Развитие программной платформы INMOST: динамические сетки, линейные решатели и автоматическое дифференцирование*

Д.В. Багаев[1], А.И. Бурачковский[1], А.А. Данилов[2], И.Н. Коньшин[2], К.М. Терехов[2,3]

МГУ, Москва[1], ИВМ РАН, Москва[2], Стэнфордский университет, Стэнфорд[3]

Представляется программная платформа INMOST, предназначенная для работы с распределенными сетками общего вида, выполнения дискретизации, формирования и решения систем линейных уравнений. Приводится краткий обзор возможностей программной платформы, а также ее последнего развития. Разработана и реализована распределенная структура данных для построения динамически сгущаемых и разгрубляемых расчетных сеток на основе восьмидеревьев. На некоторых тестовых матрицах из коллекции университета Флориды была оценена работоспособность встроенных линейных решателей платформы INMOST, а также линейных решателей, подключаемых из внешних пакетов PETSc и Trilinos. Рассматривается также разработанный механизм автоматического дифференцирования, применяемый для составления матрицы линейной системы и формирования правой части. Все разработки доступны на сайте программной платформы INMOST: www.inmost.org.

Ключевые слова: программная платформа, распределенные динамические сетки, система линейных уравнений, автоматическое дифференцирование.

1. Введение

Программная платформа INMOST (Integrated Numerical Modelling and Object-oriented Supercomputing Technologies) [1] была разработана в Институте вычислительной математики РАН. Она предназначена для обеспечения пользователя всеми необходимыми средствами при создании и исследовании различных численных моделей. Сюда входит не только работа с распределенными по процессорам сеточными данными для сеток общего вида, но и удобный интерфейс для выполнения дискретизации задачи, а также формирования и решения систем линейных уравнений.

Программная платформа INMOST предоставляет пользователю богатый функционал возможностей, обладает достаточной универсальностью применения, обеспечивает надежность и эффективность вычислений. Стоит также отметить открытость исходного кода [2], простоту установки на компьютер разработчика и удобство ее дальнейшего использования. Немаловажным также является поддержка программной платформы и ее дальнейшее развитие.

Данная работа является логическим продолжением работ [3,4] представленных на предыдущей конференции, показывая новые разработки и проведенные за это время исследования. Развитие программной платформы коснулось большинства этапов численного моделирования: была разработана методика работы с распределенными динамическими сетками на основе использования «восьмидеревьев», на этапе дискретизации была разработана технология автоматического дифференцирования для согласованного формирования линейных систем, на этапе решения линейных систем была исследована параллельная эффективность различных линейных решателей, как входящих в состав программной платформы INMOST, так и решателей из внешних подключаемых пакетов.

Программная платформа INMOST разрабатывалась с учетом следующих критериев:

- богатый функционал возможностей;
- эффективность;
- надежность;

* Работа поддержана грантами РФФИ 14-01-00830, 15-35-20991 и компанией ExxonMobil Upstream Research Company.

- универсальность;
- простота использования;
- открытость исходного кода.

В настоящее время достаточно трудно найти комплексы программ, удовлетворяющие сразу всем вышеперечисленным требованиям. Существующие в 2012 году на начало работы над INMOST решения, такие как, библиотека FMDB (Flexible distributed Mesh DataBase), библиотека MOAB (A Mesh-Oriented datABase), библиотека MSTK (MeSh ToolKit), библиотека STK (Sierra ToolKit), пакет Salome, пакет OpenFOAM (Open Source Field Operation And Manipulation CFD ToolBox) и другие, обладали рядом недостатков и в полной мере не соответствовали заявленным критериям. У существовавших на тот момент решений не всегда была легкая переносимость между различными платформами (Windows, Linux), недостаточно надежные реализации, в некоторых пакетах было невозможно, либо проблематично внедрить свои схемы дискретизации. Однако стоит отметить, что имеются общедоступные пакеты, в некоторый степени, удовлетворяющие выбранным критериям и имеющие достаточно богатый функционал, которые могли бы использоваться для вспомогательных целей. Например, пакеты ParMETIS [5] и Zoltan [6] используются в программной платформе INMOST для распределения и перераспределения данных по процессорам, а пакеты PETSc [7] и Trilinos [8] применяются для решения систем линейных уравнений.

2. Динамические сетки

При решении многих задач математической физики возникает необходимость использования динамических сеток. Это могут быть, например, задачи с движущимися объектами, моделирование течения жидкости со свободной поверхностью, моделирование течений с ударными волнами или рассмотрение задач с динамической адаптацией сетки к решению. Если процесс моделирования происходит на многопроцессорной вычислительной системе с распределенной памятью, то помимо построения адаптируемой сетки и ее изменения на каждом шаге по времени встает также задача динамического перераспределения ячеек между процессорами для более эффективного использования ресурсов вычислительной системы за счет балансировки вычислений.

Мы рассматриваем построение расчетных сеток с помощью технологии «восьмидеревьев» (octree), которая позволяет строить сетки, состоящие из кубиков, некоторые грани которых могут быть разбиты на 4 части, за счет измельчения вдвое шага сетки в соседних ячейках. Эта технология (см., например, [9]) позволяет строить сетки, адаптируемые во всех трех измерениях (см. рис. 1). Однако для большей наглядности, в дальнейших иллюстрациях мы будем приводить вид сверху, несмотря на то, что все сетки будут оставаться трехмерными. Заметим также, что построенные сетки при измельчении остаются конформными.

