ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ НАУКИ
ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ
И МАТЕМАТИЧЕСКОЙ ГЕОФИЗИКИ
СИБИРСКОГО ОТДЕЛЕНИЯ РОССИЙСКОЙ АКАДЕМИИ НАУК

ТРУДЫ
конференции
молодых ученых

Под редакцией А. С. Родионова

НОВОСИБИРСК, 2015

УДК 519.6, 551.5, 550.8
ББК 22.19, 26.23, 26.217

Труды конференции молодых ученых. Новосибирск: ИВМиМГ СО РАН, 2015. 136 с.

В данном сборнике опубликованы полные тексты докладов конференции молодых ученых ИВМиМГ СО РАН, прошедшей в апреле 2014 г. По тематике наиболее представлены доклады по вычислительной математике, математическим методам исследования динамики атмосферы и океана, математической геофизике и информатике.

Издание осуществлено при частичной финансовой поддержке Совета научной молодежи ННЦ СО РАН.

Proceedings of Young Scientist Conference. — Novosibirsk: Inst. of Comp. Math. and Math. Geoph. Publ., 2015. 136 p.

The bulletin contains full texts of reports and papers of Inst. of Comp. Math. and Math. Geoph. Young scientists conference, which have taken place on April 2013. Mostly papers are devoted to computation mathematics, methods of the atmosphere–ocean circulation modelling, mathematical geophysics and informatics.

ТРУДЫ КОНФЕРЕНЦИИ МОЛОДЫХ УЧЕНЫХ
Новосибирск, 2014 С. 3–18

Различные подходы распараллеливания прямых методов решения систем линейных уравнений с разреженной матрицей

Р. В. Андерс, А. В. Андерс, А. А. Калинкин

УДК 519.612.2

Описаны способы распараллеливания прямого метода решения систем линейных уравнений с разреженной матрицей, основанного на разложении матрицы в произведение нижнетреугольной, диагональной и верхнетреуголной матриц, на компьютерах с распределенной памятью. Предложены варианты эффективного распределения рабочей нагрузки между процессами. Для распараллеливания между вычислительными узлами используется функциональность MPI. Внутри каждого узла распараллеливание осуществляется с помощью OpenMP. Представлены результаты численных экспериментов, показывающие зависимость времени работы от числа вычислительных узлов, произведено сравнение различных подходов к балансировке рабочей нагрузки между MPI процессами, выполнено сравнение с аналогичным пакетом MUMPS.

Different approaches of parallelization of direct sparse solver based on LU decomposition for solving system of linear equations on distributed memory machines are considered. We proposed several techniques for effective workload balancing between MPI processes. To achieve better performance we used hybrid level of parallelization – MPI + OpenMP. Finally we presented the numerical experiments that demonstrate dependency between computational time and number of MPI processes. We also made comparison of different approaches for workload balancing between MPI processes. Charts with performance comparison against MUMPS were provided.

Ключевые слова: прямой метод, системы линейных уравнений, MPI.

Keywords: Direct sparse solver, linear equations, MPI.

Введение. Одной из актуальных задач алгебры разреженных матриц является решение систем линейных алгебраических уравне-

ний $Ax = b$ с разреженной матрицей. Разработка алгоритмов решения таких задач и их высокопроизводительная программная реализация, ориентированная на современные вычислительные архитектуры, представляет большой практический интерес. В данной работе речь пойдет о разработке эффективного прямого метода решения разреженных систем линейных алгебраических уравнений, основанного на разложении исходной матрицы в произведение верхнетреугольной и нижнетреугольной матриц в общем случае. В настоящее время существуют методы, ориентированные на различные режимы работы: последовательный, параллельный для систем с общей и распределенной памятью. Некоторые из таких методов уже имеют высокопроизводительные реализации. Среди известных реализаций прямых методов подобного типа можно выделить MKL PARDISO [4], SuperLU, MUMPS и др. Предложенная реализация разработана для матриц общего вида и ориентирована на компьютеры с распределенной памятью. Для коммуникаций между вычислительными узлами используется функциональность MPI, внутри одного узла распараллеливание ведется с помощью OpenMP директив. Современные процессоры имеют до 28 вычислительных ядер с общей памятью, поэтому необходимо уметь эффективно их использовать. В данном случае использована методика вложенного параллелизма. Для эффективной работы алгоритма на компьютерах с распределенной памятью большую роль играет правильная балансировка рабочей нагрузки между вычислительными узлами. Одна из целей данной работы — поиск параметров, с помощью которых можно эффективно распределить данные между вычислительными узлами. В работе предложены четыре таких параметра. Также большое значение имеет способ взаимодействия между MPI процессами. В предлагаемой программе использована техника с “потоком-почтальоном” на каждом MPI процессе. Этот поток отвечает только за пересылку данных и не участвует в вычислениях. В работе представлены результаты численных экспериментов, показывающие преимущества различных подходов к распараллеливанию. Произведено сравнение различных параметров для балансировки рабочей нагрузки между MPI процессами. Представлены графики, показывающие масштабируемость алгоритма по времени счета, а так же сравнение производительности с аналогичным пакетом MUMPS [2].

1. Постановка задачи. Пусть дана система линейных алгебраических уравнений вида

$$Ax = b. \tag{1}$$

Здесь A — произвольная (вещественно- или комплекснозначная, симметричная или несимметричная) разреженная квадратная матрица. Прямой метод решения такой системы заключается в разложении исходной матрицы в произведение нижнетреугольной, диагональной и верхнетреугольной матриц:

$$A = L \times D \times U. \tag{2}$$

Здесь L — нижнетреугольная матрица, U — верхнетреугольная матрица, D — диагональная матрица. Для случая симметричных (эрмитовых) матриц $U = L^*$. Далее это разложение будет называться факторизацией, а полученные матрицы — факторами. После получения разложения решение исходной системы (1) сводится к последовательному решению треугольных систем следующего вида:

$$Lz = b, \quad Dy = z, \quad L^\top x = y. \tag{3}$$

Так как матрица является разреженной, то для ее хранения в памяти был использован CSR формат — сжатый разреженный строчный формат хранения матриц. Отметим, что в симметричном случае для экономии памяти необходимо хранить только верхний треугольник матрицы.

Отметим, что при факторизации разреженной матрицы полученные факторы могут быть плотно заполнены, вследствие чего существенно возрастает количество потребляемой памяти, а так же теряется возможность использовать исходную разреженную структуру для распараллеливания алгоритма. Этого можно избежать с помощью подходящей перестановки строк и столбцов исходной матрицы и переходу к системе следующего вида:

$$PAP^\top x = b \tag{4}$$

где P — матрица перестановок.

Таким образом, при численном решении разреженной системы (1) с использованием предложенного метода можно выделить

Рис. 1. Общая схема

следующие этапы: вычисление матрицы перестановки и переход к системе (4); символическое разложение (построение портрета матрицы, выделение памяти для хранения ненулевых элементов); численное разложение (вычисление значений матрицы факторов и размещение их в выделенной памяти); обратный ход (решение треугольных систем уравнений).

В работе представлена параллельная реализация данного алгоритма на компьютерах с распределенной памятью, а также предложены различные способы улучшения производительности этого подхода. Наиболее трудоемкой частью предложенного алгоритма является эффективная реализация численной фазы решения.

2. Программная реализация. Общая схема предложенного алгоритма показана на рис. 1. Рассмотрим более подробно каждый шаг предложенного метода.

1. *Перестановка и символьная факторизация.* Перестановка исходной матрицы нужна для уменьшения числа ненулевых элементов в факторах, а также для применения более эффективной, в смысле распараллеливания вычислений, факторизации, а именно мультифронтального подхода. Данная перестановка в предлагаемой реализации вычисляется с помощью пакета программ METIS [3].

Пусть исходная матрица имеет структуру, показанную на рис. 2, где серым цветом обозначены ненулевые элементы, а белым — нулевые. Видно, что для такой матрицы независимо можно начать вычислять только строки 1, 2, 4. Используя алгоритм вложенных сечений, эту матрицу можно привести к виду, показанному на рис. 3.

На рис. 3 светло-серым цветом обозначены элементы, сформировавшиеся в процессе факторизации. Для переставленной матрицы независимо можно считать строки 1, 2, 4, 5, а затем также независимо строки 3, 6. Таким образом, получается граф, или дерево

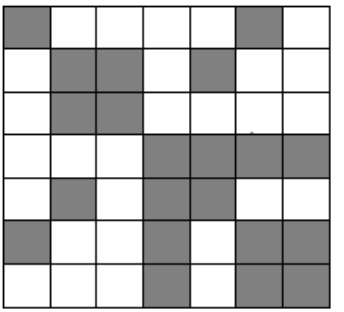


Рис. 2. Портрет исходной матрицы

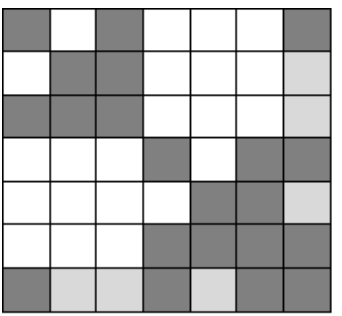


Рис. 3. Портрет переставленной матрицы

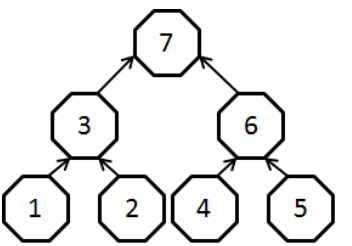


Рис. 4. Дерево зависимостей

зависимостей, согласно которому можно проводить факторизацию. Для данного примера дерево имеет вид, показанный на рис. 4.

Каждый узел дерева зависимостей есть набор столбцов исходной матрицы. Следовательно, каждому узлу можно сопоставить вес (параметры, с помощью которых можно вычислить вес узла, рассмотрены в п. 5). Отметим также, что при вычислении перестановки учитывается структура столбцов исходной матрицы таким образом, чтобы столбцы с похожей структурой располагались рядом и объединялись в группы (далее — панели; размеры панелей варьируются от 1 до 120 для оптимального использования функциональности BLAS Level 3 [4]).

В процессе символьной факторизации вычисляется структура факторов, т. е. шаблон расположения ненулевых элементов. Также в этой фазе происходит подготовка некоторых внутренних структур для дальнейшей численной факторизации.

2. *Численная факторизация.* На этапе численной факторизации проводится вычисление матриц L, D, U из представления (2). Для простоты положим, что матрица A симметрична и положительно определена. Пусть узел дерева принадлежит только одному MPI процессу, тогда его факторизацию можно вычислить последовательно для каждой панели, причем если какие-то панели независимы, то их вычисление происходит параллельно с помощью OpenMP.

Если узел дерева принадлежит более чем одному процессу, то его вычисление происходит в два этапа. Рассмотрим сначала вычисление одного элемента матрицы L. Предположим, что все элементы матрицы L с индексами, меньшими j, подсчитаны, тогда $L_{j,j}$ вычисляется следующим образом:

$$\sqrt{L_{j,j}} = A_{j,j} - \sum_{k=1}^{j-1} L_{k,j}^2, \tag{5}$$

а вычисление внедиагонального элемента имеет вид:

$$L_{i,j} = (A_{i,j} - \sum_{k=1}^{j-1} L_{k,j} \times L_{i,k}) \times L_{i,i}^{-1}, \tag{6}$$

причем в формулах (5), (6) элементы могут быть квадратными матрицами. В этом случае взятие корня и обращение выполняются с помощью функциональности LAPACK.

Операцию умножения под знаком сумм в (5) и (6) и вычитание элементов $L_{k,j}$ и $L_{i,k}$ назовем “update”. Таким образом, перед тем как приступить к вычислению узла, который принадлежит нескольким MPI процессам, каждый MPI независимо выполняет “update” своими элементами. Последующая факторизация выполняется с учетом того, что каждый узел распределен между несколькими MPI (рис. 5).

Когда 1-й MPI процесс завершает факторизацию первой панели, он рассылает ее всем остальным процессам и продолжает считать следующие панели. После получения этой панели другие процессы начинают выполнять функцию “update” элементами полученной панели, и так далее для всех панелей. Как только у какого-то из процессов появляется панель, которая больше ни от чего не зависит, он

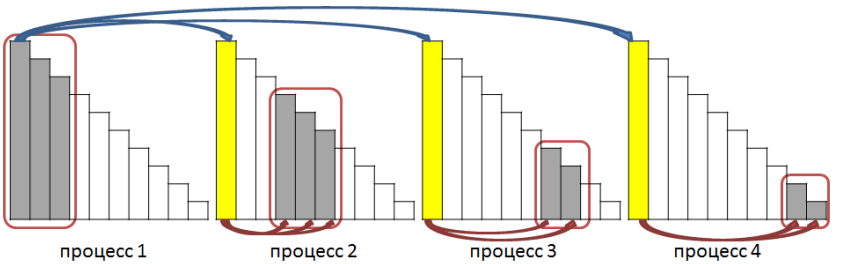


Рис. 5. Факторизация узла четырьмя процессами

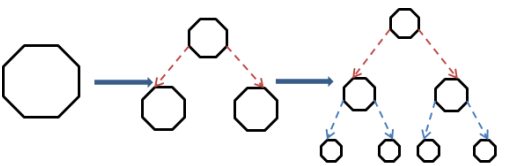


Рис. 6. Разбиение узла на новое поддерево

сразу факторизует и рассылает ее остальным процессам. Таким же образом происходит обработка узлов на следующем уровне дерева.

3. *Решение треугольных систем.* Предположим, необходимо решить систему с нижнетреугольной матрицей. Важно, что эта матрица распределена между несколькими MPI процессами. Аналогично факторизации, решение треугольной системы можно начать с нескольких мест независимо. Для параллельного решения такой системы будет также использовано дерево зависимости. Рассмотрим случай, когда узел принадлежит одному процессу. Тогда для эффективного использования всех OpenMP потоков разделяем узел на несколько (если это возможно):

В этом случае можно начать решение независимо с четырех мест, что позволяет существенно экономить на синхронизациях между OpenMP потоками. Отметим, что разбиение узла на поддерево не всегда возможно и зависит от структуры матрицы.

Пусть теперь узел распределен между несколькими MPI процессами. Тогда решение будет происходить, как показано на рис. 7.

Здесь каждый процесс имеет по одной панели $a_{i,j}$, имеющей вид квадратной матрицы. Операцию умножения $a_{i,j}$ на компоненты x_i также назовем “update”. Первый процесс, выполнив “update” четырех

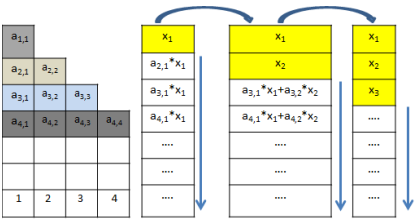


Рис. 7. Решение на распределенном узле

первых компонент вектора x, отправляет их остальным процессам и продолжает подсчет своих панелей. Второй процесс, получив данные от первого, может посчитать полностью вторую компоненту вектора решения x_2 и выполнить "update" компонентов, принадлежащих следующим процессам, и так далее для всех компонент вектора решения. Решение систем с верхнетреугольной матрицей выполняется аналогично.

3. Использование почтальона. Рассмотрим более подробно случай, когда узел дерева зависимости распределен между несколькими MPI процессами. Очевидно, что процесс, панелями которого выполняется операция "update", должен синхронизироваться со всеми остальными процессами на каждой пересылке. Следовательно, общее время вычислений будет складываться из самых длительных интервалов вычисления каждой панели. Данная проблема решается путем выделения на каждом процессе по одному "потоку-почтальону", который будет участвовать только в пересылках и не будет участвовать в вычислениях. В этом случае факторизация будет проходить следующим образом:

```
1: если поток-почтальон то
2:    открыть посылку, чтобы получить (или отправить) панель
3:    для всех панелей текущего узла
4:       повторять
5:          если панель подсчитана то
6:             отправить (или получить) панель
7:          иначе
8:             ждать пока панель не будет подсчитана
9:       пока все панели не будут подсчитаны
10: иначе    // поток не почтальон
```

```
11:  для всех панелей текущего узла
12:    повторять
13:      если панель получена или уже принадлежит текущему
         процессу то
14:        выполнить update всех своих панелей
15:      иначе
16:        ждать пока панель не будет получена
17:    пока все панели не будут подсчитаны
```

Такой подход позволяет непрерывно считать панели принадлежащие одному процессу, не прерываясь на их пересылку, что существенно сокращает время ожиданий на MPI синхронизациях.

Отметим, что операция "update" реализована с использованием OpenMP распараллеливания. При этом, если потоков мало, а "update" от каждой панели занимает достаточно много времени, то использование "почтальона" нецелесообразно, поскольку в этом случае панели обрабатываются долго, что увеличивает время ожидания для "почтальона". Для ускорения всего алгоритма эффективнее было бы использовать этот поток в вычислениях.

Заметим, что при обработке узла, который распределен между несколькими процессами, большой вклад в общее время вносят пересылки панелей, длительности пересылок зависит не только от размера посылки, но и от количества процессов, участвующих в коммуникации. Поэтому конкретный процесс, завершивший подсчет и отправку всех свои панелей на текущем узле, удаляется из локального коммуникатора, и дальнейшие коммуникации происходят без него. Такой подход особо эффективен на верхних узлах дерева, когда количество MPI процессов может доходить до 1000.

4. Использование вложенного параллелизма. Рассмотрим еще один способ оптимизации предлагаемого алгоритма. Напомним, что локально на каждом процессе факторизация панелей выполняется с использованием OpenMP распараллеливания. Причем возможность эффективно использовать OpenMP потоки прямо пропорциональна количеству панелей, которые можно считать независимо. В этом случае можно представить узел как новое дерево, как было показано на рис. 6. Однако такой подход не всегда возможен и не позволяет эффективно использовать большое количество потоков. Напомним, что матрицы L и U из представления (2) имеют блочную структуру, а все внутренние вычисления происходят с ис-

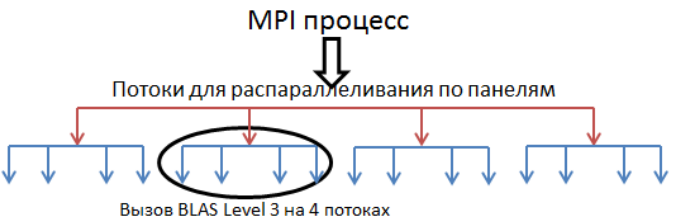


Рис. 8. Nested parallelization

пользованием функциональности BLAS Level 3. Поэтому в случае, когда нет возможности распараллеливания по панелям, предлагается использовать технику вложенного распараллеливания (nested parallelization) (рис. 8).

Такой подход дает преимущество по времени, если обрабатываемые таким способом панели достаточно большие. Численные эксперименты показывают, что такой подход эффективен на небольшом количестве матриц и в среднем позволяет достичь ускорения всей факторизации на 10 %.

5. Балансировка рабочей нагрузки между MPI процессами. Большое значение при работе на компьютерах с распределенной памятью имеет эффективная балансировка рабочей нагрузки между MPI процессами. Пусть переставленная матрица и дерево зависимости имеют вид, показанный на рис. 3, 4. Самый простой и наименее эффективный способ — распределение вычислительных узлов равномерно по дереву. Для четырех MPI процессов равномерное распределение будет выглядеть следующим образом: первый MPI процесс будет обрабатывать столбец 1, второй MPI процесс — столбец 2, третий — столбец 4, четвертый — столбец 5. После того как первый и второй MPI процессы завершат вычисление своих блоков, они могут независимо от третьего и четвертого MPI процессов начать вычисление столбца 3. Аналогично третий и четвертый MPI процессы перейдут к вычислению столбца 6 и т. д. Видно, что имеет место равномерное распределение MPI процессов по дереву. На практике зачастую данные распределены по дереву зависимости неравномерно, поэтому важную роль играет правильное распределение вычислительных ресурсов между узлами дерева.

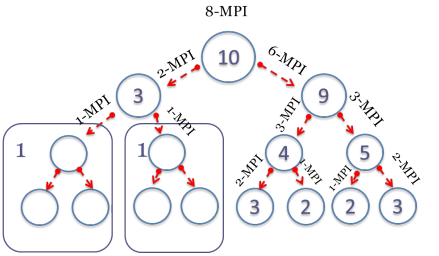


Рис. 9. Дерево зависимости для 8 MPI процессов

На рис. 9 показано дерево зависимости. Числом внутри узла обозначен его вес, т. е. величина, соответствующая объему работы по вычислению данного узла дерева. Видно, что имеет место неравномерное распределение данных: правое поддерево имеет узлы с большими весами, в то время как в левом поддереве узлы имеют маленькие веса. Если в данной ситуации применить обычное равномерное распределение вычислительных узлов, то получим ситуацию, когда одни MPI процессы имеют очень большой объем работы, а другие очень маленький или совсем не имеют, что приведет к простою части вычислительных ресурсов. Поэтому необходимо распределять MPI процессы с учетом весов узлов. Стоит отметить, что если вычисления ведутся на кластере с неравноценными вычислительными узлами, состоящем из процессоров с разной частотой или различным количеством ядер, то это также можно учитывать при распределении рабочей нагрузки между MPI процессами.

Опишем более подробно распределение MPI процессов по дереву зависимости. Обход дерева начинается с самого верхнего уровня, соответственно узел последнего уровня будут обрабатывать все MPI процессы, далее вычисляется вес каждого поддерева, исходя из некоторых параметров, который будут рассмотрены далее. После того как веса посчитаны, можно распределить MPI процессы пропорционально весам каждого из поддеревьев. В данной ситуации для вычисления левой ветки назначаются два MPI процесса, а для правой — шесть. Дальнейшее распределение MPI процессов ведется аналогичным образом. Если для обработки одной из веток остался только один MPI процесс, то вся эта ветка рассматривается как один

узел дерева. На рис. 9 видно что для вычисления узлов первого и второго уровней в левом поддереве осталось всего по одному MPI процессу, поэтому они объединены в один узел и будут обрабатываться как одно целое.

Существуют несколько параметров, с помощью которых можно вести балансировку рабочей нагрузки между MPI процессами. Первый способ распределения рабочей нагрузки — это вычисление количества ненулевых элементов в каждом узле дерева. Информация о заполненности каждого узла доступна после символьной фазы. Далее MPI процессы распределяются по дереву зависимости таким образом, чтобы на каждый MPI процесс попадало равное количество ненулевых элементов. Второй и третий способы являются обобщениями первого, а именно: второй способ заключается в том, чтобы рассматривать не число ненулевых элементов, а число столбцов в каждом узле дерева, т. е. распределять вычислительные ресурсы таким образом, чтобы на каждый MPI процесс попадало равное количество столбцов матрицы. В третьем подходе предлагается учитывать количество супернодов в каждом узле дерева. Все три подхода основаны на общей идее, а именно, распределять вычислительные ресурсы в соответствии с объемом данных на каждом узле дерева. На практике эффективность применения каждого из подходов зависит от исходной матрицы и числа MPI процессов. Четвертый подход несколько отличается от первых трех: он в большей степени учитывает вычислительную работу, а не объем данных. По результатам символьной части можно определить количество обращений к функциональности GEMM (умножение плотных матриц) при вычислении каждого узла дерева. Поэтому распределение MPI процессов будет производиться таким образом, чтобы каждый процесс имел равное количество вычислительной работы в терминах умножения плотных матриц. Каждый из предложенных параметров на различных задачах может давать лучшее распределение вычислительных ресурсов, но численные эксперименты показывают, что в большинстве случаев наиболее эффективным подходом является четвертый. Это связанно с тем, что он учитывает не только объем данных, но и структуру факторизованной матрицы, что немало важно при работе с разреженными матрицами.

6. Результаты численных экспериментов. Результаты численных экспериментов представлены в виде графиков, показываю-

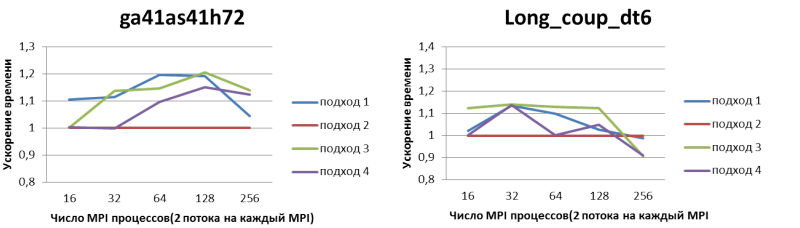


Рис. 10. Сравнение различных параметров для балансировки

щих сравнение всех предложенных подходов к балансировке рабочей нагрузки, ускорение времени счета при увеличении числа вычислительных узлов, а также сравнение с аналогичным пакетом MUMPS. Все вычисления производились на кластере Endevour, состоящем из процессоров IvyTown E5-2697 v2 @ 2.70GHz, 24 cores, 64Gb RAM. В качестве тестовых данных использовались матрицы из Florida Sparse Matrix Collection, размеры матриц варьируются от 3×10^5 до 10^7.

На рис. 10 показано ускорение численной фазы факторизации в зависимости от различных способов балансировки рабочей нагрузки. Ускорение вычислений рассчитывается относительно равномерного распределения вычислительных узлов. Нетрудно заметить, что для разных матриц и различного числа MPI процессов не существует единого наилучшего подхода. В разных случаях более эффективным оказывается тот или иной подход, однако что в большинстве случаев наилучшее распределение вычислительных ресурсов дает подход, учитывающий количество обращений к функциональности GEMM. Также можно заметить, что при правильном выборе подхода для оптимальной балансировки рабочей нагрузки можно получить ускорение фазы численной факторизации до 20 % по сравнению с равномерным распределением вычислительных ресурсов.

На рис. 11 показано уменьшение времени счета при увеличении числа вычислительных узлов. Видно, что в среднем алгоритм имеет положительную шкалируемость при увеличении ядер более чем 1000.

На рис. 12 показано сравнение предлагаемого алгоритма с аналогичным пакетом MUMPS [3] для различных конфигураций (MPI× threads: 16×12, 16×24, 32×12, 32×24, 64×12, 64×24). Тесты проводились на 20 произвольных матрицах. Видно, что почти во всех слу-

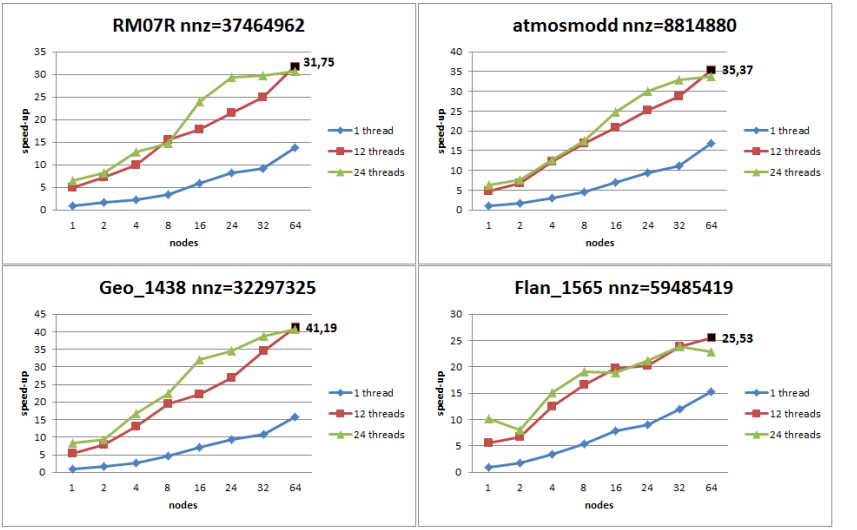


Рис. 11. Ускорение в зависимости от числа узлов.
Каждый узел содержит 24 ядра

чаях обсуждаемая реализация показывает лучшую производительность, поскольку учитывает полное время решения системы: перестановка — факторизация — решение треугольных систем.

Заключение. В представленной работе описана программная реализация прямого метода решения систем линейных уравнений, основанного на разложении матрицы в произведение верхне- и нижнетреугольных матриц, на компьютерах с распределенными данными. Описаны различные способы улучшения производительности программы: методы уменьшения заполненности факторизованной матрицы путем перестановки строк в исходной матрице (метод вложенных сечений); способы достижения эффективного параллелизма программы на распределенных данных на основе дерева зависимости; методика вложенного параллелизма. Предложены различные параметры распределения вычислительной нагрузки между

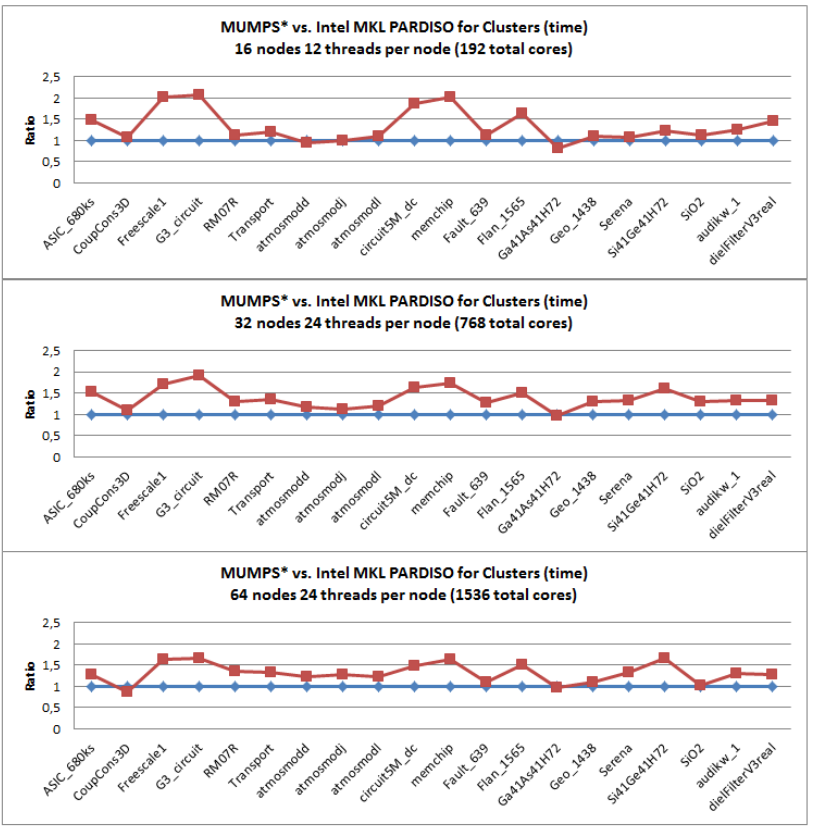


Рис. 12. S-curve Pardiso for Cluster vs. MUMPS

MPI процессами, а именно: число ненулевых элементов в факторизованной матрице в каждом узле дерева зависимости, количество столбцов или супернодов в каждом узле дерева зависимости, а также число обращений к функциональности GEMM (умножение плотных матриц) при вычислении каждого узла дерева зависимости. Были представлены результаты численных экспериментов на различ-

ных тестовых матрицах из Florida Sparse Matrix Collection, размеры матриц менялись от 3×10^5 до 10^7, число MPI процессов изменялось от 16 до 256. По результатам этих экспериментов установлено, что во многих случаях наиболее эффективным оказывается подход, учитывающий количество обращений к функциональности GEMM. Это связано с тем, что данный подход использует не только объем данных, но и структуру самой факторизованной матрицы. Как показывают результаты численных экспериментов, только при правильном выборе балансировки можно получить ускорение численной факторизации до 20 % по сравнению с обычным равномерным распределением. Представлены результаты численных экспериментов, показывающие ускорение времени счета при увеличении числа вычислительных узлов, а также произведено сравнение с аналогичным пакетом MUMPS.

Научный руководитель — А. А. Калинкин.

Список литературы

[1] Davis T. A., Hu Y. The University of Florida Sparse Matrix Collection // ACM Trans. on Math. Software. 2011. V. 38, iss. 1. P. 1:1–1:25. [Electron. resource]. www.cise.ufl.edu/research/sparse/matrices.

[2] [Electron. resource]. http://mumps.enseeiht.fr/.

[3] [Electron. resource]. http://glaros.dtc.umn.edu/gkhome/views/metis.

[4] [Electron. resource]. https://software.intel.com/en-us/intel-mkl.

Андерс Роман Викторович — аспирант Новосибирского государственного университета; e-mail: andersroman@mail.ru;
Андерс Антон Викторович — аспирант Новосибирского государственного университета; e-mail: ontoha90@mail.ru

ТРУДЫ КОНФЕРЕНЦИИ МОЛОДЫХ УЧЕНЫХ
Новосибирск, 2014 С. 19–32

Программный инструментарий HPC Community Cloud для организации взаимодействия пользователей и внешних программных систем с вычислительными центрами

С. 33–45

С. А. Вайцель

УДК 004.75

Разработан программный инструментарий HPC Community Cloud (HPC2C) в составе сервера и веб-приложения, которые унифицируют взаимодействие программных систем и пользователей с различными высокопроизводительными вычислительными системами (ВВС). Программные системы через программный интерфейс сервера (API), а пользователи — через веб-приложение, реализованное на основе API, могут обмениваться файлами с ВВС, ставить вычислительные задачи на ВВС и контролировать ход их исполнения. Разработана технология реализации интерактивных интерфейсов прикладных программ и встраивания этих интерфейсов в веб-приложение HPC2C для организации систематизированного накопления, разделения и повторного использования прикладных программ в среде HPC2C.

