УДК 519.6

ПОДХОД К ПАРАЛЛЕЛЬНОЙ РЕАЛИЗАЦИИ ЧИСЛЕННЫХ МЕТОДОВ НА НЕСТРУКТУРИРОВАННЫХ СЕТКАХ

А. Н. Андрианов[1], **К. Н. Ефимкин**[1]

Рассматривается подход, который позволяет создавать для численных методов, использующих адаптивные неструктурированные сетки, параллельные программы для многопроцессорных систем с распределенной памятью. Данный подход может применяться как при распараллеливании уже существующих последовательных программ, так и при разработке новых параллельных программ. Работа выполнена при финансовой поддержке РФФИ (код проекта 05–07–90080в).

Ключевые слова: параллельные численные методы, неструктурированные сетки, адаптивные сетки, многопроцессорные системы, распределенная память.

Введение. Метод НЕСЛА (неявный свободно-лагранжев) [1] — это метод расчета двумерных нестационарных магнитогазодинамических течений, основанный на использовании лагранжевых сеток переменной структуры и неявных полностью консервативных операторно-разностных схем магнитной гидродинамики. В процессе расчета возникает необходимость изменения структуры расчетной сетки, которая определяется тем обстоятельством, что лагранжева сетка, связанная с движущейся средой, может искажаться в процессе расчета. Задача параллельной реализации операций перестройки сетки является достаточно интересной, но здесь не рассматривается.

Одной из главных целей работы по распараллеливанию метода НЕСЛА было стремление к минимальному изменению исходного текста последовательной программы. Это стремление обусловлено тем, что объем исходных программ, реализующих рассматриваемый метод, достаточно велик.

В настоящей статье определяются основные структуры данных, способы распределения данных и вычислений, а также набор подпрограмм, поддерживающих операции с распределенными данными на параллельной системе. Рассматриваются правила преобразования последовательной программы в параллельную.

1. Последовательные программы: базовые структуры данных и основные операции.

1.1. Базовые структуры данных. Основное понятие, используемое при реализации численных методов указанного выше типа, — это сетка, состоящая из *узлов* и *ячеек*.

Сетка может рассматриваться как граф, вершинами которого являются узлы; связи между узлами представляются дугами. Узлы, между которыми есть связь, называются *соседними*. В дальнейшем будем называть граф, представляющий сетку, *графом узлов*. В общем случае ячейка является многоугольником и задается списком узлов сетки, расположенных в ее вершинах. Ниже мы рассматриваем ячейки, являющиеся минимальными треугольниками, т.е. не содержащими внутри себя другую ячейку.

На рис. 1 кругами изображены узлы сетки с номерами от 1 до 10. Три черных узла с номерами 6, 4, 7 составляют треугольную ячейку; кроме того, на рисунке приведены ячейки (5,4,1), (1,4,2), (2,4,7), (2,7,3), (3,7,10), (10,7,9), (9,7,8), (8,7,6) и (8,6,5). Соседними к узлу 4 являются узлы 1, 2, 7, 6, 5.

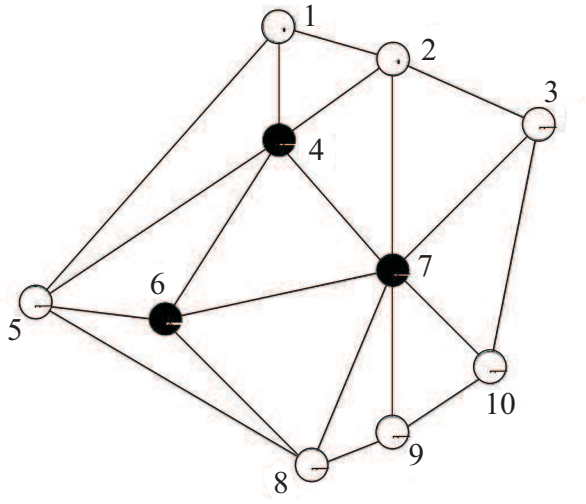

Рис. 1. Фрагмент сетки

В численных методах могут использоваться дополнительные понятия, определяемые на основании

[1] Институт прикладной математики им. М. В. Келдыша РАН, Миусская пл., 4, 125047, Москва; e-mail: and@a5.kiam.ru

