v2	82	67	62	58	59	60
Время, с, opt_v3	67	48	43	33	30	28

Таблица 6
Данные по MPI, второй и третий варианты оптимизации, NX_ALL=NY_ALL=1002

Число процессов	196		484	
Версия программы	opt_v2	opt_v3	opt_v2	opt_v3
Время, с	67	48	60	28
% времени на MPI	63.8	42.5	88.8	79.05
Объем передаваемых данных, байт	5.54×10^{10}	5.54×10^{10}	9.57×10^{10}	9.57×10^{10}

Таким образом, только изменение порядка и способа выполнения коммуникаций привело к ускорению работы программы в среднем в 1.5–2 раза. В табл. 6 приведена информация о том, как изменились данные по выполнению MPI-вызовов.

Очевидно, что в этом случае объем передаваемых данных остается тем же самым. Однако доля времени выполнения всей программы, которое пришлось на обработку MPI_вызовов, значительно снижается, что и приводит к уменьшению общего времени выполнения.

В заключение приведем сравнение всех рассмотренных вариантов программы (рис. 5).

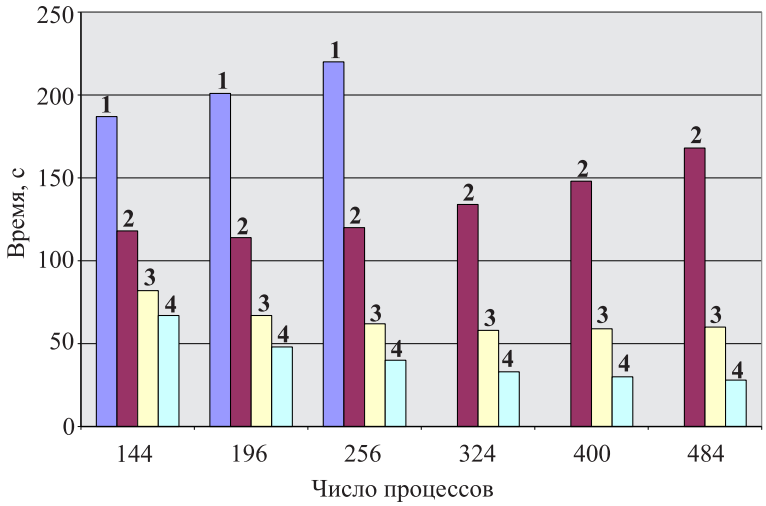

Рис. 5. Сравнение исходной и оптимизированных версий программы, NX_ALL=NY_ALL=1002: 1) basic, 2) opt_v1, 3) opt_v2, 4) opt_v3

Для варианта basic подсчитаны не все значения, поскольку время счета при увеличении числа процессов начинает существенно расти, однако увидеть общую динамику это не мешает. Видно, что в исходном варианте и после первой оптимизации при увеличении числа процессов время выполнения почти сразу начинает увеличиваться. В следующих вариантах оптимизации этого уже не наблюдается: время продолжает существенно уменьшаться и при большом числе процессов. Все осуществленные оптимизации являются достаточно простыми и могут быть легко реализованы, однако, несмотря на простоту, они позволили ускорить работу программы на порядок, а также обеспечить гораздо более высокую масштабируемость.

8. Выводы. Предложена архитектура программы, позволяющая выделить большое количество независимых параллельных процессов в общем потоке задачи.

Предлагаемый подход позволяет легко распараллеливать вычисления и свободно маневрировать, меняя соотношение “(ускорение расчета)/(количество вычислительных узлов)”, для достижения максимальной эффективности.

Проведены оптимизации исходного кода, которые позволили ускорить работу программы на порядок, а также обеспечить гораздо более высокую масштабируемость.

Как показали результаты многочисленных расчетов на вычислительных системах МГУ “Чебышев” и “Ломоносов”, структура программы достаточно эффективно приспособлена к работе на этих кластерных системах.