A software tool HPC Community Cloud (HPC2C) composed of a server and a web-application has been developed to unify interaction of program systems and users with different high performance computing systems (HPCS). Program systems access HPC2C through the server's programming interface and users work with the HPC2C through the web-application. Programming systems and users can exchange files with the HPCSs, submit computing jobs to the HPCSs and control execution of the jobs through the respective interfaces. A technology has been developed to implement interactive user interfaces for the application programs and integrate these interfaces into the web-application in order to organize systematic accumulation, sharing and reuse of application programs in the HPC2C environment.

Ключевые слова: распределенные вычислительные системы, высокопроизводительные вычисления, суперкомпьютерные технологии, облачные вычисления, параллельное программирование,

инструменты разработки программ, численное моделирование, сетевые сервисы.

Keywords: distributed computing, high performance computing, supercomputing technologies, cloud computing, parallel programming, software development tools, numerical simulation, network services.

Введение. В работе представлена модель программной системы для унификации взаимодействия программных систем и пользователей с суперкомпьютерами на основе единого формата описания вычислительных задач, централизованного инструмента взаимодействия с различными вычислительными системами, позволяющего пользователям подключать к системе собственные вычислительные ресурсы и работать над вычислительными проектами совместно с другими пользователями. Модель реализована в виде программного инструментария, состоящего из сервера управления, реализующего программный интерфейс для внешних систем, и пользовательского интерфейса на основе веб-приложения.

Работа выполняется в рамках проекта HPC Community Cloud (HPC2C) [1].

1. Программный инструментарий HPC Community Cloud. HPC Community Cloud (HPC2C) предоставляет пользователям интерфейс в виде веб-приложения, посредством которого можно получить доступ к высокопроизводительным вычислительным ресурсам в соответствии с разрешенным уровнем доступа, поставить вычислительное задание в очередь, посмотреть статусы выполняющихся задач, визуализировать результаты завершенных задач или загрузить их на компьютер пользователя. Внешним программным системам доступ к ВВС в автоматическом режиме предоставляется с помощью программного интерфейса (API). HPC2C реализует платформу для накопления и интегрирования в единый интерфейс содержимого, создаваемого сторонними разработчиками: средств конструирования программ, визуализации данных, численного моделирования, а также анализа данных.

Для дальнейшего изложения требуется определение используемых в проекте HPC2C терминов [1].

Проект (project) — программное решение, предоставляемое пользователю или разрабатываемое им. Проектами являются приложения и встраиваемые программные решения.

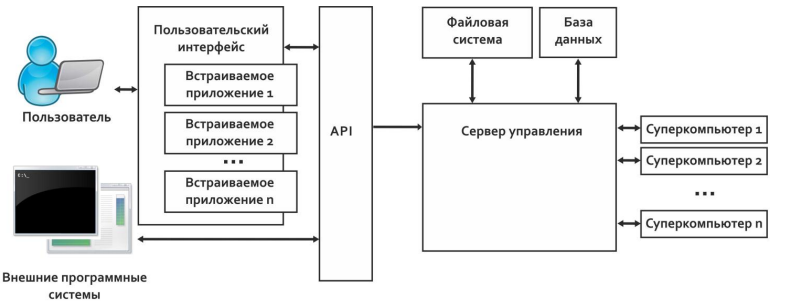
Приложение (application) — разрабатываемая пользователем посредством интегрированной в веб-приложение среды разработки программа, представленная исходным кодом. Исходный код можно редактировать и компилировать. После компиляции программы на основе полученного приложения может быть создан эксперимент.

Встраиваемое программное решение (embedded software solution) — разрабатываемое сторонним разработчиком или пользователем программное решение, внедряемое в интерфейс HPC2C. Внедряемое в том смысле, что каждое встраиваемое программное решение имеет доступный из интерфейса HPC2C интерфейс, позволяющий формулировать задачи и запускать их на доступном вычислительном оборудовании. Встраиваемые программные решения подразделяются на фреймворки и модели.

Фреймворк (framework) — встраиваемое программное решение, представляющее собой программный код, который реализует некоторую схему вычислений или метод вычислений и при этом может быть настроен и дополнен для решения конкретной задачи из некоторого относительно широкого класса задач. Например, это может быть параллельная реализация схемы вычислений в методе частиц в ячейках [2], которая может быть настроена и дополнена соответствующим кодом для того, чтобы решать задачу гравитационного взаимодействия частиц.

Модель (model) — встраиваемое программное решение, пользовательский интерфейс которого позволяет задать входные данные, запустить приложение, в некотором смысле наблюдать процесс вычислений и анализировать выходные данные. Например, в качестве модели в систему HPC2C встраивается программный комплекс клеточно-автоматного моделирования газопорошковых потоков [3].

Эксперимент (experiment) — объект, характеризующий отдельный запуск сформированной задачи на конкретных вычислительных ресурсах. Характеризуется приложением, запросом на выполнение задания и состоянием выполнения (“ожидает”, “выполняется”, “выполнено”, “редактируется”, “не выполнено”). Запрос на выполнение задания описывает требования и условия выполнения приложения: количество вычислительных узлов и количество ядер на узле, время работы с вычислительными ресурсами, операции, которые необходимо выполнить до или после запуска задачи и проч.



Архитектура HPC Community Cloud

Личный кабинет (dashboard) — это набор инструментов пользовательского интерфейса, которые позволяют пользователю взаимодействовать в системе HPC2C с проектами и экспериментами как принадлежащими пользователю, так и с теми, к которым ему предоставили доступ другие пользователи. Взаимодействие означает добавление, удаление, запуск экспериментов, просмотр статуса выполнения эксперимента и т. д.

Архитектура инструментария HPC2C представлена на рисунке. Его основу составляет сервер управления, предоставляющий программный интерфейс (API). Пользовательский интерфейс представлен в виде внешней программной системы — веб-приложения, взаимодействующего с сервером управления посредством данного API HPC2C. В систему HPC2C могут встраиваться программные пакеты, создаваемые пользователями и сторонними разработчиками. Доступ к ним предоставляется посредством пользовательского интерфейса, их взаимодействие с сервером управления организуется с помощью API HPC2C.

2. Сервер управления. Сервер управления имеет модульную структуру и содержит модули для взаимодействия с базой данных и файловым хранилищем, а также модуль, обеспечивающий взаимодействие с подключенными ВВС.

База данных хранит личные данные пользователей, информацию о подключенных ВВС, информацию о проектах, экспериментах и документах.

В файловом хранилище располагаются репозитории пользователей, имеющие следующую структуру:

/apps — файлы приложений пользователя;

/appstorage — каталог, предоставляемый программным решениям для размещения рабочих данных;

/experiments — файлы экспериментов пользователя;

/frameworks — исходные и исполняемые файлы фреймворков;

/models — исходные и исполняемые файлы моделей.

В зависимости от типа проекта его каталог имеет определенную структуру. Так, при создании приложения его каталог содержит следующие файлы и каталоги:

/bin — каталог для исполняемых файлов, собранных в различных конфигурациях;

/doc — каталог для документации к проекту;

/lib — каталог для библиотек, используемых при сборке;

/rc — каталог, содержащий входные данные приложения; при создании эксперимента на основе приложения содержимое каталога копируется в каталог эксперимента.

/src — каталог файлов с исходным кодом приложения.

/Makefile — файл,,содержащий конфигурацию сборки приложения.

Для возможности исполнения экспериментов на различном вычислительном оборудовании сервер управления HPC2C содержит модуль взаимодействия с ВВС. Он также имеет модульную структуру, в которой под модулем следует понимать набор методов, предназначенных для взаимодействия с определенным классом вычислительных систем, имеющих одни и те же особенности интерфейса и организации вычислений.

Унификация работы с вычислительными кластерами достигается посредством промежуточного формата хранения параметров запуска задачи. Эти параметры хранятся в базе данных HPC2C. Задача модуля взаимодействия с ВВС — транслировать эти параметры запуска задачи в формат, используемый в ВВС, на которой будет происходить запуск задачи.

На данный момент модуль взаимодействия с ВВС реализован как набор функций для взаимодействия с кластером NKS-30T Сибирского суперкомпьютерного центра. Также в нем присутствует метод генерации скриптов запуска задач на кластере NKS-30T. Для описания вычислительного задания генерируется скрипт вида:

```
#!/bin/bash/
#PBS -V
#PBS -q G7_q
```

```
#PBS -r n
#PBS -l select=%mpiNodes%:ncpus=%mpiPPN%
        :mpiprocs=%mpiPPN%,walltime=%walltime%
#PBS -N h2c\%id\%
#PBS -j oe
date
cd $PBS_O_WORKDIR
MPD_CON_EXT=\'date\'
mpirun -r ssh -genv I_MPI_FABRICS shm:ofa
        -n %mpiNodes%*%mpiPPN% ./a.out
date
```

Здесь *%mpiNodes%* — число запрашиваемых узлов; *%mpiPPN%* — число процессоров на узле и число потоков на нем; *%walltime%* — предполагаемое время выполнения задачи, по истечении которого прекращается выполнение задачи; *%id%* — уникальный идентификатор запускаемого эксперимента.

3. Программный интерфейс. Программный интерфейс — это набор команд, позволяющий внешним программным системам взаимодействовать с сервером управления HPC2C. Программный интерфейс реализует всю функциональность программного инструментария HPC2C. Все команды разбиты на четыре класса: работа с данными пользователя, файловой системой, проектами и экспериментами.

Класс работы с данными пользователя включает команды, позволяющие пользователю зарегистрироваться, авторизоваться, удалить профиль, а также изменить данные о себе (логин, пароль, имя и фамилию, e-mail). Также к данным пользователя относится информация об используемых им ВВС.

Изначально, чтобы получить доступ к системе HPC2C, необходимо зарегистрироваться. Команда “создать учетную запись” в случае успешной регистрации возвращает идентификационный токен. Токен является подтверждением права вызывающей стороны на выполнение этой команды. Через определенный период времени он дезактивируется и должен быть запрошен новый. Токен передается при вызове всех остальных команд программного интерфейса, все остальные команды за исключением команды “авторизовать пользователя” имеют вид

имя_команды <токен> <параметры_команды>.

Класс команд для работы с файловой системой HPC2C обеспечивает пользователю доступ к файлам и каталогам. Программный интерфейс предоставляет команды для создания, изменения, перемещения, дублирования и удаления файлов и каталогов.

В системе HPC2C постановка задачи на исполнение происходит в два этапа: работа с проектом и работа с экспериментом. Сначала пользователь создает проект, редактирует его исходные файлы, собирает, получает исполняемый файл. Затем на основе созданного проекта пользователь может создать эксперимент, который позволит запустить исполняемый файл на определенной вычислительной системе. Этот подход повышает удобство работы с заданиями, предоставляет возможность манипулировать конкретными запусками задач, многократно их запускать с различными данными.

Класс команд для работы с проектами включает команды получения списков проектов пользователя, создания, изменения, дублирования и удаления проектов, а также команду сборки проекта на ВВС.

Команда "собрать проект" предназначена для сборки проектов типа "приложение", т. е. разрабатываемых пользователем программ. При вызове данной команды сервер управления производит следующие операции:

1) анализирует конфигурацию сборки;

2) отсылает исходные данные приложения на указанную в конфигурации ВВС;

3) производит сборку проекта в соответствии с конфигурацией;

4) передает сформированные исполняемые файлы в каталог /bin соответствующего приложения.

Как было указано, эксперимент является отдельным запуском задачи. Он характеризуется исполняемой программой и входными файлами.

Класс команд для работы с экспериментами включается в себя команды получения списка экспериментов пользователя, создания, изменения, дублирования и удаления экспериментов, а также команду запуска эксперимента на ВВС.

Команда "запустить эксперимент" передает на вычислительный кластер конфигурацию запуска задачи, исполняемый файл и файлы входных данных и ставит переданную задачу в очередь. Команда "получить состояние эксперимента" запрашивает у вычислительно-

го кластера информацию о состоянии выполнения эксперимента и соответственно обновляет информацию об эксперименте внутри системы. Эксперимент может находиться в одном из шести состояний:

1) редактироваться,
2) находиться в очереди на выполнение,
3) выполняться,
4) быть приостановленным,
5) быть успешно завершенным,
6) быть завершенным с ошибкой.

Рассмотрим пример использования программного интерфейса для создания задачи, формирования на ее основе эксперимента и его запуска. Допустим, пользователю необходимо разработать и собрать программу для решения СЛАУ, а затем протестировать ее. Последовательность команд программного интерфейса для решения этой задачи имеет вид:

1) регистрация/авторизация пользователя в системе,

```
авторизовать_пользователя 'user' '12345'
```

2) создание проекта

```
создать_проект <токен> 'application'
    'SLAE_solver1'
```

3) загрузка исходных файлов программы, входных файлов в систему

```
создать_файл <токен>
    '/apps/SLAE_solver1/src/main.cpp'
    <содержимое_файла>
создать_файл <токен>
    '/apps/SLAE_solver1/rc/SLAE.txt'
    <содержимое_файла>
```

4) редактирование конфигурации сборки проекта

```
изменить_содержимое_файла <токен>
    '/apps/SLAE_solver1/Makefile'
    <измененное_содержимое_файла>
```

5) сборка проекта

```
собрать_проект <токен> 'SLAE_solver1'
```

6) создание эксперимента

```
создать_эксперимент <токен>
    'SLAE_solver1_TEST1' 'application'
    'SLAE_solver1' 'bin/a.out' 'EDITED'
    'nks-g6.sscc.ru' 4 4
```

7) постановка эксперимента в очередь

```
запустить_эксперимент <токен>
    'SLAE_solver1_TEST1'
```

8) получение состояния эксперимента

```
получить_состояние_эксперимента <токен>
    'SLAE_solver1_TEST1'
```

9) получение результата работы программы

```
выгрузить_файл_из_хранилища <токен>
    'experiments/SLAE_solver1_TEST1/RESULT.txt'
```

Программный интерфейс реализован в виде HTTP-сервиса, принимающего запросы определенного вида. Для облегчения разработки внешних приложений, работающих на стороне браузера, а также приложений на платформе Node.js, создана библиотека функций на JavaScript, реализующих соответствующие запросы.

Документация по программному интерфейсу доступна по адресу *http://hpccloud.sscc.ru/help*.

4. Пользовательский интерфейс. Интерфейс пользователя — внешняя программная система, взаимодействующая с сервером управления HPC Community Cloud посредством программного интерфейса (API HPC Community Cloud). Пользователю предоставляется возможность авторизоваться в системе, после чего он получает доступ к личному кабинету. В личном кабинете пользователю становится доступен весь функционал сервиса:

1) Просмотр списка проектов и сведений о доступных проектах (название проекта, описание проекта, автор, права доступа), а также выполнение операций над ними (создание проектов, изменение, дублирование, удаление проекта);

2) Просмотр списка экспериментов пользователя, добавление нового эксперимента, просмотр сведений о доступных экспериментах (название, описание, автор, название проекта, с помощью которого создан эксперимент, статус выполнения эксперимента), а также выполнение операций над экспериментами (создание, изменение, дублирование, удаление);

3) Изменение личных данных пользователя (ФИО, организация, e-mail, номер телефона);

4) Просмотр списка доступных ВВС, добавление записей, удаление, изменение записей.

Одна из компонент пользовательского интерфейса — интегрированная среда разработки (IDE). При работе пользователя с интегрированной средой разработки ему доступны следующие возможности:

1) Загрузка файлов исходного кода / начальных данных;

2) Редактирование файлов посредством встроенного текстового редактора;

3) Просмотр дерева каталогов, выполнение действий над файлами и папками (создание, перемещение, переименование, удаление, дублирование);

4) Изменение названия открытого проекта или эксперимента;

5) Изменение конфигурации запуска задания у эксперимента;

6) Изменение конфигурации сборки у приложения;

7) Постановка задания в очередь (запуск эксперимента);

8) Сборка приложения на ВВС.

Каждая модель или фреймворк имеет собственные интерфейсы, созданные разработчиком данной модели или фреймворка.

В целях обеспечения повсеместного доступа к данным с любых устройств, имеющих доступ в интернет, пользовательский интерфейс реализован в виде веб-приложения.

Интерфейс состоит из личного кабинета и интегрированной среды разработки. Личный кабинет HPC Community Cloud содержит списки доступных проектов, проектов пользователя и экспериментов. В интегрированной среде разработки отображаются список файлов открытого проекта или эксперимента, редактор файлов с под-

светкой синтаксиса языков C/C++ и другим функционалом (поиск, замена, сворачивание блоков кода).

В веб-приложении реализован механизм хранения сессий. При регистрации пользователя веб-сервер получает токен пользователя — уникальный идентификатор доступа к серверу управления HPC2C. Веб-сервер генерирует к этому токену случайный идентификатор сессии, связь между идентификаторами хранится в виде двух ассоциативных массивов — словарей, в одном из которых в качестве ключа выступает токен, а в качестве значения — идентификатор сессии, в другом — идентификатор сессии и токен, соответственно. При вызове команд программного интерфейса с помощью библиотеки функций сформированный запрос передается веб-серверу, который получает по идентификатору сессии токен и дополняет им запрос, а затем отправляет его серверу управления.

Документация по пользовательскому интерфейсу доступна по адресу *http://hpccloud.sscc.ru/help*.

5. Встраивание программных решений. HPC Community Cloud предоставляет среду для накопления знаний в виде фреймворков и моделей. Разрабатывать встроенные программные решения могут не только разработчики HPC Community Cloud, но и другие пользователями. Для достижения этой цели они могут использовать интегрированную среду разработки для редактирования файлов.

Назначение пользовательских интерфейсов моделей и фреймворков состоит в задании параметров для последующего исполнения приложений на вычислительных системах, запуске задачи, а также в отображении состояния вычислений и предоставлении возможности анализа результатов расчетов.

Для взаимодействия с HPC2C встраиваемые программные решения используют команды программного интерфейса HPC2C.

Чтобы обеспечить защиту системы и пользователей от произвольных действий сторонних программ, а также обеспечить должный уровень сервиса для пользователей, были приняты следующие требования к программным решениям, встраиваемым в единую среду HPC2C [1]:

1) никакие части кода программных решений не исполняются в рамках сервера управления, а только на подключенных к HPC2C ВВС под учетной записью пользователя;

2) интерфейс приложений реализуется пользователем на основе технологий, используемых в пользовательском интерфейсе (например, если пользовательский интерфейс представлен веб-приложением, то используются HTML5, CSS и JavaScript), общение с сервером выполняется через системный программный интерфейс, серверная часть программного решения пользователями не разрабатывается.

Таким образом, разработчиком программного решения создается пользовательский интерфейс, а также исполняемые файлы и файлы входных данных. Все файлы программного решения хранятся в файловом хранилище HPC2C. Файлы пользовательского интерфейса программного решения хранятся в каталоге `имя_пользователя/frameworks(models)/имя_программного_решения/GUI`.

Поскольку интерфейс встраиваемых систем предоставляется пользователю посредством веб-приложения, он должен быть реализован с использованием технологий HTML5 и JavaScript, и каталог `имя_пользователя/frameworks(models)/имя_программного_решения/GUI` должен содержать каталоги `/HTML`, `/CSS`, `/JS`. В данных каталогах должны располагаться файлы пользовательского интерфейса встраиваемого приложения. Файл `/GUI/HTML/index.html` должен реализовывать точку входа в приложение, загружать в браузер пользователя все необходимые скрипты для работы приложения. Встраиваемая программная система имеет свой репозиторий в каталоге `имя_пользователя/appstorage`, где она размещает вспомогательные файлы, необходимые для работы, а при создании экспериментов она размещает файлы экспериментов в каталоге `/имя_пользователя/experiments/имя_программного_решения`.

В настоящее время сторонними разработчиками разрабатываются и встраиваются такие программные решения, как программный комплекс визуального конструирования программ Visual Seismics [4] и фреймворк, реализующий параллельный метод частиц в ячейках [2].

В систему HPC Community Cloud сторонним разработчиком встроен программный комплекс клеточно-автоматного моделирования газопорошковых потоков [3]. Это программный комплекс, реализующий клеточно-автоматную модель FHP-GP [5]. Программный комплекс состоит из трех модулей: конструктора граничных условий, симулятора потока и модуля осреднения и визуализации, —

и позволяет проводить вычислительные эксперименты с моделью FHP-GP, задавая различные параметры и начальные условия. Конструктор граничных условий позволяет задать начальные и граничные условия задачи, вносить в расчетную область физические объекты в виде графических примитивов, таких как точка, линия, прямоугольник, блок, треугольник, круг. Симулятор имеет набор параметров, позволяющих управлять ходом моделирования. Визуализатор позволяет строить графические изображения полей давления газа, концентрации порошка, скорости потока, линии тока.

Заключение. В ходе выполнения работы были получены следующие результаты:

1) разработана архитектура программного комплекса;

2) реализован прототип системы HPC Community Cloud в составе: сервер управления; веб-приложение; библиотека, реализующая программный интерфейс (API HPC2C); модуль сервера управления, организующий выполнение и сборку программ на кластере NKS-G6 Сибирского суперкомпьютерного Центра;

3) разработаны и реализованы механизмы встраивания в HPC2C готовых программных решений, разрабатываемых сторонними разработчиками;

4) встроен программный комплекс клеточно-автоматного моделирования газопорошковых потоков.

В качестве основных направлений для дальнейшей работы были определены следующие:

1) организация распределенных вычислений на основе технологии NumGrid;

2) расширение API для внешних систем (архивация файлов, проектов и экспериментов, получение данных о кластерах и т. д.);

3) внедрение в HPC2C программных решений сторонних разработчиков (PIC-фреймворк, визуальный конструктор программ)

Научный руководитель — мл. науч. сотр. ИВМиМГ СО РАН М. А. Городничев.

Список литературы

[1] Городничев М. А., Малышкин В. Э., Медведев Ю. Г. “HPC Community cloud: эффективная организация работы научно-образовательных суперкомпьютерных центров” // Науч. вестн. НГТУ. 2013. № 3(52). С. 91–96.

[2] Волков А. С., Морский Д. В. MPI-фреймворк для программ, реализующих метод частиц-в-ячейках // Материалы 52-й Междунар. науч. студ. конф. “Студент и научно-технический прогресс”: Информ. технол. / Новосибирск: Изд-во НГУ, 2014. С. 5.

[3] Медведев Ю. Г. Программный комплекс клеточно-автоматного моделирования газопорошковых потоков // Труды Междунар. науч. конф. “Параллельные вычислительные технологии” (ПаВТ’2012), Новосибирск, 26–30 марта 2012 г.). Челябинск: Издательский центр ЮУрГУ, 2012. С. 732.

[4] Сарычев В. Г., Купчишин А. Б.. Разработка программного комплекса для конструирования программ обработки данных на высокопроизводительных вычислительных системах // 7-я Сиб. конф. по параллельным и высокопроизводительным вычислениям / Под ред. проф. А.В. Старченко. Томск: Изд-во ТГУ, 2014. С. 55–64.

[5] Медведев Ю.Г. Клеточно-автоматная модель формирования порошковой струи // Прикладная дискретная математика. 2009. № 3. С. 50–58.

Вайцель Сергей Александрович — магистрант Новосибирского государственного технического университета;
e-mail: seralwei@gmail.com

ТРУДЫ КОНФЕРЕНЦИИ МОЛОДЫХ УЧЕНЫХ
Новосибирск, 2014 С. 33–45

Априорные оценки для схем расщепления в смешанном методе конечных элементов

К. В. Воронин

Новосибирский государственный университет

УДК 519.63

Рассматриваются априорные оценки для потоковых схем расщепления в смешанном методе конечных элементов для задачи теплопереноса. Аппроксимация по пространству осуществляется с помощью конечных элементов Равьяра — Тома наименьшей степени на прямоугольных сетках. Для аппроксимации по времени используется новый подход, позволяющий строить схемы расщепления для вектора теплового потока. Основной идеей является применение известных скалярных схем расщепления для дивергенции теплового потока. Полученные априорные оценки накладывают требования дополнительной гладкости на начальный тепловой поток. Рассматриваются различные случаи, в которых необходимой гладкостью тепловой поток не обладает. Численно исследовано поведение решения и возможные способы решения возникающих проблем со сходимостью в этих случаях.

A priori estimates for flux splitting schemes in mixed FEM for heat transfer problems are considered. Space approximation is implemented using Raviart-Thomas finite elements of lowest order on rectangular meshes. For time approximation a new approach is used in order to obtain splitting schemes for the heat flux. The main idea is to construct vector splitting schemes based on scalar splitting schemes for heat flux divergence. The obtained a priori estimates impose additional smoothness requirements on the initial heat flux. In several situations the initial heat flux is not smooth enough. Solution behaviour and possible ways of overcoming arising convergence issues are studied numerically.

Ключевые слова: смешанный метод конечных элементов, схемы расщепления, теплоперенос, априорные оценки, устойчивость.

Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (номер гранта 13-01-00019) и Интеграционного проекта СО РАН № 76.

Keywords: mixed finite element method, splitting schemes, heat transfer, a priori estimates, stability.

Введение. В данной работе представлены априорные оценки для потоковых схем расщепления в смешанном методе конечных элементов применительно к задаче теплопереноса. Рассматривается слабая смешанная постановка в виде системы уравнений первого порядка в терминах “температура - тепловой поток”. Аппроксимация по пространству осуществляется с помощью конечных элементов Равьяра — Тома наименьшей степени [2] на прямоугольных сетках. Для аппроксимации по времени используется новый подход [1], позволяющий строить схемы расщепления для вектора теплового потока [3, 4]. Основной идеей является применение известных скалярных схем расщепления [5] для дивергенции теплового потока. На самом деле, любая потоковая схема расщепления может быть получена с помощью такого подхода (в некоторых случаях для этого требуется ввести дополнительные дробные шаги).

Оказывается, связь между схемами для дивергенции и для теплового потока может быть использована при исследовании аппроксимации и устойчивости потоковых схем расщепления. Во-первых, так как температура в рассматриваемой постановке полностью определяется через дивергенцию теплового потока, то процесс вычисления температуры характеризуется теми же свойствами, которыми обладает порождающая схема для дивергенции. Во-вторых, можно показать, что порядок аппроксимации для теплового потока совпадает с порядком аппроксимации порождающей схемы для дивергенции в неперестановочном случае. Более интересным является вопрос об устойчивости потоковых схем расщепления, основную трудность при этом представляет исследование глобальной устойчивости для теплового потока.

В работе представлены априорные оценки для потоковой схемы расщепления в двумерном случае, построенной на основе схемы переменных направлений для дивергенции теплового потока. Ключевой идеей получения априорных оценок является разложение теплового потока на две ортогональные компоненты — дискретно-соленоидальную и дискретно-потенциальную составляющие. Полученные априорные оценки накладывают требования наличия дополнительной гладкости на начальный тепловой поток, который вычис-

ляется через начальную температуру через дискретный аналог закона Фурье.

Как показывают приведенные в работе результаты численных экспериментов, в некоторых случаях начальный тепловой поток не обладает требуемой гладкостью, что приводит к условной аппроксимации, и, как следствие, только условной сходимости. В качестве решения возникающей проблемы предлагается использовать сглаживающие процедуры либо для начальной температуры либо для начального теплового потока.

1. Постановка задачи. Рассмотрим следующую систему дифференциальных уравнений первого порядка для температуры и теплового потока, которая описывает процесс теплопереноса в области $\mathbf{x} \in \Omega \subset R^n, n = 2, 3$ при $t \in [0, t_f]$:

$$\begin{cases} c_p \rho \dfrac{\partial T}{\partial t} + \nabla^T \mathbf{w} = f \\ \mathbf{w} = -\lambda \nabla T \end{cases} \quad \mathbf{x} \in \Omega, t \in [0, t_f].$$

Здесь T и $\mathbf{w}$ — неизвестные функции температуры и теплового потока, c_p, ρ и λ — коэффициенты теплоемкости, плотности и теплопроводности; правая часть f в первом уравнении отвечает за наличие источников тепла в Ω. На границе Ω поставлены краевые условия Дирихле или Неймана. Кроме того, задано начальное распределение температуры $T = T^0(\mathbf{x})$ для $t = 0$. Можно заметить, что первое уравнение — это закон сохранения энергии, а второе — определяющее соотношение (закон Фурье), задающее связь между температурой и тепловым потоком.

Стандартным способом можно выполнить переход к слабой смешанной постановке:

$$\begin{cases} \int\limits_\Omega c_p \rho \frac{\partial T}{\partial t} \chi + \int\limits_\Omega div\, \mathbf{w} \chi = \int\limits_\Omega f \chi, \forall \chi \in L_2(\Omega) \\ \int\limits_\Omega \frac{1}{\lambda} \mathbf{w} \cdot \mathbf{u} = \int\limits_\Omega T \nabla \mathbf{u} - \int\limits_{\partial\Omega} T \mathbf{u} \cdot \mathbf{n}, \quad \forall \mathbf{u} \in \mathbf{H}_{div}(\Omega) \end{cases},$$

где неизвестные функции температуры T и теплового потока $\mathbf{w}$ являются уже элементами пространств $C^1(0, t_f; L_2(\Omega))$ и $C(0, t_f; \mathbf{H}_{div})$, соответственно.

Далее, пусть область Ω покрыта прямоугольной сеткой. Аппроксимация по пространству осуществляется с помощью смешанного

метода конечных элементов, используются конечные элементы Равьяра — Тома наименьшей степени (для $\mathbf{H}_{div}$) и кусочно-постоянные элементы (для L_2). Таким образом, тепловой поток аппроксимируется кусочно-линейными функциями —"крышечками", а температура — кусочно-постоянными функциями. После аппроксимации по пространству возникает следующая система обыкновенных дифференциальных уравнений:

$$\begin{cases} M\dfrac{dT_h}{dt} + B^T\mathbf{w}_h = f_h \\ A\mathbf{w}_h = BT_h + g_h \end{cases},$$

где M — диагональная матрица масс для температуры; $\mathbf{A}$ — трехдиагональная матрица масс для теплового потока (симметричная, положительно определенная); $\mathbf{B}$ и $\mathbf{B}^T$ — матрицы дискретных операторов градиента и дивергенции, соответственно. В правой части второго уравнения вектор g_h соответствует неоднородным краевым условиям. Вектор f_h аппроксимирует правую часть f дифференциальной задачи.

Теперь можно записать схему с весами в следующем виде:

$$\begin{cases} (\mathbf{A} + \alpha\tau\mathbf{G})\dfrac{\mathbf{w}^{n+1} - \mathbf{w}^n}{\tau} + \mathbf{G}\mathbf{w}^n = \mathbf{B}M^{-1}(\alpha F^{n+1} + (1-\alpha)F^n) \\ M\dfrac{T^{n+1} - T^n}{\tau} + \mathbf{B}^T(\alpha\mathbf{w}^{n+1} + (1-\alpha)\mathbf{w}^n) = \alpha F^{n+1} + (1-\alpha)F^n \end{cases}$$

где $\mathbf{G} = \mathbf{B}M^{-1}\mathbf{B}^T$ аппроксимирует вторые производные по пространственным переменным. Для получения первого уравнения на тепловой поток необходимо продифференцировать на сеточном уровне закон Фурье и воспользоваться сеточным законом сохранения энергии. Начальный тепловой поток при этом вычисляется как решения уравнения $\mathbf{A}\mathbf{w}^0 = \mathbf{B}T^0$. Следует заметить, что температура вычисляется на очередном шаге по времени через дивергенцию теплового потока и потому может рассматриваться в данной постановке как вспомогательная переменная.

Основной трудностью является необходимость факторизации оператора $\mathbf{G}$. Среди известных способов можно выделить следующие:

1. Факторизация оператора в явном виде [3], например, попеременно-треугольная или факторизация типа SOR для векторного уравнения на тепловой поток.

2. Схемы на основе алгоритма Удзавы (Arbogast и др. [7]).

3. Новый подход из [1], [4], основанный на применении скалярных схем расщепления для дивергенции теплового потока.

Замечание 1. Вводя новые обозначения $\mathbf{B} = \mathbf{A}^{-1/2}\mathbf{B}M^{-1/2}$, $\mathbf{w} = \mathbf{A}^{-1/2}\mathbf{w}$ и $T = M^{1/2}T$, можно упростить все уравнения, исключив $\mathbf{A}$ и M. Например, $\mathbf{A} + \alpha\tau\mathbf{G} = \mathbf{A} + \alpha\tau\mathbf{B}M^{-1}\mathbf{B}^T$ в новых обозначениях примет вид $\mathbf{E} + \alpha\tau\mathbf{B}\mathbf{B}^T$.