© Научно-исследовательский вычислительный центр МГУ им. М. В. Ломоносова

графа узлов. Примером такого понятия, используемого ниже, является *вхождение узла в ячейки*. Например, узел 4 входит в ячейки (5,1,4), (4,1,2), (4,2,7), (7,4,6) и (5,6,4). В узлах сетки вычисляются величины, значения которых могут зависеть от значений расчетных величин в соседних узлах; в ячейках вычисляются ячеечные функции, значения которых могут зависеть от значений в узлах-вершинах данной ячейки или в соседних к ней узлах. Узлы могут быть граничными — находиться на границе расчетной области, либо внутренними — находиться внутри расчетной области. Правила расчета для граничных и внутренних узлов различны. Для простоты изложения здесь мы обсуждаем только те структуры, которые касаются задания узлов и ячеек.

Для представления этих понятий при программировании используются следующие правила и базовые структуры данных.

Нумерация узлов и ячеек. Все узлы нумеруются целыми числами от 1 до nnode, все ячейки — целыми числами от 1 до ncell. Порядок нумерации произвольный, обычно он определяется математическим методом построения сетки.

Граф узлов представляется в виде массива

integer grafnode(nnode, maxnode),

где maxnode — максимальное число возможных соседей и i-я строка массива содержит номера узлов, соседних к i-му узлу. Кроме того, задан вектор vsn(nnode), i-й элемент которого равен числу соседей i-го узла.

Представление ячеек. Ячейки представляются массивом

integer cell(ncell, 3),

где i-я строка массива содержит номера трех узлов, составляющих i-ю ячейку.

Вхождение узла в ячейки представляется массивом

integer grafcell(nnode, maxcell),

где maxcell — максимальное число возможных ячеек, в которые может входить узел, и i-я строка массива содержит номера ячеек, в которые входит i-й узел. Кроме того, задан вектор vsc(nnode), i-й элемент которого равен числу ячеек, в которые входит i-й узел.

1.2. Основные операции. В данном разделе рассматриваются характерные операторы исходной последовательной программы.

В первом примере приведен цикл по узлам. В рамках этого цикла вычисляются значения узловой функции (FJR3) по значениям ячеечных функций FI и V из ячеек, в которые входит рассматриваемый узел.

Пример 1.

```
      C цикл по узлам
          DO J = 1,NNODE
              S1 = 0.D0
              S2 = 0.D0
              DO K = 1,VSC(J)
      C используются значения FI, V в соседних ячейках
                  I = GRAFCELL(J,K)
                  S1 = S1+FI(I)*V(I)
                  S2 = S2+V(I)
              ENDDO
              FJR3(J) = S1/S2
          ENDDO
```

Во втором примере вычисляется значение ячеечной функции FI. В качестве аргументов функции используются как узловые (FIR1, FIZ1), так и ячеечные (ANR, ANZ и ZRV) функции.

Пример 2.

```
      C цикл по ячейкам
          DO I = 1,NCELL
              FI(I) = 0.0D0
              DO K = 1,3
      C используются значения FIR1, FIZ1 в узлах ячейки
```

```
            J = CELL(I,K)
            FI(I) = FI(I)+ANR(I,K)*FIR1(J)+ANZ(I,K)*FIZ1(J)
        ENDDO
        FI(I) = FI(I)/(3.0D0*ZRV(I))
    ENDDO
```

Общим для приведенных двух примеров является то, что во внешних циклах просматриваются последовательно элементы структуры (в первом случае — узлы, а во втором случае — ячейки) и вычисляемые функции определены точно на таких же структурах (переменная FJR3 определена как узловая, а переменная FI определена как ячеечная).

В следующем примере внешний цикл организован по ячейкам, однако присваивание значений производится в узловую переменную.

Пример 3.

```
C цикл по ячейкам
    DO I = 1,NCELL
        DO K = 1,3
            J = CELL(I,K)
C ниже J — это номер узла
            FJR2(J) = FJR2(J)+(ANR(I,K)*SK(I))*FI1(I) +ANZ(I,K)* FI2(I)
        ENDDO
    ENDDO
```

В заключение приведем еще один пример, иллюстрирующий вычисление редукционных функций. В этом примере проводится вычисление суммы и нахождение максимального значения. Аргументами для вычисления являются узловые функции.

