УДК 519.6

ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ МЕТОД ДЕКОМПОЗИЦИИ ОБЛАСТИ

С. П. Копысов[1], **И. В. Красноперов**[1], **В. Н. Рычков**[1]

Разрабатывается объектно-ориентированная модель для метода декомпозиции области. На основе наследования, полиморфизма и инкапсуляции данная модель позволяет проводить эксперименты с новыми алгоритмами декомпозиции: использовать различные схемы разделения области, способы хранения систем уравнений и методы их решения, варианты задания граничных условий.

1. Введение. В последнее время интенсивно развиваются методы расчета сложных инженерных объектов, которые называют методами декомпозиции, композиции, разделения области и т.д. Главная идея их состоит в следующем: основная задача, подлежащая решению, достаточно сложная, но ее можно разбить на подобласти и для каждой из них решить подзадачу соответствующим методом. Это значительно повышает скорость и эффективность расчета больших по размеру сложных систем.

Период наиболее интенсивного развития метода декомпозиции области (МДО) и его широкого распространения в современной вычислительной практике начался сравнительно недавно. Этот метод используется при численном решении уравнений в частных производных и его развитие является одним из фундаментальных направлений современной вычислительной математики.

Идея декомпозиции использовалась и ранее, но не в связи с параллельными алгоритмами, и привела к методам подструктур, подконструкций [1], макроэлементов, суперэлементов [2], фрагментов, модульэлементов, редуцированных элементов [3], а также к методам Шварца [4], матриц емкости [5] и др. Такие методы всегда использовались для того, чтобы сводить решение исходной задачи в области со сложной границей к последовательности задач в подобластях, граница которых достаточно проста.

Важным обстоятельством при решении задач с помощью МДО является то, что необходимо рассматривать совместно: прикладную задачу и математическую модель, вычислительные алгоритмы, объектно-ориентированные технологии проектирования и программирования, архитектуру компьютерной платформы. В связи с этим метод еще недостаточно развит для ряда важнейших классов задач.

Основная цель данной работы — показать пути максимального сокращения трудоемкости программной реализации новых методов декомпозиции с одновременным глубоким распараллеливанием вычислений, удобным для реализации на современных ЭВМ, и рассмотреть новый подход к созданию программной системы для МДО. Отличительными чертами этого подхода являются: прозрачность и возможность доступа ко всем деталям реализации конкретного метода или алгоритма, а также математическая ясность описания нового метода для разработчика и пользователя; открытость, то есть возможность быстрого дополнения системы новыми алгоритмами МДО; простота использования разработанных методов для практических расчетов.

Содержание настоящей работы излагается по следующей схеме. В п. 2 на примерах известных алгоритмов и вводимых новых методов декомпозиции области рассматриваются общие этапы методов декомпозиции. В п. 3 строится объектно-ориентированная вычислительная модель МДО, которую можно было бы реализовать на том или ином языке программирования, с помощью той или иной вычислительной технологии. В п. 4 показываются возможности адаптации вычислительных алгоритмов МДО к параллельной вычислительный системе с произвольной организацией памяти и связями между отдельными вычислительными узлами.

2. Методы декомпозиции области. В настоящее время в методах декомпозиции можно выделить два основных класса: методы подструктур и методы Шварца (итерационные методы подструктур). В методе подструктур решение строится на нескольких непересекающихся подобластях, а в методах Шварца — как на пересекающихся, так и непересекающихся подобластях. Методы декомпозиции допускают постоянное расширение и обобщение.

[1] Институт прикладной механики Уральского отделения РАН, ул. Горького, 222, 426000, г. Ижевск; e-mail: kopyssov@udman.ru

Построение алгоритмов декомпозиции будем осуществлять для задач, аппроксимированных методом конечных элементов. Построение алгоритмов декомпозиции в терминах исходных "непрерывных задач" рассматривалось в работе В. И. Агошкова и В. И. Лебедева [6].

Остановимся на рассмотрении эллиптических задач второго порядка. Изложение методов будет вестись в применении к некоторым задачам теории упругости. Представим несколько формулировок методов декомпозиции для представления их общих и специфических операций.

2.1. Методы подструктур. В основе метода подструктур [1] лежит идея независимого расчета отдельных подобластей и последующего учета их взаимодействия. Предполагается разделить все узлы области на два множества — внешних и внутренних узлов, а неизвестные перемещения области рассматривать в виде суперпозиции двух составляющих. Первая составляющая — перемещения, вызванные внешними силами при закреплении границ в подобластях. Перемещения каждой подобласти определяются из уравнений, включающих неизвестные, связанные только с данной подобластью. Вторая составляющая — перемещения, вызванные перемещениями границ подобласти с исключенными внутренними узлами.

Для решения задач теории упругости метод подструктур впервые был использован А.Н. Пржеминицким [1], а метод Шварца — С.Л. Соболевым [7].

Применяя этот алгоритм, разобьем область на непересекающиеся подобласти. Пусть область Ω разбита на p подобластей Ω_i с границей разделяющей подобласти Γ_B $(i = 1, p)$:

$$\Omega = \Omega_1 \bigcup \Omega_2 \bigcup \ldots \bigcup \Omega_p, \quad \Omega_i \bigcap \Omega_j = \emptyset, \quad \Gamma_B = \bigcup_{i=1}^{p} \partial\Omega_i \setminus \partial\Omega. \tag{1}$$

Для каждой подобласти Ω_i строятся системы уравнений, причем степени свободы, связанные с внутренними и внешними (граничными) узлами, разделяются. Введем обозначения: индекс I относится к внутренним точкам подобластей, а B — к границе разделяющей подобласти. Полученные системы решаются независимо в предположении, что на границе Γ_{B_i} заданы граничные условия Дирихле $u|_{\Gamma_{B_i}} = 0$ с соседними подобластями. Затем из условия равновесия определяются искомые перемещения граничных узлов u_B при одновременном их освобождении. Представим матрицу жесткости $A \in \mathbb{R}^{N\times N}$ системы уравнений в виде

$$A = \sum_{i=1}^{p} C_i^T A^i C_i, \tag{2}$$

где A^i — матрица жесткости подобласти Ω_i; $C_i \in \mathbb{R}^{M\times N}$ — матрица, отображающая вектор степеней свободы u в вектор u^i соответствующей подструктуры: $u^i = C_i u$. Группируя вектор неизвестных u^i (т.е. разделяя его компоненты на граничные степени свободы u^i_B, соответствующие Γ_{B_i}, и внутренние степени свободы, обозначаемые u^i_I), получим

$$\begin{pmatrix} A^1_{II} & & & A^1_{IB} \\ & A^2_{II} & & A^2_{IB} \\ & & \ddots & \vdots \\ & & A^p_{II} & A^p_{IB} \\ A^1_{BI} & A^2_{BI} & \ldots A^p_{BI} & A_{BB} \end{pmatrix} \begin{pmatrix} u^1_I \\ u^2_I \\ \vdots \\ u^p_I \\ u_B \end{pmatrix} = \begin{pmatrix} f^1_I \\ f^2_I \\ \vdots \\ f^p_I \\ f_B \end{pmatrix}.$$

Тогда условия равновесия i-ой подобласти записываются в виде

$$\begin{pmatrix} A^i_{II} & A^i_{IB} \\ A^i_{BI} & A^i_{BB} \end{pmatrix} \begin{pmatrix} u^i_I \\ u^i_B \end{pmatrix} = \begin{pmatrix} f^i_I \\ f^i_B \end{pmatrix},$$

где индексы I и B относятся к внутренним и граничным степеням свободы.

Перемещения подобласти Ω_i представим суперпозицией двух векторов (далее индекс i опущен)

$$u = u^0 + u^{\sim},$$

где

$$u = \begin{pmatrix} u_I \\ u_B \end{pmatrix} = \begin{pmatrix} u^0_I \\ u^0_B \end{pmatrix} + \begin{pmatrix} u^{\sim}_I \\ u^{\sim}_B \end{pmatrix}.$$

Аналогично для вектора нагрузок f можно записать

$$f = \begin{pmatrix} f_I \\ f_B \end{pmatrix} = \begin{pmatrix} f^0_I \\ f^0_B \end{pmatrix} + \begin{pmatrix} f^{\sim}_I \\ f^{\sim}_B \end{pmatrix}.$$

Здесь $u_B^0 = 0$ — перемещения граничных узлов при закреплении границы; $u_B\big|_{\Gamma_{B_i}} = 0$; u_I^0 — перемещения внутренних узлов при закреплении границы; $u_B^{\sim}$ — перемещения граничных узлов; $u_I^{\sim}$ — перемещения внутренних узлов, обусловленные $u_B^{\sim}$; f_I^0 — заданные внешние силы, приложенные к внутренним узлам; f_B^0 — узловые реакции, возникающие при закреплении границы; $f_I^{\sim} = 0$; $f_B^{\sim}$ — вектор нагрузки, включающий внешние силы и реакции взаимодействия с соседними подобластями при раскреплении границ. Тогда система уравнений для подобласти примет вид

$$\begin{pmatrix} A_{II} & A_{IB} \\ A_{BI} & A_{BB} \end{pmatrix} \begin{pmatrix} u_I^0 & u_I^{\sim} \\ & u_B^{\sim} \end{pmatrix} = \begin{pmatrix} f_I^0 & \\ f_B^0 & f_B^{\sim} \end{pmatrix}.$$

Таким образом, алгоритм метода подструктур может быть представлен следующим образом.