2. Построение схем расщепления для теплового потока. Примеры. Как уже упоминалось, основная идея предложенного подхода состоит в использовании скалярных схем расщепления [5] для дивергенции потока. Рассмотрим пример потоковой схемы расщепления в двумерном случае (без правой части):

$$\frac{w_y^{n+1/2} - w_y^n}{0.5\tau} + B_y\mathbf{B}^T\mathbf{w}^n = 0$$
$$\frac{w_x^{n+1/2} - w_x^n}{0.5\tau} + B_x\mathbf{B}^T\mathbf{w}^{n+1/2} = 0$$
$$\frac{w_x^{n+1} - w_x^{n+1/2}}{0.5\tau} + B_x\mathbf{B}^T\mathbf{w}^{n+1/2} = 0$$
$$\frac{w_y^{n+1} - w_y^{n+1/2}}{0.5\tau} + B_y\mathbf{B}^T\mathbf{w}^{n+1} = 0$$
$$\frac{T^{n+1} - T^n}{\tau} + \mathbf{B}^T\frac{\mathbf{w}^{n+1} + \mathbf{w}^n}{2} = 0$$

Здесь $\mathbf{B}^T\mathbf{w} = B_x^T w_x + B_y^T w_y$.

Замечание 2.

1. В действительности, $w_x^{n+1/2} = \frac{w_x^n + w_x^{n+1}}{2}$, и третье уравнение может быть заменено на более простое.

2. При реализации схемы возникает необходимость обращения только одномерных операторов с трехдиагональными матрицами вдоль координатных линий сетки.

Далее введем дивергенцию потока $\xi = \mathbf{B}^T\mathbf{w}$ и применим операторы B_x^T и B_y^T к парам уравнений 1–2 и 3–4, а затем сложим их. В результате получится следующая схема для дивергенции потока ξ:

$$\frac{\xi^{n+1/2} - \xi^n}{0.5\tau} + \Lambda_x\xi^{n+1/2} + \Lambda_y\xi^n = 0$$

$$\frac{\xi^{n+1} - \xi^{n+1/2}}{0.5\tau} + \Lambda_x \xi^{n+1/2} + \Lambda_y \xi^{n+1} = 0$$

с операторами $\Lambda_x = B_x^T B_x$ и $\Lambda_y = B_y^T B_y$, аппроксимирующими вторые производные по соответствующим пространственным направлениям. Можно заметить, что для ξ в итоге получилась классическая схема переменных направлений. Кроме того, уравнение для температуры можно переписать в следующем виде через ξ:

$$\frac{T^{n+1} - T^n}{\tau} + \frac{\xi^{n+1} + \xi^n}{2} = 0$$

и, очевидно, вычисление температуры полностью определяется тем, как вычисляется дивергенция теплового потока. В частности, точность вычисления температуры совпадает с точностью схемы для дивергенции. Справедливо и более сложное утверждение: порядок аппроксимации схемы для теплового потока совпадает с порядком аппроксимации схемы для дивергенции в неперестановочном случае $\Lambda_x \Lambda_y \neq \Lambda_y \Lambda_x$. Этот факт может быть проверен на примере локально-одномерной схемы для дивергенции, которая имеет второй порядок в перестановочном случае и лишь первый — в неперестановочном. Таким образом, соответствующая схема для теплового потока будет иметь в этом случае только первый порядок точности.

Замечание 4. Главной особенностью предложенного подхода является то, что можно взять любую схему для дивергенции теплового потока в качестве порождающей, и наоборот — любая схема для теплового потока соответствует какой-то схеме для дивергенции. Например, если в качестве схем для дивергенции взять схемы типа предиктор-корректор, то получатся схемы на основе алгоритма Удзавы.

При этом устойчивость схемы для дивергенции в норме вида $\| \xi \|_*$ эквивалентна устойчивости теплового потока в полунорме $\| \mathbf{B}^T \mathbf{w} \|_*$. В следующем пункте рассматривается идея доказательства глобальной устойчивости теплового потока.

В трехмерном случае можно использовать, например, схему Дугласа — Гана [6] второго порядка точности в качестве порождающей схемы для дивергенции. Соответствующая схема расщепления для теплового потока может быть записана в следующем виде:

$$\frac{w_y^{n+1/2}-w_y^n}{\tau}+B_yB_x^Tw_x^n+B_yB_y^Tw_x^n+B_yB_z^Tw_z^n=0$$

$$\frac{w_z^{n+1/2}-w_z^n}{\tau}+B_zB_x^Tw_x^n+B_zB_y^Tw_x^n+B_zB_z^Tw_z^n=0$$

$$\frac{w_x^{n+1}-w_x^n}{\tau}+B_xB_x^T\frac{w_x^{n+1}+w_x^n}{2}+B_xB_y^T\frac{w_y^{n+1/2}+w_y^n}{2}+$$
$$+B_xB_z^T\frac{w_z^{n+1/2}+w_z^n}{2}=0$$

$$\frac{w_y^{n+1}-w_y^{n+1/2}}{\tau}+B_yB_x^T\frac{w_x^{n+1}-w_x^n}{2}+B_yB_y^T\frac{w_y^{n+1}-w_y^n}{2}+$$
$$+B_yB_z^T\frac{w_z^{n+1/2}-w_z^n}{2}=0$$

$$\frac{w_z^{n+1}-w_z^{n+1/2}}{\tau}+B_zB_x^T\frac{w_x^{n+1}-w_x^n}{2}+B_zB_y^T\frac{w_y^{n+1}-w_y^n}{2}+$$
$$+B_zB_z^T\frac{w_z^{n+1}-w_z^n}{2}=0$$

$$\frac{T^{n+1}-T^n}{\tau}+\mathbf{B}^T\frac{\mathbf{w}^{n+1}+\mathbf{w}^n}{2}=0$$

Анализ данной схемы можно провести тем же способом, что и для потоковой схемы, основанной на схеме переменных направлений для дивергенции.

3. Априорные оценки и необходимая гладкость.

3.1. *Априорные оценки.* В данном пункте ограничимся рассмотрением примера потоковой схемы расщепления в двумерном случае. Ключевым моментом для дальнейшего анализа рассматриваемой схемы является тот факт, что переход от схемы для дивергенции к схеме для теплового потока может быть выполнен и после исключения дробных шагов, т. е. применение оператора дивергенции $\mathbf{B}^T$ к схеме в целых шагах для потока

$$(\mathbf{E}+\frac{\tau^2}{4}\begin{pmatrix}B_x\Lambda_y\\0\end{pmatrix}\mathbf{B}^T)\frac{\mathbf{w}^{n+1}-\mathbf{w}^n}{\tau}+\mathbf{B}\mathbf{B}^T\frac{\mathbf{w}^{n+1}+\mathbf{w}^n}{2}=0$$

приводит в точности к схеме в целых шагах для дивергенции:

$$(E+\frac{\tau^2}{4}\Lambda_x\Lambda_y)\frac{\xi^{n+1}-\xi^n}{\tau}+(\Lambda_x+\Lambda_y)\frac{\xi^{n+1}+\xi^n}{2}=0.$$

Далее, представим вектор теплового потока $\mathbf{w}^n$ как сумму двух ортогональных компонент: $\mathbf{w}^n = \mathbf{w}^{n,0} + \mathbf{w}^{n,1}$, где $\mathbf{w}^{n,0} \in Ker\mathbf{B}^T$ (ядро оператора дивергенции $\mathbf{B}^T\mathbf{w}^{n,0} = 0$) и $\mathbf{w}^{n,1} \in Im\mathbf{B}$ (ортогональное дополнение к ядру совпадает с образом оператора градиента $\mathbf{B}$). Тогда можно получить следующие два уравнения в соответствующих подпространствах:

1) в $Im\mathbf{B}$

$$\left(\mathbf{E} + \frac{\tau}{2}\mathbf{B}\mathbf{B}^T\right)\frac{\mathbf{w}^{n+1,1} - \mathbf{w}^{n,1}}{\tau} + \mathbf{B}\mathbf{B}^T\frac{\mathbf{w}^{n+1,1} + \mathbf{w}^{n,1}}{2} + \\ + (\mathbf{E} - \mathbf{Pr}_{Ker\mathbf{B}^T})\frac{\tau^2}{4}\begin{pmatrix} B_x\Lambda_y \\ 0 \end{pmatrix}\mathbf{B}^T\frac{\mathbf{w}^{n+1,1} - \mathbf{w}^{n,1}}{\tau} = 0,$$

2) в $Ker\mathbf{B}^T$

$$\frac{\mathbf{w}^{n+1,0} - \mathbf{w}^{n,0}}{\tau} + \frac{\tau^2}{4}\mathbf{Pr}_{Ker\mathbf{B}^T}\begin{pmatrix} B_x\Lambda_y \\ 0 \end{pmatrix}\mathbf{B}^T\frac{\mathbf{w}^{n+1,1} - \mathbf{w}^{n,1}}{\tau} = 0.$$

Так как точный тепловой поток должен удовлетворять закону Фурье и быть градиентом температуры, то компонента $\mathbf{w}^{\cdot,0}$ является полностью паразитической. Как следует из уравнений выше, эта компонента полностью определяется компонентой из Im$\mathbf{B}$. Поэтому для оценки компоненты из ядра необходимо получить оценки устойчивости для "регулярной" компоненты в соответствующих нормах. Введем сеточный аналог нормы пространства $\mathbf{H}_{div}$ следующим образом: $\| \mathbf{w} \|_{\mathbf{H}} = \| \mathbf{w} \|_{\mathbf{2}} + \| \mathbf{B}^T\mathbf{w} \|_2$. Здесь $\| \cdot \|_2$ — стандартная сеточная норма пространства L_2. После преобразований можно получить следующие априорные оценки для представленной двумерной потоковой схемы расщепления:

Теорема 1 (перестановочный случай, $\Lambda_x\Lambda_y = \Lambda_y\Lambda_x$). В перестановочном случае для потоковой схемы расщепления на основе схемы переменных направлений верна следующая оценка с константой $C \neq C(\tau, h)$:

$$\| \mathbf{w}^n \|_{\mathbf{H}} \leq C \| \mathbf{w}^0 \|_{\widetilde{\mathbf{H}}} \; \forall \mathbf{w}^0 \in \widetilde{\mathbf{H}}$$

где $\| \mathbf{w} \|_{\widetilde{\mathbf{H}}} = \| \mathbf{w} \|_{\mathbf{H}} + \tau^2 \| \Lambda_y\mathbf{B}^T\mathbf{w} \|_{\Lambda_x+\Lambda_y}$.

Теорема 2 (неперестановочный случай, $\Lambda_x\Lambda_y \neq \Lambda_y\Lambda_x$). В неперестановочном случае для потоковой схемы расщепления на основе схемы переменных направлений верна следующая оценка с константой $C \neq C(\tau, h)$:

$$\| \mathbf{w}^n \|_{\mathbf{H}} \leq C(1 + \frac{\tau}{h}) \| \mathbf{w}^0 \|_{\widetilde{\mathbf{H}}} \; \forall \mathbf{w}^0 \in \widetilde{\mathbf{H}},$$

где $\| \mathbf{w} \|_{\widetilde{\mathbf{H}}} = \| \mathbf{w} \|_{\mathbf{H}} + \| \Lambda_y \mathbf{B}^T \mathbf{w} \|_{\Lambda_x + \Lambda_y} + \| (E + \frac{\tau}{2}\Lambda_y)\mathbf{B}^T \mathbf{w}^0 \|_2$.

Замечание 5. Для рассмотренной в предыдущем пункте потоковой схемы расщепления в трехмерном случае могут быть получены аналогичные оценки.

Как можно заметить, априорные оценки накладывают требования дополнительной гладкости (связанные с наличием четырех производных по пространству) на начальный тепловой поток, который находится как решение уравнения $\mathbf{A}\mathbf{w}^0 = \mathbf{B}T^0$. Возникает вопрос, удовлетворяет ли начальный тепловой поток этим требованиям гладкости в различных ситуациях (неравномерные сетки, различные краевые улсовия, переменные коэффициенты и т. п.). Далее рассмотрим несколько численных экспериментов, демонстрирующих суть проблемы. В модельных примерах область $\Omega = [0,1]^2$ с однородными краевыми условиями Дирихле на границах по y и условиями Неймана по x.

Для сеточных норм погрешностей температуры и теплового потока введем следующие обозначения:

$$\varepsilon_{T,C} = \| T - T_{exact} \|_C, \varepsilon_{T,L_2} = \| T - T_{exact} \|_{L_2},$$
$$\varepsilon_{\mathbf{w},\mathbf{C}} = \| \mathbf{w} - \mathbf{w}_{exact} \|_{\mathbf{C}}, \varepsilon_{\mathbf{w},\mathbf{L_2}} = \| \mathbf{w} - \mathbf{w}_{exact} \|_{\mathbf{L_2}} .$$

3.2. *Краевые условия.* Рассмотрим два следующих тестовых решения: $T_1(t,x,y) = \cos 2\pi x \cdot \sin 2\pi y \cdot e^{-t}$ и $T_2(t,x,y) = \cos 2\pi x \cdot y \cdot \sin 2\pi y \cdot e^{-t}$ для случая равномерной сетки и постоянного отношения $\frac{\tau}{h}$ =const. В табл. 1 представлена погрешность в C и L_2-нормах для теплового потока.

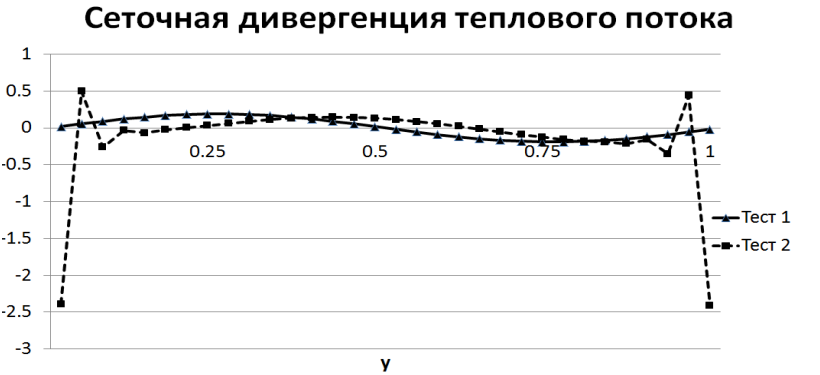
Причиной отсутствия сходимости для теста T_2 (сходимость с порядком 0 в C-норме и $1/2$ в L_2-норме) является плохая аппроксимация краевого условия Дирихле, что полностью согласуется с приведенными ранее априорными оценками.

На рисунке приведен график дивергенции начального теплового потока $\mathbf{B}^T\mathbf{w}^0$ для тестовых решений T_1 (Тест 1) и T_2 (Тест 2) вдоль фиксированной линии сетки по направлению y. Очевидно, для T_2 производные от дивергенции потока содержат отрицательные степени h, что и приводит к проблемам со сходимостью. Одним из возможных путей решения проблемы является использование интегрально-

Таблица 1

Сравнение тестовых решений T_1 и T_2,
краевые условия Дирихле — Неймана

h	T_1		T_2	
	$\varepsilon_{\mathbf{w},\mathbf{C}}$	$\varepsilon_{\mathbf{w},\mathbf{L_2}}$	$\varepsilon_{\mathbf{w},\mathbf{C}}$	$\varepsilon_{\mathbf{w},\mathbf{L_2}}$
1/32	3.5e-1	2.5e-1	**3.1e+2**	**6.4e+2**
1/64	5.9e-2	4.1e-2	**3.1e+2**	**4.6e+1**
1/128	1.3e-2	9.1e-3	**4.1e+2**	**4.3e+1**



Дивергенция начального теплового потока $\mathbf{B}^T\mathbf{w}^0$
вдоль линии сетки в направлении y: для T_1 (Тест 1), для T_2 (Тест 2)

го усреднения по Соболеву для начальной температуры (вместо использования стандартного поузлового сброса на сетку).

Замечание 6. В случае применения метода концентрации масс (диагонализации матрицы масс) для потока интегральное усреднение уже не исправляет ситуацию с неправильной аппроксимацией краевого условия Дирихле.

3.3. *Неравномерная сетка.* Теперь рассмотрим то же (см. предыдущий подпункт) тестовое решение $T_1(t,x,y) = \cos 2\pi x \cdot \sin 2\pi y \cdot e^{-t}$ на неравномерной сетке. Как и ранее, мы прежде всего следим за тепловым потоком, так как для температуры всегда имеет место сходимость со вторым порядком как в C-, так и в L_2-норме. Неравномерная сетка имеет следующий вид:

Таблица 2

Погрешность для теплового потока в случае неравномерной сетки для тестового решения T_1

h		$\tau/h^2 = \text{const}$		$\tau/h = \text{const}$	
h_1	h_2	$\varepsilon_{\mathbf{w},\mathbf{C}}$	$\varepsilon_{\mathbf{w},\mathbf{L_2}}$	$\varepsilon_{\mathbf{w},\mathbf{C}}$	$\varepsilon_{\mathbf{w},\mathbf{L_2}}$
1/30	1/36	1.8e+2	3.4e+1	1.8e+2	3.4e+1
1/60	1/72	6.0e+1	8.1e+0	1.8e+2	2.4e+1
1/120	1/144	2.0e+1	1.9e+0	2.4e+2	2.3e+1

Таблица 3

Результаты для T_2 в случае неравномерной сетки с точными значениями начального теплового потока

h		$\varepsilon_{T,C}$	ε_{T,L_2}	$\varepsilon_{\mathbf{w},\mathbf{C}}$	$\varepsilon_{\mathbf{w},\mathbf{L_2}}$
1/30	1/36	3.4e-2	1.7e-2	8.5e-1	2.1e-1
1/60	1/72	7.4e-3	3.8e-3	2.1e-1	4.3e-2
1/120	1/144	1.8e-3	9.1e-4	6.4e-2	9.6e-3

$$h_x = \text{const}, \qquad h_y = \begin{cases} h_1, & \text{если } \ y \le 0.5 \\ h_2, & \text{иначе} \end{cases}$$

Результаты приведены в табл. 2. При $\tau/h = \text{const}$ сходимость отсутствует в C-норме и имеет лишь порядок $1/2$ в L_2-норме. Можно показать с помощью непосредственных вычислений (по крайней мере, для диагонализованной матрицы масс $\mathbf{A}$), что в случае неравномерной сетки в ошибке для теплового потока присутствует слагаемое вида $(h_2 - h_1)\frac{\tau^2}{h^2}$ в окрестности изменения шага пространственной сетки. Таким образом, имеет место лишь условная сходимость.

К тому же, использование сглаживающих процедур типа интегрального усреднения для начальной температуры или начального теплового потока приводит к улучшению сходимости. В табл. 3 приведены результаты для того же тестового решения, но в начальный момент времени тепловой поток задавался точными значениями (вместо стандартного вычисления начального теплового потока как решения уравнения $\mathbf{A}\mathbf{w}^0 = \mathbf{B}T^0$ с заданной начальной температурой).

Нетрудно заметить, что точность для теплового потока существенно возросла, имеет место сходимость приблизительно со вто-

рым порядком. Данный факт является подтверждением того, что проблемы сходимости связана исключительно с недостаточной гладкостью именно начального теплового потока. Кроме того, точность вычисления температуры увеличивается при использовании точного начального потока незначительно, т. е. проблемы с аппроксимацией связаны с компонентой потока из ядра сеточного оператора дивергенции $\mathbf{B}^T$.

Заключение. В данной статье приводятся априорные оценки для потоковых схем расщепления в смешанном методе конечных элементов, построенных с помощью нового подхода. Основной идеей рассматриваемого подхода является использование скалярных схем расщепления для дивергенции теплового потока. Построенные схемы расщепления наследуют некоторые свойства порождающих схем для дивергенции потока, связанные с точностью и устойчивостью. Для потоковой схемы расщепления, предложенной в двумерном случае на основе схемы переменных направлений, получены априорные оценки устойчивости для теплового потока. В работе приведены результаты численных экспериментов, которые показывают, что в отдельных случаях могут возникать проблемы с аппроксимацией и сходимостью для теплового потока (в то время как для температуры имеет место сходимость со вторым порядком). В качестве решения возникающей проблемы предлагается использовать сглаживающие процедуры для начального теплового потока, в частности, интегральное усреднение Соболева.

Научный руководитель — д-р физ.-мат. наук Ю. М. Лаевский.

Список литературы

[1] Voronin K. V., Laevsky Y. M. An approach to the construction of flow splitting schemes in the mixed finite element method. Mathematical Models and Computer Simulations (2013) (to appear)

[2] Brezzi F., Fortin M. Mixed and hybrid finite element methods. Springer-Verlag, New York (1991)

[3] Voronin K. V., Laevsky Y. M. On splitting schemes in the mixed finite element method // Num. Analysis and Applications. 2010. N 5(2). P. 150–155.

[4] Voronin K. V., Laevsky Y. M. Splitting schemes in the mixed finite-element method for the solution of heat transfer problems // Math.l Models and Computer Simulations. 2012. N 5(2). P. 167–174.

[5] Yanenko N. N. The method of fractional steps: solution of problems of mathematical physics in several variables. Springer, 1971.

[6] Douglas J., Gunn J. E. A general formulation of alternating direction methods // Numerische Mathematik. 1964. N 6. P. 428–453.

[7] Arbogast T., Huang C.-S., Yang S.-M. Improved accuracy for alternating-direction methods for parabolic equations based on regular and mixed finite elements // Math. Models and Methods in Appl. Scie. 2007. N 17(8). P. 1279–1305.

Воронин Кирилл Владиславович — инженер Института вычислительной математики и математической геофизики СО РАН; аспирант Новосибирского государственного университета; e-mail: kvoronin@labchem.sscc.ru

ТРУДЫ КОНФЕРЕНЦИИ МОЛОДЫХ УЧЕНЫХ
Новосибирск, 2014 С. 46–54

Итерационный метод решения СЛАУ с положительно определенной симметричной матрицей на основе алгебраической декомпозиции области для машин с распределенной памятью

М. В. Жукова

УДК 519.612.2

Работа является частью пакета программ для решения СЛАУ с разреженной матрицей на машинах с распределенной памятью, основанного на предобусловленном итерационном методе решения. В качестве основного алгоритма используется метод сопряженных градиентов. Главная цель — обобщить алгоритм со случая блочно-трехдиагональных матриц на матрицы общего вида. Рассмотрены два подхода: использование предобусловливателя, построенного на основе блочно-трехдиагональной части матрицы, и изначальное переупорядочивание матрицы с целью повышения эффективности предобусловливания. В качестве алгоритма для переупорядочивания использован приближенный метод минимальной степени. Описан реализованный параллельный алгоритм с использованием MPI, приведены графики зависимости времени счета и количества итераций от числа процессов.

The work is a part of software package created for solving SLAE with sparse matrices on distributed memory machines which is based on preconditioned iterative method. Conjugate gradient method is used as the basic algorithm. The main goal of this work is to generalize algorithm for block tridiagonal matrix to the general case. Two approaches were investigated: preconditioner constructed on the basis of block tridiagonal part of the initial matrix and reordering of matrix in order to increase the efficiency of preconditioning. For the reordering-step approximate minimum degree algorithm is used. Implemented parallel algorithm using MPI is described, dependencies of solution time and iteration number on the number of MPI processes are presented.

Ключевые слова: Итерационный алгоритм, машины с распределенной памятью, MPI, параллельные вычисления, разреженные матрицы.

Keywords: Iterative method, distributed memory machines, MPI, parallel computations, sparse matrices

Введение. В настоящее время при моделировании многих явлений мы сталкиваемся с необходимостью решать системы линейных алгебраических уравнений с достаточно большими разреженными матрицами. Под большими сейчас принято понимать системы с числом неизвестных порядка $N = 10^7 - 10^9$, разреженными называются матрицы с числом ненулевых элементов $nnz = O(N)$.

Данная работа является продолжением цикла работ [1, 2] по созданию пакета программ для решения систем линейных алгебраических уравнений (СЛАУ) с разреженной матрицей на машинах с распределенной памятью. Пакет основан на предобусловленном итерационном методе решения СЛАУ, в качестве основного алгоритма используется метод сопряженных градиентов. Главная цель данной работы — обобщение случая блочно-трехдиагональных матриц на матрицы общего вида. В данном случае рассматриваются только симметричные положительно определенные матрицы.

Были выбраны два подхода: использование предобусловливателя, построенного на основе блочно-трехдиагональной части матрицы, и изначальное переупорядочивание матрицы с целью повысить эффективность предобусловливания. Для переупорядочивания был использован приближенный метод минимальной степени (Approximate minimum degree (AMD)) [3, 4].

В работе описан реализованный параллельный алгоритм с использованием MPI и приведены графики зависимости времени счета и числа итераций от количества процессов, также представлены случаи, отражающие преимущества и недостатки выбранных подходов.

1. Описание алгоритма и реализация. Необходимо решить систему $Ax = b$, где матрица $(A)_{N\times N}$ разреженная симметричная и положительно определена, $A = A^T > 0$. Представим ее в виде

$$\begin{pmatrix} A_{1,1} & A_{1,2} & ... & A_{1,n} \\ A_{1,2} & A_{2,2} & ... & A_{2,n} \\ ... & ... & ... & ... \\ A_{1,n} & A_{2,n} & ... & A_{n,n} \end{pmatrix},$$

где $A_{i,j}$ — некоторый блок, размер которого определяется в зависимости от числа процессов, используемых при вычислениях.

СЛАУ решается методом сопряженных градиентов с предобусловливателем, основанным на алгебраической декомпозиции. Для построения предобусловливателя B используется блочно-трехдиагональная часть матрицы с перекрытиями: $B^{-1} = \sum_{i=1}^{n-1} R_i \begin{pmatrix} A_{i,i} & A_{i,i+1} \\ A_{i,i+1} & A_{i+1,i+1} \end{pmatrix}^{-1} R_i^T$, где R_i — блочная матрица $2 \times n$ с единичной матрицей E на местах $(1, i)$ и $(2, i+1)$:

$$\begin{pmatrix} 0 & 0 & ... & E & 0 & ... & 0 \\ 0 & 0 & ... & 0 & E & ... & 0 \end{pmatrix}.$$

Программная реализация. Так как в качестве основного итерационного алгоритма в данной работе выбран предобусловленный метод сопряженных градиентов, то необходимо реализовать умножение матрицы на вектор, обращение блоков — для построения предобусловливателя, и скалярное произведение. Важно отметить, что подобный способ построения предобусловливателя позволяет хранить данные распределенно, часть матрицы и вектора на каждом процессе.

На процессе с номером i хранится часть матрицы вида

$$\begin{pmatrix} A_{1,i} & A_{2,i} & ... & A_{i,i} & A_{i,i+1} & ... & A_{i,n} \\ A_{1,i+1} & A_{2,i+1} & ... & A_{i,i+1} & A_{i+1,i+1} & ... & A_{i+1,n} \end{pmatrix},$$

где $A_{i,j}$ — некоторый блок, N — количество неизвестных, n — число MPI процессов, используемых при вычислениях. Размер блока определяется как $\lfloor \frac{N}{n+1} \rfloor$ для процессов $i = 1...n-1$ и $N - \lfloor \frac{N}{n+1} \rfloor \cdot (n-1)$ для последнего. Аналогично храним две блочные компоненты для вектора решения и правой части. В таком случае для реализации предобусловливателя на каждом из процессов понадобится лишь часть:

$$\begin{pmatrix} A_{i,i} & A_{i,i+1} \\ A_{i,i+1} & A_{i+1,i+1} \end{pmatrix}.$$

При таком распределении данных скалярное произведение реализуется очевидным образом, и необходимо лишь собрать полученные

данные с каждого процесса на выделенный — главный. Для умножения матрицы на вектор используется функциональность Sparse BLAS из Intel ®MKL [7], умножение происходит на каждом процессе:

$$\begin{pmatrix} A_{1,i} & A_{2,i} & ... & A_{i,n} \\ A_{1,i+1} & A_{2,i+1} & ... & A_{i+1,n} \end{pmatrix} \cdot \begin{pmatrix} X_1 \\ ... \\ X_i \\ ... \\ X_n \end{pmatrix} = \begin{pmatrix} Y_i \\ Y_{i+1} \end{pmatrix},$$

полученные в результате блоки Y_i собираются в полный вектор. Также необходимо на каждом процессе обращать часть матрицы, используемой для предобусловливателя. Таким образом для процесса с номером i с помощью Intel ®MKL PARDISO вычисляем

$$\begin{pmatrix} A_{i,i} & A_{i,i+1} \\ A_{i,i+1} & A_{i+1,i+1} \end{pmatrix}^{-1} \cdot \begin{pmatrix} X_i \\ X_{i+1} \end{pmatrix} = \begin{pmatrix} Z_{i,1} \\ Z_{i,2} \end{pmatrix}.$$

Необходимо отметить, что, учитывая способ построения предобусловливателя, это только частичный результат, поэтому понадобятся дополнительные пересылки с процессов $i-1, i+1$ компонент $Z_{i-1,2}$ и $Z_{i+1,1}$ соответственно. В итоге на i-м процессе имеем:

$$\begin{pmatrix} Z_{i,1} + Z_{i-1,2} \\ Z_{i,2} + Z_{i+1,1} \end{pmatrix}.$$

В результате алгоритм подразделяется на несколько этапов: на первом происходит инициализация данных и их рассылка по процессам, затем в рамках функциональности Intel ®MKL PARDISO происходит подготовка предобусловливателя, последний этап — решение системы, именно его продолжительность будет представлена в дальнейшем на графиках. При использовании алгоритма переупорядочивания матрицы (AMD) он включается в первый этап.

2. Переупорядочивание матрицы. Зачастую для повышения эффективности, например, при решении с помощью прямых методов, используются алгоритмы переупорядочивания матрицы, которые позволяют уменьшить заполнение, портрет матрицы. Задача поиска наилучшего переупорядочивания является *NP*-полной, поэтому для ее решения используются различные эвристики, которые не могут гарантировать результат.

Одним из популярных и достаточно простых алгоритмов переупорядочивания является метод минимальной степени (Minimum degree (MD)). В данной работе с целью улучшить предобусловливатель было выбрано его приближение (AMD), которое хоть и может давать худший, чем непосредственно алгоритм минимальной степени, результат, но требует меньше вычислительных затрат.

Алгоритм минимальной степени (MD). Алгоритмы переупорядочивания матрицы основаны на представлении ее в виде графа $G = (V, E)$, где $V = \{1...N\}$ — множество вершин; E — множество ребер графа : $(i, j) \in E \leftrightarrow a_{i,j} \neq 0$, заданное ненулевыми элементами матрицы $A = (a_{i,j})_{N \times N}$.

Для алгоритма минимальной степени также необходимо задать величину $\deg(v) = |\{u \mid \exists e \in E : (u, v) = e\}|$ — степень вершины $v \in V$,, определяемую как число вершин, смежных к данной, иначе говоря, число ненулевых элементов в выбранной строке матрицы.

Несложный алгоритм состоит в последовательном исключении вершин из графа. Очередная выбираемая вершина i должна иметь минимальную степень, $i : \ deg(i) = min_k \ deg(k)$, после удаления ее номер заносится в перестановку $\pi = \pi \cup \{i\}$. Одним из недостатков данного алгоритма является необходимость пересчитывать степени вершин на каждом шаге исключения. Вычислительная сложность алгоритма равна $O(|V|^2 \cdot |E|)$, или, в более привычных терминах, $O(N^2 \cdot nnz)$.

Приближение для алгоритма и программная реализация. С целью уменьшения вычислительных затрат вместо алгоритма минимальной степени часто используется алгоритм Multiple Minimum Degree (MMD), при выполнении которого на каждом шаге исключается более одной вершины, либо приближенный метод Approximate Minimum Degree (AMD). Идея AMD заключается в том, чтобы избавиться от постоянного пересчитывания степени вершины, используя вместо этого некоторую верхнюю границу. Подробное описание алгоритма, выражение для верхней границы и ее точность по отношению к реальной степени, а также сравнение с другими алгоритмами для переупорядочивания матрицы можно найти в [3, 4]. Реализация алгоритма взята из [5]. Необходимо отметить, что вычислительная сложность *AMD* равна $O(|V| \cdot |E|)$, или, что то же самое, $O(N \cdot \ nnz)$.

3. Численные эксперименты. Тестирование разработанного параллельного алгоритма проводилось на симметричных положи-

а *б*

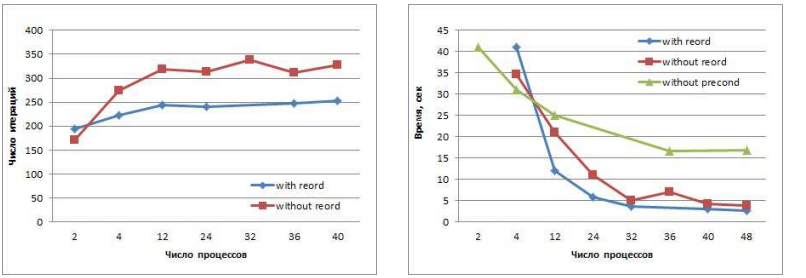


Рис. 1. Матрица crankseg, $N = 52\,804$, $nnz = 10\,614\,210$

Таблица 1

Сравнение числа итераций для матрицы crankseg, $N = 52\,804$, $nnz = 10\,614\,210$

Число процессов	При использовании предобусловливателя	Без предобусловливателя
4	171	2979
12	274	2987
48	327	2990
60	360	2995
180	406	2982

тельно определенных матрицах из *The University of Florida sparse matrix collection* [6] с некоторым заранее заданным вектором решения. Условие остановки итерационного процесса: $\|\dfrac{B^{-1}r_k}{B^{-1}r_0}\| < \varepsilon$, $\varepsilon = 10^{-9}$ (r_k — невязка). Вычисления выполнялись на кластере NKS-30T (nks-g6.sscc.ru) с 6-ядерными процессорами X5670 2.93 GHz (Westmere) в составе блейд-серверов HP BL2x220c; на каждое вычислительное ядро приходится 2Gb оперативной памяти.