Пример 4.

```
C вычисление суммы SL2 и максимума S2
C цикл по узлам
    SL2 = 0.0
    S2 = 1.D-10
    DO J = 1,NNODE
        R1 = FJR2(J)–FJR3(J)
        SL2 = SL2+R1*R1*RV(J)/1.D5
        FPR = DABS(FJR(J))
        FJR6(J) = FJR1(J)
        FJR1(J) = R1
        SS2 = DABS(R1/(FPR+1.D0))
        S2 = DMAX1(S2,SS2)
    ENDDO
```

В разделе 2.6 для рассмотренных примеров будет показано, какие преобразования необходимо выполнить при переходе к параллельной версии программы.

2. Параллельная программа на треугольных сетках: общие принципы создания.

2.1. Критерии распределения данных и вычислений. Высокая эффективность вычислений на параллельной системе с распределенной архитектурой обычно достигается в случаях

(1) равномерной загрузки процессоров вычислениями и

(2) минимизации простоев на обмены данными между процессорами.

В рассматриваемой задаче эти два критерия учитываются при распределении по процессорам *узлов* статической неструктурированной треугольной сетки. Остальные структуры данных, представляющие ячейки, значения расчетных переменных в узлах и ячейках и т.п., распределяются в соответствии с распределением узлов. Такой подход объясняется тем фактом, что граф узлов является необходимой и основной информационной структурой, определяющей все остальные понятия и структуры. По-видимому, можно использовать и другой вариант — взять за основу распределение *ячеек*, однако этот вариант является менее универсальным, так как понятие ячейки используется не во всех численных методах решения задач на нерегулярных сетках.

Критерий (1) при *начальном распределении* узлов удовлетворяется достаточно просто — для этого надо взять [(nnode-1)/p]+1 узлов на процессор (p — число процессоров). В случае динамически изменяемых сеток в процессе вычислений число узлов в процессорах при перестройке сетки меняется, причем неравномерно, поэтому для удовлетворения критерия (1) необходимо устраивать дополнительно *перерас-*

пределение узлов между процессорами.

Распределение узлов с учетом критерия (2) является самостоятельной математической задачей, для решения которой известны несколько способов, например, часто используется свободно распространяемое семейство пакетов METIS [2] или программы, разработанные в Институте математического моделирования РАН [3]. Неплохие результаты [4] показал также геометрический метод распределения [5], в качестве одного из критериев распределения в котором используется минимизация числа связей между процессорами (а не только минимизация объема пересылаемых данных), что часто дает лучшие результаты. Этот метод и был применен для начального распределения узлов в рассматриваемом классе задач.

2.2. Подход к созданию параллельной программы. Способы распределения базовых структур данных и работы с ними на вычислительной системе с распределенной архитектурой являются ключевым моментом при выборе путей создания параллельных программ рассматриваемого класса.

Понятно, что информация, содержащаяся в базовых структурах данных, может быть представлена многими способами и структурами данных, например, grafnode(nnode,maxnode) и grafcell(nnode,maxcell) могут быть представлены одним массивом grafnc(nnode,neigh,2). Отличия такого типа не являются существенными, если структуры из последовательной программы "близки" к базовым. В случае применения в последовательной программе структур данных, "плохо сводимых" к базовым, предлагаемая методика распараллеливания может оказаться неприменимой.

Замечание 1. Кроме базовых структур численные сеточные методы, конечно, используют векторы значений расчетных переменных, доступ к элементам которых осуществляется за счет индексации по значениям базовых структур данных. Например, если вектор real T(nnode) представляет собой значения температуры в узлах, то цикл

```
DO J = 1, NNODE
    S = 0.0
    DO I = 1, VSN(J)
        S = S+T(GRAFNODE(j, i))
    ENDDO
ENDDO
```

позволяет найти сумму S температур во всех узлах, соседних с j-м узлом. С точки зрения распараллеливания правила распределения таких массивов по процессорам являются вторичными и следуют из правил распределения основных структур данных, представляющих сетку и приведенных ниже. С точки зрения возможности автоматического распараллеливания задач рассматриваемого класса приведенный выше пример демонстрирует одну из основных трудностей — косвенная индексация является типичным способом индексации и препятствует автоматическому определению информационных зависимостей.