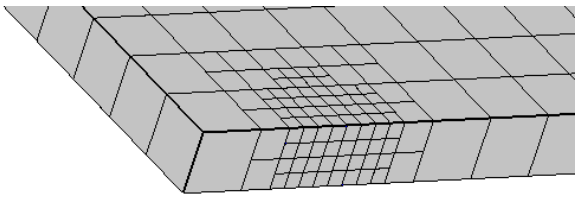

Рис. 1. Сгущение сетки во всех 3-х измерениях

Для рассмотрения динамических сеток на основе «восьмидеревьев» мы использовали модельный пример со сгущением и разгрублением сетки вслед за движением курсора мыши (см. рис. 2).

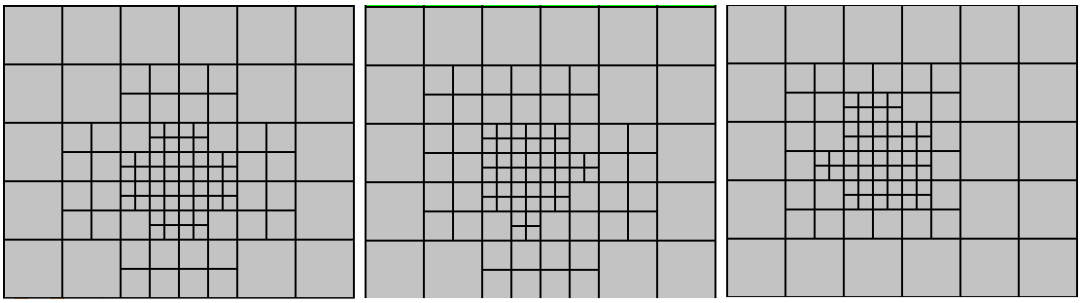

Рис. 2. Сгущение сетки вслед за движением курсора мыши

При разработке многопроцессорной версии работы с сетками на основе технологии «восьмидеревьев», необходимо было разработать структуру данных, которая была бы локальной для каждой из ячеек сетки. Это необходимо для того чтобы можно было сохранять ее в тегах программой платформы INMOST, прикрепляемых к каждой ячейке, для дальнейшей организации обменов при согласованном изменении сетки на границе между процессорами (см. рис. 3, где различными, по возможности, цветами показаны ячейки, принадлежащие различным процессорам). В рассматриваемую структуру данных входит информация об «уровне» ячейки (т.е. количестве ее «предков»), «расположении» самой ячейки внутри «родительской», а также вся наследуемая информация от «родительских» ячеек. Информацию от «родительских» ячеек необходимо сохранять внутри «потомков», т.к. «родительская» ячейка удаляется из сетки при разбиении ее на 8 «потомков» с вдвое меньшим шагом сетки, а хранить ее неявно в некоторой глобальной структуре не представляется возможным из-за использования многопроцессорной архитектуры с распределенной памятью.

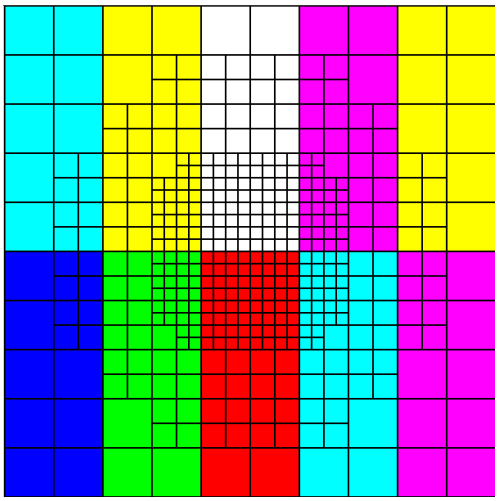

Рис. 3. Согласованное измельчение сетки

После того как построена согласованная сетка на нескольких процессорах, необходимо проанализировать количество полученных ячеек на каждом из процессоров, а также их отличие от среднего значения. В принципе, может оказаться так, что большая часть сетки находится на одном из процессоров, в этом случае ситуация будет не слишком отличаться от последовательного решения задачи (см. рис. 4). Таким образом, при работе с динамическими сетками одним из важных этапов работы программной платформы INMOST становится балансировка сетки.