На рис. 1–4 показана зависимость числа итераций (рис. 1*а*–4*а*) и продолжительности выполнения этого этапа решения системы (рис. 1,*б*–4,*б*), от числа процессов:

Например, для матрицы *crankseg* ($N = 52\,804, nnz = 10\,614\,210$) из табл. 1 видно, что при использовании предобусловливателя значительно уменьшалось число итераций. На рис. 1 видно, что это дает

а *б*

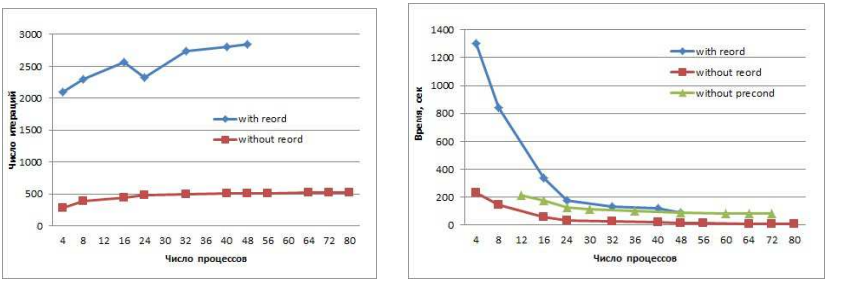


Рис. 2. Матрица nd24k, $N = 72\,000$, $nnz = 28\,715\,634$

а *б*

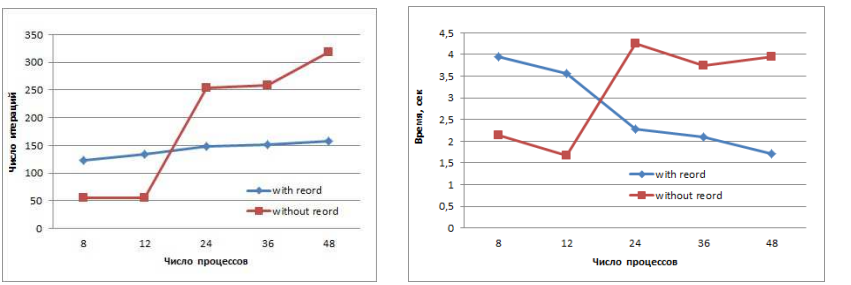


Рис. 3. Dubcova3, $N = 146\,689$, $nnz = 3\,636\,649$

выигрыш и во времени решения, однако использование AMD не дало значительного улучшения.

Для *nd24k* (рис. 2) видно, что переупорядочивание только ухудшало ситуацию, что может объясняться изменением портрета матрицы.

Для матриц, рассматриваемых далее, время и число итераций при решении без предобусловливателя значительно больше, поэтому показаны только случаи с использованием предобусловливателя, и с (без) AMD.

На рис. 3, 4 показаны результаты, отражающие преимущества использования алгоритма переупорядочивания матрицы. На рис. 4 также видно, что и время, и число итераций заметно сократилось. Необходимо также отметить, что для этого случая увеличение числа процессов не дает выигрыша — нет масштабируемости.

а *б*

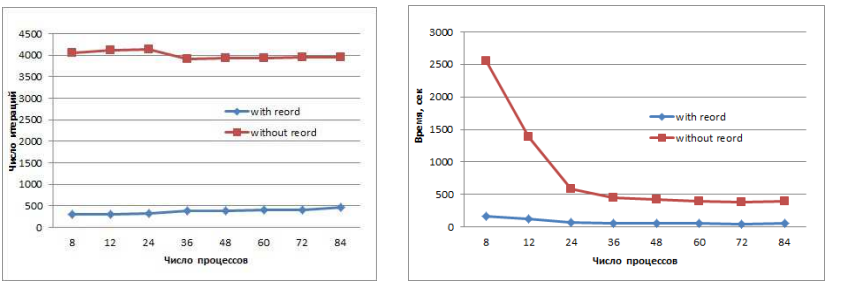


Рис. 4. ldoor, $N = 952\ 203$, $nnz = 46\ 522\ 475$

Заключение. В данной работе реализован метод сопряженных градиентов с предобусловливателем для решения систем уравнений с симметричной положительно определенной матрицей на машинах с распределенной памятью. Предложены два варианта предобусловливателя — блочно-трехдиагональный и блочно-трехдиагональный после процедуры перестановки AMD. Приведено описание алгоритмов, выполнено численное сравнение. Анализ полученных результатов не позволяет выделить какой-либо из рассмотренных предобусловливателей с точки зрения как количества итераций, так и времени решения, однако показывает перспективность применения алгоритмов декомпозиции для матриц произвольного вида.

Научный руководитель — А. А. Калинкин

Список литературы

[1] Kalinkin A. A., Laevsky Yu. M., Iterative solver for systems of linear equations with a sparse stiffness matrix for clusters, Sib. Zh. Vychisl. Mat. 2012. V. 15, N 2. P. 223–228.

[2] Андерс А. В., Андерс Р. В. Итерационный метод решения систем линейных уравнений с разреженной матрицей для машин с распределенной памятью // Материалы конф. молодых ученых ИВМиМГ СО РАН, Новосибирск, 24–26 апр. 2012 г. Новосибирск: ИВМиМГ, 2014.

[3] Amestoy P. R., Davis T. A., Duff I. S. An approximate minimum degree ordering algorithm // SIAM J.l on Matrix Analysis and Appl. 1996. N 17. P. 886–905.

[4] Amestoy P. R., Davis T. A., Duff I. S. Algorithm 837: AMD, An approximate minimum degree ordering algorithm // ACM Trans. on Math. Software. 2004. V. 30, N 3. P. 381–388.

[5] Amestoy P. R., Davis T. A., Duff I. S. [Electron. resource]. https://www.cise.ufl.edu/research/sparse/amd.

[6] The University of Florida Sparse Matrix Collection. [Electron. resource]. https://www.cise.ufl.edu/research/sparse/matrices.

[7] Intel ®Math Kernel Library. [Electron. resource]. https://software.intel.com/sites/products/documentation/hpc/mkl/mklman/mklman.pdf

Жукова Мария Викторовна — 2-й курс магистратуры ММФ НГУ;
практикант-магистр ЗАО "Интел А/О";
e-mail: maria.v.zhukova@mail.ru

ТРУДЫ КОНФЕРЕНЦИИ МОЛОДЫХ УЧЕНЫХ
Новосибирск, 2014 С. 55–63

Система вывода алгоритмов на вычислительных моделях и интерпретатор для реализации алгоритмов на вычислительных системах с общей памятью

А. Б. Купчишин

УДК 004.4’2

Представлены проект и прототип программного комплекса, состоящего из системы вывода линейных алгоритмов на простых вычислительных моделях, системы исполнения алгоритмов на вычислительных системах с общей памятью, а также библиотеки, включающей процедуры для решения задачи поиска гипоцентра микросейсмической активности в однородной среде.

The paper presents the project and the prototype of the software system consisting of a module for structural synthesis of parallel algorithms on simple computational models, an interpreter module to execute algorithms on shared memory computers, and a library that includes subroutines for solving the problem of detection of a microseismic activity hypocenter in a homogeneous medium.

Ключевые слова: Гипоцентр, вывод, алгоритм, вычислительная модель, процедура, микросейсмика, сейсмика.

Keywords: Hypocenter, concluded algorithm computational model, the procedure, microseismic, seismic.

Введение. При решении крупномасштабных задач с использованием высокопроизводительных вычислительных систем в настоящее время повсеместно используются инструменты низкоуровневого параллельного программирования, такие как MPI, OpenMP, CUDA, PThreads и т. д.. Этим определяется необходимость ручной адаптации алгоритма решения задачи к архитектуре конкретного вычислительного устройства, а также необходимость выполнения большого объема рутинной работы, не имеющей непосредственного отношения к решению задачи, такой как программирование всех коммуникаций вычислительных узлов, контроль порядка чтения/записи данных в памяти параллельными потоками, управление и синхронизация по-

токов и т. д. Поэтому актуальной задачей является создание системы автоматического вывода алгоритмов на вычислительных моделях [1] и системы исполнения (интерпретатора) выведенных алгоритмов на высокопроизводительном вычислительном оборудовании. В настоящей работе представлен проект и прототип программного комплекса, предназначенного для разработки и исполнения программ решения задач обработки сейсмических данных. Комплекс состоит из системы вывода потоковых алгоритмов на простых вычислительных моделях, системы исполнения построенных алгоритмов на вычислительных системах с общей памятью и библиотеки параллельных процедур, реализующих алгоритм когерентного суммирования для поиска гипоцентра микросейсмической активности.

1. Постановка задачи. Цель работы — создание системы автоматического вывода алгоритмов на вычислительных моделях и системы исполнения (интерпретатора) выведенных алгоритмов на высокопроизводительных вычислительных машинах. В качестве входа для задачи вывода алгоритмов по методу структурного синтеза [1] подается вычислительная модель, набор входных величин для алгоритма, набор величин, которые должны быть вычислены алгоритмом, набор нефункциональных требований. Вычислительная модель представляет собой двудольный граф, в котором одно множество вершин – это величины (объекты) предметной области, а другое множество вершин – операции, позволяющие по одним объектам вычислять другие. Задача интерпретатора — в динамике отображать алгоритм, представленный в виде множества операций, над которыми наведен частичный порядок, на ресурсы вычислительной системы. Выбор подходящих процедур для реализации операций алгоритмов также осуществляется исполнительной системой.

2. Архитектура программного комплекса. Данная работа является частью проекта создания системы автоматизации разработки параллельных программ на основе структурного синтеза [5].

Система автоматизации разработки параллельных программ на основе структурного синтеза состоит из следующих компонентов (рис. 1):

1) Визуальная среда (реализует интерфейс пользователя [6], позволяет конструировать алгоритмы и вычислительные модели).

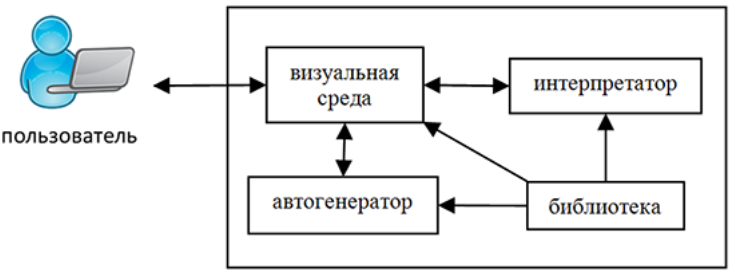


Рис. 1. Система автоматизации разработки параллельных программ на основе структурного синтеза

2) Автогенератор (система вывода алгоритмов по методу структурного синтеза, позволяет автоматически находить алгоритм преобразования исходных данных в искомые).

3) Интерпретатор (система исполнения построенных алгоритмов).

4) Библиотека (набор операций-процедур предметной области, адаптированных под различные вычислительные платформы).

Пользователь конструирует алгоритм решения задачи посредством визуальной среды (внешне разрабатываемый компонент [6]), которая предоставляет ему необходимый для этого инструментарий, в терминах предметной области из объектов и операций из библиотеки. Также пользователь с помощью визуального конструктора может задавать вычислительную модель и ставить задачу вывода алгоритмов на заданных вычислительных моделях. Автогенератор позволяет извлекать алгоритм из вычислительной модели, при этом достаточно указать, какие объекты известны, а какие нужно найти. Результатом работы пользователя с визуальной средой и автогенератором является построенный алгоритм. Алгоритм сохраняется в скрипт-файле в понятной интерпретатору форме: файл содержит описание множества объектов и операций алгоритма, описание всех взаимосвязей вида объект-операция. Интерпретатор, получая на вход скрипт-файл, выполняет алгоритм, вызывая соответствующие операциям процедуры из библиотеки.

3. Интерпретатор. Интерпретатор работает в терминах асинхронной модели вычислений, которая была предложена в [7]. Асинхронная программа (А-программа) — это конечное множество

А-блоков Ak, определенных над памятью M. Каждый А-блок Ak = <tr(ak),ak> состоит из спусковой функции tr(ak) и операции ak. Выполнение А-программы включает вычисление значения спусковой функции tr(ak) для всех или части А-блоков при текущем состоянии памяти M и выполнение одного, части или всех А-блоков Ak, так что (tr(ak) = true) при текущем состоянии памяти M.

Выполнение А-программы заканчивается, если больше ни один А-блок выполниться не может, т. е. tr(ak) = false для всех А-блоков в некоторый момент времени.

Каждая операция имеет набор входных и выходных переменных, т. е. набор объектов, с которыми данная операция связана. Объекты в этом случае выступают в качестве памяти M, над которой определены А-блоки. Триггер-функция А-блока срабатывает, если входные объекты в операцию содержат конкретные значения, а выходные объекты не заполнены. Выполнение операции происходит, если соответствующая триггер функция возвращает true, при этом связанные объекты блокируются, входные объекты становятся не заполненными, а выходные объекты заполняются вычисленными значениями. По завершении выполнения операции связанные объекты освобождаются.

Блоки операций, не имеющие транзитивной информационной зависимости по данным, могут быть выполнены параллельно. Таким образом, раскрывается весь потенциал параллельного исполнения алгоритма. Процедуры из библиотеки, соответствующие операциям алгоритма, могут быть реализованы независимо с использованием параллельных технологий. Поэтому параллельное исполнение программы в целом включает два уровня параллелизма: параллельное исполнение инструкций внутри отдельных операций алгоритма и параллельное исполнение независимых операций.

4. Библиотека. Библиотека представляет собой расширяемый набор элементарных процедур предметной области, адаптированных под различные вычислительные платформы и набор пользовательских процедур, т. е. сконструированных пользователем алгоритмов, добавленных в библиотеку в качестве структурной операции. Также в библиотеке хранится описание вычислительных моделей.

Специализированные библиотеки могут быть разработаны для различных предметных областей. В настоящей работе осуществляется наполнение библиотеки для решения двух задач: первая —

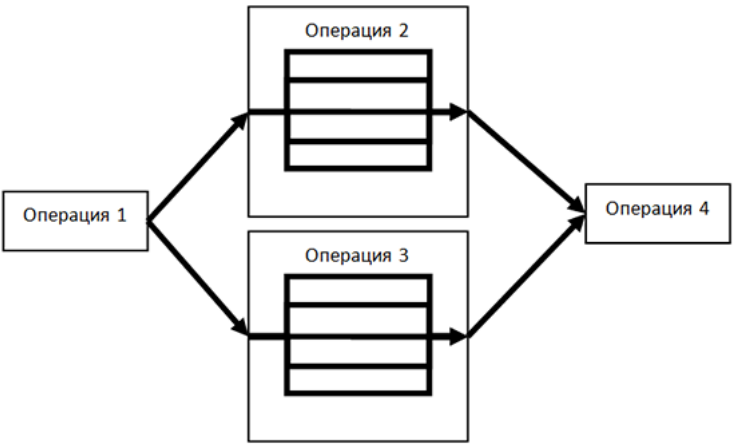


Рис. 2. Параллелизм на уровне операций и алгоритма в целом

это генерация тестовых моделей геологического строения и физико-механических свойств исследуемых грунтов, что включает такие операции, как построение сеток и моделирование распространения сейсмических волн, вторая — это локализация источников сейсмической активности.

В настоящее время библиотека содержит реализации необходимых операций для решения задачи поиска гипоцентра микросейсмической активности в однородной среде методом когерентного суммирования [2].

Чтобы определить координаты гипоцентра микросейсмической активности на поверхности исследуемого объема Земли, раскладывают сейсмоприемники, улавливающие возмущение источника и шум, сопоставимый по силе с самим возмущением. На рис. 3 точками обозначены сейсмоприемники, а цветом — их состояние.

На рис. 4 показаны примеры суммирований для гипоцентра и некоторой произвольной точки: треугольниками обозначены сейсмоприемники на поверхности; звездочкой — источник (гипоцентр); Data — схематически показанные сейсмические данные (каждая линия отражает развертку во времени состояния датчика, всплеск на линии — приход сигнала от источника на датчик в соответствующий момент); Moveout — данные и спрямленный годограф; Stack — результат суммирования.

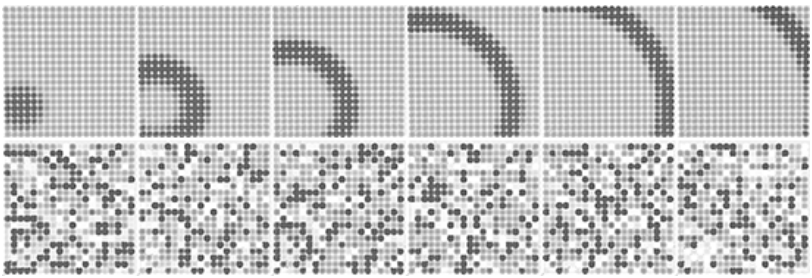


Рис. 3. Регистрация датчиками распространения волны в идеальных условиях без шумов (верхний ряд) и с шумом (нижний ряд). Точками обозначена сеть сейсмоприемников, расположенных на поверхности земли, цветом — состояние приемника

В методе когерентного суммирования по годографу прямой волны сначала объем исследуемого пространства разбивается на кубическую сетку, в узлы которой последовательно помещается предполагаемый источник. С учетом информации о строении среды рассчитывается время прихода сигнала $TS(D,P)$ от выбранного узла P до каждого из датчиков D. Пусть $G(t,D,P)$ обозначает измерение, записанное датчиком D в момент времени t от начала наблюдений. Далее для всех $t = \overline{0, T_{end} - max_{\forall D}(TS(D,P))}$, где T_{end} — время записи последнего измерения датчиками, производится вычисление суммы

$$S(t,P) = \sum_{\forall D} G(TS(D,P) + t, D, P).$$

Узел, для которого наблюдается максимальное значение среди сумм $S(t,P)$, полагается местоположением искомого источника. Соответствующее значение t позволяет определить время наступления сейсмического события. Узел, для которого результат суммирования максимален, соответствует координатам искомого источника.

Реализованный алгоритм когерентного суммирования сочетает следующие особенности [5]:

— параллелизм в распределенной памяти с разделением работы между процессами по обрабатываемым временным отрезкам, т. е. между процессами распределяются блоки значений t.

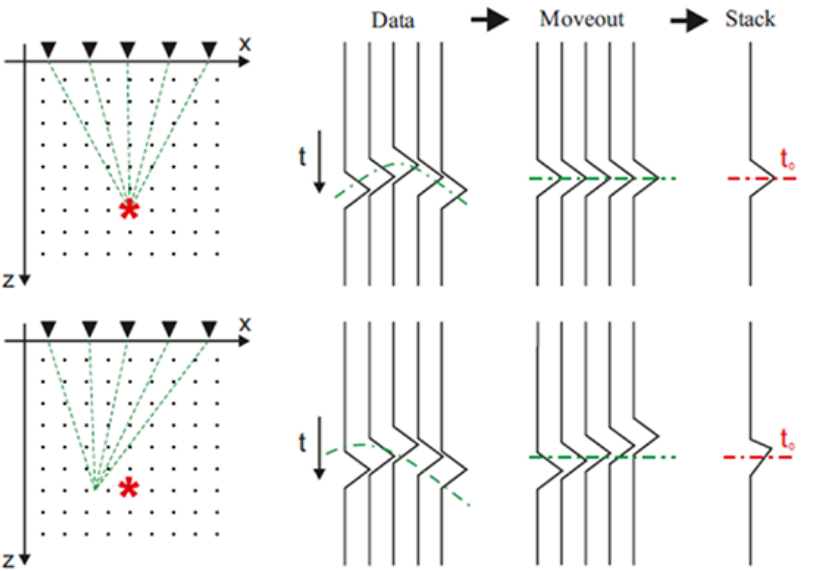


Рис. 4. Обнаружение гипоцентров сейсмической активности с помощью метода когерентного суммирования

— параллелизм в общей памяти с разделением работы между потоками по исследуемому пространству, т. е. между потоками распределяется множество узлов P.

— двойная буферизация, поскольку значительное время занимает считывание состояния датчиков из файлов, то организовано чтение файлов на фоне обработки ранее прочитанных данных.

— уточнение исследуемой области: в исходном исследуемом объеме задается относительно грубая сетка, на которой обнаруживается с помощью описанного алгоритма узел, наиболее близкий к источнику. Вокруг обнаруженного узла задается область меньшего размера, в которой строится сетка из узлов с более плотным расположением. Такие шаги уточнения подобласти продолжаются до заданного размера подобласти.

5. Автогенератор. Система вывода должна извлечь из вычислительной модели алгоритм, позволяющий вычислить искомые величины по заданным, при этом из множества выводимых на дан-

ной вычислительной модели алгоритмов должен быть выбран алгоритм, отвечающий заданным нефункциональным требованиям, таким как наибольшее быстродействие (из выводимых алгоритмов) на данном вычислителе, минимальные требования к памяти, максимальная масштабируемость и т. д.

В настоящий момент автогенератор реализует поиск алгоритма в простых вычислительных моделях по принципу обхода графа в ширину с заданной глубиной в два этапа.

1) Итерационный вывод всех объектов, которые можно получить из набора исходных объектов. Одна итерация состоит из попытки получить все объекты по заданным правилам вычислительной модели из исходных объектов и объектов, полученных на предыдущих итерациях.

2) Если заданная глубина поиска достигнута или на какой-то итерации больше ничего нельзя вывести, то алгоритм переходит во второй этап, в котором отслеживается минимально необходимая (в общем случае удовлетворяющая нефункциональным требованиям) цепочка преобразований до целевого объекта из исходных, если таковой был получен.

Заключение. Реализованы: автоматический поиск алгоритмов на простых вычислительных моделях по принципу обхода графа в ширину; интерпретатор алгоритмов в общей памяти в терминах асинхронной модели вычисления; набор операций, решающий задачу поиска гипоцентра микросейсмической активности методом когерентного суммирования.

Дальнейшая работа предполагает перевод исполнительной системы в распределенную память, реализацию возможности конструировать нелинейные схемы программ, интеграцию инструмента визуального конструирования в среду HPC Community Cloud [3, 4], учет нефункциональных требований в системе автоматического вывода алгоритма.

Научный руководитель — м.н.с. М. А. Городничев

Список литературы

[1] Вальковский В. А., Малышкин В. Э. Синтез параллельных программ и систем на вычислительных моделях Новосибирск: Наука. Сиб. отд-ние, 1988.

[2] Колесников Ю. И., Хогоев Е. А., Полозов С. В., Донцов М. В., 2004. Применение сейсмоэмиссионной томографии для локализации сейсмических источников. // Сб. докл. Междунар. конф. “Сейсмические исследования Земной коры”, Новосибирск. С. 129–134.

[3] Городничев М. А., Малышкин В. Э., Медведев Ю. Г. HPC Community cloud: эффективная организация работы научно-образовательных суперкомпьютерных центров // Науч. вестн. НГТУ. 2013. № 3 (52). С. 91–96.

[4] Вайцель С. А., Городничев М. А. HPC Community Cloud: разработка инструментария для повышения уровня взаимодействия пользователей с объединенными HPC-системами // Тез. докл. 7-й Сиб. конф. по параллельным и высокопроизводительным вычислениям, Томск, 12–14 нояб. 2013 г. Томск: Изд-во ТГУ, 2014. С. 82–84.

[5] Сарычев В. Г., Купчишин А. Б. Разработка программного комплекса для конструирования программ обработки данных на высокопроизводительных вычислительных системах // Тез. докл. 7-й Сиб. конф. по параллельным и высокопроизводительным вычислениям / Под ред. проф. А. В. Старченко. Томск: Изд-во ТГУ, 2014. С. 55–64.

[6] Сарычев В. Г. Среда визуального конструирования параллельных программ // Материалы школы-конференции молодых ученых “Современные проблемы прикладной математики и информатики”, Новосибирск, 9–13 июня 2014 г. Новосибирск: Б. и., 2014. С. 66.

[7] Котов В. Е., Нариньяни А. С. Асинхронные вычислительные процессы над памятью // Кибернетика. 1966. № 3. С. 64–71.

Купчишин Александр Борисович — студент Новосибирского государственного технического университета; e-mail: vsegdatrezv@mail.ru

ТРУДЫ КОНФЕРЕНЦИИ МОЛОДЫХ УЧЕНЫХ
Новосибирск, 2014 С. 64–75

Методы оценки надежности сетей с ограничением на диаметр

С. Н. Нестеров
Новосибирский государственный университет

УДК 519.6

Задача расчета K-терминальной надежности графа $G = (V, E)$ с ограничением $R_k(G, D)$ на диаметр D определяется как вероятность того, что для любой пары вершин из множества терминалов $K \subseteq V$ существует связывающий их надежный путь длины не более D ребер, и является $\mathcal{NP}$-трудной. Для подсчета точного значения $R_k(G, D)$ выбран алгоритм Cancela и Petingi с предлагаемыми методами ускорения (редукция ребер и упорядочивание ребер). Обобщен на случай K-терминальной надежности с ограничением на диаметр метод кумулятивных оценок надежности сети, предложенный Won и Karray, позволяющий ограничить значение надежности данной сети заранее, до вычисления точного значения. Эффективность предложенных методов показана в результате численных экспериментов.

A problem of computation diameter constrained network reliability for a chosen graph $G = (V, E)$ is a probability that every two nodes from a given set of terminals $K \subseteq V$ are connected with a path of length less or equal to a given integer D. This problem is known to be NP-hard. The factoring method by Cancela and Petingi(2001) was chosen for calculation of a precise value of $R_k(G, D)$ with proposed new methods of speeding computation. Algorithm by Won and Karray(2010), which cumulatively updates the lower and upper bounds of the classic network reliability, was extended for diameter constrained terms. Numerical experiments show effectiveness of proposed methods.

Ключевые слова: надежность сети, ограничение на диаметр, упорядоченное ветвление, ветвление по цепи, параллельный алгоритм, кумулятивные оценки.

Keywords: network reliability, diameter constraint, series-parallel transformation, factoring method, cumulative estimation

Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (код проекта14-07-31069).

Введение. В данной статье исследуются сети, элементы которых подвержены случайным отказам. Обычно в подобных случаях сеть моделируется графом, для каждого элемента которого задана вероятность его присутствия [1]. Одним из основных показателей надежности такой сети является вероятность связности заданного подмножества узлов (терминалов). Данный показатель достаточно хорошо изучен, существует много различных точных и приближенных методов его расчета [1, 2, 3, 4, 5, 6]. Необходимость расчета и оценки показателей надежности возникает, прежде всего, при структурной оптимизации сетей [7] как на этапе проектирования, так и при расширении существующих структур.

Однако на практике часто требуется обеспечить не просто существование пути между каждой парой целевых узлов, а существование пути, проходящего по ограниченному числу звеньев. Например, если T — ограничение на время передачи данных между двумя узлами, то количество транзитных узлов, участвующих в передаче данных, не должно превышать T/t, где t — требуемое время для обработки данных на каждом узле сети. Ряд протоколов накладывают ограничение на количество пересылок для каждого пакета данных, чтобы избежать заторов из-за зацикливания, например, это некоторые peer-to-peer сети, такие как Freenet, Gnutella и др. При передаче данных в беспроводных сенсорных сетях [8, 9] также важное значение имеет ограничение на количество транзитных узлов. Для анализа надежности таких сетей был предложен другой показатель — вероятность связности сети с ограничением на диаметр [10], т. е. вероятность того, что любые два полюса сети соединены путем, состоящим из ограниченного числа звеньев. Данная характеристика для заданного графа $G(V, E)$, терминального множества $K \subset V$ и диаметра D обозначается как $R_K(G, D)$ или $R_K^D(G)$.

Вычислительная сложность задачи для конкретных значений $k = |K|$ и D хорошо изучена [13]. В табл. 1 приведены значения вычислительных сложностей для различных значений k и D:

Одна из основных причин, делающих расчет надежности с ограничением на диаметр существенно более трудоемким по сравнению с другими показателями сетевой надежности, — это отсутствие методов снижения количества рекурсий. Например, для расчета классической надежности сети, являющейся частным случаем K-терминальной надежности сети с ограничением на диаметр, ко-

Таблица 1

Сложность вычислений для различных значений диаметра D и количества терминалов $k = |K|$

		$\longleftarrow$ k $\longrightarrow$		
		2	$3 \dots n-1$	n
	2	O(n) - [10]	O(n) - [14]	$\mathcal{NP}$-hard - [13]
$\uparrow$	3			
d	$\vdots$	$\mathcal{NP}$-hard - [10]		$\mathcal{NP}$-hard - [13]
$\downarrow$	$n-2$			
	$\geq n-1$	$\mathcal{NP}$-hard - [15]		$\mathcal{NP}$-hard - [16]

гда терминальное множество совпадает со множеством вершин, а ограничение на диаметр отсутствует, используются методы редукции, декомпозиции, направленное ветвление и другие методы, которые еще не адаптированы или в силу различных причин не могут быть применены для расчета $R_K^d(G)$.

В данной работе предлагаются новые методы для ускорения точного расчета надежности сети с ограничением на диаметр: редукция ребер и направленное ветвление.

В работе [3] представлен метод кумулятивных оценок значения классической надежности сети, обобщенный и примененный нами для уточнения оценок значения $R_K^D(G)$.

Основные определения и обозначения. Будем представлять сеть с ненадежными звеньями с помощью неориентированного случайного графа $G = (V, E)$, для каждого ребра $e = (s, t)$ которого задана вероятность его присутствия в графе $r_e = r_{(s,t)}$, что соответствует надежности звена. Задано выделенное множество вершин $K \subset V$ — терминалов, соответствующих узлам сети, для которых необходимо обеспечить возможность устанавливать соединение друг с другом.

Задача расчета K-терминальной надежности графа $G = (V, E)$ с ограничением на диаметр D, $R_k(G, D)$, определяется как вероятность того, что для любой пары вершин $s, t \in K$ существует связывающий их надежный путь длины не более D ребер.

Метод факторизации. Наиболее широко используемый метод вычисления надежности сети — метод факторизации, основанный на

рассмотрении двух возможных состояний выбранного канала e: если ребро отказало, его можно удалить из сети, если оно надежно, то может быть стянуто. В случае K-терминальной надежности с ограничением на диаметр формула факторизации для графа $G = (V, E)$ принимает следующий вид:

$$R_K(G, D) = r_e \times R_K(G\backslash e, D) + (1 - r_e) \times R_K(G/e, D), \qquad (1)$$

где G/e — сеть, образованная из G удалением ребра e из E; $G\backslash e$ — сеть, в которой вероятность ребра e устанавливается равной 1.

Множества вершин V и терминалов K остаются теми же. Рекурсии продолжаются либо до получения несвязного графа, либо до получения графа малой размерности, для которого надежность можно рассчитать непосредственно.

В работе [10] предложен модифицированный метод ветвления для расчета надежности сети, который работает существенно быстрее, чем вышеописанный классический метод факторизации в случае с ограничением на диаметр. Основная идея состоит в переходе от графов к спискам путей P. На начальном этапе формируется список всех путей, длина которых не превышает D для каждой пары полюсов. Это автоматически исключает из рассмотрения все ребра, через которые не проходит ни один такой путь, например, так называемые "прикрепленные деревья" без полюсов. Для каждого оставшегося ребра e формируется список $P(e)$ всех содержащих его путей. При рекурсивных вызовах процедуры в качестве параметров не передаются подграфы, вместо них передается шесть параметров, описывающих состояние соответствующего подграфа с точки зрения путей из P.

Редукция ребер. Одним из эффективных методов ускорения расчета $R_K(G)$ является последовательно-параллельное преобразование, удаляющее из графа вершины степени 2 [1, 3, 4] и замещающее параллельные ребра, которые могут в результате возникнуть, одним ребром. В случае с ограничением на диаметр возможно объединение двух ребер, инцидентных вершине степени 2, при условии, что она не является полюсом. На рис. 1 изображена такая ситуация. Любой путь, связывающий два каких-либо полюса и проходящий через ребро (s, v), будет обязательно проходить и через ребро (v, t), так как v — не полюс. Следовательно, $P(s, v) = P(v, t)$ и ребра (s, v) и (v, t) могут быть заменены на ребро (s, t) с надежностью $r_{(s,v)} \times r_{(v,t)}$.