Замечание 2. Здесь мы хотим обратить внимание на дополнительные проблемы, возникающие при переходе от последовательной программы к параллельной. Возникает вопрос, следует ли менять последовательный код (очень большой и уже отлаженный), чтобы получить какие-то дополнительные достоинства в параллельной версии, или следует "подстраиваться" под этот последовательный код. Вопрос далеко не праздный, если учесть, что в предельном случае вместо распараллеливания придется создавать параллельную программу заново, причем по далеко не лучшей спецификации — последовательному коду. С другой стороны, без изменения (как правило, на уровне исходного текста) последовательной программы все равно не обойтись — данная статья показывает, например, насколько разные структуры данных и программы работы с ними используются в последовательной и параллельной реализациях. Последовательная программа и функционально эквивалентная ей параллельная программа — существенно различные по сложности объекты; преодоление этого различия в сложности требует значительных интеллектуальных усилий. Рассматриваемая нами методика учитывает требование минимизации изменений кода исходной последовательной программы.

Таким образом, кроме определения методов распределения базовых структур данных и работы с ними, важным моментом при использовании данной методики для распараллеливания является стремление как можно меньше менять код последовательной программы. Другими словами, преследуется цель определить набор поддерживающих распараллеливание подпрограмм с возможно более простым интерфейсом (либо вообще скрытым от пользователя) и правила их включения в последовательную программу.

Набор (библиотека) поддерживающих подпрограмм реализован таким образом, что параметрами подпрограмм являются только имена переменных — расчетных величин, определенных пользователем. В приведенном выше цикле не все соседние узлы j-го узла могут находиться в том же процессоре, что и узел j (а следовательно, и значения переменной T(grafnode(j,i))); для правильного выполнения этого цикла в параллельной программе надо обеспечить подкачку необходимых значений переменной T. При помощи

набора поддерживающих подпрограмм эта подкачка осуществляется вызовом CALL MKSnode(T) перед рассматриваемым циклом. Все необходимые для работы подпрограммы MKSnode структуры и информация о правилах распределения данных скрыты от пользователя.

Инициализация внутренних структур проводится путем вызова одной из подпрограмм библиотеки перед началом основного цикла расчета.

Таким образом, описываемый способ создания параллельной программы на неструктурированных сетках (способ распараллеливания уже существующей последовательной программы) основан на

— методе распределения базовых структур данных и вычислений;

— разработке поддерживающего набора подпрограмм;

— правилах модификации текста исходной последовательной программы.

Ниже эти элементы методики рассматриваются подробнее.

2.3. Принципы распределения базовых структур данных. Граф зависимостей между узлами сетки, представленный при помощи массива grafnode(nnode, maxnode), распределяется между k процессорами с номерами i=1,. . .,k.

Часть графа зависимостей, распределенная некоторому процессору, будем называть *локальным графом* этого процессора, а весь граф — *полным*, или *глобальным графом*. Такой же смысл будем связывать с терминами *локальный* и *глобальный* и для других структур данных, имеющих распределенные аналоги.

Введем параметр mm, характеризующий максимальный размер локального графа и равный максимально возможному числу узлов, распределенных на любой из процессоров i.

Нумерация узлов в параллельной программе. Как указано выше, все узлы сетки (глобального графа) нумеруются целыми числами от 1 до nnode — исходными номерами. В параллельной программе узлы имеют *глобальную* нумерацию (нумерация узлов глобального графа; в общем случае она не совпадает с исходной нумерацией) и *локальную* нумерацию (нумерация узлов локального графа).

Применяя правило распределения узлов к *исходным* номерам узлов, получим для k процессоров k множеств N_i, i=1,. . .,k, номеров узлов, распределенных на i-й процессор. В каждом из множеств N_i узлы нумеруются *локальными* номерами 1, 2, 3,. . . . Таким образом, каждый узел характеризуется парой (i, n(i)), где i — номер процессора, в котором расположен узел, n(i) — номер узла в локальной нумерации. *Глобальный* номер Nn узла определяется соотношением

$$\mathrm{Nn} = \mathrm{mm} * \mathrm{i} + \mathrm{n(i)}, \tag{1}$$

где i — номер процессора.