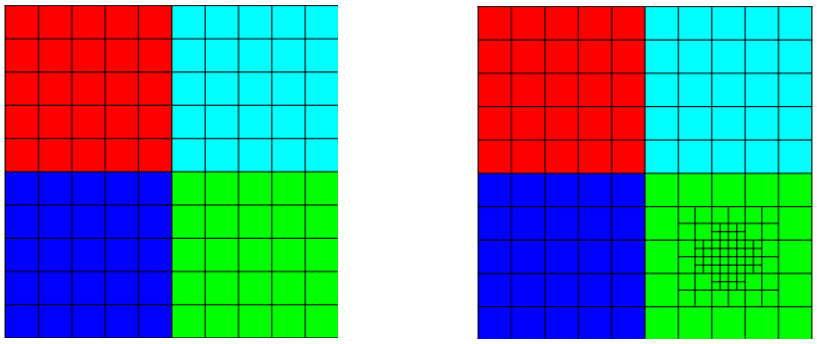

Рис. 4. Необходимость балансировки сетки

Наряду с выполнением обменов данными между процессорами, программная платформа INMOST позволяет производить перераспределение сетки между процессорами, обеспечивая сбалансированность вычислительной нагрузки. Удобнее всего воспользоваться внешними пакетами ParMetis [5] или Zoltan [6], применение которых при сборке INMOST указывается флагами «USE_PARTITIONER_PARMETIS=ON» или «USE_PARTITIONER_ZOLTAN=ON», соответственно. Перераспределение сетки в рамках программной платформы INMOST характеризуется следующими «действиями»:

- «Partition» – распределение сетки без использования информации о текущем распределении, оно может занимать значительное время и обычно вызывается только один раз перед началом вычислений;
- «Repartition» – перераспределение сетки, которое, по возможности, оставляет большинство ячеек на тех же процессорах, где они и находились до вызова перераспределения, это «действие» также может занимать достаточно большое время, но зато оно старается минимизировать пересылки данных при обмене ячейками, может вызываться, например, на каждой 10-й итерации по времени;
- «Refine» – попытка сбалансировать распределение небольшим количеством обменов ячейками между соседними процессорами, качество балансировки не гарантировано, однако время выполнения минимально, так что это перераспределение может вызываться на каждом шаге по времени.
- Выбор «действия» и конкретного типа перераспределения осуществляется с помощью функции Partitioner::SetMethod(), а само перераспределение проводится при вызове функции Partitioner::Evaluate() и Mesh::Redistribute().

На рис. 5 показаны три сетки, распределенные на 9 процессоров: исходная сетка, при использовании равномерного геометрического распределения; сетка, полученная после выполнения двухуровневой адаптации к центральной точке; и сетка, полученная после применения перераспределения «Repartition» с целью сбалансировать количество узлов на различных процессорах.

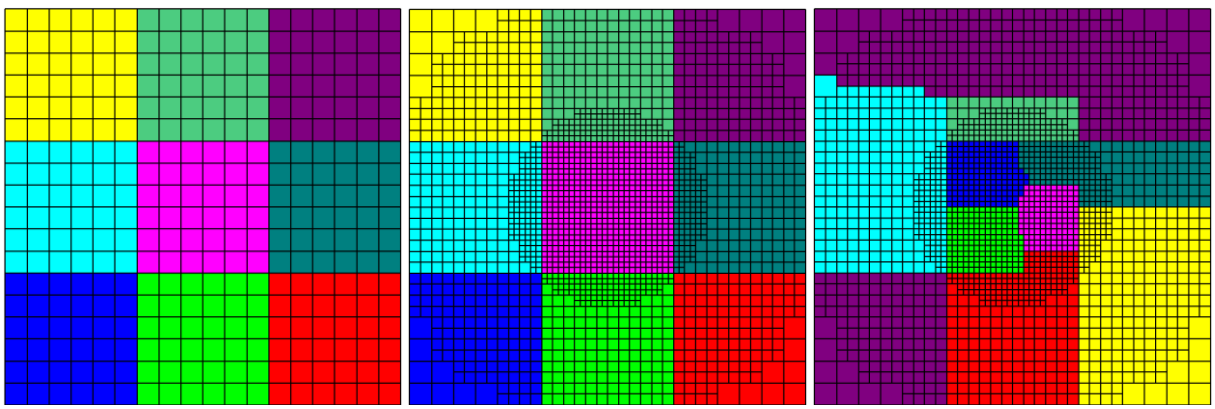

Рис. 5. Распределенные сетки: исходная, адаптированная, после балансировки

Таким образом, описан механизм работы с распределенными адаптивными динамическими сетками на основе технологии «восьмидеревьев». В принципе, сторонние пользователи программной платформы INMOST, могут напрямую применять разработанную распределенную структуру данных. Для этого необходимо в соответствии с потребностями решаемой задачи переопределить функции «cell_should_split()» и «cell_should_unite()», которые контролируют процессы сгущения и разгрубления сетки. Описанный пример работы с динамическими сетками выложен в открытый доступ на сайт INMOST [2] в разделе «Examples».

3. Решение систем линейных уравнений

После построения расчетной сетки и проведения дискретизации исходной задачи возникает необходимость решения полученной системы линейных уравнений. В программной платформе INMOST реализован специальный модуль «Solver», предназначенный для составления и решения линейных систем. INMOST предоставляет большой выбор различных решателей линейных систем, некоторые из них непосредственно встроены в программную платформу, а некоторые подключаются как отдельные пакеты, например, PETSc [7] или Trilinos [8]. Среди встроенных линейных решателей можно выделить:

- Inner ILU2;
- Inner DDPQILUC;
- Inner MPTILUC;
- Inner MPTILU2;
- Inner FCBIILU2;
- Inner K3BIILU2.

Среди дополнительных внешних линейных решателей, поддерживаемых платформой INMOST, мы в дальнейшем будем рассматривать следующие:

- Trilinos Aztec;
- Trilinos Belos;
- Trilinos ML;
- Trilinos Ifpack;
- PETSc.

