ФАБРИКА МОДЕЛЕЙ

А.И.Илюшин

1. Мотивация.
2. Архитектура реализации.
3. Программный уровень.
4. Вычислительный уровень.
5. Примеры.

1. Мотивация.

   В настоящее время, как мне кажется, актуален переход от традиционного способа построения вычислительных моделей (физическая область -> вычислительная модель в виде единого целого) к построению итоговой модели из подмоделей (физическая область -> подобласти -> вычислительные модели подобластей –> полная модель в виде сборки из подмоделей).

   Можно указать, как минимум, три причины для такого перехода:

   1. Счет на многопроцессорных вычислительных системах требует разбиения программной модели на части. В настоящее время преобладает явно неэффективный подход, когда физическую область, по самой своей сути состоящую из параллельно эволюционирующих и взаимодействующих друг с другом частей, отображают сначала на единую вычислительную схему с последовательным алгоритмом интерпретации, затем программируют этот последовательных алгоритм, и, наконец, пытаются «распараллелить» эту последовательную программу. Очевидно, что ошибка подхода именно в исходном сведении параллельной системы к последовательной. Если этот этап ликвидировать, то само понятие «распараллеливание» теряет всякий смысл и его употребление в конкретном проекте означает порочность самой концепции, заложенной в проект, так как консервируется неэффективная схема (область – последовательный алгоритм – распараллеливание).
   2. Вычислительная мощность современных систем такова, что в принципе позволяет строить большие комплексные модели, учитывающие не несколько «основных» факторов, а множество разнородных. Например, модель волн в прибрежных областях можно построить, учитывая лишь механику движения волн, а можно учесть влияние солнечной активности, состояние атмосферы, состояние мантии, движение планет и т.д. Такой учет требует строить конкретную модель путем композиции уже независимо разработанных моделей в смежных областях.
   3. В случае сборки становится возможно создание хранилища моделей. Модели из этого хранилища могут быть использованы для построения составных моделей. Могут храниться:

      - модели для сравнительно простых «базовых» физических областей (аналог библиотеки стандартных подпрограмм);
      - готовые модели, предлагаемые разработчиками бесплатно или на платной основе (хранилище приложений в интернете);
      - возможно использование хранилища в качестве инструментария для реализации больших проектов, когда по единому плану одновременно реализуются части одной большой модели.

2. Архитектура реализации.

2.1. Схема построения модели.

Создание модели предлагается разбить на четыре этапа:
- определение структуры физической области (составляющие ее части и связи между ними). Прикладной математик(и), ставящий задачу, создает «ручное» описание для каждой подобласти, которое может интерпретировать человек (в отличие от описания, которое может прочесть и проинтерпретировать машина, например на языке C++).
- создание вычислительных моделей для частей области, определенных на первом этапе. На этом этапе прикладной математик создает вычислительную схему для конкретной части, а программист для этой схемы создает «решатель», который может интерпретировать машина.
- математик выбирает протокол взаимодействия частей-решателей (см. 2.2.2 и 2.2.4) .
- программист создает каждую подмодель из «решателя» и «адаптеров», реализующих протоколы взаимодействия с подмоделями, «соседними» к описываемой подмодели.
- программист создает «машинное» описание структуры программной модели (в виде перечня подмоделей и связей между ними).

2.2. Основные компоненты.

2.2.2 Решатели.

Основными компонентами, из которых строится модель, являются «решатели» для подобластей. Предполагается, что в основном это «старые» последовательные решатели, разработанные в традиционном стиле. Встраивание в новую архитектуру осуществляется с помощью объектов-оболочек. «Новые» решатели рассматриваются как стандартные вычислительные модели для базовых типов физических областей. Их совокупность играет роль библиотеки шаблонов (прототипов) для получения компонент составных моделей.

2.2.2 Объекты-адаптеры.

Каждый объект-адаптер реализует конкретный протокол согласования граничных значений на внутренних границах между подобластями после фазы независимого счета очередного шага по времени (псевдо времени для статических моделей). Предполагается, что будет создан свой шаблон объекта-адаптера для каждого протокола взаимодействия. Желательным вариантом было бы создание одного протокола (и соответственно одного шаблона) для каждого типа уравнений. Это означает, что программист, занимающийся сборкой модели, не должен разрабатывать протокол взаимодействия и описывать реализацию объекта-адаптера. Он должен его создать, используя имеющийся шаблон.

2.2.3. Объекты-оболочки.

Все решатели должны быть приведены к стандартному виду. Они должны представлять собой программные объекты в смысле используемого языка программирования (C++, Python) со стандартизованным набором операций и стандартизованными форматами данных (например, CGNS для формата данных представления областей). Отметим, что приведение решателя к стандартному О-О виду с помощью объектов-оболочек – это

малозатратная техническая работа, которая выполняется при создании «библиотеки» подмоделей. Решатель будет подобъектом в объекте-оболочке.