Шаг 1. Пусть на границе подобласти заданы перемещения $u_B^0 = u_B^{\sim} = u_I^{\sim} = 0$ и $f_I = f_I^0$ в Ω_i; $f_I^{\sim} = f_B^{\sim} = 0$. Найдем u_I^0, решив независимо на каждом процессоре систему вида

$$A_{II} u_I^0 = f_I. \tag{3}$$

Шаг 2. Для каждой подобласти вычислим

$$f_B^0 = A_{IB} A_{II}^{-1} f_I \tag{4}$$

(где f_B^0 — матрица граничных реакций) при заданных силах f_I, необходимых для выполнения условия $u_B|_{\Gamma_{B_i}} = 0$.

Шаг 3. При $u_B|_{\Gamma_{B_i}} \neq 0$, когда граница раскреплена, перемещения $u^{\sim}$ определяются как

$$S_{BB} u_B^{\sim} = f_B^{\sim}, \tag{5}$$

где

$$S_{BB} = A_{BB} - A_{BI} A_{II}^{-1} A_{IB}. \tag{6}$$

Здесь S_{BB} — матрица граничных жесткостей или дополнение Шура для подобласти. Вектор $f_B^{\sim} = f_B - f_B^0$ определяется как

$$f_B^{\sim} = f_B - A_{BI} A_{II}^{-1} f_I. \tag{7}$$

Шаг 4. Перемещения $u_I^{\sim}$ вычислим независимо в каждой подобласти ($u_I^0 = 0$):

$$u_I^{\sim} = -A_{II} A_{IB} u_B^{\sim}. \tag{8}$$

Шаг 5. Определим перемещения внутренних узлов:

$$u_I = u_I^0 + u_I^{\sim}. \tag{9}$$

Матрица S_{BB} положительно определена и симметрична, а число ее обусловленности имеет порядок $k(S_{BB}) \approx O(h^{-1})$. Отметим, что порядок матрицы S_{BB} значительно меньше порядка исходной матрицы A, но может быть все-таки большим для ее обращения за приемлемое время. Поэтому необходимы более эффективные подходы построения вычислений.

2.2. Методы Шварца (итерационные методы подструктур). Как известно, метод Шварца позволяет получать решение для сложной области, являющейся суммой двух простых замкнутых областей $\Omega = \Omega_1 + \Omega_2$, последовательным приближением, если для каждой из подобластей решение известно [5]. Сходимость алгоритма Шварца в задаче трехмерной теории упругости в перемещениях доказана в [7] при задании граничных перемещений, а в случае задания напряжений — в [8]. Отметим, что условия сходимости определяются не только различными вариантами граничных условий, но и размером самих подобластей. Данный метод дает возможность конструировать множество разнообразных алгоритмов путем наложения границ различными способами и сочетанием вариантов граничных условий, встречающихся в практике.

В последние годы появились алгоритмы с итерациями по непересекающимся подобластям (произвольного количества) и использованием чередующихся типов граничных условий во вспомогательных краевых задачах (см., например, [9]).

Блочные методы Гаусса–Зейделя и Якоби являются алгебраическими обобщениями итерационных алгоритмов декомпозиции Шварца. Блочный метод Якоби идентичен аддитивному методу Шварца, а метод Гаусса–Зейделя — мультипликативному методу Шварца.

Приведем параллельный алгоритм решения системы $Au = f$ мультипликативным методом Шварца, полагая, что введено упорядочивание по (1). Тогда [10]

$$A = \sum_{i=1}^{p} C_i^T A^i C_i, \quad \begin{pmatrix} A_{II}^i & A_{IB}^i \\ A_{BI}^i & A_{BB}^i \end{pmatrix} \begin{pmatrix} u_I^i \\ u_B^i \end{pmatrix} = \begin{pmatrix} f_I^i \\ f_B^i \end{pmatrix}.$$

Пусть $A = U + L + D$, где D — диагональная, L — нижняя и U — верхняя треугольные матрицы с нулевыми диагоналями. Тогда можно записать

$$Du^{k+1} = f - Lu^{k+1} - U\,u^k$$

и

$$\sum_{i=1}^{p} C_i^T D_i C_i u^{k+1} = \sum_{i=1}^{p} C_i^T (f_i - L_i C_i u^{k+1} - U_i C_i u^k).$$

Для внутренних узлов каждой подобласти Ω_i будем вычислять

$$D_{II}^i u_I^{i,k+1} = f_I^i - L_{II}^i u_I^{i,k+1} - U_{II}^i u_I^{i,k} + U_{IB}^i u_B^{i,k}.$$

Система уравнений для граничных узлов имеет вид

$$\sum_{i=1}^{p} C_B^{i,T} D_{BB}^i u_B^{i,k+1} = \sum_{i=1}^{p} C_B^{i,T} (f_B^i - L_{BI}^i u_I^{i,k+1} + L_{BB}^i u_B^{i,k+1} - U_{BB}^i u_B^{i,k}).$$

Следует отметить, что возможности метода Шварца в области декомпозиции больших дискретных систем далеко не исчерпаны. Данный подход дает возможность строить множество разнообразных алгоритмов.

2.3. Другие варианты методов декомпозиции. При решении многих задач МКЭ (метода конечных элементов) приходится иметь дело с системами с симметричными положительно определенными ленточными матрицами, которые могут быть плохо обусловленными. Такие матрицы насчитывают несколько десятков тысяч элементов и более, поэтому перед решением больших задач необходимо проводить преобразования систем, улучшающие их параллельные свойства и уменьшающие общий объем вычислений, и строить параллельный процесс вычислений для преобразованных систем.

2.3.1. Поэлементная декомпозиция. Рассмотрим конечно-элементные системы уравнений, в которых глобальные матрицы жесткости не формируются в явном виде, а вводятся тем или иным способом.

Одно из главных достоинств итерационных методов в сочетании с МКЭ заключается в том, что допускается построение вычислений, при которых опускается операция формирования глобальной матрицы жесткости системы уравнений. Это имеет особое значение при рассмотрении пространственных конечных элементов высоких порядков с большим числом степеней свободы на адаптивно перестраивающихся сетках. При адаптивном перестроении сетки в *h-версии* МКЭ на каждом шаге уточнения необходимо частично переформировать коэффициенты матрицы жесткости системы. В этом случае применение метода подструктур, которому свойственно выделение крупноблочного параллелизма, становится нецелесообразным. Поэтому рассматриваемый ниже подход представим как неявный метод декомпозиции, для которого распараллеливание отражает среднезернистость вычислений (уровень отдельных конечных элементов).

В основе поэлементной декомпозиции лежит преобразование связности, базирующееся на топологических операциях объединения и разъединения [10].

Пусть область Ω разбита на p подобластей так же, как в (1). Обычно в МКЭ считается, что конечно-элементная сетка собрана и каждый узел принадлежит нескольким элементам. Предположим обратное: множество элементов разобрано, так что любой узел принадлежит одному элементу и нумерация степеней свободы в элементах выполнена от 1 до n. Пусть подобласть Ω_i содержит по N_i узлов и M_i элементов, причем $M = \sum_{i=1}^{p} M_i$, $N < \sum_i^p N_i$. В результате получим блочно-диагональную матрицу A, которую удобно хранить по процессорам, вводя матрицу связности

$$A = \sum_{e=1}^{M} C_e^T \tilde{A}^e C_e, \tag{10}$$

где $A \in \mathbb{R}^{N\times N}$ — глобальная матрица жесткости, $\tilde{A}^e \in \mathbb{R}^{n\times n}$ — локальная элементная матрица жесткости, $C_e \in \mathbb{R}^{n\times N}$ — булева матрица (матрица связности), выражающая соотношение между независимыми

степенями свободы индивидуальных элементов и независимыми перемещениями для всей области в целом, для которой справедливы соотношения

$$D = \sum_{e=1}^{M} C_e^T C_e, \qquad I = \sum_{e=1}^{M} C_e^T I^e C_e \qquad C_e^{-1} = C_e^T.$$

Здесь D — диагональная матрица, содержащая для каждого узла число элементов, которым принадлежит данный узел, и I — единичная матрица.

Тогда (10) можно записать в виде

$$A = (C_1^T C_2^T \dots C_M^T) \begin{pmatrix} \tilde{A}^1 & 0 & \dots & 0 \\ 0 & \tilde{A}^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \tilde{A}^M \end{pmatrix} \begin{pmatrix} C_1 \\ C_2 \\ \vdots \\ C_M \end{pmatrix}. \tag{11}$$

Таким образом, матрицы $\tilde{A}^e$ независимо хранятся на процессорах. В случае перестроения системы уравнений, связанном с адаптацией сетки или локальным изменением степени аппроксимирующих полиномов, корректируются только матрицы перестраиваемых элементов и списки глобальной связанности. Соотношения, подобные (10), мы уже вводили в методе подструктур (соотношения (2)), что соответствовало более крупному разделению на подобласти. Введем следующее упорядочивание матрицы жесткости системы:

$$A = \sum_{i=1}^{p} C_i^T A^i C_i. \tag{12}$$

В свою очередь,

$$A^i = \sum_{e=1}^{M_i} C_e^T \tilde{A}^e C_e. \tag{13}$$

На каждом процессоре будем хранить M_i несовместных элементных матриц жесткости $\tilde{A}^e$ и векторов $\tilde{f}^e$.