Целью данного раздела было исследование свойств вышеперечисленных линейных решателей на некоторых задачах из наиболее популярной и представительной коллекции тестовых матриц Университеты Флориды [10]. В качестве тестовых мы выбрали 5 достаточно плохообусловленных матриц (см. табл. 1).

Таблица 1. Свойства тестовых матриц из коллекции университета Флориды

Матрица	*N*	NZA	NZA/N
tmt_unsym	917 825	4 584 801	5,0
tmt_sym	726 713	5 080 961	7,0
ML_Laplace	377 002	27 582 698	73,1
Transport	1 602 111	23 487 281	14,6
CoupCons3D	416 800	17 277 420	41,4

Рассматриваемые матрицы имеют различное происхождение:

- «tmt_unsym» – моделирование задач электромагнетизма;
- «tmt_sym» – моделирование задач электромагнетизма;
- «ML_Laplace» – дискретизация уравнения Пуассона;
- «Transport» – конечноэлементная дискретизация 3-х мерной задачи переноса;

- «CoupCons3D» – полностью связная задача пороэластичности.

Численные эксперименты проводились на кластере ИВМ РАН [11]. Характеристики вычислительных узлов, используемых для проведения расчетов:

- Compute Node Asus RS704D-E6;
- 12 ядер (два 6-ядерных процессора Intel Xeon X5650@2.67ГГц);
- Оперативная память: 24 Гб.;
- Дисковая память: 280 Гб.;
- Операционная система: SUSE Linux Enterprise Server 11 SP1 (x86_64).

Кластер ИВМ РАН поддерживает самые последние версии компиляторов Intel для языков программирования C, C++ и Fortran. Для поддержки параллельного решения линейных систем использовалась версия INMOST, скомпилированная с поддержкой версии MPI 5.0.3.

Стоит отметить, что производительность линейного решателя может зависеть от конфигурации запуска приложения. Например, на рис. 6 показано время решения задачи «tmt_unsym» на различных количествах вычислительных ядер. При этом оказалось, что время работы итерационной схемы на 4-х ядрах одного узла оказалось больше чем аналогичное время на одном ядре того же узла (при этом замедления этапа факторизации не наблюдалось). При проведении тех же самых вычислений на 4-х ядрах, выделенных на 4-х различных узлах, уже наблюдалось ожидаемое ускорение этапа проведения итераций. Скорее всего, это вызвано особенностями доступа к памяти при интенсивном ее использовании на этапе проведения итераций.

На представленных рисунках 7-11 показана зависимость общего времени решения линейной системы от используемого количества вычислительных ядер 1, 4, 12, 36, 72 при решении каждой из 5 рассматриваемых задач. Следует отметить, что не все линейные решатели справились с этими задачами, в том случае, когда при уменьшении начальной невязки в 10^6 раз сходимость не была достигнута за 2000 итераций, задача считалась не решенной и данные по этому решателю на графике не приводились. Стоит отметить, что встроенные линейные решатели показали достаточно высокую надежность и масштабируемость.

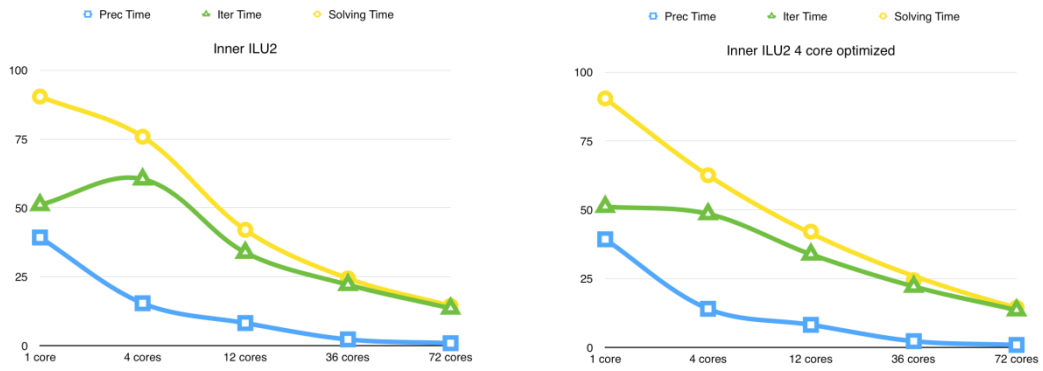

Рис. 6. Производительность (в сек.) линейного решателя Inner_ILU2 для задачи «tmt_unsym» при различных конфигурациях запуска на 4 ядра

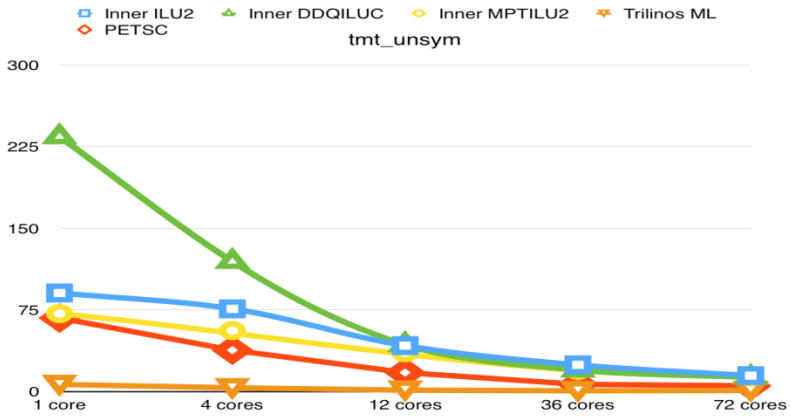

Рис. 7. Производительность (в сек.) линейных решателей для задачи «tmt_unsym»