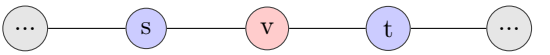


Рис. 1. Редукция ребер, инцидентных вершине степени 2

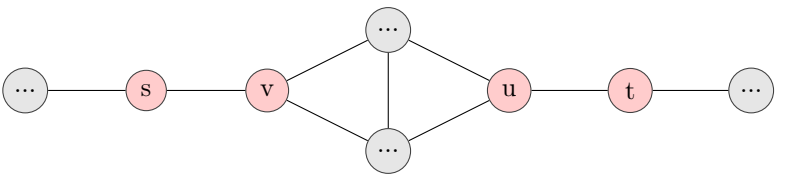


Рис. 2. Редукция ребер в общем случае

Параллельное преобразование в данном случае нереализуемо. Предположим, что вершины s и t были связаны ребром f. Тогда для вновь образованного ребра $g = (s, v)$ в общем случае неверным будет равенство $P(g) = P(f)$, так как через f может проходить большее число путей ограниченной длины между всевозможными парами полюсов, чем через g.

Если формирование списков путей $P(e)$ осуществлять после преобразования, необходимо учесть, что образованное ребро внесет больший вклад в длину любого проходящего через него пути. Если преобразование осуществлять после формирования списка, то нужно просто оставить в нем одно из двух совпадающих множеств, например $P(s, v)$.

В общем случае вместо вершины v может быть произвольный подграф без полюсов (рис. 2). Тогда любой путь, связывающий два каких-либо полюса и проходящий через ребро (s, v), будет обязательно проходить через ребро (u, t). Следовательно, остается верным равенство $P(s, v) = P(u, t)$ и можно совершать ветвление сразу по этим двум ребрам.

Таким образом, на предварительном этапе появляется возможность редуцировать цепи произвольной длины в одно ребро и объединять пары ребер в одно при определенных условиях, что уменьшает количество рекурсий и, соответственно, время работы алгоритма.

Упорядочивание ребер для направленного ветвления. Метод факторизации будет правильно работать при любом выборе разрешающего ребра, однако вопрос оптимального выбора ребра, ускоряющий время работы алгоритма, остается открытым [10]. В неко-

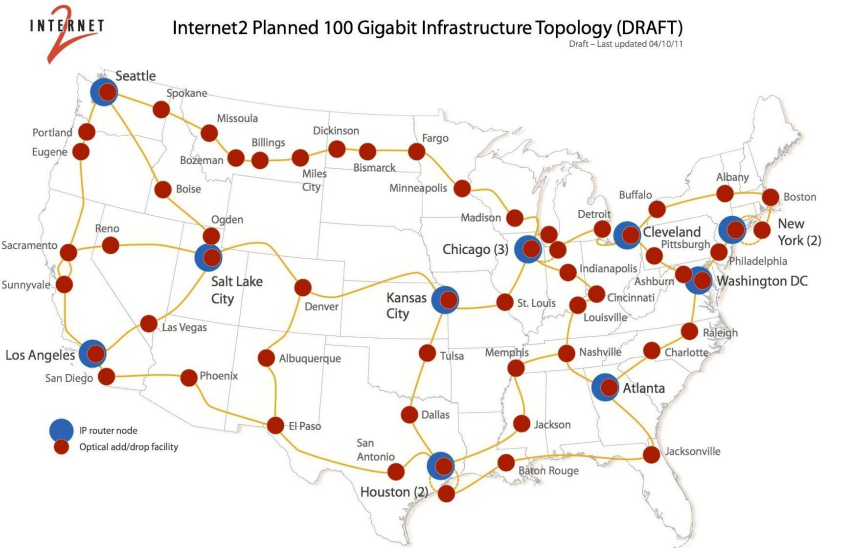


Рис. 3. Структура научно-образовательной сети Internet2

торых случаях упорядочивание ребер для направленного ветвления существенно влияет на количество рекурсий.

Предлагается упорядочивать ребра по критерию максимума путей из P, содержащих эти ребра. Другими словами, при очередном выборе ребра для ветвления предпочтение отдается ребру, которое входит в наибольшее количество путей из P. В большинстве случаев это дает ускорение расчета, но все же не является оптимальным.

Результаты численных экспериментов. Предложенные методы ускорения расчета надежности были протестированы на структуре научно-образовательной сети Internet2, изображенной на рис. 3 (данный рисунок имеется в свободном доступе по адресу http://www.extremetech.com/computing/98283-the-next-internet-a-quest-for-speed/2). В графе сети 58 вершин и 67 ребер. В качестве полюсов естественным образом были выбраны маршрутизаторы сети (IP router nodes), значение диаметра полагалось равным $15, 20, 25$.

За основу был взят модифицированный метод ветвления Fact [10], который был улучшен тремя способами: сортировкой ребер(Fact + sort), редукцией ребер (Fact + red) и сортировкой и редукцией ребер

Таблица 2

Алгоритм	$D = 15$		$D = 20$		$D = 25$	
	t	rec	t	rec	t	rec
Fact	>24ч	-	>24ч	-	>24ч	-
Fact + sort	>24ч	-	>24ч	-	>24ч	-
Fact + red	8с	109798	87с	1462308	8м	9581810
Fact + sort + red	4с	38016	36с	699552	2м	1935311

(Fact + sort + red). В таблице приведено время расчета надежности данной структуры для каждого из тестируемых алгоритмов (t), а также количество рекурсий (rec), т. е. количество вызовов процедуры факторизации в процессе расчета. Для расчетов использовался компьютер с четырехъядерным процессором AMD A8-450M 1,9 GHz.

Как видно из табл. 2, наиболее существенный вклад в ускорение расчета надежности данной сети с ограничением на диаметр вносит редукция ребер, а использование сортировки позволяет дополнительно ускорить расчет.

Уточнение оценок надежности. В работе [3] предложен алгоритм, позволяющий обновлять верхний и нижний пределы значения классической надежности сети в процессе ветвления.

Предположим, что заданная сеть G в процессе факторизации приведена в L подсетей $G_1, \ldots, G_L$, таких что для каждой легко вычисляется надежность, т. е. если P_l для $l \in \{1, \ldots, L\}$ — вероятность получения реализации G_l, то $\sum_{l=1}^{L} P_l = 1$ и надежность G может быть представлена в виде

$$R(G) = \sum_{l=1}^{L} P_l R(G_l),$$

где $R(G_l)$ - надежность подсети G_l. Из этого выражения легко можно сделать следующий вывод: для каждого $k \in \{1, \ldots, L\}$ выполняется неравенство

$$\sum_{l=1}^{k} P_l R(G_l) \le R(G) \le 1 - \sum_{l=1}^{L} P_l (1 - R(G_l)),$$

что верно, так как $R(G) = 1 - \sum_{l=1}^{L} P_l (1 - R(G_l))$.

Из этого вывода и следует алгоритм обновления нижней и верхней границ надежности $R(G)$. Обозначим через RL_l и RU_l нижнюю и верхнюю границы, обновленные для $G_1, \ldots, G_L$, и установим значения RL_0 и RU_0 равными нулю и единице, соответственно. Каждый раз, когда вычисляется надежность для G_l, алгоритм обновляет значения RL_l и RU_l:

$$RL_l = RL_{l-1} + P_l R(G_l),$$
$$RU_l = RU_{l-1} - P_l(1 - R(G_l)).$$

Как только l увеличивается, RL_l и RU_l приближаются к $R(G)$. Как только RL_l или RU_l достигают заранее заданного R_0, выполнение алгоритма заканчивается. Таким образом можно заранее установить такое допустимое для нас значение R_0, чтобы остановить работу алгоритма, не дожидаясь вычисления точного значения, например, если необходимо, чтобы сеть была не менее надежна, чем R_0.

В случае, когда этот принцип применяется к алгоритму Cancela и Petigi, роль итоговых реализаций сетей $G_1, \ldots, G_L$ будут играть те случаи, когда значение G_l устанавливается как $G_l \leftarrow RContract \leftarrow 1$ или $G_l \leftarrow RDelete \leftarrow 0$. Существует необходимость каждый раз отправлять параметр — значение P_l, которое устанавливается равным 1 перед началом работы алгоритма, и каждый раз умножается на r_e или $1 - r_e$ перед уходом в рекурсию. Далее представлен псевдокод алгоритма, обновляющего оценки надежности.

Эксперименты уточнения оценок. Алгоритм уточнения оценок надежности был протестирован на структуре глобальной IP-сети Intellinet, изображенной на рис. 4 (данный рисунок имеется в свободном доступе). В графе сети 47 вершин и 69 ребер. В качестве полюсов были выбраны приватные узлы (Private NAPs).

На рис. 5, 6 показаны графики уточнения значения надежности сети Intellinet для значений диаметра 15 и 20.

Заключение. Разработан алгоритм, вычисляющий K-терминальную надежность сети с ограничением на диаметр, уточняющий оценки надежности, и написана программа, реализующая этот алгоритм. Предложен и экспериментально подтвержден аналог известного последовательно-параллельного преобразования в вычислении надежности сети для случая с ограничением на ее диаметр. Изучалась проблема влияния упорядочивания ребер для выбора в качестве разрешающих при расчете надежности сети с ограничением на

```
1  Function FACTO(Input, P_l)
2  |  if nowT − startT > limitT or RL > fixVal or RU < fixVal
   |  then
3  |  |  return
4  |  end
5  |  e ← случайное ребро : 0 < r_e < 1
6  |  contractEdge(Input, e, P_l) // случай, отвечающий
   |  стягиванию ребра
7  |  deleteEdge(Input, e, P_l) // случай, отвечающий удалению
   |  ребра
8  end
9  Function contractEdge(Input, e, P_l)
10 |  P_l ← P_l * r_e
11 |  foreach p = (s, ..., t) in P(e), где feasible(p) = true do
12 |  |  links(p) ← links(p) − 1
13 |  |  if connected(s, t) = false and links(p) = 0 then
14 |  |  |  connected(s, t) ← true
15 |  |  |  connectedPairs ← connectedPairs + 1
16 |  |  |  if connectedPairs = k×(k−1)/2 then
17 |  |  |  |  RL ← RL + P_l
18 |  |  |  |  return
19 |  |  |  end
20 |  |  end
21 |  end
22 |  FACTO(Input, P_l)
23 end
24 Function deleteEdge(Input, e, P_l)
25 |  P_l ← P_l * r_e
26 |  foreach p = (s, ..., t) in P(e), где feasible(p) = true do
27 |  |  feasible(p) ← false
28 |  |  np(s, t) ← np(s, t) − 1
29 |  |  if np(s, t) = 0 then
30 |  |  |  RU ← RU − P_l
31 |  |  end
32 |  end
33 |  FACTO(Input, P_l)
34 end
```

Рис. 4. Структура глобальной IP сети Intellinet

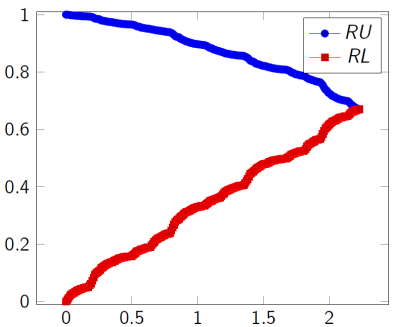


Рис. 5. $d = 15$, $P_e = 0.7$, $T \approx 12,31$ часов

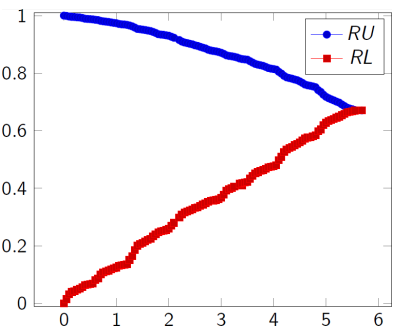


Рис. 6. $d = 20$, $P = 0.7$, $T \approx 59,5$ часов

диаметр методом ветвления. Обобщен на случай K-терминальной надежности метод кумулятивных оценок значений надежности сети и исследован при помощи экспериментов.

Цель дальнейшей работы — параллельная реализация нахождения точного и оценочного значения надежности с ограничением на диаметр. Также предполагается рассмотрение более сложных приемов редукции размерности задачи, в том числе путем декомпозиции сети, и дальнейшее изучение выбора разрешающего ребра для проведения факторизации.

Научный руководитель — к.ф.-м.н. Д. А. Мигов

Список литературы

[1] Colbourn Ch. J. The combinatorics of network reliability. N.Y.: Oxford Univ. press, 1987.

[2] Satyanarayana A., Chang M.K. Network reliability and the factoring theorem // Networks. 1983. V. 13. P. 107-120.

[3] Won J.-M., Karray F. Cumulative update of all-terminal reliability for faster feasibility decision // IEEE trans. on reliability. 2010. V. 59, № 3. P. 551–562.

[4] Rodionov A., Migov D., Rodionova O. Improvements in the efficiency of cumulative updating of all terminal network reliability // IEEE Transactions on Reliability. 2012. V. 61, № 2. P. 460-465.

[5] Родионов А.С. К вопросу ускорения расчета коэффициентов полинома надежности случайного графа // Автоматика и телемеханика. 2011. № 7. С. 134-146.

[6] Цициашвили Г.Ш., Осипова М.А., Лосев А.С. Асимптотика вероятности связности графа с низконадежными ребрами // Прикладная дискретная математика. 2013. № 1(19). С. 93-98.

[7] Нечунаева К.А. Оптимальное по показателям связности объединение сетей в условиях структурных и стоимостных ограничений // Пробл. информ. 2010. № 2. С. 18-26.

[8] Shakhov V., Choo H. Reliability of wireless sensor network with sleeping nodes // Lect. Notes On Comput. Sci. 2007. V. 4490. P. 530-533.

[9] Shakhov V. Protecting Wireless Sensor Networks from Energy Exhausting Attacks // Lect. Notes on Comput. Sci. 2013. V. 7971. P. 184-193.

[10] Cancela H., Petingi L. Diameter constrained network reliability: exact evaluation by factorization and bounds // Proc. of the Intern. conf. on industrial logistics. Japan, 2001. P. 359-356.

[11] Мигов Д. А. Расчет надежности сети с ограничением на диаметр с применением точек сочленения // Автоматика и телемеханика. 2011. № 7. С. 69–74.

[12] Migov D., Rodionov A. Decomposing graph with 2-node cuts for diameter constrained network reliability calculation // Proc. of the 7th Intern. conference on ubiquitous information management and communication (ACM ICUIMC 2013). Kota Kinabalu, Malaysia, 2013. ACM New York, USA. Article No. 39. 6 p.

[13] Canale E., Romero P. Diameter constrained reliability : Computational complexity in terms of the diameter and number of terminals. 2014.

[14] Canale E., Cancela H., Robledo F., Rubino G., Sartor P. On computing the 2-diameter-constrained K-reliability of networks // Intern. Transactions in Operational Research. 2013. V. 20. P. 49-58.

[15] Rosenthal A. Computing the reliability of complex networks // SIAM J. on Appl. Math. 1977. V. 32. P. 384-393.

[16] Provan J., Ball M. The complexity of counting cuts and of computing the probability that a graph is connected // SIAM J. Comput. 1983. V. 12. P. 410-421.

[17] Мигов Д. А., Нестеров С. Н., Родионов А. С. Методы ускорения расчета надежности сетей с ограничением на диаметр // Вестн. СибГУТИ. № 1, 2014. С. 49-56.

Нестеров Сергей Николаевич — магистрант
Новосибирского государственного университета;
e-mail: cepera_666@inbox.ru

ТРУДЫ КОНФЕРЕНЦИИ МОЛОДЫХ УЧЕНЫХ
Новосибирск, 2014 С. 76–85

Реализация модели многочастичного газа FHP-MP на гибридном кластере

А. С. Подстригайло
Новосибирский государственный университет

УДК 004.94

Разработан и реализован параллельный алгоритм модели FHP-MP, позволяющий вести расчет задачи на мультикомпьютерах с графическими ускорителями (гибридном кластере). Проведены оценки наличия дисбаланса вычислений, предложены и реализованы алгоритмы динамической балансировки нагрузки как между графическими ускорителями одного вычислительного узла, так и между разными вычислительными узлами.

Developed and implemented a parallel algorithm FHP-MP model, allows for a calculation on multicomputers graphics accelerators (hybrid cluster). Estimates of having an imbalance calculations proposed and implemented algorithms for dynamic load balancing between both graphics cards one computing node, and between different computing nodes.

Ключевые слова: клеточный автомат, графический ускоритель, параллельная реализация, балансировка.

Keywords: cellular automata, GPU, parallel implementation, balancing.

Введение. Для имитационного моделирования потоков жидкости и газа в 70-х годах прошлого века было разработано семейство дискретных клеточно-автоматных моделей Lattice Gas [1]. Эти модели имеют ряд ограничений, в частности, граничные условия позволяют задавать только неподвижные твердые объекты (стенки), моделирование околозвуковых скоростей влечет искажение результата и т. д. Для решения этих проблем в ИВМиМГ СО РАН была

Работа выполнена при финансовой поддержке гранта Президента РФ для государственной поддержки молодых российских ученых (номер гранта МК-3644.2014.9).

предложена новая КА модель, названная FHP-MP, подробно описанная в работе [2]. Данная модель является обобщением классической булевой модели FHP, в которой допускается более одной частицы по каждому направлению скорости. Приведем краткое описание модели: пространство моделирования представляет собой гексагональную решетку, состояние каждой клетки представлено вектором целых чисел, частицы покоя в рассматриваемой модели отсутствуют. Процесс моделирования итеративный. Каждая итерация состоит из двух фаз: столкновения и движения. На фазе сдвига каждая частица, обладающая ненулевой скоростью, перемещается в смежную ячейку, соответствующую направлению ее скорости; при этом скорость частицы в новой ячейке остается прежней. На фазе столкновения необходимо случайно равновероятно выбрать одно из состояний вектора, такое что выполняются законы сохранения массы и импульса частиц. Характерной особенностью модели является большая вычислительная сложность фазы столкновения. В то же время, данный алгоритм обладает высоким уровнем параллелизма, что позволяет, в частности, использовать модель вычислений на GPU. Данная работа посвящена параллельной реализации модели FHP-MP на мультикомпьютере с графическими ускорителями (гибридном кластере). Описываются алгоритмы динамической балансировки нагрузки как между графическими ускорителями одного вычислительного узла, так и между разными вычислительными узлами.

1. Постановка задачи. Рассматриваемая задача разбивается на несколько этапов:

1. Реализация модели FHP-MP непосредственно на одном графическом ускорителе;

2. Анализ степени дисбаланса вычислений на графическом ускорителе в рамках данной модели;

3. Реализация модели FHP-MP на одном вычислительном узле кластера с несколькими графическими ускорителями;

4. Разработка и реализация алгоритма балансировки нагрузки между видеокартами одного вычислительного узла кластера;

5. Реализация модели FHP-MP на нескольких вычислительных узлах кластера;

6. Разработка и реализация алгоритма балансировки нагрузки между графическими ускорителями разных вычислительных узлов кластера.

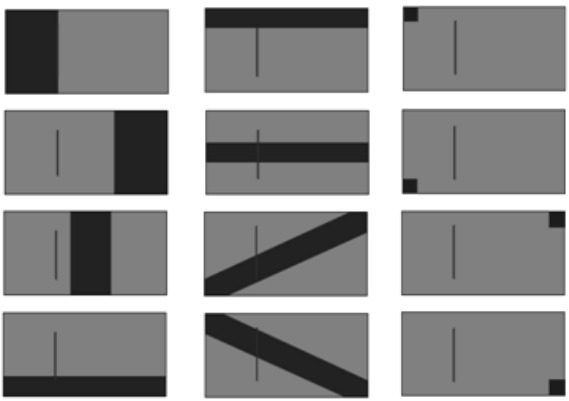


Рис. 1. Серия экспериментов 1-го типа. Темным цветом показаны области с плотным заполнением частицами, светлым — с разряженным. Вертикальной чертой обозначена отражающая стенка.

2. Параллельная реализация модели FHP-MP на одном графическом ускорителе. Для проверки существования дисбаланса в рамках одной видеокарты было реализовано два алгоритма столкновений разной сложности: кубической [3] и линейной [4]. Поскольку штатные средства профилировки не позволяют изучить загрузку мультипроцессоров видеокарты по отдельности, для определения степени дисбаланса загрузки мультипроцессоров был поставлен следующий вычислительный эксперимент:

— Рассматривались поля размером 1024×512, 2048×1024, 4096×2048 клеток;

— На расстоянии четверти поля от левого края была поставлена отражающая стенка, не доходящая до верха и низа поля, — таким образом, частицы способны обходить ее;

— Верхняя и нижняя границы поля отражающие, правая и левая — проникающие, т. е. частицы, вылетающие за правую границу прилетают с левой, и наоборот.

Для сравнения поставлены серии экспериментов двух типов. В первом случае на поле выделялась область с “плотным” заполнением, по каждому направлению в этой клетке вылетало по 1, 10 или 100 частиц (т. е. всего 6, 60 или 600 частиц в клетке), в зависимости от эксперимента; в оставшейся области в каждой клетке в трех случайных направлениях разлеталось по одной частице (рис. 1).

Таблица 1

Численные результаты эксперимента 1-го типа,
случай с плотным заполнением левой четверти массива
(Среднее время обсчета клетки (область 1024×512), мкс)

N	CPU-N3	CPU-N	GPU-N3	GPU-N
6	0,243	0,115	0,012	0,008
60	30,351	0,222	2,505	0,012
600	—	1,196	—	0,024

Таблица 2

Численные результаты эксперимента 2-го типа
(Среднее время обсчета клетки (область 1024×512), мкс)

M	CPU-N3	CPU-N	GPU-N3	GPU-N
3	0,200	0,114	0,009	0,008
17	5,408	0,236	0,328	0,011
152	—	1,215	—	0,020

В экспериментах второго типа область равномерно заполнялась частицами, разлетающимися в случайных направлениях, так, чтобы их суммарное количество соответствовало общему числу частиц в эксперименте первого типа.

Приведем численные результаты экспериментов, рассчитанных на графическом ускорителе GTX 680, и сравним их с последовательной реализацией тех же алгоритмов на процессоре Core i7. Хотя общее время обсчета модели при переходе к большей области закономерно возрастает, время обсчета одной клетки существенно не меняется, поэтому приведем результаты для области 1024×512 клеток.

В табл. 1, 2 N — количество частиц в области с плотным заполнением (в данном случае представлены результаты эксперимента при плотном заполнении четверти поля слева); — количество частиц в каждой клетке при равномерном распределении суммарного количества частиц по области. Прочерки в таблице объясняются тем, что общее время расчета на CPU для кубического алгоритма на 50 итерациях уже для 60 частиц составляет примерно 15 мин., поэтому расчеты для 600 частиц с точки зрения анализа производительности и наличия дисбаланса были признаны нецелесообразными; соответствующие результаты для версии GPU также не вычислялись.

Таблица 3

Отношение времени вычислений экспериментов
1-го и 2-го типов
(Среднее время обсчета клетки (область 1024×512), мкс)

CPU-N3	CPU-N	GPU-N3	GPU-N
1,216	*1,006*	1,285	*1,007*
5,613	*0,939*	7,631	*1,105*
—	*0,984*	—	*1,197*

Рассматривая отношения времени вычисления первого эксперимента ко второму (табл. 3), можно заметить, что эти отношения для CPU и GPU (выделено курсивом) почти не различаются, что говорит о том, что в эксперименте первого типа на GPU дисбаланс вычислений отсутствует: на CPU дисбаланс, очевидно, отсутствует ввиду последовательной реализации, а если бы присутствовал дисбаланс на GPU, то эксперимент 1-го типа ускорился бы в разы меньше, чем 2-го, что на практике не реализуется. Аналогичные результаты были получены для остальных размеров полей и экспериментов серии 1-го типа. Таким образом, было установлено, что в случае модели FHP-MP при вычислениях на одной видеокарте не наблюдается дисбаланса вычислительной нагрузки, т. е. планировщик блоков потоков справляется с распределением задачи по вычислительным ядрам видеокарты самостоятельно, и дополнительная балансировка нагрузки в рамках одного ускорителя не требуется. Кроме того, для дальнейшего развития модели был выбран алгоритм столкновений с линейной сложностью.

3. Параллельная реализация модели FHP-MP на узле гибридного кластера. В табл. 4 представлены результаты расчета модели, расширенной для выполнения на нескольких видеокартах в рамках одного узла кластера НКС-30Т+GPU. Нижняя строчка таблицы показывает, что использование всех трех видеокарт дает существенный прирост производительности. Однако существует ряд экспериментов, в которых основная вычислительная нагрузка приходится на одну видеокарту. В этом случае, как показано в табл. 5, прирост производительности от использования дополнительных графических ускорителей не наблюдается.

Таблица 4

Численные результаты эксперимента 1-го типа
(Среднее время обсчета клетки (область 4096×2048), мкс)

N	CPU-N	GPU-N		
		Fermi x1	Fermi x2	Fermi x3
6	0,149	0,009	0,005	0,004
60	0,289	0,011	0,006	0,005
600	1,673	0,042	0,021	0,014

Таблица 5

Численные результаты эксперимента 1-го типа для области
с плотным заполнением частиц в нижней четверти поля
(Среднее время обсчета клетки (область 4096×2048), мкс)

N	GPU-N		
	Fermi x1	Fermi x2	Fermi x3
6	0,007	0,005	0,004
60	0,010	0,008	0,006
600	0,038	0,036	0,034

Для компенсации этого факта реализован следующий алгоритм динамической балансировки между видеокартами. Во время выполнения итерации подсчитывается суммарное количество частиц по всем направлениям в каждой строке M (рис. 2). На основе этих данных можно рассчитать теоретическое "эталонное" количество частиц Мт, равное M деленное на количество графических ускорителей в узле, которое нужно отдать каждой видеокарте для балансировки нагрузки. Опираясь на эту границу и зная, сколько частиц содержит каждая строка, алгоритм отдает для вычисления каждой видеокарте такое число последовательных строк, чтобы разница между теоретическим Мт и практическим Мп была минимальна.

В табл. 6 представлены численные результаты того же эксперимента, что в табл. 5, с условием включенной динамической балансировки. Видно, что алгоритм хорошо компенсирует замеченный дисбаланс нагрузки между графическими ускорителями в узле.

4. Параллельная реализация модели FHP-MP на нескольких узлах гибридного кластера. В реальных задачах газо- и гидродинамики используются комплексные модели большой слож-

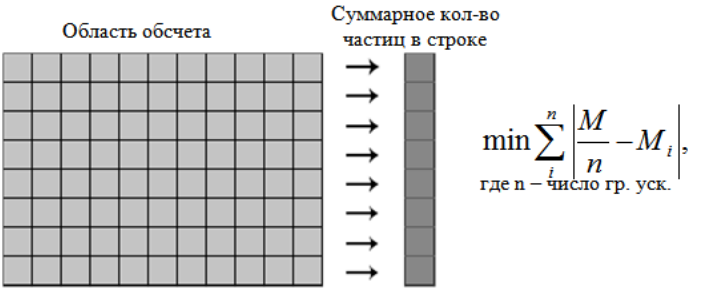


Рис. 2. Подсчет частиц по направлениям. Все частицы, находящиеся в строке массива направления (светлый прямоугольник), суммируются в соответствующую ячейку общего вектора частиц (темный столбец)

Таблица 6

Численные результаты эксперимента 1-го типа для области с плотным заполнением частиц в нижней четверти поля, с балансировкой (Среднее время обсчета клетки (область 4096×2048), мкс)

N	GPU-N		
	Fermi x1	Fermi x2	Fermi x3
6	0,007	0,005	0,004
60	0,010	0,008	0,006
600	0,038	0,024	0,014

ности, и при прохождении волны между разными узлами также может наблюдаться дисбаланс нагрузки, поэтому для предотвращения этого был введен дополнительный слой балансировки между соседними узлами кластера. Поскольку постоянное перевыделение памяти влечет большие накладные расходы и замедляет алгоритм, было решено изначально выделять в видеопамяти область большего размера, с запасом места в так называемых “карманах” по бокам, равным по ширине половине расчетной области каждый (рис. 3).

В случае полного заполнения “кармана” возможны три стратегии поведения:

— если один из карманов свободен, сдвинуть расчетную область и перераспределить свободное место поровну между карманами;

— временно запретить использование данного узла для балансировки;

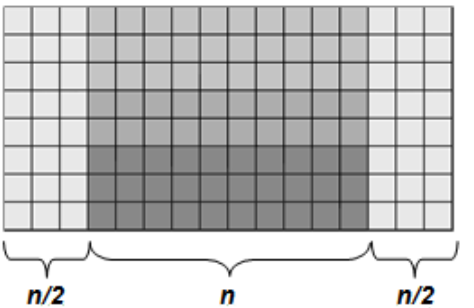


Рис. 3. Схема выделения памяти с “карманами”

— если возможно, перевыделить в видеопамяти массив большего объема, поскольку в карманах изначально заложено достаточно свободного пространства, такие ситуации будут редки и не слишком скажутся на производительности алгоритма.

Балансировщик по времени расчета итерации определяет загруженные узлы, и если разница времен расчета больше заданного порога, вычисляет направление, в котором необходимо провести балансировку нагрузки, и выставляет каждому узлу флаги приема или передачи данных в зависимости от роли узла. Балансировка происходит децентрализованно, локально между соседними узлами. В начале балансировки все узлы, используя массовую рассылку, получают информацию о вертикальных границах своих соседей, таким образом, загруженный узел может легко определить размер доступного ему в узле-реципиенте свободного места в кармане и выслать столько столбцов, чтобы гарантированно не выйти за границы области (в текущей реализации алгоритма этот параметр равен четверти ширины кармана принимающей стороны). В свою очередь, каждый узел, выполнив успешную проверку флагов приема, может определить, от какого узла ждать сообщения и какого размера. Поскольку линейный алгоритм столкновений менее подвержен дисбалансу, для проверки эффективности работы балансировщика нагрузки между узлами в условиях, приближенных к сложности реальных моделей, проводилось тестирование модели с алгоритмом столкновения кубической сложности (N3). В табл. 7 приведены численные результаты эксперимента без дополнительной балансировки, в табл. 8 — в случае, когда балансировка между узлами включена.

Таблица 7

Численные результаты эксперимента 1-го типа для области с плотным заполнением частиц в левой четверти поля, без балансировки (Среднее время обсчета клетки (область 4096×2048), мкс)

N	GPU-N3		
	Node x1	Node x2	Node x3
6	0,004	0,002	0,002
60	0,082	0,076	0,070
120	0,485	0,463	0,440

Таблица 8

Численные результаты эксперимента 1-го типа для области с плотным заполнением частиц в левой четверти поля, с балансировкой (Среднее время обсчета клетки (область 4096×2048), мкс)

N	GPU-N3		
	Node x1	Node x2	Node x3
6	0,004	0,002	0,002
60	0,071	0,064	0,051
120	0,417	0,397	0,321

При использовании данной реализации алгоритма балансировки достигается ускорение расчета задачи порядка 15–30 % в случае алгоритма столкновения кубической сложности. Для большей эффективности реализации в случае практических задач со сложностью порядка O(N2) или O(N3) необходима дальнейшая оптимизация алгоритма балансировки нагрузки между узлами.

Заключение. В работе представлена реализация модели FHP-MP для мультикомпьютера с несколькими графическими ускорителями (гибридном кластере). Были разработаны и реализованы алгоритмы динамической балансировки нагрузки для данной модели как между графическими ускорителями в одном вычислительном узле, так и между различными вычислительными узлами, которые позволяют компенсировать возникающий в процессе вычислений дисбаланс нагрузки.

Научный руководитель — к.ф.-м.н. К. В. Калгин Константин Викторович.

Список литературы

[1] Hardy J., Pomeau Y., de Pazzis O. 2D Lattice-Gas model // J. Math. Phys. 1973. N 14. P. 1746.

[2] Медведев Ю. Г. Многочастичная клеточно-автоматная модель потока жидкости FHP-MP // Вестн. Том. гос. ун-та. Сер.: Управление, вычислительная техника и информатика. 2009. № 1(6). С. 33–40.

[3] Kalgin K. Optimization of transition rule computation algorithm in multiparticle lattice gas. // Proc. of Combinatorics. 2010. P. 148.

[4] Дубовик А. С. Алгоритмы выбора нового состояния клетки в многочастичной конечно-автоматной модели FHP-MP // Труды конф. "Новые информационные технологии в исследовании сложных структур", 2012.

Подстригайло Алена Сергеевна — магистрант Новосибирского государственного университета;
e-mail: dina119@ya.ru

ТРУДЫ КОНФЕРЕНЦИИ МОЛОДЫХ УЧЕНЫХ
Новосибирск, 2014 С. 86–94

Разработка среды визуального конструирования параллельных программ

В. Г. Сарычев

Новосибирский государственный технический университет

УДК 004.4’2

Рассматривается подход к разработке предметно-ориентированных систем для визуального конструирования параллельных программ на основе формального представления знаний о предметной области в виде вычислительной модели. Представлен проект и прототип системы визуального конструирования параллельных программ, позволяющей конструировать простые вычислительные модели и линейные алгоритмы в визуальном редакторе, сохранять их в текстовом представлении в файлах.