Скалярные произведения также вычисляется поэлементно. Представим вектор правой части $f \in \mathbb{R}^N$ в виде

$$f = \sum_{e=1}^{M} C_e^T \tilde{f}^e, \tag{14}$$

где $\tilde{f}^e \in \mathbb{R}^n$. Таким образом, для получения глобального вектора необходима операция сборки $f = \sum_{i=1}^{p} C_i^T \tilde{f}_i$, которая сопряжена с обменами между процессорами.

В дальнейшем потребуется другой тип векторов, который введем следующим образом:

$$\overline{f} = (C_1 f C_2 f \dots C_M f)^T, \tag{15}$$

где $\overline{f} \in \mathbb{R}^M$. Вектора $\overline{f}_i$ такого типа могут храниться по процессорам как частями, так и полностью, но главное, что величины имеют глобальные значения.

На каждом процессоре P_i вектора могут храниться в виде векторов различных типов (14), (15); соответственно скалярные произведения векторов можно вычислить несколькими способами:

$$(r, r) = r^T r = \overline{r}^T C Q C^T \overline{r} = \tilde{r}^T C^T \tilde{r}, \tag{16}$$

где $Q = 1/C^T C$.

Наиболее трудоемкой операцией в вычислительных затратах метода сопряженных градиентов является многократное вычисление произведения глобальной матрицы $A \in \mathbb{R}^{N \times N}$ на глобальный вектор $p \in \mathbb{R}^N$:

$$q = Ap. \tag{17}$$

Подставим (10) в (17) и получим

$$q = \sum_{e=1}^{M} C_e^T \tilde{A}^e C_e p. \tag{18}$$

При перемножении матрицы на вектор (18) можно выделить два этапа вычисления: произведение элементных матриц $\tilde{A}^e$ на элементный несвязанный вектор $C_e p$, которое может выполняется параллельно: $v_e = \tilde{A}^e C_e p$; преобразование $C_e^m v_e$ результирующего элементного вектора размерности $n \times M$ к глобальному вектору размерности N. Запишем поэлементное перемножение с учетом (15):

$$q = Ap = C^T \tilde{A} C p = C^T \tilde{A} \overline{p} = C^T \tilde{q}, \tag{19}$$

где $\tilde{q} = \tilde{A}\overline{p}$.

Реализация алгоритма метода сопряженных градиентов без непосредственной сборки глобальной матрицы системы уравнений позволяет эффективно реализовать данный метод на параллельной вычислительной системе с распределенной памятью [11].

2.3.2. Многофронтальный метод как метод декомпозиции. Фронтальный метод впервые введен в рассмотрение в [12] и фактически является модификацией метода Гаусса решения систем. В этом методе факторизация матрицы проводится как последовательность факторизаций и исключений плотных подматриц, называемых фронтальными. Метод часто использовался для решения систем в МКЭ, но применяется и для решения систем общего вида.

Метод Айронса может быть рассмотрен как для полностью сформированных матриц, так и для систем, формирование которых еще не закончено (когда в *глобальной* матрице A учтены (просуммированы) еще не все матрицы конечных элементов). Изначально этот метод использовался как раз для систем второго типа с целью экономии памяти. Рассмотрим подробнее этот метод для таких систем.

Исключения переменных начинаются прежде, чем закончено суммирование в (13). Фактически на каждом шаге происходит исключение k-го неизвестного из каждого уравнения системы. Рассмотрим соотношения для матриц жесткости и вектора правой части при исключении k-го неизвестного:

$$A_{ij} = A_{ij} - \frac{A_{ik} A_{ki}}{A_{kk}}, \qquad f_i = f_i - \frac{A_{ik} f_i}{A_{kk}},$$

где i и j — строка и столбец ($i \neq k, j \neq k$). Так как матрица A ленточная, то изменяются лишь несколько строк матрицы жесткости. Поэтому исключение можно проводить уже на стадии формирования глобальной матрицы жесткости, когда добавлена локальная матрица жесткости последнего конечного элемента, включающая k-ю степень свободы. Неизвестные исключаются в порядке, определяемом нумерацией элементов. В рамках программной реализации это означает, что необходимо хранить лишь часть матрицы A_{ij}, которая изменяется на каждом шаге разложения. Эта часть также является симметричной и положительно определенной матрицей, порядок которой решающим образом влияет на быстродействие и объем памяти. Если при минимизации ширины ленты определенным образом упорядочиваются узлы сетки, то при минимизации ширины фронта упорядочивается нумерация конечных элементов.

Если на каждом этапе область, покрытая множеством собранных конечных элементов, — односвязная, то метод определяется как однофронтальный. Если же множество собранных конечных элементов покрывает несколько отдельных областей с независимыми границами, метод называется многофронтальным.

Преимущества многофронтального метода заключаются в следующем: меньшие затраты времени и памяти, когда матрица коэффициентов имеет ленточный характер; произвольность способа нумерации узлов (или переменных), имеет значение только порядок, в котором элементы объединяются в глобальную систему; можно производить локальное сгущение сетки элементов без перенумерации; есть возможность получения матриц жесткости подобластей; в отличие от других прямых методов, имеются большие перспективы для распараллеливания.

2.3.3. Метод декомпозиции, основанный на механическом взаимодействии подструктур. Данный метод основан на разъединении (декомпозиции) контактирующих тел и отдельном их рассмотрении [13]. Этот подход применим к решению задач на основе декомпозиции контактирующих тел; методы, базирующиеся на нем, назовем методами декомпозиции контактирующих тел. Для решения задачи выполняется итерационная процедура, основанная на независимом решении обычных задач в подобластях (линейных или нелинейных) с последовательным удовлетворением кинематических и силовых условий контактного взаимодействия (линейных или нелинейных). Задавая те или иные условия на контактных поверхностях можно получить метод декомпозиции решения краевых задач теории упругости с выделением вспомогательных границ при условии идеального контакта на них, т.е. в качестве кинематических условий на границе выставляются условия взаимного непроникновения, а в качестве силовых — отсутствие взаимодействия по касательной к границе. Вспомогательные границы определяются из условий эффективного параллельного решения. Итерационная процедура будет выполняться до тех пор, пока на границе взаимодействия не будут удовлетворены контактные кинематические и силовые условия.

Рассматривая различные задачи, решаемые с помощью приведенных выше методов декомпозиции области, можно заметить, что формулировки используют разные абстракции: матричные, графовые, физические и т.д. Не ограничивая общности можно считать, что ключевыми понятиями для реализации МДО являются: область; подобласть; границы раздела подобластей; глобальная и локальные матрицы жесткости; граничные условия на подобластях; методы решения систем линейных уравнений в подобластях и на общих границах; сетки; векторы; матрицы; тензоры; графы и др. Отметим также, что методы декомпозиции области на матричном уровне могут рассматриваться как специальные итерационные или прямые методы блочного типа.

Рассмотрим методы декомпозиции области с точки зрения программирования и построим такую вычислительную модель, которую можно было бы реализовать на том или ином языке программирования, с помощью той или иной вычислительной технологии. Для этого используем методологию объектно-ориентированного проектирования и программирования, которая позволяет строить абстрактные модели и описывает, каким образом они могут быть запрограммированы. Здесь и далее под моделью будем понимать программную объектно-ориентированную модель. Проектирование программного обеспечения подробно обсуждается в [14 – 16].

3. Объектно-ориентированная реализация метода декомпозиции. Метод декомпозиции области позволяет эффективно решать сложные задачи в различных отраслях науки. Его применение на практике сдерживается значительной трудоемкостью программной реализации. Хорошо известно, сколько времени и сил тратится на попытки создания расчетных программ, которые тем более далеки от совершенства, когда разработчики не являются профессиональными программистами. Большая часть существующего на данный момент программного обеспечения написана на процедурных языках программирования (Fortran, Си), поэтому эти пакеты достаточно сложны в использовании и практически не расширяемы. Отсюда возникает потребность в объектно-ориентированном программном обеспечении для метода декомпозиции, который позволит решить большую часть из вышеперечисленных проблем.

Объектно-ориентированный подход к программированию собственно МДО в отличие от традиционного позволяет разрабатывать все алгоритмы, функции и методы в соответствии с законами математических абстракций (математическими законами преобразования объектов).

Основные свойства МДО должны быть отражены в системе объектов и компонентов. Кроме описания классов, объектно-ориентированная модель включает в себя диаграмму связей сущностей, динамическую и функциональную модель системы. Первая показывает взаимоотношение между объектами, вторая — описание изменений, которые происходят с объектами и их связями во время работы описываемой системы. Функциональная модель описывает вычисления в системе. Все это в совокупности позволяет хорошо понять различные аспекты и свойства разрабатываемой программной системы, что в последующем существенно облегчает ее реализацию, тестирование, сопровождение, разработку новых версий и модификацию.

Большинство программных реализаций МДО построены на основе того или иного метода аппроксимации дифференциальной задачи, чаще всего на основе обычного МКЭ (см., например, [17]). Поэтому вся модель объектов обычно тесным образом связывается с моделью объектов этих методов. Наиболее приемлемым решением является определение системы объектов, общей для всех МДО, и ее последующее расширение путем добавления новых объектов, реализующих узко-специальные алгоритмы [18].

Несмотря на представительность библиографии по объектно-ориентированным конечно-элементным системам [19, 20], следует обратить внимание на отсутствие в настоящее время сколь-нибудь систематического изложения объектно-ориентированной модели МКЭ. Отметим чрезвычайную редкость использования объектно-ориентированного подхода для реализации МДО.