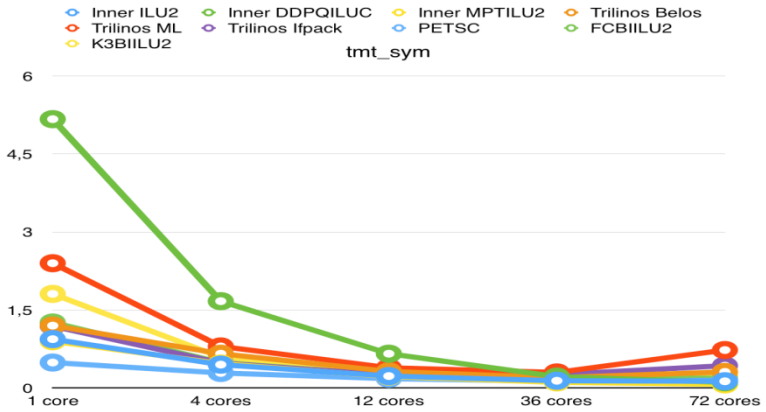

Рис. 8. Производительность (в сек.) линейных решателей для задачи «tmt_sym»

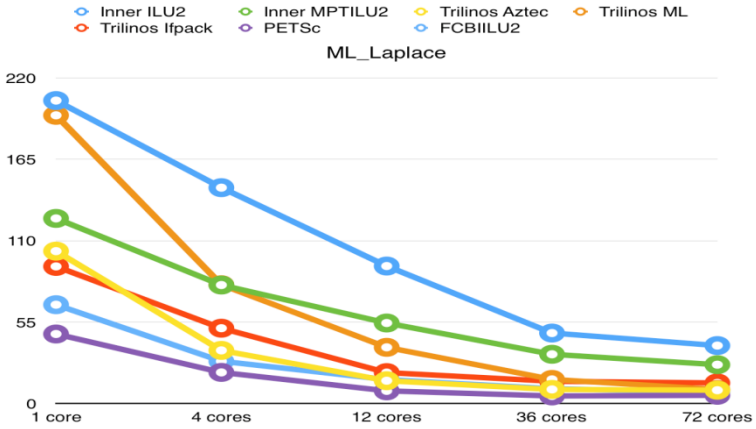

Рис. 9. Производительность (в сек.) линейных решателей для задачи «ML_Laplace»

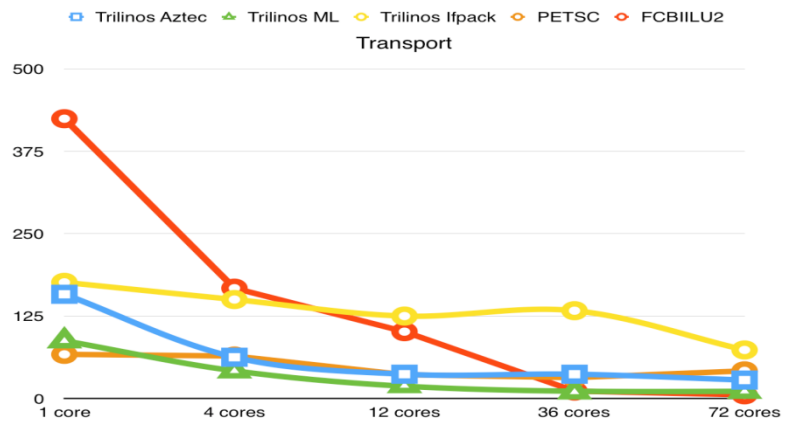

Рис. 10. Производительность (в сек.) линейных решателей для задачи «Transport»

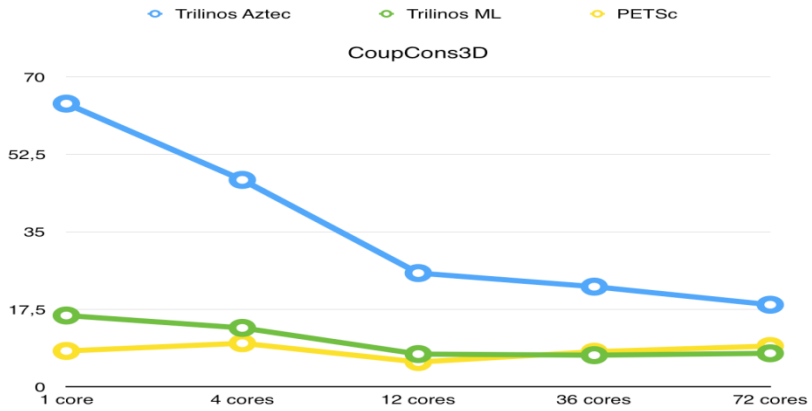

Рис. 11. Производительность (в сек.) линейных решателей для задачи «CoupCons3D»

4. Автоматическое дифференцирование

Программная платформа INMOST включает в себя модуль автоматического дифференцирования. Основная функция модуля – облегчение разработки математических моделей, требующих построения матрицы первых производных (матрицы якобиана). Построение якобиана, как правило, требуется для моделей с неявным интегрированием по времени, либо для стационарных моделей. Данный модуль особенно полезен при разработке сложных нелинейных моделей, например, переноса смеси химически активных материалов.