2.2.4. Протоколы, реализуемые объектами-адаптерами.

Целью введения этих протоколов является принципиальное снижение трудозатрат при сборке составной вычислительной модели из готовых моделей подобластей. Перечислим этапы создания вычислительной модели по предлагаемой схеме:

1. Выбор уже существующих решателей для подобластей.
2. Приведение решателей к стандартному объектно-ориентированному виду путем создания объектов-оболочек. Это малозатратная техническая операция, так как в интерфейсе объекта оболочки должны существовать лишь операция начала работы и операции обмена результатами шага с соседями.
3. Выбор шаблонов адаптеров (шаблоны – это программные классы, реализующие конкретные протоколы взаимодействия объектов-решателей) для каждой взаимодействующей пары решателей.
4. Для каждого решателя программирование создания объектов-адаптеров для каждого «соседнего» решателя с использованием соответствующих конкретных шаблонов взаимодействия для рассматриваемой пары. Объект-решатель связывается со всеми своими адаптерами двусторонними ссылками.
5. Определение топологии связей между решателями, например, средствами системы OST. После этого в момент счета все решатели будут связаны в одну систему через связи между соответствующими адаптерами (решатель 🔔 адаптер конкретного соседа 🔔адаптер соседа для рассматриваемого решателя <= >соседний решатель).

   Таким образом, задача создания составной модели сводится в основном к выбору уже существующих типов решателей для подобластей, выбору уже существующих типов адаптеров и заданию топологии связей между решателями.

2.2.5. «Правдоподобное» обоснование возможности создания небольшого числа типов адаптеров для обширного класса физических областей.

Рассмотрим предлагаемую общую схему счета. Продвижение на один шаг по времени состоит из следующих этапов:

1. Независимый счет подобластей. В качестве начальных значений берутся значения из предыдущего шага. В качестве граничных значений на внутренних границах между подобластями берутся «угаданные» значения (например, из предыдущего шага). В результате получаем «правильные» значения во внутренней части подобласти и «неправильные» в некоторой приграничной полуполосе. Вся приграничная полоса – это объединение полуполосы рассматриваемой подобласти и прилегающих к ней полуполос соседних подобластей. Предполагается, что за счет использования соответствующего шага по времени размер полуполосы (и соответственно

объем пересчета этой полуполосы), как минимум на десятичный порядок меньше всей подобласти. Заметим, что это требование намного слабее случая явной схемы, где ширина полуполосы равна одной ячейке.

2. Итерационный пересчет полуполос для каждой подобласти. Чтобы определить критерий окончания итераций и критерии выбора скорректированных граничных значений для пересчета полуполос, введем понятие «фантомной» энергии. Для этого рассмотрим два варианта выполнения шага. Первый вариант – счет всей области. В этом случае по результатам счета возможен расчет потоков энергии через внутренние границы за время шага. Вообще говоря, можно говорить обо всех типах потоков, фигурирующих в ответствующих законах сохранения (для упрощения изложения остановимся только на энергии). Тогда при отсутствии источников/приемников в каждой точке (ячейке в дискретном случае) границы сумма потоков с разных сторон границы ?на нормали к ней? Эти слова наверное нужно опустить равна нулю в случае расчета одного шага для всей области в целом. В случае независимого счета одного шага расчета для каждой подобласти отдельно сумма потоков с разных сторон границы в каждой точке границы будет отлична от нуля, если «угадано» неправильное граничное значение. Конечно, это утверждение справедливо только тогда, когда это не граничное условие второго рода, которое задает именно поток. С целью сокращения словесных оборотов речи мы можем назвать эту сумму «фантомной» энергией, имея в виду, что в физической системе она отсутствует, а появляется только в возможной физической интерпретации факта задания «неправильных» граничных значений. Величину этой энергии предлагается использовать в качестве критерия окончания итераций, так как из выполнения законов сохранения в окрестности каждой точки на границе следует корректность полученного решения.
   Для граничных условий первого и второго рода нужны разные адаптеры. Для граничных условий второго рода при «неправильном» граничном значении должна существовать существенная переменная, описывающая состояние области, значения которой слева и справа от границы при независимом счете подобластей разойдутся.
   Сказанное выше является лишь «правдоподобным» описанием схемы счета. В конкретных алгоритмах должны быть точно определены:
   - тип граничных условий;
   - способ получения из имеющихся двух временных слоев для окрестности точки на внутренней границе значения потока;
   - обратное преобразование скорректированного потока (например, новое значение потока равно полусумме (без учета знаков) потоков с двух сторон границы) в граничные значения для очередной итерации полуполосы.