Diffpack [21] — это объектно-ориентированная среда для решения уравнений в частных производных. В основе этого коммерческого проекта лежит несколько фундаментальных абстракций, которые хорошо разделены в функциональности: структуры данных, линейные и нелинейные солверы, уравнения в частных производных, параллельные вычисления и вспомогательные утилиты. Первоначально Diffpack не включал методы декомпозиции области и многосеточные алгоритмы. Однако позднее к существующему пакету уравнений в частных производных был добавлен модуль, реализующий эти функциональные возможности. Как отмечено в [22], это вызвало многочисленные трудности включения нового класса.

Отметим также, что к объектно-ориентированному подходу при реализации элементов МДО обращаются еще в двух открытых системах: PETSc (Portable Extensible Toolkit for Scientific Computation) [23] и SPOOLES (Sparse Object-Oriented Linear Equations Solver) [24].

В методах декомпозиции выделим несколько подзадач, причем некоторые из них можно рассматривать как отдельные самостоятельные задачи. В соответствии с этим выделим в МДО несколько сущ-

ностей, или объектов. Разобьем все объекты на несколько групп, которые должны решать следующие задачи.

1. Разделить область на подобласти.

2. Поставить в соответствие узловым степеням свободы номера уравнений. Отображение может иметь значительное влияние на объем вычислений, требуемых для решения матричных систем уравнений, и на размер памяти, необходимой для хранения этих матриц.

3. Сформировать матричные системы уравнений, учитывая вклады от элементов или узлов. Вклады определяются применяемой схемой сборки системы.

4. Ввести граничные условия. Данная операция может повлечь за собой преобразование элементных и узловых вкладов или добавление в систему новых переменных.

5. Решить матричные системы уравнений.

6. Обновить глобальные узловые характеристики в соответствии с полученным решением.

Хорошо разработанная объектно-ориентированная программная система позволит изменять и проводить эксперименты с новыми алгоритмами решения просто, с приложением минимума усилий, использовать различные схемы упорядочивания, способы хранения систем уравнений и методы их решения, способы обработки граничных условий задачи и т.п. С учетом этого была спроектирована и реализована система объектов. Все объекты рассматриваемый модели можно разделить на четыре группы:

— классы расчетной модели: с их помощью создается расчетная модель задачи (область, узлы, граничные условия и т.п.);

— численные классы: выполняют все вычисления и хранение данных задачи (матрицы, векторы, системы уравнений, графы);

— классы решения задачи: отвечают за решение задачи (алгоритм решения, схемы сборки систем уравнений, наложения граничных условий, способы упорядочивания уравнений и т.п.);

— классы метода декомпозиции: необходимы при решении задачи МДО (подобласть, решатель систем подобласти и т.д.).

В рамках данного подхода был проведен теоретический объектно-ориентированный анализ основных задач МДО. Результатом проведенного анализа являются разработанные детальные классификации объектов, вычислительных алгоритмов и возникающих вычислительных задач метода декомпозиции области. Классификации представляют собой иерархию наследуемых классов и составляют конструктивную основу для расширения и программной реализации библиотеки МДО.

3.1. Классы расчетной модели. Объекты этого вида используются при создании расчетной модели задачи, т.е. они представляют собой узлы, элементы, граничные условия, нагрузки, области и т.п. Для больших задач важно, чтобы создание модели происходило простым и понятным способом. Основные классы расчетной модели задачи и обозначения отношений между классами представлены на диаграмме (рис. 1).

Класс "редактор расчетной модели" (`ModelBuilder`) — это базовый объект, который отвечает за построение расчетной модели задачи, т.е. создает узлы, элементы, граничные условия и т.п. Конструктор класса имеет только один параметр: ссылку на объект класса `Domain`. У данного класса существует единственный метод `BuildModel()`, при вызове которого и формируется построение модели. Создание модели может происходить по информации из файла, из графического редактора, CAD-системы и т.д. Соответственно, с помощью наследования от базового класса создаются различные варианты редакторов модели.

Класс "область" (`Domain`) является контейнером, содержащим в себе компоненты расчетной модели задачи, такие как узлы, граничные условия, подобласти. Начальные данные в область добавляются классом редактора расчетной модели. Каждый объект, хранящийся в области, наследуется от класса `DomainComponent`. Объект класса `Domain` связан с классом построения области `DomainBuilder` и классом решения задачи `AnalysisModel`. Интерфейс данного класса имеет следующие функции:

1. Методы по добавлению и удалению компонентов в область. При добавлении объекта в область выполняется проверка допустимости данной операции. Важно, чтобы проверка выполнялась во время построения области, а не во время решения задачи, иначе в итерационных алгоритмах МДО данная процедура будет повторяться на каждом шаге решения.

2. Методы по представлению компонента области или их коллекции.

3. Методы по обновлению состояния области. Например, при фиксации текущего состояния области вызывается метод `Commit()`, который вызывает аналогичные методы у всех компонентов области.

4. Методы, информирующие о текущем состоянии области (например, граф связности узлов, число компонентов и т.п.).

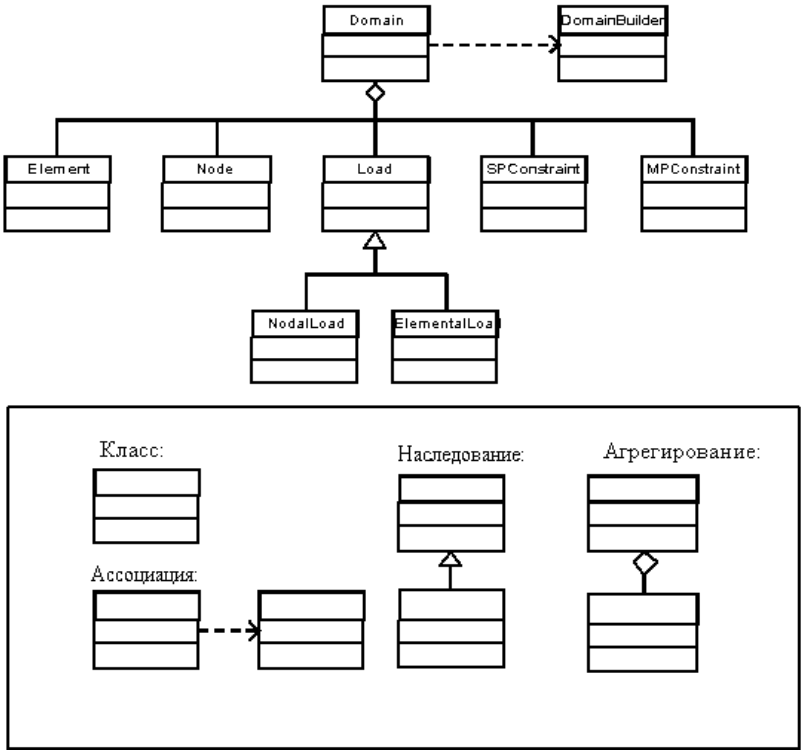

Рис. 1. Классы расчетной модели

5. Методы, сообщающие классу решения `Analysis` об изменении области и, следовательно, возможности модификации модели решения задачи.

Класс "узел" (`Node`). Объект класса `Node` представляет собой дискретную точку в области, в которой вычисляется искомое решение задачи. Объект данного класса хранит такие характеристики узла, как координаты в области, результаты решения задачи (перемещение, скорость, ускорение и т.п.) и информацию о нагрузках и ограничениях, а также функции по работе с этими характеристиками. Конструкторы объектов `Node` позволяют использовать их в решении одномерных, двумерных и трехмерных узлов, при этом количество степеней свободы в узле выбирается таким, каким это необходимо в задаче. Объекты класса `Node` используются в решении как статических, так и динамических задач. Это достигается введением в класс временных и постоянных значений характеристик узла. Временное значение — это значение, которое еще не было принято за искомое решения задачи. В ходе решения задачи это временное значение может быть либо отвергнуто, либо может стать постоянным (т.е. искомым решением задачи).

Класс "элемент" (`Element`) — это абстракция конечного элемента. Его основной функциональностью является построение элементных матриц и вектора сил с учетом существующих нагрузок в локальных координатах. От абстрактного класса `Element` путем наследования создаются конкретные типы элементов или конечно-разностные схемы. С каждым объектом класса `Element` связано несколько объектов `Node` и объектов класса `Material` (класс, предоставляющий описание физико-механических свойств). Такой подход позволяет изолировать функции элемента от функций алгоритма решения задачи и наследовать общие функции элемента в подклассах.

Классы граничных условий задачи (`SPConstraint` **и** `MPConstraint`). Граничные условия задачи могут быть заданы двух типов: одноточечные (граничные условия), где задается значение для одной степени свободы, и многоточечные — выражающие взаимосвязь нескольких степеней свободы.

В модель были введены два класса `SPConstraint` и `MPConstraint`, соответственно. Объекты этих классов информируют объекты в модели решения задачи о том, что ограничения существуют, и о том, какие они имеют значения.

С каждым объектом класса `SPConstraint` связан один объект класса `Node`. Класс `SPConstraint` имеет методы по получению номера степени свободы граничного узла, определения того, является ли

условие однородным.

Класс `MPConstraint` связан с двумя или более объектами `Node`. Функциональность класса заключается в предоставлении матрицы ограничений в определенный момент времени и номеров связанных степеней свободы.