Конкретные примеры работы модуля автоматического дифференцирования также выложены в открытый доступ на сайт INMOST [2] в разделе «Examples».

4.1 Основные структуры и детали реализации

Матрица якобиана строится автоматически при выполнении арифметических операций над данными. Автоматическое построение основано на правиле цепочки. Правило цепочки гласит, что вне зависимости от сложности дифференцируемого выражения, итоговое выражение является суммой частных производных с коэффициентами. Арифметические операции для построения выражений определены для трех типов данных:

- константы;
- переменной с одной частной производной;
- переменной с множеством частных производных.

Первый тип данных является стандартным типом «double», два последних типа данных представлены в виде отдельных классов.

Класс переменной с одной частной производной содержит значение переменной и позицию производной в матрице якобиана. Этот класс предназначен для экономного представления в памяти основных неизвестных модели.

Класс переменной с множеством частных производных содержит значение переменной и разреженный вектор пар значений. Каждая пара значений состоит из коэффициента производной и позиции производной в матрице якобиана. Класс переменной с множеством частных производных предназначен для представления в памяти результата арифметических операций над основными неизвестными модели.

Каждая из арифметических операций технически представлена в виде шаблона класса. В каждом шаблоне класса имеются функции для получения значения «GetValue()» и для вычисления производных «GetJacobian()». Функция для получения производных реализует правило дифференцирования сложной функции, соответствующее заданной классом арифметической операции. Например, функция «GetJacobian()» для арифметической операции «cos(x)» должна реализовывать «-sin(x)dx». При этом «dx» является вызовом функции «GetJacobian()» для аргумента «x» с множителем «-sin(x)». Для удобства построения арифметических выражений, каждому шаблону класса сопоставляется шаблон перегруженной функции, например, «sin(x)»; либо шаблон перегруженного оператора, например, оператора «+». Шаблон перегруженной функции «sin(x)» возвращает шаблон класса «sin_expression». В зависимости от класса аргумента «x», например, «double», компилятор строит фактическую функцию «sin(x)», которая возвращает фактический класс «sin_expression<double>». В итоге на этапе компиляции программы выстраивается дерево на основе шаблонов классов (см. рис. 12), а функция «GetJacobian()» реализует обход данного дерева в глубину. Для ускорения вычислений каждый класс запоминает значение результата соответствующей арифметической операции, чтобы не требовался обход дерева в глубину для функции «GetValue()». В том числе в ряде случаев запоминаются промежуточные коэффициенты для умножения частных производных.

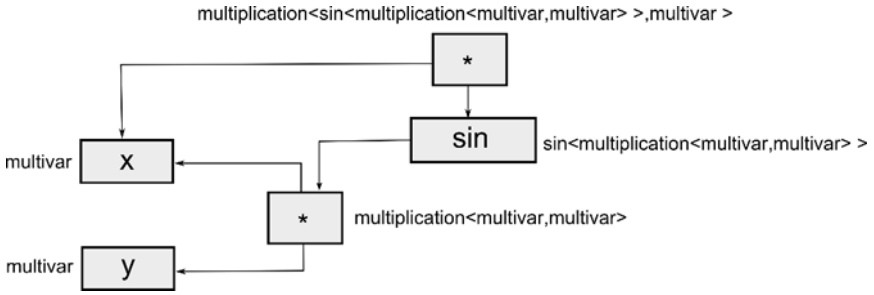

Рис. 12. Схематическое изображение построения дерева классов для арифметической операции «sin(x*y)*x»

Данный механизм является достаточно гибким и позволяет реализовать даже такую арифметическую операцию, как обращение к интерполированным табличным данным.

Рассмотрим арифметическую операцию z = x*y, где переменная «x» содержит частные производные δa+δb+δc, а переменная «y» содержит частные производные δb+δc+δd. Результатом функции «GetJacobian()» для данного выражения является разреженный вектор частных производных yδa+(x+y) δb+(x+y) δc+xδd (см. рис. 13). Для вычисления выражения в общем случае требуется операция над парой разреженных векторов: умножить данный вектор на коэффициент и прибавить к исходному вектору.