СПИСОК ЛИТЕРАТУРЫ

1. *Jiřík R., Peterlík I., Ruiter N., Fousek J., Dapp R., Zapf M., Jan J.* Sound-speed image reconstruction in sparse-aperture 3D ultrasound transmission tomography // IEEE Trans. on Ultrasonics, Ferroelectrics, and Frequency Control. 2012. **59**, N 2. 254–264.
2. *Li C., Duric N., Littrup P., Huang L.* In vivo breast soundspeed imaging with ultrasound tomography // Ultrasound Med. Biol. 2009. **35**, N 10. 1615–1628.
3. *Backushinsky A., Goncharsky A., Romanov S., Seatzu S.* On the identification of velocity in seismics and in acoustic sounding. Pubblicazioni dell’istituto di analisa globale e applicazioni. Serie “Problemi non ben posti ed inversi”. Issue 71. Firenze, 1994.
4. *Бакушинский А.Б., Козлов А.И., Кокурин М.Ю.* Об одной обратной задаче для трехмерного волнового уравнения // Журн. вычисл. матем. и матем. физики. 2003. **43**, № 8. 1201–1209.
5. *Гончарский А.В., Овчинников С.Л., Романов С.Ю.* Об одной задаче волновой диагностики // Вестн. Моск. ун-та. Сер. 15. Вычисл. матем. и киберн. 2010. № 1. 7–13.
6. *Kokurin M.Y.* Stable iteratively regularized gradient method for nonlinear irregular equations under large noise // Inverse Problems. 2006. **22**, № 1. 197–208.

7. *Головина С.Г., Романов С.Ю., Степанов В.В.* Об одной обратной задаче сейсмики // Вестн. Моск. ун-та. Сер. 15. Вычисл. матем. и киберн. 1994. № 4. 16–21.
8. *Chavent G.* Deux resultats sur le probleme inverse dans les equations aux derivees partielles du deuxieme ordre en tet sur l'unicite de la solution du probleme inverse de la diffusion // C. R. Acad. Sci. Paris. 1970. N 270. 25–28.
9. *Natterer F., Wubbeling F.* A propagation-backpropagation method for ultrasound tomography // Inverse Problems. 1995. **11**, № 6. 1225–1232.
10. *Beilina L., Klibanov M.V.* Approximate global convergence and adaptivity for coefficient inverse problems. New York: Springer, 2012.
11. *Гончарский А.В., Романов С.Ю.* О двух подходах к решению коэффициентных обратных задач для волновых уравнений // Журн. вычисл. матем. и матем. физики. 2012. **52**, № 2. 1–7.
12. *Овчинников С.Л., Романов С.Ю.* Организация параллельных вычислений при решении обратной задачи волновой диагностики // Вычислительные методы и программирование. 2008. **9**, № 1. 338–345.
13. *Гончарский А.В., Романов С.Ю.* Об одной задаче ультразвуковой томографии // Вычислительные методы и программирование. 2011. **12**, № 1. 317–320.
14. *Гончарский А.В., Романов С.Ю.* Суперкомпьютерные технологии в разработке методов решения обратных задач в УЗИ-томографии // Вычислительные методы и программирование. 2012. **13**, № 1. 235–238.
15. Описание суперкомпьютера "Ломоносов" (http://parallel.ru/cluster/lomonosov.html).
16. *Foster I.* Designing and building parallel programs: concepts and tools for parallel software engineering. Reading: Addison Wesley, 1995.
17. Описание средства профилировки mpiP (http://mpip.sourceforge.net/).
18. *Vetter J.S., McCracken M.O.* Statistical scalability analysis of communication operations in distributed applications // Proc. of the 8th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPOPP-2001). New York: ACM, 2001. 123–132.
19. *Антонов А.С.* Параллельное программирование с использованием технологии MPI. М.: Изд-во Моск. ун-та, 2004.

Поступила в редакцию
10.04.2012