3.2. Численные классы. Метод декомпозиции требует интенсивных вычислений и, следовательно, систему классов для них. Основу их составляют задачи вычислительной линейной алгебры. Далее будут кратко описаны некоторые из них.

Матрицы и векторы. Классы матриц и векторов в основном предназначены для хранения информации о системах уравнений и характеристиках узлов и элементов. Интерфейсы классов предоставляют полный набор арифметических операций, в основном в форме перегружаемых операторов. От базового класса матрицы наследуются специальные классы матриц (симметричные, ленточные и т.п.), которые по соображениям эффективности удобно использовать в некоторых случаях.

Класс решения систем линейных уравнений. Двумя базовыми классами этого типа являются класс линейной системы уравнений `LinearSOE` и класс решателя `LinearSolver`. Класс `LinearSOE` выполняет хранение данных системы уравнений. В соответствии с тем, каким методом решается система, класс агрегирует наиболее удобный тип матриц и векторов. Основным методом класса `LinearSolver` является метод `Solve()`. В нем реализуются основные шаги решения системы уравнений. Матрица системы может храниться различными способами, в соответствии с этим и реализованы подклассы `BandSPDLinearSOE` (ленточная положительно определенная матрица), `SparseSPDLinearSOE` (разреженная положительно определенная матрица).

3.3. Классы решения задачи. Для облегчения повторного использования программного кода, расширяемости и гибкости необходимо применить объектно-ориентированные принципы к разработке алгоритма решения задачи. Следуя этим принципам, сначала нужно определить основные процессы, выполняемые в ходе решения задачи методом декомпозиции, выделяя их в отдельные классы, и затем определить интерфейс этих классов. Важно, чтобы интерфейсы позволяли работать классам вместе и добавлять новые классы без изменения старых. Основные классы модели решения задачи представлены на рис. 2.

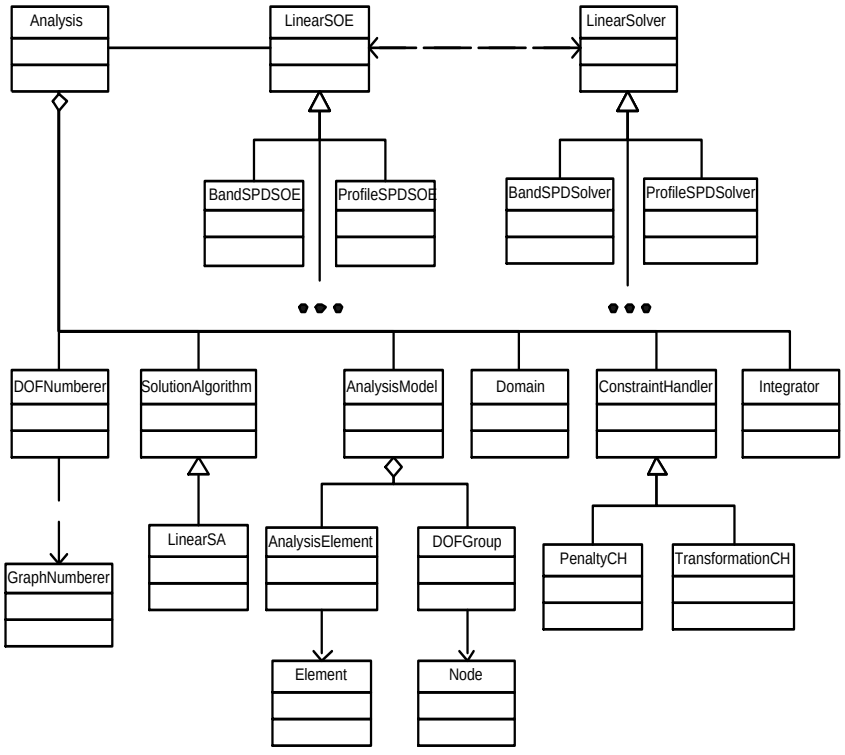

Рис. 2. Классы модели решения задачи

В представленной модели объект решения задачи — это объединение объектов следующих типов.

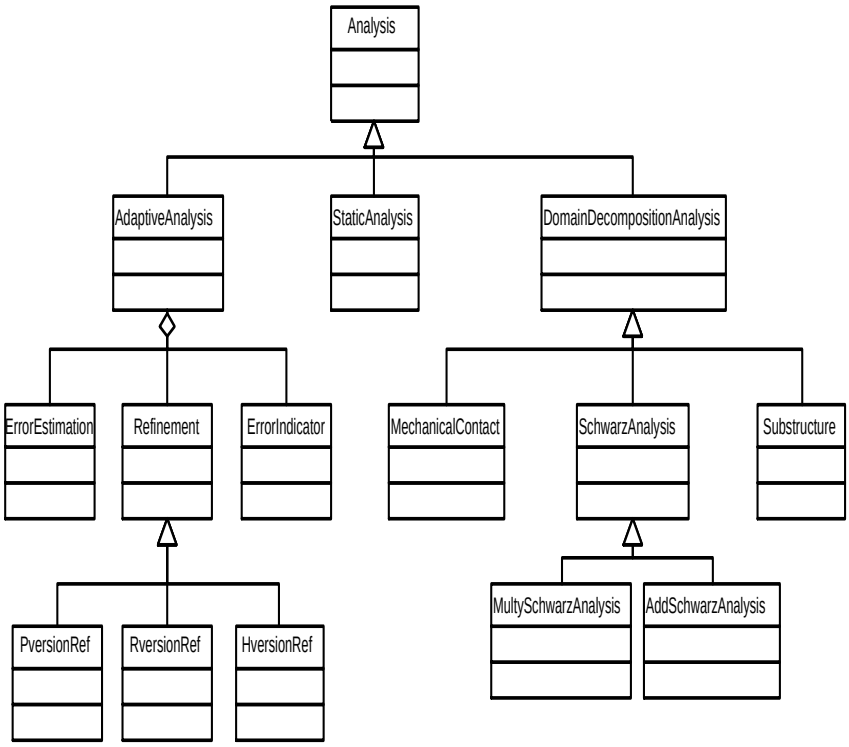

Рис. 3. Классы решения задачи `Analysis`

1. Алгоритм решения — объект, управляющий шагами решения задачи.

2. Модель решения — контейнер, содержащий объекты класса `DOFGroup`. Класс `DOFGroup` (группа степеней свободы) введен для того, чтобы устранить необходимость хранения в геометрическом классе `Node` отображение между степенями свободы и номерами уравнений. Потомки класса `DOFGroup` отвечают за обработку ограничений задачи, что избавляет другие объекты от этого.

3. Интегратор — объект, определяющий вклады классов `DOFGroup` и `SubDomain` в глобальную систему уравнений и обновляющий результаты решения в них в соответствии с решением системы уравнений.

4. Обработчик граничных условий — объект, обрабатывающий граничные условия с помощью создания соответствующих объектов класса `DOFGroup`.

5. Упорядочивание степеней свободы — объект, устанавливающий связи между номерами уравнений системы и степенями свободы в объектах `DOFGroup`.

Рассмотрим назначение каждого из классов.

Класс решения задачи (`Analysis`) является классом, агрегирующим основные объекты задачи (см. рис. 3). Предназначен для проверки корректности объектов и установки необходимых связей между ними при выполнении своих функции для запуска решения. Вычислительные операции выполняются непосредственно объектами, входящими в состав класса `Analysis`. Основными функциями класса являются `Analyze()` (основная функция решения данной задачи) и `DomainChanged()` (вызывается, когда необходимо сообщить, что область решения изменилась и нужно перестроить модель решения задачи в соответствии с этим).

Наследуясь от базового класса `Analysis`, реализуются различные способы решения задачи. На рис. 3 показаны некоторые из них. Отметим классы, которые построены для адаптивного решения задачи (по *p-версии* — трехмерные задачи деформирования, по *h-* и *r-версии* — двухмерные): класс оценки погрешности, класс стратегии уточнения, критерий перестроения и т.д.

Класс "алгоритм решения" (`SolutionAlgorithm`). Методы этого класса управляют шагами решения задачи и определяют последовательность операций, которые будут вызваны у входящих в состав класса `Analysis` объектов.

Класс интеграции систем уравнений (`Integrator`). Объект класса `Integrator` предназначен для формирования системы уравнений задачи, определения вкладов объектов классов `DOFGroup` и

AnalysisElement в главную систему уравнений и обновления вычисляемых в задаче характеристик. Класс Integrator является абстрактным классом.

Для решения статических задач используется класс IncrementalIntegrator, наследующийся от класса Integrator и выполняющий следующие задачи.

1. Установление связей между объектами модели решения AnalysisModel и линейной системы уравнений LinearSOE.

2. Формирование линейных систем уравнений. Эти методы перебирают объекты AnalysisElement и DOFGroup и вызывают из них функции по формированию локальных матриц и векторов.

3. Определение вкладов объектов AnalysisElement и DOFGroup в главную систему уравнений и того, как эта система уравнений будет строиться.

4. Установление связи между решением системы уравнений и степенями свободы в узлах сетки.

Класс IncrementalIntegrator также является абстрактным. Наследники этого класса реализуют его абстрактные методы в зависимости от того, какой тип задачи решается.