The paper studies an approach to develop domain specific systems for visual construction of parallel programs based on the representation of domain knowledge in the form of computational models. A design and a prototype of a system for visual constuction of parallel programs is presented. The system allows a user to compose simple computational models and algorithms without loops, save them in a textual representation in files.

Ключевые слова: визуальное программирование, интегрированная среда разработки, простая вычислительная модель, конструирование алгоритмов, синтез программ, высокопроизводительные вычисления, суперкомпьютерные технологии, параллельное программирование, инструменты разработки программ.

Keywords: visual programming, integrated development environment, a simple computational model, construction of algorithms, synthesis programs, high performance computing, supercomputing technology, parallel programming, software development tools.

Введение. Настоящая работа выполняется в рамках проекта лаборатории Синтеза параллельных программ ИВМиМГ СО РАН по разработке системы накопления и вывода алгоритмов в различных предметных областях на основе вычислительных моделей, генерации программ на основе метода структурного синтеза [1].

Цель работы — создание системы визуального конструирования и визуализации алгоритмов и простых вычислительных моделей и обеспечение их представления для организации хранения и подачи алгоритмов на вход интерпретатора. В основе системы визуального конструирования лежит формальное описание предметной области в виде вычислительной модели [1]. В описании должна содержаться информация о том, какие объекты и с помощью каких преобразований могут быть получены из других объектов. Преобразования реализуются программными процедурами, которые извлекаются из библиотек, либо должны быть разработаны специально для данной модели.

Для передачи алгоритма интерпретатору среда визуального конструирования генерирует программу с необходимой реализацией объектов и операций и управлением на множестве операций в виде скрипт-файла, оставляя интерпретатору свободу для планирования параллельного исполнения операций, притом что некоторые операции могут быть структурно раскрыты для более глубокого планирования.

1. Постановка задачи. Структурный подход к синтезу программ предполагает накопление знаний о предметной области в виде вычислительной модели и последующий вывод алгоритмов и программ на вычислительных моделях. Разрабатываемый визуальный конструктор должен стать инструментом разработчика и пользователя вычислительной модели. Функциональность такого конструктора должна не только предоставлять удобные инструменты и средства для конструирования алгоритмов из уже готовых операций, но и давать возможность пользователю внедрять собственные процедуры и переменные, тем самым расширяя простую вычислительную модель.

Простая вычислительная модель представляет собой двудольный граф, в котором одно множество вершин — переменные, а другое множество вершин — операции, позволяющие по одним переменным "вычислять" другие. По мере развития знаний о предметной области модель наполняется переменными и связывающими их операциями. Алгоритм, описывающий порядок вычисления одних переменных из других, является вычислительной моделью. На рис. 1 показан пример простой вычислительной модели: множество

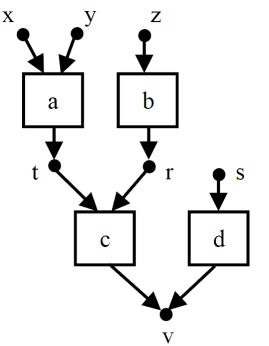


Рис. 1. Пример простой вычислительной модели

$V = \{x, y, z, t, r, s, v\}$ — переменные, множество $F = \{a, b, c, d\}$ — операции.

В качестве подграфов простая вычислительная модель может содержать множество алгоритмов. Алгоритмы, которые содержатся в простой вычислительной модели, могут быть автоматически извлечены из нее по заданным входным и выходным переменным. Могут быть реализованы как структурные операции (в этом случае требуется задание алгоритма их реализации), так и элементарные (указывается процедура из библиотеки).

Операции могут быть реализованы для параллельного исполнения на высокопроизводительных вычислительных системах и адаптированы под различные архитектуры. Для запуска на высокопроизводительных вычислительных системах необходимо реализовать интерфейсы взаимодействия (API) с этими системами [2].

2. Визуальный конструктор. Пусть заданы следующие определения для описания элементов визуального конструктора:

— *переменная* (типизированный объект, представляющий некоторую сущность предметной области);

— *операция* (характеризуется типом входных и выходных переменных и позволяет “вычислять” по входным переменным выходные);

— *стрелка* (соединяет переменные и операции в соответствии с характеризующими их типами и определяет направление передачи данных);

— *сцена* (рабочая область в границах которой пользователь конструирует алгоритм программы, размещая и редактируя объекты (переменные, операции, стрелки), используя доступные ему инструменты);

— *проект* (отображает текущее состояние разработки программы и служит контейнером для набора переменных, операций и стрелок);

— *обозреватель решений* (отображает доступные проекты в упорядоченном представлении в виде дерева и позволяет осуществлять быстрый доступ как к проектам, так и к их элементам. Служит контейнером для проектов);

— *скрипт-файл* (эквивалентная запись алгоритма с необходимой реализацией переменных и операций и управлением на множестве операций, понятная интерпретатору);

— *скрытые переменные* (переменные, которые не отображаются на сцене и не видны пользователю. К скрытым переменным относятся параметры, определяющие нефункциональные свойства реализации операций (например, "последовательная" , "параллельная для модели общей памяти" , "с использованием CUDA" и т. д.) и переменные, осуществляющие функцию "посредников", при соединении операций напрямую;

— *входные переменные* (подаются на вход первых операций (операции с которых начинается обработка данных в графе, имеющие глубину равную нулю; инициализируются пользователем).

Визуальный конструктор представляет собой среду разработки параллельных программ и предоставляет графический интерфейс, в котором есть необходимые инструменты для построения, редактирования и отладки алгоритмов, а также присутствуют соответствующие элементы управления для сборки и отправки сконструированных алгоритмов на вход интерпретатора. Есть возможности сохранения и загрузки конструируемых алгоритмов, пополнения библиотеки пользовательскими программными решениями и их модификации. В рамках интерфейса пользователь может создавать проект, содержащий более одного алгоритма, и динамически переключаться между этими алгоритмами. Для удобства управления проектами и содержащимися в них объектами используется обозреватель решений. Древовидная структура обозревателя дает пользователю возможность перемещаться между проектами и настраивать доступные параметры объектов.

Для конструирования алгоритмов доступны два режима. Используя первый режим, пользователь самостоятельно составляет алгоритм из переменных, операций и стрелок, находящихся в библиотеке. Во втором режиме, часть действий пользователя автоматизируется за счет использования структурного синтеза. Задаются множества входных и выходных переменных и передаются на выполнение специальной командой. В результате, если входных переменных достаточно для получения выходных, на рабочей области появляется необходимый набор операций и переменных для получения требуемого алгоритма.

3. Логика объектов интерфейса. Реализации операций могут отличаться друг от друга. С точки зрения среды визуального конструирования операция является либо элементарной, при этом она реализуется как некоторый внешний программный модуль и может быть разработана с использованием различных языков программирования (C/C++, Fortran и т. п), либо структурированной и заключать внутри себя некоторую вычислительную модель, которая может быть раскрыта или свернута в визуальном конструкторе. Также, возможны полиморфные переменные и операции. Например, пользователь указывает, что необходимо реализовать произведение матриц, но разреженная матрица или плотная, формат представления матриц, соответствующей операции могут быть либо определены автоматически из контекста, либо настраиваются пользователем.

В процессе конструирования алгоритма, в момент соединения операций и переменных стрелками, выполняется проверка правильности в смысле соответствия типов соответствующих входов и выходов. При несоответствии типов связь не устанавливается. В качестве подсказки пользователю, в момент соединения объектов, цветом подсвечиваются все входы, соответствующие данному типу. Исключается возможность запуска неправильно сконструированного алгоритма.

Для ускорения и простоты работы пользователя возможно соединение операций напрямую, при этом пользователь не создает дополнительных переменных, которые бы являлись “посредниками” между операциями. Такой подход позволяет, упростить визуальное представление программы и сделать ее алгоритм более наглядным, сохранив при этом всю ее структуру во внутреннем представлении.

4. Модель исполнения программ. Понятия и термины приведены в соответствии с [3].

Интерпретатор [4] получает на вход скрипт-файл, сформированный визуальным конструктором по алгоритму, построенному пользователем. Скрипт-файл оставляет свободу планирования выполнения операций интерпретатору.

Каждая операция имеет набор входных и выходных переменных. Некоторые операции в алгоритме являются информационно зависимыми, т. е. переменные, которые "вырабатывает" одна операция поступают на вход другой операции. Основную часть алгоритмов составляют транзитивно информационно зависимые операции, т. е. попарно информационно зависимые. Операция может быть выполнена, если ее входные переменные уже вычислены. Выполнение программы завершается, когда ни одна операция выполниться не может.

Операции, между которыми нет информационной зависимости, могут быть исполнены параллельно при наличии достаточного количества вычислительных ресурсов. Структурированные операции раскрываются при исполнении и независимые подоперации также могут выполняться параллельно. В свою очередь, реализация каждой из элементарных операций может предусматривать параллельное исполнение инструкций внутри операции. Для этого могут быть использованы обычные средства параллельного программирования такие как MPI, OpenMP.

5. Компоненты программного комплекса. Визуальный конструктор программ является частью программного комплекса для конструирования и выполнения параллельных программ на высокопроизводительных вычислительных системах.

Программный комплекс имеет модульную архитектуру, включающую следующие компоненты:

— *интерфейс пользователя* (модуль взаимодействия пользователя с программной системой, предоставляющий управление проектом и инструменты для работы с алгоритмами и их составляющими: операциями, переменными и стрелками);

— *логика объектов, представленных в интерфейсе* (модуль, содержащий набор правил и методов, обеспечивающих корректность конструируемых программ в смысле синтаксиса внутреннего представления алгоритмов);

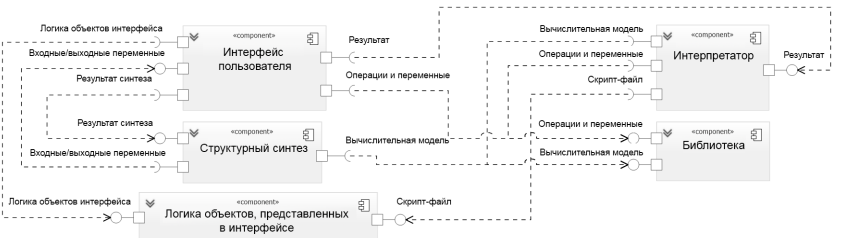


Рис. 2. Диаграмма компонентов системы

— *структурный синтез* (модуль, позволяющий синтезировать (получать) алгоритмы по набору входных и выходных переменных на основе простой вычислительной модели из библиотеки);

— *библиотека* (набор взаимосвязанных между собой переменных и операций описывающих предметную область и формирующих простую вычислительную модель);

— *интерпретатор* (устанавливает управление на множестве операций на основе информации о простой вычислительной модели, полученной из библиотеки, и осуществляет планирование исполнения алгоритма).

На рис. 2 изображена диаграмма взаимодействия компонентов программного комплекса в целом.

6. Связь программы с другими программами. Скрипт-файл отражает внутреннее представление сконструированного алгоритма и используется для передачи алгоритмов на вход интерпретатора.

Интерпретатор представляет собой отдельную программу и использует библиотеку операций и переменных из определенной предметной области. В общем случае, библиотека может содержать более одной предметной области и соответственно, более одной простой вычислительной модели. Библиотека предоставляет визуальному конструктору простые вычислительные модели, наборы операций и переменных и их описание.

Заключение. Разработана архитектура системы визуального конструирования и хранения алгоритмов и вычислительных моделей, в том числе: разработан язык внутреннего представления алгоритмов и вычислительных моделей, реализован прототип среды

визуального конструирования алгоритмов, обеспечивается проверка корректности связей элементов конструируемых алгоритмов на основе типизации, реализована возможность создавать, сохранять, загружать и модифицировать простые вычислительные модели и потоковые алгоритмы, добавлять алгоритмы в существующие простые вычислительные модели, также обеспечивается полиморфизм объектов предметной области и операций и генерация программы в виде скрипт-файла с необходимой реализацией объектов предметной области и операций.

Научный руководитель — м.н.с. ИВМиМГ СО РАН М. А. Городничев.

Список литературы

[1] Вальковский В. А., Малышкин В. Э. Синтез параллельных программ и систем на вычислительных моделях. Новосибирск: Наука. Сиб. отдние, 1988.

[2] Городничев М. А., Малышкин В. Э., Медведев Ю. Г. HPC Community cloud: эффективная организация работы научно-образовательных суперкомпьютерных центров // Науч. вестн. НГТУ. 2013. № 3(52). С. 91–96.

[3] Малышкин В. Э., Корнеев В. Д. Параллельное программирование мультикомпьютеров. Сер.: Учебники НГТУ'. Новосибирск : Изд-во НГТУ, 2006.

[4] Сарычев В. Г., Купчишин А. Б. Разработка программного комплекса для конструирования программ обработки данных на высокопроизводительных вычислительных системах // 7-я Сиб. конф. по параллельным и высокопроизводительным вычислениям / Под ред. проф. А. В. Старченко. Томск: Изд-во Том. ун-та, 2014. С. 55–64.

Сарычев Виктор Геннадьевич — магистрант Новосибирского государственного технического университета;
e-mail: sarychevmail@gmail.com

ТРУДЫ КОНФЕРЕНЦИИ МОЛОДЫХ УЧЕНЫХ
Новосибирск, 2014 С. 94–104

Вероятностная модель пространственно-временных полей суточных сумм жидких осадков по данным для Новосибирской области

О. В. Сересева

УДК 519.6, 551.501

Рассматривается вероятностная модель пространственно-временных полей суточных сумм жидких осадков по данным Новосибирской области для теплого полугодия (с мая по октябрь). Поле сумм осадков строится в виде произведения двух независимых полей: полей индикаторов выпадения осадков и полей сумм осадков с заданным на каждой станции одномерным распределением и корреляционной функцией определенного вида, параметры которой выбираются из условия близости к реальной корреляционной функции.

The multiplicative approach to construction of numerical stochastic models of spatial and spatio-temporal fields of daily sums of liquid precipitation is considered. The approach is that the indicator fields of precipitation with the given correlation function both probabilities of precipitation and field of the daily precipitation with the appropriate correlation function and one-dimensional distribution of probability are independently simulated. A final field is the product of these fields. All models are based on real data obtained on Novosibirsk weather stations.

Ключевые слова: численная стохастическая модель, поля осадков, индикаторные поля, осадки.

Keywords: numerical stochastic model, precipitation field, indicator field, liquid precipitation.

Введение. В данной работе рассматривается численная стохастическая модель пространственно-временных полей суточных сумм жидких осадков на станциях и на регулярной сетке. Поле сумм осадков строится в виде произведения двух независимых полей: полей индикаторов выпадения осадков и полей сумм осадков с заданным

Работа выполнена при финансовой поддержке ведущих научных школ Российской Федерации (грант Президента Российской Федерации НШ-5111.2014.1).

на каждой станции одномерным распределением и корреляционной функцией определенного вида, параметры которой выбираются из условия близости к реальной корреляционной функции. Подход к моделированию индикаторных полей близок к подходам, рассмотренным в работах [1, 3, 4, 7, 8, 10, 12] и основан на пороговом преобразовании специально подобранного гауссовского поля. Также рассмотрен способ моделирования пространственно-временных неоднородных полей индикаторов осадков, в котором гауссовские поля на станциях моделируются с помощью векторной модели авторегрессии с блочно-теплицевой матричной ковариационной функцией, а элементы этой матрицы оцениваются по реальным выборкам.

Пространственно-временное поле можно представить в виде последовательности пространственных полей, в которой временные и пространственно-временные корреляционные связи задаются на основе реальной информации. В наиболее простом случае однородного по пространству и стационарного по времени поля корреляционная функция может быть задана в виде произведения пространственной и временной корреляционных функций, что соответствует прямому произведению пространственных и временных корреляционных матриц, построенных по этим функциям для поля на заданной сетке. Методы моделирования гауссовских полей с такой корреляционной структурой хорошо известны [8].

В качестве исходной информации были использованы данные суточных сумм количества жидких осадков для теплого полугодия (с мая по октябрь), измеренных на 47 станциях, покрывающих Новосибирскую область площадью покрытия около 158 000 км^2 (вся площадь Новосибирской области составляет примерно 178 000 км^2). Исследуемая в работе часть Новосибирской области находится западнее р. Обь и представляет собой преимущественно равнинный рельеф (120 м над уровнем моря).

Данные охватывают пятнадцатилетний период с 1969 по 1983 гг. Следует отметить тот факт, что все 47 станций содержат достоверную информацию и в представленном наборе данных не имеется пропущенных значений. Минимальное количество осадков, фиксируемое на станциях равно 0,1 мм. Количество осадков меньше 0,1 мм, но больше нуля, фиксируется как след осадков, а нули обозначают отсутствие осадков.

1. Численная стохастическая модель пространственно-временных полей сумм жидких осадков на станциях. Пространственно-временное поле $\{\eta_{i,t}\}$ $(i = 1, ..., m, t = 1, ..., n)$ суточных сумм жидких осадков на нерегулярной сетке (сети станций) строится в виде произведения двух независимых полей: поля индикаторов осадков $\{\omega_{i,t}\}$ и условного поля суточных сумм осадков $\{\chi_{i,t}\}$ (при условии, что осадки были).

Рассмотрим неоднородное по пространству и стационарное по времени пространственно-временное поле индикаторов суточных сумм жидких осадков $\{\omega_{i,t}\}$ $(i = 1, ..., m,\ \ t = 1, ..., n)$ на станциях с координатами x_i, y_i, где $m = 47$ — число рассматриваемых станций, n — число суток. Это поле можно представить как стационарную по времени последовательность векторов $(\vec{\omega}_1^{\top}, ..., \vec{\omega}_n^{\top})^{\top} = \vec{\omega}_{(n)}$, $\vec{\omega}_t = (\omega_{t1}, ..., \omega_{tm})^{\top}$ $(t = 1, ..., n)$ индикаторов суточных сумм осадков на станциях, принимающих значения $\omega_i = 1$ с независящими от времени вероятностями p_i, $\omega_i = 0$ с соответствующими вероятностями $1 - p_i$ и блочно-теплицевой корреляционной матрицей $S_{(n)} = (S_{k-l}) = (S_\tau)$ $(k, l = 1, ..., n)$ с блоками $S_\tau = (s_{ik,jl}) = (s_{ij,\tau})$ $(i, j = 1, ..., m, \tau = 0, ..., n-1)$. Поле индикаторов строится в виде порогового преобразования гауссовского пространственно-временного поля $\{\xi_{i,t}\}$ с нулевым средним в виде

$$\omega_{i,t} = \begin{cases} 1, & \xi_{i,t} \le c_i \\ 0, & \xi_{i,t} > c_i \end{cases} . \tag{1}$$

Здесь величины c_i определяются из уравнений

$$p_i = P(\omega_i = 1) = \frac{1}{\sqrt{2\pi}} \int\limits_{-\infty}^{c_i} e^{-\frac{1}{2}z^2} dz \tag{2}$$

при заданных вероятностях выпадения осадков p_i на станциях. Элементы блочно-теплицевой корреляционной матрицы $G_{(n)} = (G_{k-l}) = (G_\tau)$ $(k, l = 1, ..., n)$ гауссовского поля $\{\xi_{i,t}\}$ с блоками $G_\tau = (g_{ik,jl}) = (g_{ij,\tau})$ $(i, j = 1, ..., m, \tau = 0, ..., n-1)$, связаны с соответствующими элементами корреляционной матрицы $S_{(n)}$ поля $\{\omega_{i,t}\}$ известными соотношениями, рассмотренными, например в [11] . Неоднородность поля задается вероятностями p_i, которые зависят от координат станций x_i, y_i, но не зависят от времени [2], а также блочно-теплицевой

ковариационной матрицей $S_{(n)}$, блоки которой не являются теплицевыми матрицами и симметричны только на главной диагонали. Эта матрица определяет матричную ковариационную функцию векторного стационарного процесса.

Необходимое для построения пространственно-временного поля $\{\omega_{i,t}\}$ гауссовское поле $\{\xi_{i,t}\}$ на станциях или стационарная последовательность гауссовских векторов $(\vec{\xi}_1^{\top}, \ldots, \vec{\xi}_n^{\top})^{\top} = \vec{\xi}_{(n)}$, где $\vec{\xi}_t = (\xi_{1t}, \ldots, \xi_{mt})^{\top}$ — векторы размерности m с матричными корреляционными функциями (или блочно-теплицевой ковариационной матрицей $G_{(n)}$) моделировались с помощью векторной модели авторегрессии

$$\vec{\xi}_t = B_1^{\top}[q]\vec{\xi}_{t-1} + \ldots + B_q^{\top}[q]\vec{\xi}_{t-q} + C_q\vec{\varphi}_t, \;\; t = 1, \ldots, n,$$

начальные векторы для которой строились с помощью процедуры, рассмотренной в [8]. $\vec{\varphi}_t = (\varphi_{t1}, ..., \varphi_{tm})^{\top}$ – независимые гауссовские векторы размерности m. Матрицы $B_1^{\top}[q], \ldots, B_q^{\top}[q]$ определяются из уравнения $J_{(q)}G_{(q)}^{-1}J_{(q)}\vec{G}_q = \vec{B}[q]$, в котором $\vec{B}[q] = \left(B_1^{\top}[q], \ldots, B_q^{\top}[q]\right)^{\top}$, $\vec{G}_q = (G_1^{\top}, \ldots, G_q^{\top})^{\top}$. Нижняя треугольная матрица C_q определяется из соотношений $C_qC_q^{\top} = G_0 - \vec{B}^{\top}[q]J_{(q)}R_{(q)}J_{(q)}\vec{B}[q]$, где $J_{(q)}$ — блочная матрица перестановок, в которой на побочной диагонали стоят блоки в виде единичных матриц размерности m, а все остальные блоки такой же размерности – нулевые блоки. В качестве корреляционной матрицы $G_{(n)}$ была использована матрица, полученная решением уравнений при заданных значениях выборочной блочно-теплицевой матрицы $S_{(n)}$ для индикаторного поля, вычисленная по рассматриваемой выборке.

Для построения пространственно-временного поля $\{\chi_{i,t}\}$ с корреляционной матрицей $Q_{(n)}$ используется метод обратных функций распределения [8], и его элементы вычисляются с помощью преобразования

$$\chi_{i,t} = F_i^{-1}(\Phi(\zeta_{i,t})), \tag{3}$$

где $\zeta_{i,t}$ — элементы гауссовского поля $\{\zeta_{i,t}\}$ с корреляционной матрицей $H_{(n)}$. Элементы матрицы $H_{(n)}$ связаны с элементами матрицы $Q_{(n)}$ известными соотношениями $h_{ij,\tau} = h(q_{ij,\tau})$, конкретный вид которых приведен в [9]. Этот метод требует, как правило, корректировки корреляционной матрицы $H_{(n)}$, чтобы она стала положительно

определенной, в случае, если она таковой не является. В ряде случаев можно воспользоваться методом, основанным на нормализации реальных данных [6], который, хотя и является менее точным, но не требует дополнительной корректировки корреляционной матрицы.

Итоговое поле осадков, как отмечалось выше, строится в виде произведения полей $\{\omega_{i,t}\}$ и $\{\chi_{i,t}\}$:

$$\eta_{i,t} = \omega_{i,t}\chi_{i,t}.$$

Корреляционная структура поля $\{\eta_{i,t}\}$ определяется корреляциями полей $\{\omega_{i,t}\}$ и $\{\chi_{i,t}\}$.

Описанный подход предназначен для моделирования пространственно-временных полей осадков на станциях. Так как данный подход ориентирован на использование корреляционных матриц, то он позволяет строить поля на станциях с учетом их реальной структуры. Эти поля могут быть как однородными, так и неоднородными. В ряде случаев необходимо строить параметрические модели полей осадков, как на станциях, так и в узлах регулярной сетки. Для этих целей необходимо использовать аппроксимации реальных корреляционных функций на станциях некоторыми специальными корреляционными функциями в приближении однородного поля.

2. Статистическая структура полей суточных сумм жидких осадков. Аппроксимация корреляционных функций и одномерных распределений. По синхронным рядам наблюдений за количеством выпавших осадков на станциях, которые представлены в виде массива $\{\eta^*_{i,t}\}$, строились ряды индикаторов осадков, и по ним в стационарном приближении оценивались вероятности выпадения осадков $p^*_i = p_i$, а также блочно-теплицева ковариационная матрица

$$S^*_{(n)} = S_{(n)} = \begin{pmatrix} S_0 & S_1 & \dots & S_{n-1} \\ S_1^\top & S_0 & \dots & S_{n-2} \\ \dots & \dots & \dots & \dots \\ S_{n-1}^\top & S_{n-2}^\top & \dots & S_0 \end{pmatrix},$$

с помощью стандартных формул для оценки вероятностей и матричных ковариационных функций. С помощью этих формул оценивается только первая блочная строка матрицы $S^*_{(n)} = S_{(n)}$, а вся остальная матрица достраивается с учетом ее блочной тепливости. Здесь S_k $(k = 0, \dots, n-1)$ — матрицы вида $S_k = (s^k_{ij})$

$(i, j = 1, \ldots, m)$. Если поставить в соответствие каждому элементу s_{ij}^k матриц S_k координаты станций x_i, y_i и x_j, y_j, между которыми определены эти корреляции, то выражение $s_{ij}^k = s^k(x_i, y_i; x_j, y_j)$ представляет собой корреляционную функцию, определенную на сети станций. При $k = 0$ эта функция представляет собой пространственную корреляционную функцию поля индикаторов на станциях. Последовательность $s^0(x_i, y_i; x_j, y_j)$, $s^1(x_i, y_i; x_j, y_j), \ldots$ является пространственно-временной корреляционной функцией поля на станциях. Аналогично определяются корреляционные функции поля количества осадков при условии их наличия, а также поля сумм осадков. Обозначим их через $q_{ij}^k = q^k(x_i, y_i; x_j, y_j)$ и $r_{ij}^k = r^k(x_i, y_i; x_j, y_j)$. Оценки для осадков вычисляются по формулам, аналогичным формулам для поля индикаторов. Оценки корреляций для поля количества осадков имеют специфику, состоящую в том, что оценка производится по числу случаев, различных для каждой пары станций. В связи с этим матрица, соответствующая оцениваемой корреляционной функции, может не быть положительно определенной. Ниже приведен способ ее корректировки для достижения положительной определенности.

В данной работе все вычисления проводились для полугодового периода с мая по октябрь отдельно для каждого месяца, причем усреднения были сделаны по данным объема $n \times Y = 450$, где $n = 30$) — число дней в июне, а $Y = 15$ — число лет наблюдений. Полученные характеристики p_i и $S_{(n)}$ использовались в качестве входных параметров для модели. Ниже будут проведены результаты для пространственных полей при k =0.

Для аппроксимации корреляционных функций $s^0(x_i, y_i; x_j, y_j)$, $q^0(x_i, y_i; x_j, y_j)$ и $r^0(x_i, y_i; x_j, y_j)$ поля индикаторов, поля количества осадков при условии их наличия и поля осадков на станциях, определяемые соответствующими выборочными корреляционными матрицами $S_0 = (s_{ij})$, $Q_0 = (q_{ij})$ и $R_0 = (r_{ij})$ $(i, j = \overline{1, m})$, оцененными по реальным рядам наблюдений, используется корреляционная функция вида

$$\varphi(x_i, y_i; x_j, y_j) = \varphi(x_i - x_j, y_i - y_j) = \\ = \exp(-[\alpha(x_i - x_j)^2 + \beta(x_i - x_j)(y_i - y_j) + \gamma(y_i - y_j)^2]^\theta) \qquad (4)$$

с выбором параметров α, β, γ и θ для каждого из полей исходя из условия минимума среднего квадрата разности между фактически-

ми и аппроксимирующими функциями. В дальнейшем функции вида (4), определенные в узлах регулярной сетки, используются для моделирования полей осадков на выбранной сетке. Отметим, что формулу (4) можно также использовать для корректировки матриц $H_{(n)}$ и $G_{(n)}$ при условии, что используется приближение однородного поля. Если рассматривается неоднородное поле, то можно использовать корректировку вида

$$\hat{H}_{(n)} = (1-\varepsilon)H_{(n)} + \varepsilon I_{(n)},$$

где $\varepsilon \in (0,1)$ выбирается из условия, чтобы максимальное по модулю отрицательное собственное число матрицы $\hat{H}_{(n)}$ стало небольшим положительным. Для пространственно-временных полей в качестве корреляционных функций можно использовать произведение пространственной корреляционной функции (4) на соответствующую временную корреляционную функцию.

В табл. 1 приведены оцененные коэффициенты $\alpha, \beta, \gamma, \theta$ для всех месяцев теплого полугодия поля индикаторов, в табл. 2 — поля осадков.

На рисунке приведены изолинии корреляционной функции $r_{ij}^k = r^k(x_i, y_i;\, x_j, y_j)$ пространственного поля осадков для мая, июля и октября (слева) и корреляционной функции $s_{ij}^k = s^k(x_i, y_i;\, x_j, y_j)$ индикаторного поля осадков для мая, июля и октября (справа). Видно, что структура полей заметно меняется от месяца к месяцу.

На всех рисунках изолиний корреляционных функций $r_{ij}^k = r^k(x_i, y_i;\, x_j, y_j)$ наблюдается преобладающая юго-западная ориентация эллипсов, что соответствует характерной для рассматриваемого района розе ветров. Что касается ориентации эллипсов для корреляционных функций индикаторов осадков, то их наклон объясняется, по видимому, тем, что при подсчете корреляций для индикаторного поля в значения корреляций слабые и интенсивные осадки вносят равный вклад, в то время как для полей осадков — интенсивные осадки в направлении преобладающих ветров вносят больший вклад по сравнению со слабыми.

Специфика учета неоднородности поля суточных сумм жидких осадков по одномерным распределениям состоит в том, для оценки этих распределений на станциях доступный объем данных для каждой станции сравнительно невелик. В рамках рассматриваемого

Таблица 1

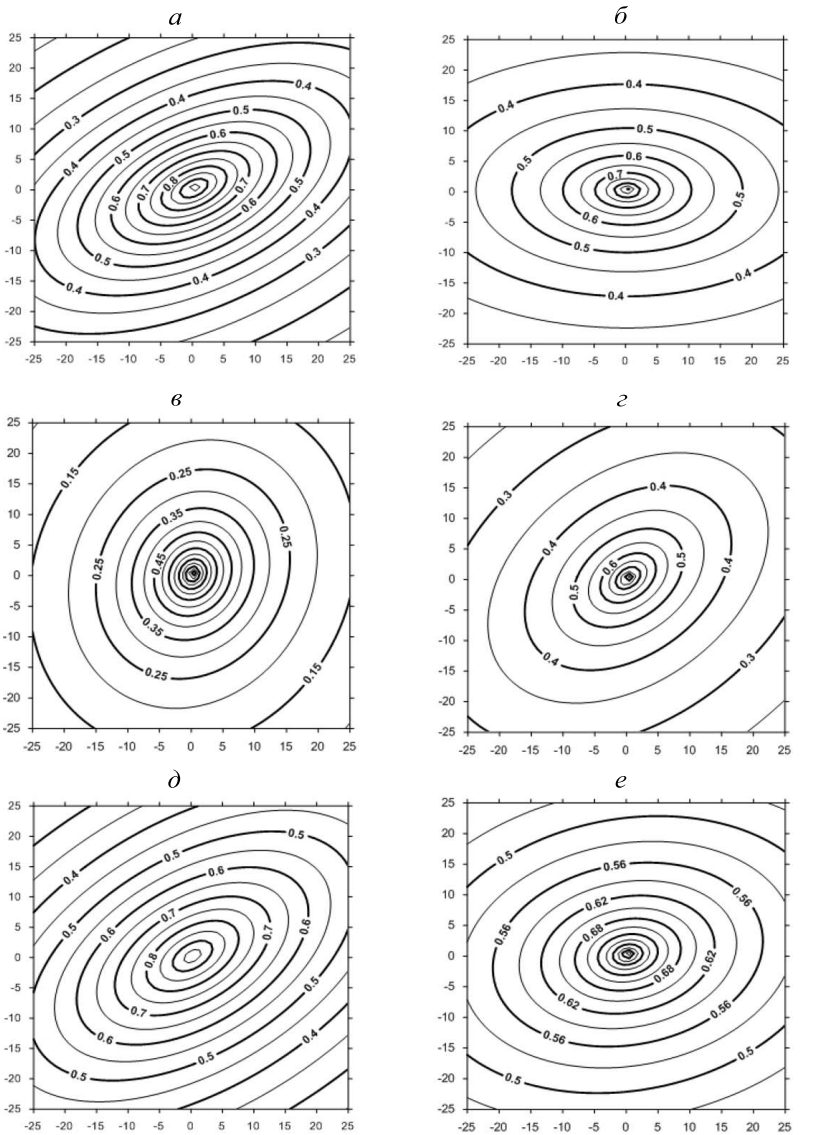
	α	β	γ	θ
Май	0.00075	-0.00001	0.0024	0.261
Июнь	0.0012	-0.0002	0.0018	0.236
Июль	0.00301	-0.0024	0.0035	0.223
Август	0.0016	-0.0004	0.0028	0.227
Сентябрь	0.0005	-0.00001	0.0013	0.260
Октябрь	0.00019	-0.00008	0.00038	0.218

Таблица 2

	α	β	γ	θ
Май	0.00173	-0.0024	0.00353	0.4477
Июнь	0.0044	-0.0016	0.0056	0.3118
Июль	0.0132	-0.0033	0.0105	0.301
Август	0.0022	-0.0035	0.0071	0.324
Сентябрь	0.0018	-0.0016	0.0024	0.413
Октябрь	0.00089	-0.0011	0.00141	0.450

в данной работе подхода необходимо оценить распределения при условии, что осадки имели место, т. е. распределения оцениваются по числу случаев наличия осадков на каждой станции. В качестве иллюстрации степени неоднородности поля по одноточечным характеристикам приведем некоторые характеристики пространственной изменчивости среднего значения сумм осадков при условии их выпадения, а также стандартное отклонение их оценки по имеющемуся объему выборки. Стандартное отклонение оценки среднего значения в среднем по всем станциям равно 0.66 мм, а изменчивость среднего составляет величину 2.84 мм. Учет такой изменчивости существенен при решении многих задач статистической метеорологии, особенно в тех случаях, когда необходимо учитывать специфику осадков в конкретных сравнительно небольших регионах.