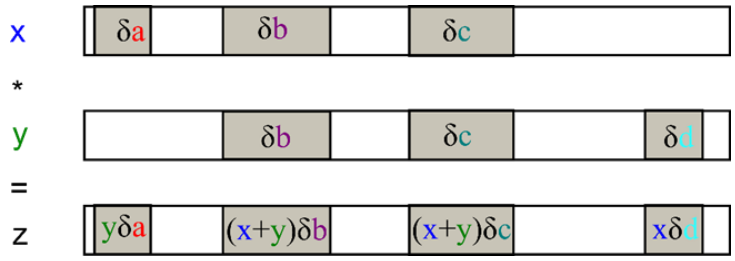

Рис. 13. Линейная комбинация разреженных векторов

Эта операция также называется линейной комбинацией разреженных векторов. Аналогичная операция требуется при вычислении предобуславливателей для итеративных методов решения систем линейных уравнений, основанных на неполном LU-разложении с отбрасыванием малых элементов. Эффективной структурой данных для выполнений этой операции в задаче неполного LU-разложения является плотный связный список. В данном приложении используется неупорядоченный плотный связный список, вставка новых элементов в который имеет сложность $O(1)$ (см. рис. 14). Список состоит из двух массивов: массива значений «A» и массива следующего ненулевого элемента «J». По умолчанию массив «J» заполнен значением «EOR», которое соответствует максимальному численному значению. «EOR» обозначает отсутствие ненулевого элемента. На нулевой позиции массива «J» стоит позиция первого ненулевого элемента. При вставке нового ненулевого элемента в список в позицию *i*:

- значение «EOR» в *i*-ой позиции заменяется на значение нулевой позиции массива «J»;
- в список «A» записывается значение элемента;
- а в нулевую позицию списка «J» записывается значение *i*.

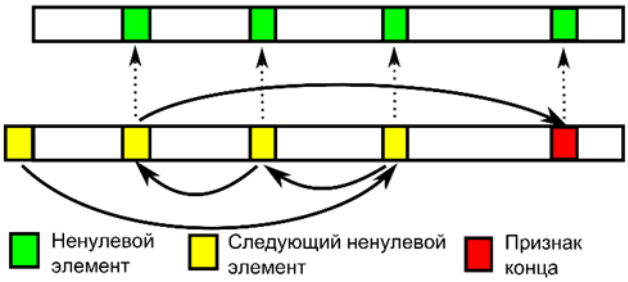

Рис. 14. Структура типа неупорядоченный плотный связный список

Последующее добавление значений в позицию *i* заключается в модификации массива «A». В программной платформе плотный связный список реализован классом «Sparse::RowMerger».

В итоге, в операции z = x*y функция «GetJacobian()» вызывается автоматически для дерева классов, соответствующего x*y и результат выполнения записывается в «z».

4.2 Вспомогательные структуры

В программной платформе реализован класс «Residual», который содержит вектор, соответствующий невязке, а также матрицу якобиана. Работая с классом «Residual» пользователь платформы напрямую модифицирует структуры типа «Sparse::Matrix» и «Sparse::Vector», которые будут непосредственно переданы в метод, реализующий решение систем линейных уравнений.

Для контроля количества основных неизвестных модели, их активации и перенумерации, реализован класс «Automatizator». Данный класс также регламентирует размер плотного связного списка, используемого для ускорения операции линейной комбинации разреженных векторов. Пользователь передает в класс «Automatizator» ярлык Tag, соответствующий основным неизвестным на элементах сетки, через функцию «RegisterDynamicTag». При вызове функции «EnumerateDynamicTags» класс нумерует все основные неизвестные и выделяет

память под дополнительные структуры данных. В данной реализации класса «Automatizator» предполагается, что все неизвестные модели хранятся на одной расчетной сетке, в будущем функционал может быть расширен на множество сеток.

Для запоминания промежуточных результатов вычислений реализована возможность ассоциировать с элементами сетки данные, представляющие собой значения с разреженным вектором частных производных. Хранимые данные могут напрямую использоваться в дальнейших арифметических операциях. Поддерживается так же хранение подобных данных в файлах форматов «.pmf» и «.xml», а также обмен такими данными между процессорами в параллельном режиме.

С помощью механизма шаблонов классов реализована также абстракция хранимых арифметических операций. Данная абстракция также представляет собой дерево классов, которое при подстановке элемента превращается в описанное ранее дерево классов, соответствующее хранимой арифметической операции. Такой механизм позволяет реализовывать функции на основе автоматического дифференцирования, арифметическое выражение аргумента которых заранее не известно и не может быть задано при помощи шаблонной реализации функции. Например, в одних случая требуется вычислить градиент давления, в других случаях требуется также учесть градиент капиллярного давления. В обоих случаях градиент вычисляется одинаковым образом, но аргумент заранее не известен.

Данный механизм включает описание условных операций и операций, описывающих шаблон разностного оператора на смежных элементах. За счет этого механизм позволяет полностью отделить описание модели от вычислений. Такое свойство может быть полезным в будущем для переноса вычислений невязки и построения матрицы якобиана на ускорители вычислений (например, GPU), без необходимости пользователю программировать на особом языке ускорителя. Анализируя хранимое выражение, главный процессор может не только построить код для ускорителя, выполняющий данное выражение, но и подготовить данные для выполнения на этом ускорителе.

5. Заключение

Продемонстрирована методика построения распределенных динамических сеток на основе использования технологии «восьмидеревьев» в рамках структур, используемых в программной платформе INMOST. Приведены результаты построения сеток и процесса их балансировки.

На ряде матриц из коллекции Университета Флориды была исследована параллельная эффективность различных линейных решателей, как встроенных в программную платформу INMOST, так и подключаемых из внешних пакетов PETSc и Trilinos. Была показана достаточно надежная и эффективная работа встроенных линейных решателей.