Класс модели решения задачи (AnalysisModel) содержит объекты классов AnalysisElement и DOFGroup и предоставляет доступ к ним. Интерфейс класса имеет следующие функции:

— создание объектов AnalysisElement и DOFGroup с помощью объекта класса ConstraintHandler;

— доступ к объектам классов AnalysisElement и DOFGroup с помощью их индивидуальных идентификаторов из других объектов класса Analysis;

— информация о связности как объектов DOFGroup, так и индивидуальных степеней свободы, возвращаемая в виде графов; граф степеней свободы необходим для определения размера и разреженности системы уравнений; граф связности объектов DOFGroup нужен для назначения номеров уравнений степеням свободы.

В классе модели решения задачи не содержится информации о нагрузках и граничных условиях задачи. Объекты граничных условий сами напрямую применяют себя к узлам и элементам, что отражается на их матрицах и векторах. Объекты ограничений не нужны в модели решения, так как объект "обработчик ограничений" создает на их основе объекты AnalysisElement и DOFGroup, которые и вносят необходимую информацию в систему уравнений.

Класс группы степеней свободы (DOFGroup). Каждый объект DOFGroup представляет степени свободы, связанные с геометрическим узлом, или новые степени свободы, которые появляются, например, при обработке ограничений задачи с помощью метода множителей Лагранжа. Класс выполняет следующие функции: хранение информации об отображении связи между степенями свободы узла и номерами уравнений; предоставление функций, которые помогают объекту Integrator формировать матрицы и векторы; обновление пробных значений вычисляемых характеристик узла.

Класс обработчика граничных условий (ConstraintHandler). Данный класс создает объекты классов DOFGroup и AnalysisElement и добавляет их в объект AnalysisModel. Кроме того, обрабатывает граничные условия задачи (т.е. объекты классов SPConstraint и MPConstraint), создавая объекты специальных классов, наследуемых от DOFGroup и AnalysisElement. Тип создаваемых элементов зависит от типа объекта ConstraintHandler. На данный момент реализованы обработчики ограничений методом штрафных функций, методом множителей Лагранжа и методом конденсации.

Класс упорядочивания степеней свободы (DOFNumberer) назначает каждой степени свободы задачи номер уравнения. Это делается с помощью графа связности объектов DOFGroup. Задачи упорядочивания степеней свободы играют важную роль при дальнейшем решении, так как от этого зависит разреженность матрицы системы уравнений, ширина ее ленты и т.п. Класс DOFNumberer является абстрактным. Его наследники реализуют те или иные алгоритмы упорядочивания матрицы системы уравнений.

3.4. Классы решения задачи МДО. Основными классами метода декомпозиции являются: класс подобласти SubDomain; класс разделителя области DomainPartitioner; класс решения задачи МДО DomainDecompositionAlgorithm; класс алгоритма метода декомпозиции DomainDecompositionAlgorithm и классы решения систем уравнений в подобласти. Взаимосвязи между ними и основными классами других подсистем показаны на рис. 4.

Класс решения МДО (DomainDecompositionAnalysis). Агрегируются объекты ConstraintHandler, Integrator, DOFNumberer, AnalysisModel, LinearSOE, DomainSolver, DomainDecompositionAlgorithm. Все эти объекты выполняют численные операции метода декомпозиции. Этот подход близок к подходу, который применялся ранее в классе решения задачи Analysis, и также позволяет просто выбирать метод декомпозиции, схему хранения системы уравнений, решатель системы уравнений и т.д. Количество объектов класса DomainDecompositionAnalysis связано с количеством подобластей задачи. По сути,

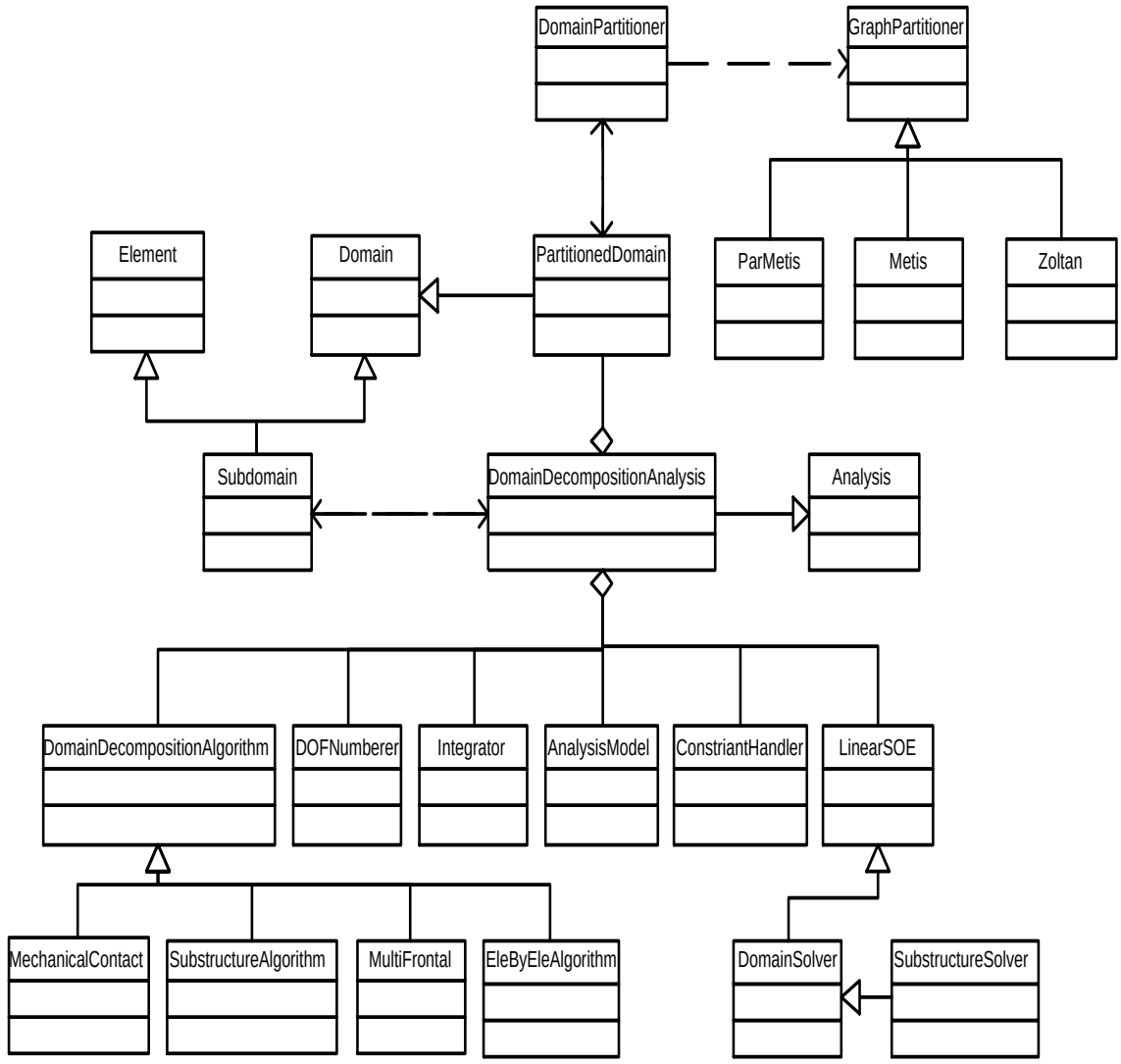

Рис. 4. Диаграмма классов модели метода декомпозиции области

класс `DomainDecompositionAnalysis` и агрегируемые им объекты представляют собой одну ветвь решения метода декомпозиции. Таким образом, все операции данной ветви происходят внутри класса `DomainDecompositionAnalysis`. Основными методами класса являются: `DomainChanged()` — вызывается классом разделителя подобласти после завершения процесса разделение для обновления модели решения задачи; методы, вызываемые классом подобласти `SubDomain` для вычисления и получения матриц и векторов подобласти.

Все вычисления непосредственно производятся объектами, которые агрегирует класс решения МДО `DomainDecompositionAnalysis`. Например, формирование матрицы и вектора правой части подобласти выполняет класс `Integrator`, а само решение системы и формирование матриц и векторов, необходимых для граничной системы уравнений, осуществляет класс `DomainSolver`.

Класс подобласти (`SubDomain`) наследуется от классов области (`Domain`) и класса элемента (`Element`). Это позволяет ему иметь необходимые свойства как области, так и компонента области. В интерфейс класса были введены дополнительные функции, которые можно разделить на две группы.

1. Функции по работе с граничными и внутренними узлами подобласти (добавление, удаление и т.п.). Это позволяет иметь информацию о том, какие узлы в подобласти являются внутренними, а какие лежат на границе с другими подобластями. Эта информация поступает от объекта разделителя области,

избавляя класс подобласти от необходимости определять границу самому.

2. Функции, связанные с численными задачами метода декомпозиции, такими как вычисление матриц и векторов систем уравнений. Класс подобласти сам по себе не выполняет эти операции. Их вызов влечет вызов методов из связанного с подобластью класса `DomainDecompositionAlgorithm`. Такой подход позволяет экспериментировать с различными методами декомпозиции, не меняя класс подобласти.

Класс "алгоритм метода декомпозиции" (`DomainDecompositionAlgorithm`). Данный класс ответственен за последовательность операций при обновлении решения задачи в узлах подобласти. Сначала запрашивается у подобласти последнее значение решения задачи для граничных узлов подобласти; эти данные передаются объекту `DomainSolver`. Далее решатель производит вычисление значений внутренних узлов. В завершение у объекта `Integrator` вызывается метод по обновлению состояния подобласти.