Метод аппроксимации эмпирического распределения для случая однородного поля рассмотрен в работе [5]. Этот метод основан на интерполяции эмпирической функции распределения, построенной по данным на всех станциях, кубическими сплайнами с аппроксимацией ее правой части распределением Вейбулла. В данной работе задание одномерных распределений осуществлялось следующим способом. Эмпирические функции одномерного распределения



Изолинии корреляционной функции $r_{ij}^k = r^k(x_i, y_i; x_j, y_j)$
пространственного поля осадков (*а* — май, *в* — июль, *д* — октябрь)
и корреляционной функции $s_{ij}^k = s^k(x_i, y_i; x_j, y_j)$
индикаторного поля осадков (*б* — май, *г* — июль, *е* — октябрь)

Таблица 3

i	μ_i^*	μ_i	$abs(\mu_i^* - \mu_i)$	σ_i^*	σ_i	$abs(\sigma_i^* - \sigma_i)$
12	5,716	6,238	0,522	10,782	10,741	0,041
16	5,261	5,3018	0,1192	12,323	7,265	5,058
19	5,162	5,376	0,214	10,86	7,598	3,263
24	4,662	5,102	0,440	7,677	8,007	0,330
25	4,819	5,144	0,325	5,572	5,871	0,300
41	5,464	5,941	0,477	6,445	6,786	0,341
47	6,03	6,677	0,647	10,76	11,206	0,446

для каждой станции оценивались в опорных точках $u_1^{(i)}, u_2^{(i)}, ..., u_{n_i}^{(i)}$, где i — номер станции, n_i — число опорных точек эмпирической функции распределения. Эмпирическая функция распределения $F(u_1^{(i)}) = 0, F(u_2^{(i)})...F(u_{n_i}^{(i)})$, $u_1^{(i)}$=0.1 мм, аппроксимировалась следующим образом. В интервалах $[u_1^{(i)}, u_2^{(i)}], ..., [u_{n_i-1}^{(i)}, u_{n_i}^{(i)}]$ функция интерполировалась линейно, а в интервале $[u_{n_i}^{(i)}, \infty)$ задавалась функцией $F(x) = \exp(-\alpha x^\beta)$, $\alpha = -\ln(1 - F(u_{n_i}^{(i)}))/(u_{n_i}^{(i)})^\beta$, где α выбирается из условия $F(u_{n_i}^{(i)}) = F(x)$. Для каждой станции число опорных точек различно, причем выбор их определялся из следующих двух условий. Первое условие состоит в том, чтобы выполнялись неравенства $F(u_1^{(i)}) < F(u_2^{(i)}) <, ..., < F(u_{n_i}^{(i)})$, а второе определялось условием достижения приемлемой близости средних и дисперсий модельного и эмпирического распределений. При этом выбором параметра β можно улучшить результат. В табл. 3 для некоторых станций i приведены средние значения μ_i, μ_i^* и стандартные отклонения σ_i, σ_i^* суточных сумм осадков при условии их выпадения, вычисленные по модельным функциям распределения и по реальным выборкам.

Рассмотренные модели могут быть использованы для оценки вероятностных характеристик суммарных осадков на заданной территории, характеристик аномальных осадков, в задачах гидрологии при построении совместных моделей осадков и речного стока, задачах, связанных с оценкой вероятности возникновения лесных пожаров и др.

Автор выражает искреннюю благодарность сотрудникам ИВМиМГ СО РАН В. А. Огородникову за внимание к работе, а также А. А. Леженину за полезные замечания.

Научный руководитель — д.ф.-м.н. В. А. Огородников.

Список литературы

[1] Анисимова А. Численное моделирование индикаторных случайных полей жидких осадков // Труды конф. молодых ученых ИВМиМГ СО РАН. Новосибирск, 1997. С. 3–15.

[2] Дробышев А. Д., Марченко А. С., Огородников В. А., Чижиков В. Д. Статистическая структура временных рядов суточных сумм жидких осадков в равнинной части Новосибирской области // Труды ЗапСибНИИ Госкомгидромета. 1989. Вып. 86. С. 44–66.

[3] Evstafieva A. I., Khlebnikova E. I., Ogorodnikov V. A. Numerical stochastic models for complex of time series of weather elements. // Rus. J. of Num. Analysis and Math. Modelling. 2005. V. 20, N 6. P. 535–548.

[4] Kleiber W., Katz R. W., Rajagopalan B. Daily spatiotemporal precipitation simulation using latent and transformed Gaussian processes // Water Resources Res. 2012. V. 48. P. 11105–11114.

[5] Марченко А. С. Аппроксимация эмпирического распределения вероятностей суточных сумм жидких осадков // Труды ЗапСибНИИ Госкомгидромета. 1989. Вып. 86. С. 66–74.

[6] Марченко А. С., Семочкин А. Г. $F\Phi\Phi F$-метод моделирования наблюдаемых реализаций временных рядов // Численные методы статистического моделирования. Новосибирск: ИВМиМГ, 1987. С. 14–22.

[7] Ogorodnikov V. A., Khlebnikova E. I., Kosyak S. S. Numerical stochastic simulation of joint non-Gaussian meteorological series // Rus. J. of Num. Analysis and Math. Modelling. 2009. V. 24, N 5. P. 467–480.

[8] Ogorodnikov V. A. and Prigarin S. M. Numerical modelling of random processes and fields: Algorithms and applications. VSP. Utrecht, 1996.

[9] Пригарин С. М. Методы численного моделирования случайных процессов и полей. Новосибирск: ИВМиМГ СО РАН, 2005.

[10] Semenov M. A., Barrow E. M. Use of a stochastic weather generator in the development of climate change scenarios // Climatic Change. 1997. V. 35. P. 397–414.

[11] Смирнов Н. В. ,Большев Л. Н. Таблицы для вычисления функций нормального распределения. М.: Изв. АН СССР, 1962.

[12] Ukhinova O. S., Ogorodnikov V. A. Stochastic models of spatial-time fields of precipitation sums // Proc. of the 6^{th} St. Peterburg WorkShop on simulation, 2009. P. 193–197.

Сересева Ольга Владимировна — мл. научный сотрудник Института вычислительной математики и математической геофизики СО РАН; e-mail: seresseva@mail.ru

ТРУДЫ КОНФЕРЕНЦИИ МОЛОДЫХ УЧЕНЫХ
Новосибирск, 2014 С. 105–121

Моделирование распространения волн в однородных средах с криволинейной свободной поверхностью

П. А. Титов

УДК 519.6

Предложен параллельный алгоритм, создана программа численного моделирования распространения волновых полей в однородных 2D- средах с криволинейной свободой поверхностью. Использован метод отображений: построение в физической области криволинейной сетки, согласованной с геометрией свободной поверхности, и дальнейший "перенос" задачи на "расчетную" область простой геометрической формы (прямоугольник), в которой задача может решаться известными методами. Для решения задачи в "расчетной" области используется комплексирование пошагового метода преобразований Лагерра по времени и конечно-разностного метода по пространственным переменным. Проведены расчеты на многопроцессорной системе для различных форм свободной поверхности.

An algorithm for the simulation of wave fields in 2D- media with curved free surface is proposed. In this work the mapping method is being used. It is based on the construction of curvilinear grid agreed with the geometry of the free surface in the domain of interest, which is then to be mapped into "calculation" rectangular domain covered by a uniform grid. To solve the transformed problem in the "calculation" domain, previously proposed stepwise Laguerre time method and spatial differential method are used. Results of numerical simulation on a multiprocessor system for various forms of the free surface are presented.

Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант 13-01-00231) и программы фундаментальных исследований РАН № 4, проект 4.9. "Модельные и экспериментальные исследования вулканических структур методами активной и пассивной сейсмологии".

Ключевые слова: геофизика, математическое моделирование, криволинейные сетки, пошаговый метод Лагерра, метод конечных разностей, волновое уравнение, 2D.

Keywords: geophysics, mathematical modeling, curvilinear grids, stepwise Laguerre method, finite diference method, wave equation, 2D.

Введение. Известно,что эффективным инструментом исследования процессов распространения волн в различных моделях сложно построенных сред является математическое моделирование. Из всех известных методов численного моделирования распространения упругих волн в случае сложно построенных 2D неоднородных упругих сред наиболее гибким является разностный метод. Однако применение данного метода неизменно приводит к возникновению сталкиваются с проблемой возникновения дифракционных волн при отражении волны от свободной поверхности. Связано это с тем, что для применения этого метода необходимо разбиение физической области на квадратные или прямоугольные ячейки, вследствие чего свободная поверхность заменяется ступенчатой функцией. Пример использования разностного метода приведен на рис. 1, 2.

В работе рассматривались области с прямолинейными левой, правой и нижней границами (пример в верхней части рис. 1), для построения криволинейной сетки использована формула:

$$q^1(x,z) = x,\ q^2(x,z) = (1 - \frac{z}{H})f_1(x) + \frac{z}{H}f_2(x),$$

где $f_1(x)$ — форма свободной поверхности; $f_2(x)$ — форма стороны области, противоположной свободной поверхности; H — высота "расчетной" области; W — ширина. В рассматриваемых областях $f_2(x)$ — константа.

Подобные дифракционные волны вносят погрешность в решение. Таким образом, была поставлена задача об избавлении от дифракционных волн. Для этого в работе предлагается использовать криволинейные сетки.

В последнее время все большую популярность приобретает моделирование с использованием криволинейных сеток. Это весьма гибкий инструментарий, нашедший уже свое применение при численном решении сингулярно возмущенных задач, моделировании потоков плазмы в камере токамака и др. Способы построения сеток, а

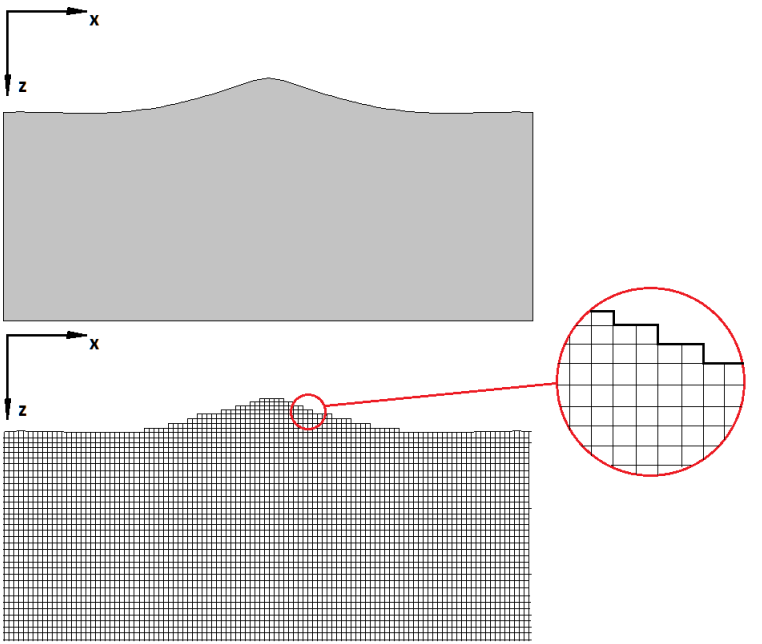


Рис. 1. Область и соответствующее ей разбиение на квадратные ячейки

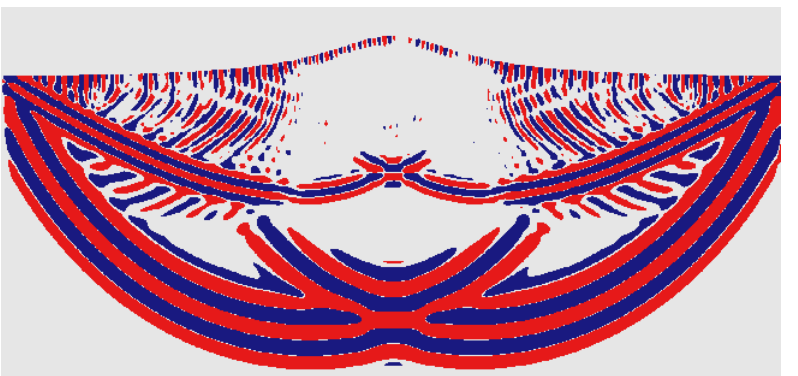


Рис. 2. При отражении от свободной поверхности видны дифракционные волны (возле поверхности)

также их применение для решения вышеупомянутых задач, приведены в работе [1].

Данная работа особенно важна для моделирования магматического вулкана Эльбрус, который имеет чрезвычайно сложное строение. Проведенные в 2010 г. экспериментальные сейсмические наблюдения в штольне показали, что вулкан относится к типу "живущих", определились частоты, на которых наблюдается деятельность вулкана, однако интерпретация полученных результатов без адекватной математической модели практически невозможна.

1. Построение криволинейной сетки. Существует много методов построения криволинейной сетки. Основные из них описаны в работе [1]. Поскольку в данной работе рассматриваются однородные области несложной формы, то для построения сетки был выбран метод трансфинитной интерполяции (TFI-метод), описанный в [1], с. 42–47. Рассмотренный также метод построения сетки с использованием обращенных уравнений Бельтрами ([1], с. 19–42) не показал существенных преимуществ по сравнению с TFI-методом, по крайней мере на областях, рассматриваемых в работе.

Криволинейная сетка в "физической" области получается в результате взаимно-однозначного отображения равномерной сетки "расчетной" области. Таким образом, необходимо установить взаимно-однозначное соответствие между "физической" и "расчетной" областями. "Физическая область" находится в пространстве переменных (q^1, q^2) , а "расчетная" — в пространстве переменных (x, z).

2. Постановка задачи. В данной работе численное моделирование процессов распространения волн проводится на основе решения волнового уравнения. Для расчета теоретических сейсмограмм, возникающих в результате воздействия сосредоточенного источника, расположенного в среде, необходимо определить значения смещения u в каждой точке области в каждый момент времени. Эти значения находятся из решения начально-краевой задачи, ставящейся в пространстве (q^1, q^2) . В случае однородной среды задача примет вид:

$$\begin{gathered}\frac{\partial^2 u(q^1,q^2,t)}{\partial t^2} = V^2\left(\frac{\partial^2 u(q^1,q^2,t)}{\partial q^1 \partial q^1} + \frac{\partial^2 u(q^1,q^2,t)}{\partial q^2 \partial q^2}\right) + F(q^1,q^2,t), \\ u|_{\partial\Gamma} = 0,\ \frac{\partial u}{\partial n}|_{\partial S} = 0,\ u|_{t=0} = 0,\ \frac{\partial u}{\partial t}|_{t=0} = 0,\end{gathered} \tag{2.1}$$

V^2 — квадрат скорости распространения волн в среде; ∂S — свободная поверхность; $\partial\Gamma$ — граница области без свободной поверхности. В качестве источника сигнала используется источник типа "центр давления", т. е.

$$F(q^1,q^2,t) = \big[\sin(\pi(t-1)) + 0.8\sin(2\pi(t-1)) + \\ + 0.2\sin(3\pi(t-1))\big]\ \delta(q^1-q_0^1, q^2-q_0^2).$$

Здесь (q_0^1, q_0^2) — координаты источника.

После построения сетки необходимо осуществить "перенос" исходной задачи, поставленной в "физической" области, на "расчетную" область. Для этого выполним замену переменных и перепишем задачу в переменных (x,z).

Введем следующие обозначения:

$$B_2(\xi) \ = \ \left(g^{22}\frac{\partial^2\xi}{\partial x^2} \ - \ 2g^{12}\frac{\partial^2\xi}{\partial x\partial z} \ + \ g^{11}\frac{\partial^2\xi}{\partial z^2}\right)$$

где

$$\begin{aligned} g^{11} &= \frac{\partial q^1}{\partial x}\frac{\partial q^1}{\partial x} + \frac{\partial q^2}{\partial x}\frac{\partial q^2}{\partial x} = 1 + \frac{\partial q^2}{\partial x}\frac{\partial q^2}{\partial x} \\ g^{12} &= \frac{\partial q^1}{\partial x}\frac{\partial q^1}{\partial z} + \frac{\partial q^2}{\partial x}\frac{\partial q^2}{\partial z} = \frac{\partial q^2}{\partial x}\frac{\partial q^2}{\partial z} \\ g^{22} &= \frac{\partial q^1}{\partial z}\frac{\partial q^1}{\partial z} + \frac{\partial q^2}{\partial z}\frac{\partial q^2}{\partial z} = \frac{\partial q^2}{\partial z}\frac{\partial q^2}{\partial z} \end{aligned}$$

В "расчетной" области уравнения (2.1) примут вид:

$$\begin{cases} \dfrac{\partial^2 u}{\partial t^2} \ = \ \dfrac{V^2}{J^2}\left(B_2(u) - \dfrac{1}{J}B_2(q^2)\dfrac{\partial u}{\partial z}\right) + F(t,x,z), \\ \\ u|_{\partial\Gamma} = 0,\ \dfrac{\partial u}{\partial n}|_{z=0} = 0,\ u|_{t=0} = u^0 = 0,\ \partial u\partial t|_{t=0} = u^1 = 0, \end{cases} \tag{2.2}$$

где

$$J = det\left(\frac{\partial q}{\partial x}\right).$$

Здесь

$$u(q^1,q^2) = u(q^1(x,z), q^2(x,z)) = u(x,z),$$

$$F(t, q^1, q^2) = F(t, q^1(x, z), q^2(x, z)) = F(t, x, z).$$

Далее к задаче (2.2) применяется преобразование Лагерра по времени.

3. Пошаговый метод Лагерра для решения волнового уравнения. Впервые данный метод был применен в работе [3].

В отличие от классических преобразований Фурье и Лапласа, применение преобразования Лагерра приводит к решению задачи для системы дифференциальных уравнений, не зависящей от параметра разделения, роль которого в данном случае исполняет номер проекции Лагерра.

В результате применения этого преобразования получается задача для проекций Лагерра с номером с правой частью, рекуррентно зависящей от проекций меньших номеров. После применения конечно-разностных аппроксимаций по пространственным переменным, решение исходной задачи сводится к решению системы линейных алгебраических уравнений со многими правыми частями, для решения которых можно использовать различные современные подходы решения линейных систем.

В работе рассматривается модификация метода, а именно преобразование Лагерра используется на последовательности конечных интервалов по времени. Полученное решение в конце одного временного отрезка используется в качестве начальных данных для решения задачи на следующем временном отрезке и т. д. Связано это с тем, что для получения решения с хорошей точностью через длительный промежуток времени необходимо очень большое количество проекций Лагерра, что критично скажется в процессе численной реализации. Подробно пошаговый метод преобразования Лагерра описан в работе [2].

При использовании данного подхода появляется необходимость выбора четырех параметров: N — количество проекций Лагерра; h — масштабный множитель, необходимый для аппроксимации решения функциями Лагерра; μ — экспоненциальный коэффициент весовой функции, используемый для нахождения решения на конечном временном интервале; τ — длительности этого интервала. Способ выбора этих параметров предложен в работе [2].

Ограничимся основными терминами и непосредственно видом задачи после применения к ней преобразования Лагерра.

Полиномы Лагерра имеют вид:

$$L_0(t) = 1,$$
$$L_1(t) = 1 - t,$$
$$L_{k+1}(t) = \frac{1}{k+1}\left((2k+1-t)L_k(t) - kL_{k-1}(t)\right), \quad k \geqq 1,$$
$$\dot{L}_k(t) = -\sum_{i=0}^{k-1} L_i(t), \quad k \geqq 1.$$

Функции Лагерра имеют вид:

$$l_n(t) = e^{-\frac{t}{2}} L_n(t), \quad n \geqq 0.$$

Они образуют полную ортонормированную систему в $L_2(0, \infty)$.

Преобразование

$$v_n = \int_0^\infty v(t) l_n(ht) dt$$

называется *прямым преобразованием Лагерра*, а v_n — проекциями Лагерра функции $v(t)$.

Обратное преобразование Лагерра имеет вид:

$$v(t) = \sum_{n=0}^{\infty} v_n l_n(ht).$$

Идея метода состоит в получении приближенного значения

$$v(\tau) \simeq \sum_{n=0}^{N} v_n l_n(\tau h),$$

и значения

$$u(\tau) = e^{\mu\tau} v(\tau), \ \dot{u}(\tau) = \mu u(\tau) + e^{\mu\tau}\dot{v}(\tau)$$

выбираем в качестве начальных данных для решения задачи при $t \geqq \tau$.

Теперь покажем нашу задачу после примененного к ней преобразования Лагерра. Задача сводится к решению линейной системы из N уравнений:

$$\left[\left(\mu+\frac{h}{2}\right)^2+A\right]v_0=\left(\frac{h^2}{2}+2\mu h\right)v^0+hv^1+f_0,$$
$$\left[\left(\mu+\frac{h}{2}\right)^2+A\right]v_1=-\left(h^2+2\mu h\right)v_0+\left(\frac{3}{2}h^2+2\mu h\right)v^0+$$
$$+hv^1+f_1, \qquad (3.1)$$
$$\dots$$
$$\left[\left(\mu+\frac{h}{2}\right)^2+A\right](v_n-2v_{n-1}+v_{n-2})+2\mu h(v_{n-1}-v_{n-2})+$$
$$+h^2v_{n-1}=f_n-2f_{n-1}+f_{n-2}, \quad n\geqq 2$$
$$v_n|_{\partial\Gamma}=0,\ n=0,...,N$$

Условие на свободной поверхности, аналогично случаю, описанному в [4], преобразуется к виду:

$$\frac{\partial v_i}{\partial n}=a\frac{\partial v_i}{\partial z}+b\frac{\partial v_i}{\partial x}=0,\ \ i=0,...,N,\ \ b=\frac{-g^{12}}{\sqrt{g^{11}}J},\ \ a=\frac{\sqrt{g^{11}}}{J} \qquad (3.2)$$

где

$$A(v_i)=\frac{-V^2}{J^2}\left(B_2(v_i)-\frac{1}{J}B_2(q^2)\frac{\partial v_i}{\partial z}\right),\qquad i=0,...,N,$$
$$f_n=f_n(x,z)=\int_0^\infty F(t,x,z)l_n(ht)dt.$$

Решив систему, получаем:

$$v(\tau)\simeq\sum_{n=0}^{N}v_nl_n(\tau h)$$

а затем

$$u(\tau)=e^{\mu\tau}v(\tau),\qquad \dot u(\tau)=\mu u(\tau)+e^{\mu\tau}\dot v(\tau)$$

выбираем в качестве начальных данных для решения задачи при $t>\tau$.

4. Метод решения задачи (3.1), (3.2.) Для решения задачи (3.1), (3.2.) используется комплексирование метода конечных разностей по пространству и пошагового метода Лагерра.

Для численного решения необходимо переписать задачу (3.1), (3.2.) в разностном виде. Все уравнения имеют второй порядок по пространству (уравнения 4.1–4.6 предложены автором).

Расчетная область имеет размеры $W \times H$, $M \times N$ точек.

Для каждой проекции Лагерра и соответствующей ей правой части разностная задача имеет вид:

$$\left((\mu + \frac{h}{2})^2 + \widehat{A_2}\right) v_{i,j} \;=\; G_{i,j} \qquad i = 2, ..., M-1,\ j = 2, ..., N-1$$

на $\partial\Gamma$: $v_{1,j} \;=\; v_{M,j} \;=\; v_{i,N} \;=\; 0,\ 1 \leqq i \leqq M,\ 1 \leqq j \leqq N$

на ∂S : $i = 2, ..., M-1$

$b \geqq a, a \geqq 0, b \geqq 0$:

$$v_{i,1} = \frac{2}{3b}\left(2(b-a)v_{i+1,1} + \frac{a-b}{2}v_{i+2,1} + 2av_{i+1,2} - \frac{a}{2}v_{i+2,3}\right)$$

$a \geqq b, a \geqq 0, b \geqq 0$:

$$v_{i,1} = \frac{2}{3a}\left(2(a-b)v_{i,2} + \frac{b-a}{2}v_{i,3} + 2bv_{i+1,2} - \frac{b}{2}v_{i+2,3}\right) \qquad (4.1)$$

$a \geqq -b, a \geqq 0, b \leqq 0$:

$$v_{i,1} = \frac{2}{3a}\left(2(a+b)v_{i-1,1} + \frac{a+b}{2}v_{i-2,1} + 2av_{i-1,2} - \frac{a}{2}v_{i-2,3}\right)$$

$-b \geqq a, a \geqq 0, b \leqq 0$:

$$v_{i,1} = \frac{-2}{3b}\left(2(a+b)v_{i-1,1} + \frac{a+b}{2}v_{i-2,1} + 2av_{i-1,2} - \frac{a}{2}v_{i-2,3}\right)$$

$b = 0, a = 1$:

$$v_{i,1} = \frac{2}{3}\left(2v_{i,2} - \frac{1}{2}v_{i,3}\right),$$

где

$$\widehat{A}(u_{i,j}) \;=\; \frac{V_{i,j}^2}{J_{i,j}^2}\left(\widehat{B_2}(u_{i,j}) - \frac{1}{J_{i,j}}\widehat{B_2}(q_{i,j}^2)\frac{u_{i,j+1} - u_{i,j-1}}{2\Delta_z}\right); \qquad (4.2)$$

$$\widehat{B_2}(u_{i,j}) = \left(g^{22}_{i,j} \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{{\Delta_x}^2} - \right.$$
$$- \frac{g^{12}_{i,j}}{2} \frac{u_{i+1,j+1} - u_{i+1,j-1} - u_{i-1,j+1} + u_{i-1,j-1}}{\Delta_x \Delta_z} + \qquad (4.3)$$
$$\left. + g^{11}_{i,j} \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{{\Delta_z}^2} \right);$$

$$g^{11}_{i,1} = 1 + \frac{q^2_{i+1,1} - q^2_{i-1,1}}{2\Delta_x} \frac{-5q^2_{i,2} + 8q^2_{i,3} - 3q^2_{i,4}}{2\Delta_z},\ i = 2, ..., M-1,$$

$$g^{11}_{i,j} = 1 + \frac{q^2_{i+1,j} - q^2_{i-1,j}}{2\Delta_x} \frac{q^2_{i+1,j} - q^2_{i-1,j}}{2\Delta_x},$$
$$i = 2, ..., M-1,\ j = 1, ..., N-1 \qquad (4.4)$$

$$g^{12}_{i,j} = \frac{q^2_{i+1,j} - q^2_{i-1,j}}{2\Delta_x} \frac{q^2_{i,j+1} - q^2_{i,j-1}}{2\Delta_z}, i = 2, ..., M-1,\ j = 2, ..., N-1$$

$$q^{22}_{i,j} = \frac{q^2_{i,j+1} - q^2_{i,j-1}}{2\Delta_z} \frac{q^2_{i,j+1} - q^2_{i,j-1}}{2\Delta_z}, i = 2, ..., M-1,\ j = 1, ..., N-1$$

$$J_{i,j} = \frac{q^2_{i,j+1} - q^2_{i,j-1}}{2\Delta_z},\ i = 2, ..., M-1,\ j = 2, ..., N-1$$
$$J_{i,1} = \frac{-3q^2_{i,1} + 4q^2_{i,2} - q^2_{i,3}}{2\Delta_z},\ i = 2, ..., M-1 \qquad (4.5)$$

$$q^2_{i,j} = S_i + \frac{j-1}{M-1}(H - S_i)$$

Формула свободной поверхности:

$$S_i = S((i-1)\Delta_x),\ i = 1, ..., M.$$

Для решения системы (4.1) использован метод простой итерации, подробно описанный в [5].

Выбор этого метода обусловлен тем, что система (4.1), переписанная в матричном виде, будет иметь матрицу с большим диагональным преобладанием, что обеспечит быструю сходимость метода.

Далее, получив все значения $v_{n\ i,j}$, находим

$$v_{i,j}(\tau) = \sum_{n=0}^{N} v_{n\ i,j} l_n(\tau h).$$

Отсюда, используя формулы

$$\dot{v}(t) = w(t) = \sum_{n=0}^{\infty} w_n l_n(ht),$$

$$w_0 = h\left(\frac{v_0}{2} - v(0)\right), w_n = w_{n-1} + \frac{h}{2}\left(v_n + v_{n-1}\right), n = 1, 2, ...$$

найдем $\dot{v}_{i,j}$, затем

$$u_{i,j}(\tau) = e^{\mu\tau} v_{i,j}(\tau),\ \dot{u}_{i,j}(\tau) = \mu u_{i,j}(\tau) + e^{\mu\tau}\dot{v}_{i,j}(\tau).$$

Значения

$$u^0_{i,j} = u_{i,j}(\tau),\ u^1_{i,j} = \dot{u}_{i,j}(\tau)$$

берутся в качестве начальных данных для следующего временного отрезка:

$$v^0_{i,j} = u^0_{i,j},\ v^1_{i,j} = -\mu u^0_{i,j} + u^1_{i,j}.$$

На отрезке $[0, \tau]$ начальные данные имеют вид:

$$u^0_{i,j} = u_{i,j}(0),\ u^1_{i,j} = \dot{u}_{i,j}(0)$$

$u_{i,j}(0)$ и $\dot{u}_{i,j}(0)$ определяются из начальных данных задачи. $v_{i,j}(\tau)$ и $\dot{v}_{i,j}(\tau)$ служат начальными данными для следующего временного отрезка. Далее, аналогично, находим $v_{i,j}(2\tau)$, $\dot{v}_{i,j}(2\tau)$ и т. д.

5. Программная реализация. При помощи языка Fortran, пакетов MPI и OpenMP созданы два варианта параллельной программы и проведены численные расчеты на кластере МСЦ МВС-10П (на узле 2 процессора Intel Xeon E5-2690, 64ГБ оперативной памяти, 2 сопроцессора Intel Xeon Phi 7110X) и на кластере ССКЦ НКС-30Т(на узле 2 процессора Intel Xeon E5450, E5540 или E5670 и 16ГБ, 16ГБ или 24 ГБ оперативной памяти, соответственно).

Реализация на CPU. Одному MPI-процессу соответствует одно ядро CPU. MPI-топология линейная. Для оменов данными используются блокирующие пересылки.

Реализация на Intel Xeon Phi (offload-режим). Одному MPI-процессу соответствует одно ядро CPU и один сопроцессор Intel Xeon Phi. MPI-топология линейная.

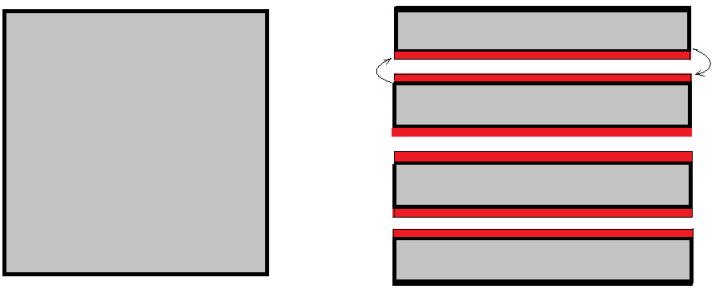


Рис. 3. Декомпозиция расчетной области на слои; красным цветом выделены слои перекрытия, через которые осуществляется обмен между соседними MPI-потоками

На рис. 3 показана декомпозиция области.

На рис. 4 приведены результаты ускорений работы программы.

6. Результаты расчетов. Ниже приведены результаты расчетов двух примеров: для области с “горкой” и для области со “впадиной”. В обоих случаях форма поверхности выражается одной и той же функцией ($\cos(x)$), но с разными знаками ($+$ или $-$).