Также была представлена технология работы модуля автоматического дифференцирования программной платформы INMOST, который обеспечивает большую наглядность и надежность выполнения дискретизации исходной задачи.

Литература

1. Василевский Ю.В., Коньшин И.Н., Копытов Г.В., Терехов К.М., INMOST – программная платформа и графическая среда для разработки параллельных численных моделей на сетках общего вида, Москва: Изд-во Московского университета, 2013, 144 с.
2. INMOST – a toolkit for distributed mathematical modeling. URL: http://www.inmost.org.
3. Danilov A., Terekhov K., Konshin I., Vassilevski Y. The structure of INMOST program platform and its usage for numerical modeling problems // Proceedings of Russian Supercomputing Days 2015, P. 104–109.
4. Konshin I., Kaporin I., Nikitin K., Vassilevski Y. Parallel linear systems solution for multiphase flow problems in the INMOST framework // Proceedings of Russian Supercomputing Days 2015, P. 96–103.

5. ParMETIS: Parallel Graph Partitioning and Fill-reducing Matrix Ordering. URL: http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview (дата обращения: 15.06.2016).
6. Zoltan: Parallel Partitioning, Load Balancing and Data-Management Services. URL: http://www.cs.sandia.gov/zoltan (дата обращения: 15.06.2016).
7. PETSc (Portable, Extensible Toolkit for Scientific Computation). URL: https://www.mcs.anl.gov/petsc (дата обращения: 15.06.2016).
8. The Trilinos Project. URL: https://trilinos.org (дата обращения: 15.06.2016).
9. Terekhov K., Vassilevski Yu. Two-phase water flooding simulations on dynamic adaptive octree grids with two-point nonlinear fluxes // Russ. J. Numer. Anal. Math. Modelling (2013) V. 28, No. 3, P. 267–288.
10. The University of Florida Sparse Matrix Collection. URL: https://www.cise.ufl.edu/research/sparse/matrices (дата обращения: 15.06.2016).
11. Кластер ИВМ РАН. URL: http://cluster2.inm.ras.ru (дата обращения: 15.06.2016).

INMOST software platform development: dynamic grids, linear solvers, and automatic differentiation

D.V. Bagaev[1], A.I. Burachkovskiy[1], A.A. Danilov[2], I.N. Konshin[2], K.M. Terekhov[2,3]

Moscow State University, Moscow, Russia[1], Institute of Numerical Mathematics of Russian Academy of Sciences, Moscow, Russia[2], Stanford University, Stanford, USA[3]

Software platform INMOST is designed to operate with distributed grids of general form, perform a discretization of the problem, and solve the linear equation systems. A brief overview of the software platform and discussion of the latest advances in its development are given. We developed and implemented a distributed data structure that allows a user to dynamically build a refined and rarefied octree based grids. For set of test matrices from Florida University sparse matrix collection we estimate the performance of inner linear solvers of INMOST platform, as well as external linear solvers from PETSc and Trilinos packages. In addition we consider the developed automatic differentiation mechanism for assembling the coefficient matrix and right-hand side of the linear system. All designs are accessible at the INMOST software platform site: www.inmost.org.

Keywords: software platform, distributed dynamic grids, system of linear equations, automatic differentiation.

References

1. Vassilevski Yu., Konshin I., Kopytov G., Terekhov K. INMOST – a software platform and graphical environment for development of parallel numerical models on general meshes. Moscow State Univ. Publ., Moscow, 2013, 144 p. (in Russian).
2. INMOST – a toolkit for distributed mathematical modeling. URL: http://www.inmost.org.
3. Danilov A., Terekhov K., Konshin I., Vassilevski Y. The structure of INMOST program platform and its usage for numerical modeling problems // Proceedings of Russian Supercomputing Days 2015, P. 104–109.
4. Konshin I., Kaporin I., Nikitin K., Vassilevski Y. Parallel linear systems solution for multiphase flow problems in the INMOST framework // Proceedings of Russian Supercomputing Days 2015, P. 96–103.
5. ParMETIS: Parallel Graph Partitioning and Fill-reducing Matrix Ordering. URL: http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview (дата обращения: 15.06.2016).
6. Zoltan: Parallel Partitioning, Load Balancing and Data-Management Services. URL: http://www.cs.sandia.gov/zoltan (дата обращения: 15.06.2016).
7. PETSc (Portable, Extensible Toolkit for Scientific Computation). URL: https://www.mcs.anl.gov/petsc (accessed: 15.06.2016).
8. The Trilinos Project. URL: https://trilinos.org (accessed: 15.06.2016).
9. Terekhov K., Vassilevski Yu. Two-phase water flooding simulations on dynamic adaptive octree grids with two-point nonlinear fluxes // Russ. J. Numer. Anal. Math. Modelling (2013) V. 28, No. 3, P. 267–288.
10. The University of Florida Sparse Matrix Collection. URL: https://www.cise.ufl.edu/research/sparse/matrices (accessed: 15.06.2016).
11. INM RAS cluster URL: http://cluster2.inm.ras.ru (accessed: 15.06.2016) (in Russian).