Класс решения систем уравнений в подобласти (`DomainSolver`). Функцией класса `DomainSolver` является выполнение всех численных операций метода декомпозиции. Он наследуется от класса решателя линейных уравнений `LinearSolver`, поэтому может использоваться и для решения обычных систем уравнений. Например, реализованы решатели подобласти, наследованные от классов решателей обычных, симметричных ленточных и профильных систем уравнений. В отличие от класса `LinearSolver`, в интерфейс класса `DomainSolver` добавлены несколько необходимых для решения систем уравнений в подобласти функций и атрибутов. Это такие свойства и операции, как количество внутренних и граничных степеней свободы подобласти, нахождение дополнения Шура, вычисление значений внутренних степеней свободы при известных значениях граничных и т.д. Так как эти операции зависят от типа метода декомпозиции, существует несколько подклассов решателя систем уравнений подобласти (решатель систем уравнений методом подструктур и т.д.).

Класс разделенной области (`PartitionedDomain`) наследуется от класса области `Domain`. Кроме основных функций области, представленных выше, он может хранить набор подобластей. Соответственно, в интерфейс класса добавлены необходимые при работе с подобластями функции: предоставление графа подобластей, массива граничных узлов подобластей и т.п.

Класс разделителя области (`DomainPartitioner`) ответственен за разделение области на подобласти. Разделение области является одной из важнейших операций метода декомпозиции. МДО предполагает балансировку нагрузки на уровне прикладной системы, которая связана с эффективным распределением данных между подобластями, таким, чтобы вычислительные затраты на каждом узле были примерно равны. Эта задача может решаться как в однородной, так и в неоднородной вычислительной среде. В расчетной модели МДО были введены объекты `PartitionedDomain` и `DomainPartitioner`, описывающие соответственно граф подобластей и метод его разбиения (`GraphPartitioner`). Таким образом, для эффективного распределения данных необходимо разработать систему объектов, унаследованных от `GraphPartitioner`, которые включают различные методы разбиения графа, а также подобрать для конкретной задачи наиболее приемлемый алгоритм разбиения или наделить объект `DomainPartitioner` некоторыми элементами экспертной системы, для того чтобы по каким-либо критериям автоматически или полуавтоматически производить выбор. Обзор и сравнение различных алгоритмов разделения сеток приведен в [25, 26].

Процесс разделения области классом `DomainPartitioner` происходит следующим образом. Сначала класс получает граф связности элементов области и делит его с помощью одного из алгоритмов разделения графов. Далее, основываясь на полученном разделенном графе, выполняется добавление компонентов области (узлов, элементов, ограничений и нагрузок) в определенную в соответствии с полученными результатами подобласть.

4. Возможности параллельной реализации. Способы применения метода декомпозиции области чрезвычайно разнообразны. Метод хорошо приспособлен к использованию на суперкомпьютерах и дает возможность адаптировать вычислительные алгоритмы к параллельной вычислительный системе с произвольной организацией памяти и связями между отдельными вычислительными узлами, позволяя эффективно использовать их при решении многомерных задач математической физики.

Алгоритмы МДО естественным образом допускают крупноблочное распараллеливание процесса решения задачи. Можно выделить два направления в декомпозиции области. Первое связано с методами явного деления на подобласти, где параллельные вычисления осуществляются на высоком уровне, т.е. число ветвей параллельного алгоритма равно числу подобластей. Практически такой подход дает значительное ускорение, если каждый процессор решает в своей подобласти свою подзадачу. В каждой подобласти задача может быть решена одним из эффективных методов. Однако возможность крупноблочного распараллеливания задачи имеется не всегда. Второе направление, где распараллеливание отражает среднезернистость задачи, — неявные методы декомпозиции подобластей (см. п. 2.3.1). Основные

параллельные операции выполняются на уровне матрично-векторных вычислений. Рассмотрение на этом уровне удобно использовать, если приходится отображать на вычислительную систему задачи с сильно меняющимися областями. Следующий уровень распараллеливания — уровень, отражающий мелкозернистость задачи и связанный с отдельными арифметическими операциями.

Крупнозернистое распараллеливание вычислений в МДО возможно при построении алгоритмов с большим числом достаточно крупных параллельных ветвей. При разработке такого программного обеспечения необходимо уделять значительное внимание роли декомпозиции задачи. Основным на данном этапе будет выявление параллельной сущности алгоритма МДО, создание параллельных процессов, описание потоков данных, балансировка нагрузки между процессами, диспетчеризация и синхронизация процессов.

Разработка программных комплексов для проведения крупномасштабных вычислительных экспериментов на параллельных вычислительных системах представляет собой сложную в теоретическом и практическом плане задачу. Существует большое количество работ, посвященных различным аспектам параллельного программирования. Типичными примерами такого рода задач служат программы для реализации отдельных алгоритмов из области линейной алгебры [23], программы для обработки изображений и т.д. Очевидно, что разработка параллельных программ МДО, практического уровня сложности представляет собою многоэтапный технологический процесс. Под технологией программирования здесь подразумеваются все этапы разработки параллельной программы, начиная с анализа задачи, выбора модели программы, декомпозиции задачи на параллельные процессы и заканчивая вопросами анализа производительности и организации вычислительного эксперимента.

Один из возможных подходов к разработке параллельного алгоритма состоит из следующих шагов:

— декомпозиция (разбиение) исходной задачи на элементарные задачи в подобластях;

— определение всех необходимых взаимодействий (коммуникаций) между задачами в подобластях;

— объединение задач в подобластях в соответствии с определенной стратегией, учитывающей в том числе количество процессоров в системе;

— оценка полученного алгоритма на масштабируемость (при увеличении объема исходных данных и/или изменении числа процессоров в мультипроцессорной системе).

Приведем код программы на C++, выполняющей расчет задачи с помощью метода подструктур (3) – (9), который может быть распараллелен.

В программе создаются четыре подобласти, каждая из которых использует обратную схему Катхилла–Макки для нумерации степеней свободы и профильную схему хранения матриц систем уравнений подобластей. Система уравнений для дополнения Шура (5) также решается прямым методом с указанной схемой хранения и упорядочивания. Сначала создается объект класса `PartitionedDomain` и связанные с ним объекты классов разделителя подобласти и разделителя графов (строки 1–3). После формирования объекта класса редактора расчетной модели из него вызывается метод `BuildModel()`, в ходе выполнения которого создаются объекты классов `Element`, `Node` и записываются в объект области (строки 4–5). Далее в цикле строятся четыре объекта класса `DomainDecompositionAnalysis` и `SubDomain` вместе со связанными с ними объектами (строки 6–18). Они представляют собой четыре независимые ветви алгоритма метода подструктур. После этого у объекта области вызывается метод `Partition()`, выполняющий разделение области на подобласти (строка 19). В строках 20–28 создается главный объект класса решения задачи `theAnalysis` и агрегируемые им объекты. Объект `theAnalysis` является главной (или управляющей) ветвью алгоритма метода декомпозиции. В последней его строке вызывается метод `Analyze()`, который запускает процесс решения задачи.

Объектно-ориентированная технология и объектная модель метода декомпозиции позволяют довольно легко выделить объекты и их функции, которые могут выполняться одновременно и независимо друг от друга. Приведенный выше пример показывает, как представленная объектно-ориентированная модель метода декомпозиции позволяет распределить данные (строки 6–19) и обработать их параллельно (строка 29) на уровне явного разделения области. Данная модель позволяет строить гибкие программы, что иллюстрируют главные вычислительные объекты Analysis, AnalysisModel, Algorithm и др. (строки 20–29), с помощью различных комбинаций которых можно, выполнив общие задачи метода декомпозиции, эффективно управлять процессом вычисления. Рассмотрим это подробнее.

1. Определяются задачи (`theAnalysis`), которым необходима информация из всех подобластей, и задачи — выполняющиеся независимо на каждой подобласти (четыре объекта `ddanalysis`).

Первые — последовательно работают с помощью объектов, входящих в главный объект решения задачи. Остальные – параллельно работают с помощью объектов в классе решения методом декомпозиции `DomainDecompositionAnalysis`, вызываются и синхронизируются объектами главного объекта решения

```
1  GraphPartitioner *metis = new Metis();
2  DomainPartitioner *partitioner = new DomainPartitioner(metis);
3  PartitionedDomain *domain = new PartitionedDomain(partitioner);
4  ModelBulider modelbuilder(domain);
5  modelbuilder.BuildModel();
6  int NumberOfSubdomains = 4;
7  for (int i = 1; i <= 4; i++)
       {
8        Subdomain subdomain(i);
9        AnalysisModel *model = new AnalysisModel();
10       DomainDecompositionAlgorithm *algo = new SubstructureAlgorithm();
11       StaticIntegrator  *integrator = new LoadControl(1.0,1.0,1.0,1.0);
12       ConstraintHandler *handler = new PenaltyConstraintHandler(1.0e8,1.0e8);
13       RCM  *theRCM = new RCM();
14       DOFNumberer *numberer = new DOFNumberer(*theRCM);
15       ProfileSPDSolver  *solver = new SPDSubstructureSolver();
16       LinearSOE *soe = new ProfileSPDSOE(*solver);
17       DomainDecompositionAnalysis *ddanalysis =
                                          new SubstructureAnalysis (subdomain,
                                                                    model,
                                                                    algo,
                                                                    integrator,
                                                                    handler,
                                                                    numberer,
                                                                    soe);
18       domain.AddSubdomain(subdomain);
        }
19 domain.Partition(NumberOfSubdomains);
20 AnalysisModel *model = new AnalysisModel();
21 SolutionAlgorithm *algo = new LinearSA();
22 StaticIntegrator *integrator = new LoadControl(1.0, 1.0, 1.0,1.0);
23 ConstraintHandler *handler = new PenaltyConstraintHandler(1.0e8,1.0e8);
24 RCM  *scheme = new RCM();
25 DOFNumberer  *numberer = new DOFNumberer(*scheme);
26 BandSPDSolver  *solver = new BandSPDSolver();
27 LinearSOE *soe = new BandSPDSOE(*Solver);
28 StaticAnalysis theAnalysis(domain,
                             handler,
                             numberer,
                             model,
                             algo,
                             soe,
                             integrator);
29 theAnalysis.Analyze();
```