Более подробное описание выполнено для области с “горкой” (рис. 5). На рис. 5 указано количество длин волн (размер всей области не важен с точки зрения изучаемых эффектов, имеют значение размеры криволинейной части поверхности). Красная точка соответствует расположению источника. На рис. 6 показана соответствующая криволинейная сетка, на рис. 7 — результаты моделирования. Скорость волны во всех расчетах равна 1 км/с. На рисунках видно, что в отличие от снимков волнового поля на рис. 2, на рис. 7 отсутствуют дифракционные волны при отражении от поверхности. Аналогичные расчеты проведены для области со “впадиной”. Рис. 8 — область, рис. 9 — снимки волнового поля.

На рис. 9 можно также наблюдать отсутствие дифракционных волн при отражении от свободной поверхности. Проведенные расчеты показывают перспективность дальнейшего использования криволинейных сеток для моделирования волновых полей в средах со сложной геометрией свободной поверхности.

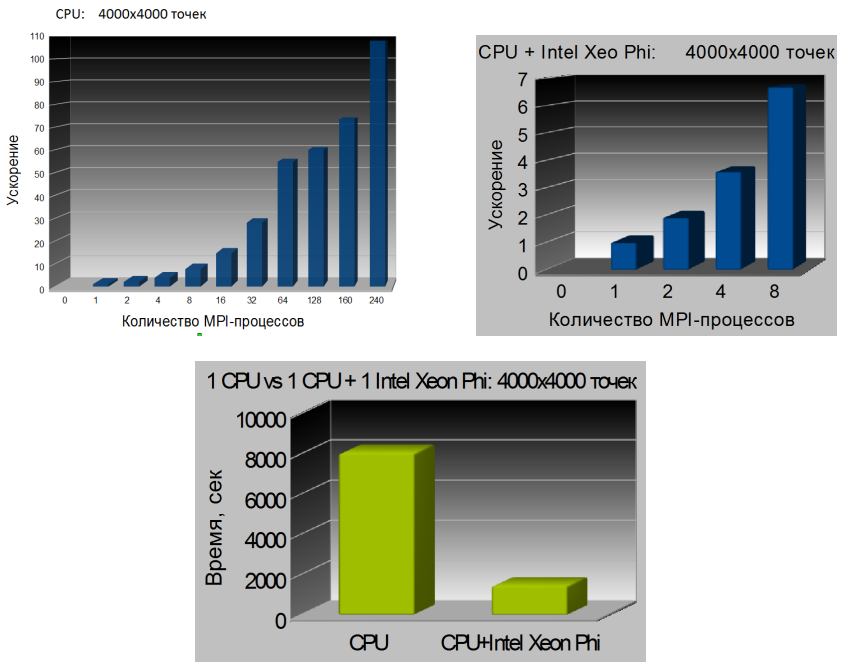


Рис. 4. Результаты сравнительных тестов

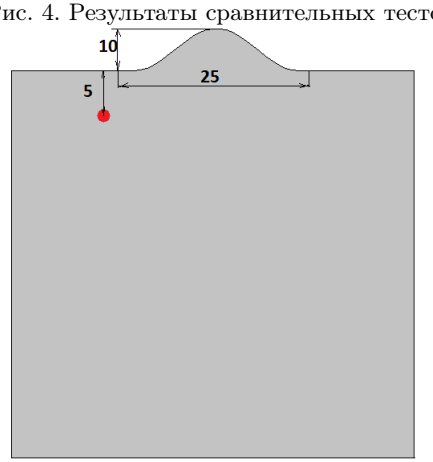


Рис. 5. Область с “горкой”

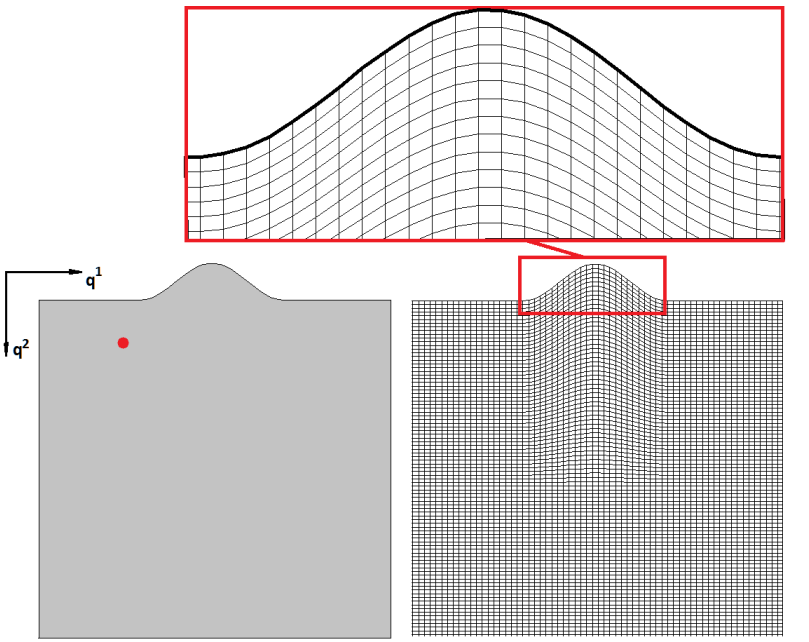


Рис. 6. Область с “горкой”

Заключение. Рассмотрено применение пошагового метода Лагерра для моделирования 2D волновых полей в средах с криволинейной свободной поверхностью. Получена аппроксимация граничных условий на свободной поверхности второго порядка по пространству. Удалось избавиться от дифракционных волн при отражении от поверхности.

Разработан параллельный алгоритм и две программы для моделирования 2D волновых полей в случае волнового уравнения и проведены численные расчеты на многопроцессорных вычислительных комплексах. Также проведены расчеты с использованием ускорителей Intel Xeon Phi, приведены сравнительные тесты ускорений. Предполагается в дальнейшем применить разработанный алгоритм для моделирования волновых полей в сложнопостроенных 3D-средах

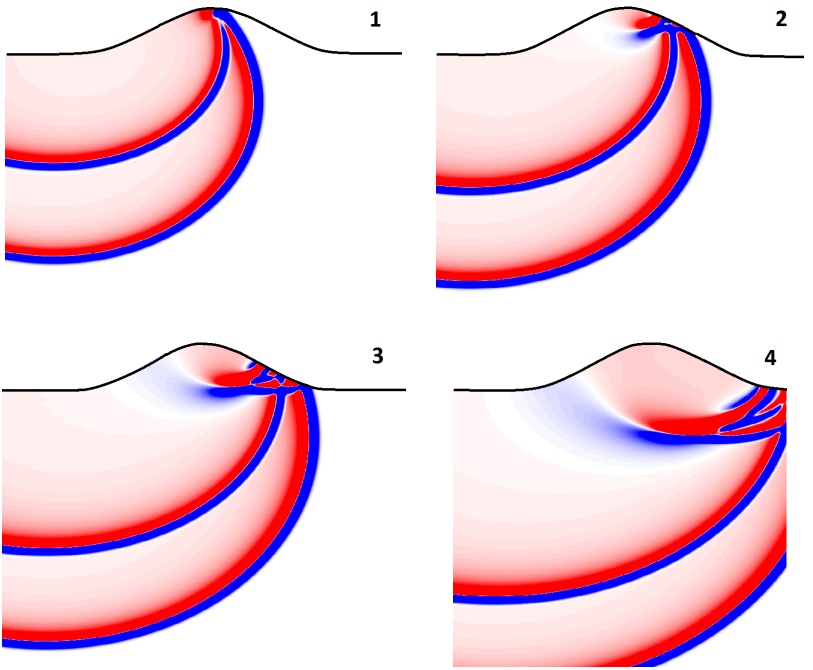


Рис. 7. Снимки волнового поля в последовательные моменты времени 1, 2, 3, 4

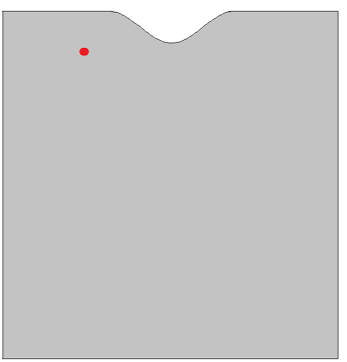


Рис. 8. Область с “впадиной”

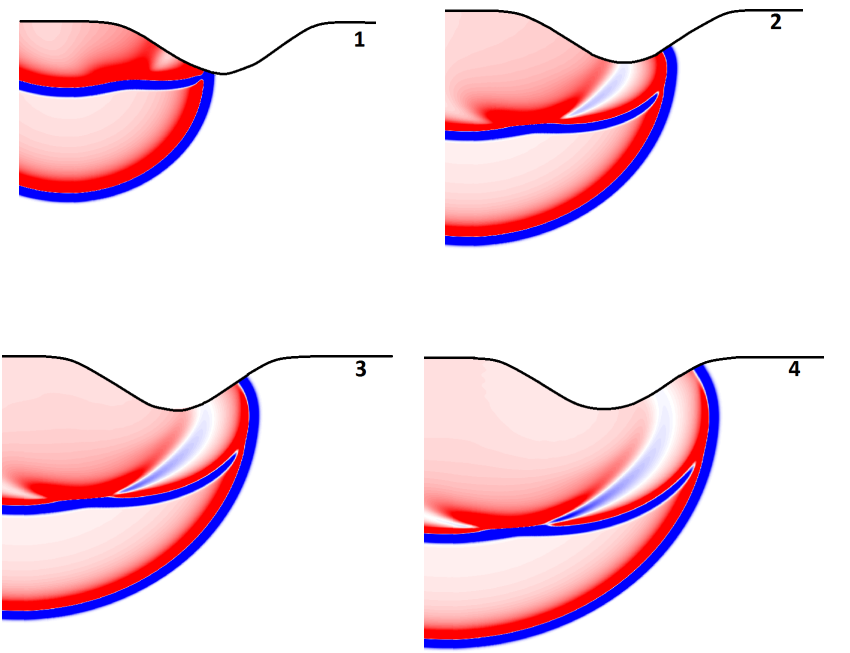


Рис. 9. Снимки волнового поля в последовательные моменты времени 1–4

с криволинейной свободной поверхностью, характерной для магматических вулканов.

Научный руководитель — д-р техн. наук Б. М. Глинский

Список литературы

[1] Лисейкин В. Д., Рычков А. Д., Кофанов А. В. Технология адаптивных сеток для решения прикладных задач. Новосибирск: Изд-во НГУ, 2011. ISBN 978-5-94356-981-4.

[2] Демидов Г. В., Мартынов В. Н. Пошаговый метод решения эволюционных задач с использованием функций Лагерра // СибЖВМ. 2010. Т. 13, № 4. С. 413–422.

[3] Mikhailenko B. G. Spectral Laguerre method for the approximate solution of time dependent problem // Appl. Math. Lett. 1999. № 12. P. 105–110.

[4] Хакимзянов Г. С., Шокин Ю. И. Лекции по разностным схемам на подвижных сетках. Ч. 2. Задачи для уравнений в частных производных с двумя пространственными переменными" // Алма-Ата: Редакционно-издательский центр КазНУ им. Аль-Фараби, 2006. С. 87–90.

[5] Фадеев Д. К., Фадеева В. Н. Вычислительные методы линейной алгебры // М.: Физматгиз, 1960. С. 214–220.

[6] Reinders J., Jeffers J. Intel Xeon Phi coprocessor high performance programming. Morgan Kaufman Publ Inc, 2013.

Титов Павел Андреевич — аспирант Института вычислительной математики и математической геофизики СО РАН; e-mail: tapawel@gmail.com

ТРУДЫ КОНФЕРЕНЦИИ МОЛОДЫХ УЧЕНЫХ
Новосибирск, 2014 С. 122–133

Средства задания прямого управления во фрагментированных программах и их применение на примере явного метода решения уравнения Пуассона

А. А. Ткачёва

УДК 004.272.2; 004.45

Рассмотрена проблема эффективного выполнения фрагментированной программы (ФП) в системе фрагментрованного программирования LuNA. Для повышения производительности выполнения ФП были разработаны средства задания прямого управления в виде предикатной сети Петри. Реализован модуль прямого управления RuSh. Представлено исследование производительности модуля RuSh на задаче решения уравнения Пуассона явным методом. Результаты сравнивались с реализацией выполнения ФП в системе LuNA и выполнением ФП в системе LuNA совместно с RuSh.

The problem of efficient execution of fragmented program (FP) under the fragmented parallel programming system LuNA is considered. To achieve good performance of FP execution the control flow means was designed in the form of predicate Petri nets. The control flow module RuSh was developed. Evaluation of RuSh performance was examine on the s solution of Poisson equation by explicit method. Obtained results were compared with FP execution under system LuNA and FP execution under system LuNA together with Rush.

Ключевые слова: параллельное программирование, системные алгоритмы параллельной обработки данных, предикатная сеть Петри.

Keywords: parallel programming, system algorithms of parallel processing data, predicate/transition nets.

Введение. Развитие параллельных вычислительных технологий позволяет решать задачи численного моделирования, которые раньше невозможно было реализовать на последовательных компьютерах. Однако возможности параллельных вычислений непосредственно связаны со сложностями параллельного программирования.

Для высокой производительности прикладная программа должна обладать следующими динамическими свойствами: настройка на имеющиеся ресурсы вычислителя, динамическая балансировка загрузки процессоров, осуществление коммуникаций на фоне счета и т. д. Сейчас в большинстве случаев эти свойства реализуются вручную с помощью MPI, но делаются попытки реализации систем программирования и библиотек, в которых динамические свойства обеспечиваются автоматически, например LuNA, Charm++, SMP Superscalar, PLASMA.

Фрагментированное программирование — технология параллельного программирования, предназначенная для реализации больших численных задач на суперкомпьютере. В ней фрагментированная программа (ФП) представляется декларативно в виде множества фрагментов данных (ФД) и фрагментов вычислений (ФВ), и такая фрагментированная структура ФП сохраняется в ходе вычислений, что позволяет автоматически обеспечивать динамические свойства программы. Такой подход позволяет программировать на более высоком уровне и не заботиться о сложностях системного параллельного программирования. При этом описание фрагментированного алгоритма (ФА) является платформенно независимым, а настройку на конкретный вычислитель обеспечивает runtime-система системы фрагментированного программирования. В настоящее время в лаборатории синтеза параллельных программ ИВМиМГ СО РАН разрабатывается система фрагментированного программирования LuNA (Language for Numerical Algorithm) [1, 2].

Runtime-системе LuNA необходимо одновременно выполнять множество функций, таких как задание динамического распределения ресурсов, определение ФВ, готовых к исполнению, и назначение на исполнение одного из них, нескольких, всех или ни одного. Из-за этого на управление вычислениями система LuNA тратит много времени, в зависимости от задачи это может привести к стократному ухудшению времени исполнения ФП по сравнению с ручной реализацией с использованием MPI.

Чтобы снизить накладные расходы, целесообразно определить статически прямое управление там, где это возможно, оставив некоторую необходимую степень недетерминизма выполнения ФП для возможности параллельного исполнения на мультикомпьютере и обеспечения динамических свойств выполнения программы.

Для этих целей был разработан модуль прямого управления RuSh (Runtime Shell), являющийся компонентом системы LuNA. Его задачей является параллельное выполнение некоторого подмножества ФВ, входящего в ФП, под прямым управлением без необходимости дополнительных проверок ФВ на готовность к исполнению.

В работе представлено исследование производительности модуля RuSh на задаче решения уравнения Пуассона явным методом при трехмерной фрагментации области. Результаты сравнивались с реализацией выполнения ФП в системе LuNA и выполнением ФП в системе LuNA совместно с RuSh.

1. Постановка задачи. Задачи численного моделирования отличаются регулярной структурой: массовость данных и операций (одни и те же алгоритм обработки ФД, размер ФД, примерно совпадающий объем вычислений при обработке). Использование этого свойства позволяет уменьшить сложность задачи управления, так как на множестве однородных по количеству вычислений ФВ алгоритма одно решение по управлению может быть использовано для всех ФВ. Такой подход позволяет разбить большую и трудоемкую задачу управления вычислениями во ФП на множество более простых и менее затратных подзадач.

Разрабатываемые средства задания прямого управления должны удовлетворять следующим требованиям:

— позволять задавать желаемый порядок для достаточно широкого класса численных алгоритмов; — задаваемое управление должно допускать эффективную реализацию на мультикомпьютере.

2. Подход к решению. Прямое управление [3] может задаваться различными способами:

— как отношение частичного порядка;

— циклами типа while или типа for;

— как сеть Петри.

Отношение частичного порядка хотя и является универсальным способом задания управления, однако такой способ труден для человеческого восприятия. Задание управления с помощью циклов типа while или типа for и сети Петри более наглядно.

Использование циклов типа while или типа for позволяет накладывать управление между итерациями цикла, т. е. использование специализированных циклов, все итерации которых выполняются либо независимо, либо последовательно, одна за другой. Недостат-

ком этого средства задания прямого управления является то, что если тело цикла состоит из ФВ, которые можно выполнять параллельно, эти ФВ будут выполняться все равно последовательно.

Использование в качестве средства задания прямого управления предикатной сети Петри [4] позволяет задавать управление для более общего случая, а именно задавать параллелизм выполнения ФВ и внутри тела циклов, и между итерациями циклов.

Идея использования предикатной сети Петри состоит в том, что каждому ФД соотносится соответствующее место, а ФВ — переход, предикат условия срабатывания ФВ — предикату условия срабатывания перехода. Нахождение фишки в месте означает, что соответствующий ФД вычислен и доступен для множества тех ФВ, для которых ФД является входным ФД. Если во всех входных местах некоторого перехода есть фишки, то это означает, что должен быть запущен на выполнение соответствующий ФВ.

Тогда порядок выполнения ФВ будет определяться предикатной сетью Петри, при этом, в отличие от runtime-системы LuNA необходимо проверять только наличие фишек в местах.

3. Реализация модуля прямого управления RuSh. Средства для задания управления в ФП были реализованы в виде программного модуля RuSh, который обеспечивает параллельное выполнение ФП в общей памяти, в которой порядок выполнения ФВ определяется функционированием предикатной сети Петри.

Особенности и ограничения реализации:

— в разработанном программном модуле поддерживается только задание управления в виде безопасной предикатной сети Петри (не более одной фишки в месте) — это условие ограничивает реализацию обмена ФД между ФВ одним буфером памяти;

— кроме того, можно задавать частичное распределение ресурсов, а именно задавать использование одного буфера памяти для хранения разных ФД.

4. Явный метод решения уравнения Пуассона. Уравнение Пуассона имеет вид

$$\frac{\partial^2 F}{\partial x^2} + \frac{\partial^2 F}{\partial y^2} + \frac{\partial^2 F}{\partial z^2} = p,$$

При этом оно аппроксимируется следующей 7-точечной разностной схемой второго порядка:

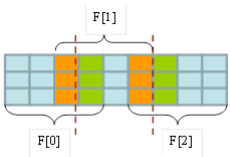


Рис. 1. Схема фрагментации данных при одномерной фрагментации области для явного метода решения уравнения Пуассона

$$\frac{F_{i+1,j,k} - 2F_{i,j,k} + F_{i-1,j,k}}{h_x^2} + \frac{F_{i,j+1,k} - 2F_{i,j,k} + F_{i,j-1,k}}{h_y^2} + \\ + \frac{F_{i,j,k+1} - 2F_{i,j,k} + F_{i,j,k-1}}{h_z^2} = p_{i,j,k}, \qquad (1)$$

Явный метод решения уравнения Пуассона является итерационным, на каждой итерации происходит уточнение решения относительно начального для некоторой области моделирования. Формула вычисления значений на новой итерации получается, если выразить $F_{i,j,k}$ из схемы (1). В явном методе для вычисления сеточного значения на новой итерации все используемые в формуле значения берутся с предыдущей итерации. Итерационный процесс продолжается до сходимости $\max_{i,j,k} |F_{i,j,k}^{m+1} - F_{i,j,k}^{m}| < \epsilon$. Здесь индекс m — номер итерации.

В основе выбранного способа распараллеливания явного метода лежит принцип фрагментации области моделирования, который в данном случае эквивалентен пространственной декомпозиции области. При этом область моделирования разбивается на слои, и расчет каждого слоя на каждой итерации выполняется отдельным ФВ.

На рис. 1 приведен пример фрагментации данных для двумерного пространства моделирования при одномерной фрагментации на три фрагмента.

Так как для расчета $F_{i,j,k}$ используется 7-точечный шаблон, то для параллельной реализации решения уравнения Пуассона требуется обмен теневыми гранями, а именно обмен значениями $F_{i,j,k}$ на границах разрезания слоев между соседними слоями области моделирования.

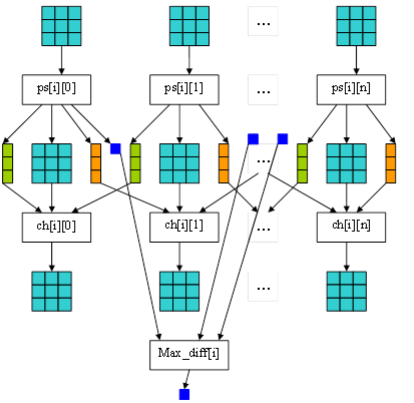


Рис. 2. Схема ФА i-й итерации явного метода решения уравнения Пуассона для двумерной области моделирования и одномерной фрагментации

Фрагментированный алгоритм i-итерации явного метода решения уравнения Пуассона изображен на рис. 2, где n — количество фрагментов, на которые разбивается область моделирования; $ps[i][j](j = 0, ..., n)$ — ФВ, обрабатывающий $j-$ слой области моделирования на i-й итерации, $in = (F[i][j])$, $out = (F'[i][j], right[i][j-1], left[i][j+1], mx[i][j])$; $ch[i][j](j = 0, ..., n$ — ФВ, вычисляющий обмен теневыми гранями для j-го слоя, $in = (F'[i][j], left[i][j], right[i][j])$, $out = (F[i+1][j])$; $Max_diff[i]$ — ФВ, вычисляющий редукцию невязки на i-й итерации, $in = (mx[i][j], j = 0..n)$, $out = (mxm[i+1])$.

Использование сети Петри для реализации ФА явного метода решения уравнения Пуассона, приведенной на рис. 3, ограничено памятью одного узла, так как программный модуль RuSh работает в общей памяти мультикомпьютера, в то время как система LuNA может работать как с общей, так и с распределенной памятью.

На рис. 3 изображена сеть Петри для i-й итерации явного метода решения уравнения Пуассона, где $pu[i][j]$, $(j = 0, ..., n)$ — переход, срабатывание требует выполнения ФВ $ps[i][j]$; $c[i][j]$, $(j = 0, ..., n)$ — переход, срабатывание которого требует выполнения ФВ $ch[i][j]$; $calc[i]$ — переход, срабатывание которого требует выполнения ФВ $Max_diff[i]$.

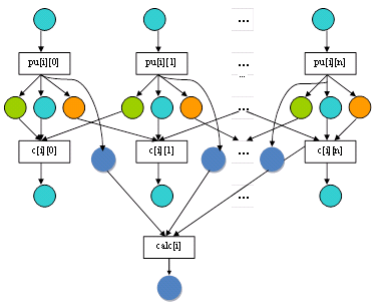


Рис. 3. Сеть Петри для i-й итерации явного метода решения уравнения Пуассона (двумерная область моделирования и одномерная фрагментация области)

Поэтому для параметров задачи, при которых данные не помещаются в память одного узла, имеет смысл использовать комбинацию базового алгоритма выполнения ФП LuNA и модуля RuSh для выполнения ФП в общей памяти.

Структурированный ФВ (СФВ) является аналогом процедуры в языках программирования, для него определяется множество входных и выходных ФД и всех ФВ, входящих в СФВ и выполняющихся в рамках одного узла вычислителя.

Все ФВ, вычисляющие j-й слой на i-й итерации ($j = 0, ..., n$) объединим в группу СФВ фиксированного размера, и внутри СФВ порядок выполнения ФВ определяется функционированием предикатной сети Петри и выполняется в RuSh, а обмен данными между СФВ осуществляется с помощью базового алгоритма выполнения ФП системы LuNA.

На рис. 4 приведена схема ФА одной итерации явного метода решения уравнения Пуассона в случае комбинированного использования алгоритмов выполнения LuNA и RuSh, где m — количество СФВ, на которые разбиваются все ФВ, вычисляющие j-й слой на i-й итерации ($j = 0, ..., n$); $L = n/m$ — количество ФВ, объединенных в рамках одного СФВ; $P[i][k]$ ($k = 0, ..., m$) — СФВ, вычисляющий несколько сгруппированных ФВ на i-й итерации. $in = (F[i][j]|j = k*L..(k+1)*L)$, $out = (F'[i][k*L]$, $F[i+1][j]|j = k*L+1..(k+1)*L-1$, $F'[i][(k+1)*L], left[i][(k+1)*L+1], right[i][k*L-1])$. В рамках СФВ порядок определяется функционированием предикатной сети Петри (см. рис. 3); $[i][k]$,

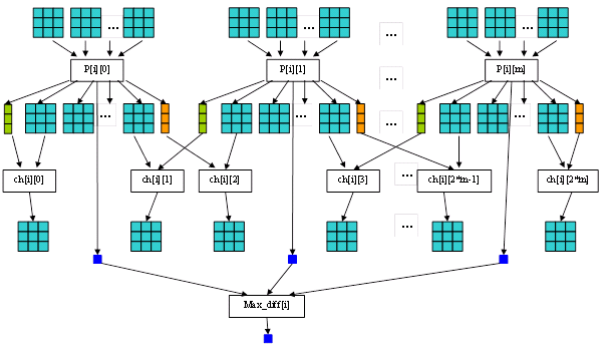


Рис. 4. Схема ФА i-й итерации явного метода решения уравнения Пуассона при использовании структурированных ФВ

$(k = 0, ..., m)$ — ФВ, вычисляющий обмен теневыми гранями для тех слоев области моделирования, которые находятся на границе. $in = (type, F'[i][j], border[i][j])out = (F[i+1][j])$.

4. Исследования производительности исполнения ФП. Для тестирования в общей памяти использовался 6-ядерный вычислитель со следующими параметрами: Intel Xeon CPU X5600 2.8Ghz. При запуске программ использовались четыре рабочих потока в LuNA и в RuSh.

В качестве прикладной задачи для исследования производительности был выбран явный метод решения уравнения Пуассона со следующими параметрами: трехмерная область моделирования, 7-точечный шаблон, трехмерная фрагментация данных.

Необходимо было провести следующие тесты.

1. Изменение накладных расходов в модуле RuSh в зависимости от количества ФВ, для которых задано прямое управления предикатной сетью Петри.

2. Сравнение времени работы распараллеленной ручной реализации прикладной задачи с использованием MPI и ФП прикладной задачи в системе LuNA в зависимости от размера задачи.

3. Сравнение производительностей следующих реализаций:

— выполнение ФП решения прикладной задачи в системе LuNA;

— выполнение ФП решения прикладной задачи с заданным прямым управление с помощью предикатной сети Петри в модуле RuSh;

Рис. 5. Зависимость удельного времени работы на вычисление одного ФВ от количества ФВ, обрабатываемых в RuSh. Размер ФД $1 \times 1 \times 1$ и 100000 итераций

— выполнение ФП решения прикладной задачи комбинированно в LuNA и RuSh, а именно: все ФВ, отвечающие за расчет i-й итерации, делятся на группы СФВ (для каждого СФВ порядок выполнения ФВ определен сетью Петри) и исполняются в RuSh; обмен данными между СФВ осуществляется с помощью LuNA.

4. Изменение времени на вычисление одного ФВ в зависимости от размера СФВ.

Тест 1. На рис. 5 представлены результаты теста 1. Заметим, что с увеличением количества ФВ, на которых задано прямое управление, время не увеличивается для достаточно большого числа ФВ.

Тест 2. На рис. 6 видно, что на достаточно большом размере прикладной задачи время работы в системе LuNA сравнимо по уровню с ручной реализацией в MPI.

Тест 3. На рис. 7 представлены результаты теста 3. Если сравнить время, затрачиваемое на выполнение ФП в системе LuNA и в модуле RuSh, то выполнение ФП в RuSh занимает намного меньше времени из-за отсутствия накладных расходов связанных с управлением. Рассмотрим теперь комбинированный случай: с одной стороны время работы увеличивается по сравнению с использованием только RuSh, так как тратится время на организацию управления вычисле-

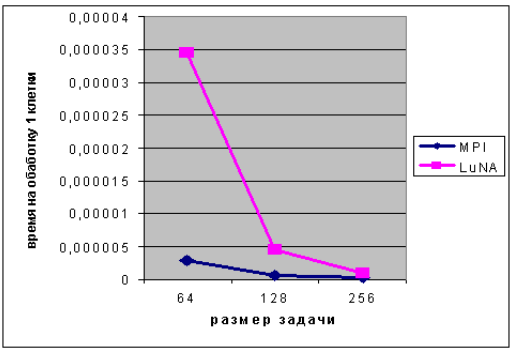


Рис. 6. Сравнение времени работы реализаций прикладной задачи в MPI и LuNA (8 процессов, количество ФД 8, итераций 10) в зависимости от размера задачи

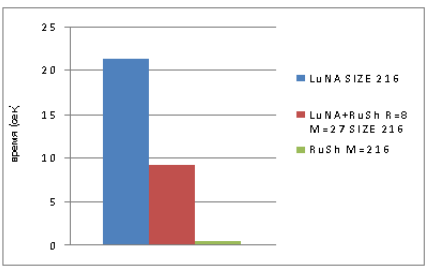


Рис. 7. Сравнение времени, затраченного на выполнение ФП с использованием LuNA, LuNA+RuSh, RuSh для степени фрагментации 216 для явного метода решения уравнения Пуассона (размер сетки: $100 \times 100 \times 100$, 40 временных итераций). SIZE — количество ФД, на которые разбивается область моделирования; M — количество ФВ внутри СФВ; R — количество СФВ, каждый такой СФВ обрабатывается на одном узле в RuSh

ниям между СФВ, с другой — даже частичное использование RuSh позволяет почти в два раза повысить производительность выполнения ФП.

Тест 4. На рис. 8 видно, что увеличение количества ФВ, обрабатываемых в рамках одного СФВ, приводит к росту производительности исполнения ФП.

Рис. 8. Зависимость времени обработки одного ФВ от размера СФВ (одномерная фрагментация, размер ФД $20 \times 20 \times 20$, 400 итераций)

5. Результаты тестирования. Результаты тестов показали, что для задачи достаточно большого размера производительность системы LuNA сравнима по уровню с ручной реализацией в MPI. А использование модуля прямого управления RuSh для исполнения СФВ, управления в которых задано предикатной сетью Петри, повышает производительность системы LuNA в общей памяти. При этом накладные расходы модуля RuSh на управление выполнением СФВ в несколько раз меньше, чем в системе LuNA, и чем больше количество ФВ, обрабатываемых в рамках одного СФВ, тем больше выигрыш по производительности.

6. Заключение. В работе исследованы способы применения средств задания прямого управления в ФП одновременно с базовым алгоритмом выполнения ФП системы LuNA на примере решения уравнения Пуассона явным методом. Были проведены тесты производительности в общей памяти, которые позволяют говорить о том, что можно повысить производительность выполнения ФП на мультикомпьютере, используя прямое управление.

В дальнейшем предполагается провести аналогичное тестирование в распределенной памяти и исследовать применимость прямого управления для решения других задач численного моделирования.

Научный руководитель — д.т.н., проф. В. Э. Малышкин.

Список литературы

[1] Malyshkin V. E., Perepelkin V. A. LuNA fragmented programming system, main functions and peculiarities of run-time subsystem // Proc. of the 11-th conf. on parallel computing technologis LNCS 6873. Springer, 2011. P. 53–61.

[2] Kraeva M. A., Malyshkin V. E. Assembly technologyfor parallel realization of numerical models on MIMD-Multicomputers // J.l on Future Generation Computer Systems. Elsiever Sci. 2001. V. 17, N. 6. P. 755–765.

[3] Малышкин В. Э., Корнеев В. Д. Параллельное программирование мультикомпьютеров. Сер.: "Учебники НГТУ". Новосибирск: Изд-во НГТУ, 2006. 296 стр.

[4] Genrich H. G. Predicate/Transition nets // Lect. Notes in Computer Science 254. Springer-Verlag. 1987. P. 207–247

Ткачёва Анастасия Александровна —
мл. науч. сотр. Ин-та вычислительной математики
и математической геофизики СО РАН
e-mail: tkacheva@ssd.sscc.ru

Содержание

ТРУДЫ
конференции
молодых ученых

Оформлено в системе LaTeX, макрос NCC

Компьютерная подготовка к изданию
О. Г. Заварзиной

Лицензия ИД № 02202 от 30.06.2000 г.
Подписано в печать 9.07.2015 г.
Формат $60 \times 84\,{}^{1}/_{16}$. Гарнитура Computer Modern Roman.
Усл. печ. л. 8,1 . Уч.-изд. л. 7,8. Тираж 70 экз. Заказ № .

ОмегаПринт, лицензия ПЛД № 57-58 от 27.05.99,
630090, Новосибирск, просп. акад. Лаврентьева, 6