3. Программный уровень
   Рассмотрим подробнее алгоритмы работы объектов-оболочек для решателей и адаптеров, а также их интерфейсы. Объекты-оболочки должны лишь приводить интерфейс для работы с решателями к некоторой

стандартной форме и их создание должно являться несложной технической задачей. Адаптеры должны реализовывать основной алгоритм согласования граничных значений на внутренних границах между подобластями после независимого счета подобластей для одного шага. В идеале должно существовать небольшое число типов адаптеров, реализованных высококвалифицированными математиками и программистами. Таким образом, создания набора адаптеров можно рассматривать как «вынесение за скобки» в «системную часть» тех действий, которые каждый создатель вычислительной модели был вынужден программировать самостоятельно. При этом сами эти действия во многих случаях были по сути одними и теми же.

На данном этапе разработки такая идеальная ситуация трудно достижима. Поэтому адаптеры придется индивидуально создавать для частных типов уравнений. Приведем схему взаимодействия (передач управления) между рассмотренными выше компонентами при функционировании вычислительной модели. За основу этой схемы взята схема функционирования вычислительной модели, использованная системе OST:

3.1.1. После загрузки объектов-оболочек в процессоры вычислительной системы в каждом из них вызывается функция RUN. Решатели входят в состав объектов-оболочек в качестве подобъектов. Оказалось, что адаптеры естественно включить в описание формальных соседей. А именно при описании окружения для объекта, входящего в состав модели, в виде списка формальных соседей, конкретный тип адаптера указывается в описании каждого соседа. Поэтому после распределения объектов-оболочек по процессорам все нужные компоненты автоматически оказываются в оперативной памяти и готовы к счету.

3.1.2. Объект-оболочка, выполняя операцию RUN, вызывает в решателе операцию счета первого шага в соответствии с соглашениями, специфичными для конкретного решателя. В качестве граничных условий на внутренних границах задаются «угаданные» значения (например, начальные значения). Решатель считает первый шаг и возвращает управление в оболочку.

3.1.3. Объект-оболочка выполняет вызовы операций, описанных в соседях. Содержательно эти вызовы, как правило, соответствуют некоторым физическим взаимодействиям между частями модели. Например, это может быть передача импульса соседу. Перечень этих вызовов специфичен для каждого типа решателей (естественно, что у пары соседей должен иметься согласованный набор операций и на некотором этапе загрузки модели должна автоматически выполняться проверка типов). Далее каждый вызов попадают в служебный объект внутри OST (объект-связи) и происходит согласование:

- локальных времен вызывающего и вызываемого объектов. А именно вызов выполняется, если локальные времена взаимодействующих объектов совпадают;
- граничных условий на внутренней границе между подобластями, которые описываются вызывающим и вызываемым объектами. Это согласование выполняется адаптерами и представляет собой итерационный процесс, сопровождающийся вызовами решателей для обсчета приграничных полуполос и обменами результатам каждой итерации между адаптерами двух соседей, расположенных, вообще говоря, в разных процессорах.

После окончания работы по согласованию результатов счета текущего шага со всеми соседями объект оболочка вызывает счет следующего шага в решателе.

3.2. Объект-оболочка.

В настоящий момент предполагается, что объект-оболочка умеет выполнять следующий набор операций:

- операцию RUN – начало работы.
- операция «счет шага» для указанной области с указанными граничными значениями.
- набор операций, определяемый типом используемого решателя. Это операции, соответствующие содержательным взаимодействиям частей вычислительной модели. В сложившейся практике как следствие широкого использования MPI это просто операции обмена данными с соседями. При этом содержательный смысл операции «закодирован» в самих передаваемых данных. В нашем случае вызов этой операции означает для объекта оболочки, что как в соседней подобласти, так и в рассматриваемой подобласти окончен шаг и в случае получения аналогичных вызовов ото всех соседей (или возвратов ото всех соседей для аналогичных операций, выданных рассматриваемым объектом-оболочкой) можно выполнять следующий шаг. В качестве аргумента операции используется указатель на просчитанную подобласть. Заметим, что факт возможного дублирования вызовов слева и справа от границы по поводу окончания шага может быть легко обработан объектом-оболочкой. Например, можно после проверки не делать вызов к соседу, если текущий объект-оболочка уже обработал вызов от соседа по поводу данного шага. Если же два вызова с двух сторон все таки выполнились одновременно, то обработка этого факта также не вызывает затруднений.

3.3. Объект-адаптер.

В настоящий момент предполагается, что объект-адаптер умеет выполнять следующий набор операций:

- окончена первая фаза шага для подобласти с «неправильными» граничными условиями. Операция вызывается управляющей программой OST”а при вызове объектом-оболочкой операции обмена результатами счета шага с соседней подобластью.
- операция обмена данными с соседним адаптером.

Объект адаптер должен выполнить итерационный процесс согласования граничных значений с адаптером соседа. При этом происходят вызовы объекта-оболочки для пересчета приграничной полосы со скорректированными граничными значениями. Далее адаптер должен «продолжить» через OST исходный вызов в указанный соседний объект. Адаптеры для разных соседей и для одного объекта-оболочки должны действовать согласовано. Например, они могут выполняться последовательно (ведь все происходит на одном процессоре) до тех пор, пока для всех них не выполниться условия окончания итераций.