задачи. Каждый объект подобласти `Subdomain` и все объекты, которые он агрегирует, создаются в отдельном процессе. Это позволяет, во-первых, разделить данные задачи между различными процессами. Во-вторых, так как все методы объект "подобласть" выполняет с помощью входящих в него объектов, операции над подобластью будут выполняться параллельно. Например, при решении систем уравнений для внутренних степеней свободы подобласти объект алгоритма решения задачи вызывает асинхронно у каждой подобласти функцию решения этих уравнений. Далее каждый объект подобласти в своем процессе, то есть параллельно, вызывает метод `SolveInternal()` у своего объекта решателя уравнений подобласти `DomainSolver`. После этого полученные результаты также параллельно записываются в со-

ответствующие узлы и элементы подобласти. В это время объект алгоритма решения задачи ожидает завершения функции на всех подобластях или выполняет в своем процессе какую-либо другую задачу.

2. Процесс разделения области на подобласти выполняется объектом `DomainPartitioner`. Как было отмечено, для разделения области используется объект разделителя графов. Если размер расчетной модели задачи небольшой, то можно использовать последовательный способ разделения, но когда граф имеет большое количество узлов или необходима динамическая балансировка нагрузки, то процесс разделения области должен выполняться параллельно. В этом случае используется объект параллельного разделителя графов (например объект, инкапсулирующий возможности какой-либо параллельной библиотеки (ParMetis, Zoltan)).

Выше показано, как представленная объектная модель метода декомпозиции позволяет разделять данные задачи между различными процессами и выполнять различные задачи параллельно над данными. В то же время, использование главного объекта решения задачи и связанных с ним объектов, наряду с объектами решения метода декомпозиции, дает возможность выполнять общие этапы метода декомпозиции и эффективно управлять процессом параллельного решения.

Объектно-ориентированный стиль может быть использован для разработки параллельных распределенных программ, если в их основе лежат технологии, которые позволили бы запускать объекты в разных процессах и вычислительных узлах. Атрибуты таких объектов можно поставить в соответствие той или иной части распределенных данных, методы — процессам обработки этих данных. Различные способы реализации методов позволяют имитировать последовательные и параллельные вычисления.

В настоящее время существует несколько технологий параллельного объектно-ориентированного программирования. В нашем случае выбрана технология объектно-ориентированного распределенного программирования CORBA и ее возможность асинхронного вызова функций объектов (AMI). На этой основе реализована технология параллельных распределенных компонентов [27], которая позволяет активировать объекты на разных вычислительных узлах и взаимодействовать с ними синхронно или асинхронно. Таким образом, пример, рассмотренный ранее, без значительных изменений становится параллельным и распределенным.

5. Заключение. Рассмотрены с точки зрения программирования методы декомпозиции области и построена объектно-ориентированная вычислительная модель, которую можно реализовать на том или ином языке программирования, с помощью той или иной вычислительной технологии. В рамках данного подхода проведен теоретический объектно-ориентированный анализ основных задач МДО, в результате чего дана детальная классификации объектов, вычислительных алгоритмов и возникающих вычислительных задач метода декомпозиции. Представлена иерархия наследуемых классов, которая составляет конструктивную основу программной реализации библиотеки МДО и ее расширения. Представлены возможности адаптации алгоритмов МДО к параллельной вычислительный системе с произвольной организацией памяти и связями между отдельными узлами.

Работа выполнена при поддержке Российского фонда фундаментальных исследований (проект 02–07–90265) и Уральского отделения РАН (грант молодых ученых).

СПИСОК ЛИТЕРАТУРЫ

1. *Przemieniecki J.S.* Theory of matrix structural analysis. N.Y.: McGaw-Hill, 1968.
2. Метод суперэлементов в расчетах инженерных сооружений / В.А. Постнов, С.А. Дмитриев, Б.К. Елтышев, Л.А. Родионов. Л.: Судостроение, 1979.
3. Метод редуцированных элементов для расчета конструкций / Е.Я. Вороненок, О.М. Палий, С.В. Сочинский. Л.: Судостроение, 1990.
4. *Марчук Г.И.* Методы вычислительной математики. М.: Наука, 1989.
5. *Хокни Р., Иствуд Дж.* Численное моделирование методом частиц. М.: Мир, 1987.
6. *Агошков В.И., Лебедев В.И.* Операторы Пуанкаре–Стеклова и методы разделения области в вариационных задачах // Вычислительные процессы и системы. Вып. 2. М.: Наука, 1985. 173-227.
7. *Соболев С.Л.* Алгоритм Шварца в теории упругости // ДАН СССР. 1936. **4**, № 6. 235–238.
8. *Никольский Е.Н.* Алгоритм Шварца в задаче теории упругости в напряжениях // Докл. АН СССР. 1960. **135**, № 3. 549–552.
9. *Вабищевич П.Н.* Итерационные методы декомпозиции областей с налеганием для эллиптических краевых задач // Дифф. уравнения. 1996. **32**, № 1. 923–927.
10. *Копысов С.П.* Методы декомпозиции и параллельные схемы метода конечных элементов. Препринт ИПМ УрО РАН. Ижевск, 1999.
11. *Novikov A.K., Kopyssov S.P.* A parallel element-by-element conjugate gradient method with decreased communication costs // Intern. Summer School “Iterative Methods and Matrix Computations”. Rostov-on-Don, 2002. 450–454.

12. *Irons B.M.* A frontal solution program for finite-element analysis // Intern. J. on Numerical Methods in Engineering. 1970. N 2. 5–32.
13. *Kopyssov S.P., Ustyshanin S.L.* Domain decomposition based on mechanical contact of substructures // Intern. Conf. OFEA'2001 "Optimization of Finite Element Approximations & Splines and Wavelets". St. Petersburg, 2001. 77–78.
14. *Буч Г.* Объектно-ориентированный анализ и проектирование с примерами приложений на C++. М.: Бином, 1999.
15. *Гайсарян С.С.* Объектно-ориентированные технологии проектирования прикладных программных систем (http://www.citforum.ru/programming/oop_rsis/index.shtml).
16. *Семенов В.А.* Объектная систематизация и парадигмы вычислительной математики // Программирование. 1997. № 4. 14–25.
17. *Hagger M.J.* Automatic domain decomposition on unstructured grids (DOUG) // Advances in Computational Mathematics. 1998. **9**, N 3, 4. 281–310.
18. *Рычков В.Н., Красноперов И.В., Копысов С.П.* Объектно-ориентированная параллельная распределенная система для конечно-элементного анализа // Матем. моделирование. 2002. **14**, № 9. 81–86.
19. *Dubois-Pelerin Y., Pegon P.* Improving modularity in object-oriented finite element programming // Communications in Numerical Methods in Engineering. 1997. **13**. 193-198.
20. *Mukunda G. R., Sotelino E. D., Hsieh S.H.* Distributed finite element computations using object-oriented techniques // Engineering with Computers. 1998. **14**, N 1. 59–72.
22.. *Cai X.* Domain decomposition in high-level parallelization of PDE codes // Proc. of the 11th Intern. Conf. on Domain Decomposition Methods. Greenwich, 1998. 388–395.
22. *Bruaset A. M., Langtangen H. P., Zumbusch G.* Domain decomposition and multilevel methods in Diffpack // Proc. of the 9th Intern. Conf. on Domain Decomposition Methods. Bergen, 1996. 655–662.
23. *Balay S., Gropp W., McInnes L.C., Smith B.* PETSc 2.0 users manual. Argonne National Laboratory. Argonne, 1999.
24. *Aschraft C., Pierce D., Wah D.K., Wu J.* The reference manual for SPOOLES, release 2.2.: an object oriented software library for solving sparse linear systems of equations. Boeing Shared Services Group. Seattle, 1999.
25. *Копысов С.П., Новиков А.К.* Параллельные алгоритмы адаптивного перестроения и разделения неструктурированных сеток // Матем. моделирование. 2002. **14**, № 9. 91–96.
26. *Копысов С.П.* Программное обеспечение динамической балансировки нагрузки // Материалы Международной конференции "СуперЭВМ и многопроцессорные вычислительные системы". Таганрог, 2002. 157–160.
27. *Копысов С.П., Красноперов И.В., Рычков В.Н.* Реализация объектно-ориентированной модели метода декомпозиции на основе параллельных распределенных компонентов CORBA // Вычислительные методы и программирование. 2003. **4**, № 1. 19–36 (http://num-meth.srcc.msu.su).

Поступила в редакцию
28.02.2003