Введенные нумерации для узлов удовлетворяют двум важным требованиям: существует однозначное правило получения локального номера по глобальному; переход от глобального номера к локальному осуществляется за одну операцию деления нацело.

Исходные номера и правило распределения узлов по процессорам применяются для построения локальных и глобальных номеров и для получения, таким образом, *начального распределения* узлов по процессорам. Эта процедура является фактически процедурой инициализации распределенных структур для представления узлов в параллельной программе. Исходные номера в параллельной программе не используются.

Замечание 3. *Исходный* номер узла N является номером строки массива grafnode, в которой находится информация об узле с номером N (в глобальной нумерации), а локальный номер n(i) — номером строки массива grafnoded (существующего в каждом процессоре), в которой находится информация об узле с номером n(i) (в локальной нумерации). Структура массива grafnoded описана ниже.

Замечание 4. В разных процессорах локальные номера узлов могут совпадать; это не приводит к неоднозначности, так как узел определяется парой (i, n(i)).

Замечание 5. Ниже именование распределенных структур данных, являющихся аналогами глобальных структур данных, осуществляется за счет добавления символа d к соответствующему имени глобальной структуры, например, глобальный граф узлов grafnode или локальный (распределенный) граф узлов grafnoded.

Распределение ячеек вычислительной сетки является вторичным — ячейки распределяются по процессорам в соответствии с тем, как распределены по процессорам узлы. При этом применяется следующее *правило распределения ячеек*: если три вершины ячейки принадлежат процессору i, то эта ячейка принадлежит процессору i; если две вершины ячейки принадлежат процессору i, а одна вершина принадлежит процессору j, то эта ячейка принадлежит процессору i (необходимые для расчетов значения в узле из процессора j будут пересылаться в процессе вычислений); три вершины одной ячейки не могут принадлежать трем разным процессорам — это неприемлемое распределение узлов.

Нумерация ячеек в параллельной программе производится по правилам, во многом схожим с изложенными выше правилами нумерации узлов в параллельной программе. При этом *глобальный* номер Nc ячейки определяется соотношением

$$\mathrm{Nc} = \mathrm{mmj} * \mathrm{i} + \mathrm{n(i)}, \tag{2}$$

где i — номер процессора и mmj — аналог параметра mm для ячеек.

2.4. Распределенные базовые структуры данных.

2.4.1. Представление локального графа узлов в процессоре. Локальные узлы и их соседство задаются локальным графом, который представляется в виде массива integer grafnoded(mm, maxneigh), где параметр mm равен максимально возможному числу узлов, распределенных на любой процессор, i-я строка массива содержит номера узлов, соседних к i-му локальному узлу, и maxneigh — максимальное число возможных соседей. Кроме того, задан вектор vsnd(mm), i-й элемент которого равен числу соседей i-го узла.

2.4.2. Связь локальных графов узлов из разных процессоров. В массиве grafnoded(mm, maxneigh), представляющем локальный граф узлов в текущем процессоре, имеются ссылки и на номера узлов, размещенных в других процессорах. Соседний узел, который располагается в другом процессоре, задается номером, который является ссылкой на компоненту вектора vneig(mm+1:2*mm). Эта компонента содержит глобальный номер узла Nn, по которому с учетом правила нумерации глобальных и локальных узлов и представления (1) можно определить номер процессора i, в котором расположен этот узел, и его локальный номер n(i) в этом процессоре. Как показывает практика, величина 2*mm, используемая в качестве верхней границы векторов, избыточна и ее можно уменьшить.

Пример 5. Пусть mm = 10 и массив grafnoded для первого процессора имеет вид, представленный на рис. 2.

Все узлы, которые имеют локальные номера, бóльшие чем mm = 10, являются узлами из других процессоров. Их глобальные номера находятся в векторе vneig:

vneig(11)	vneig(12)	vneig(13)	vneig(14)
25	28	31	38

vneig(15)	vneig(16)	...	vneig(20)
32	26		

1	2	4	11	12	5
2	1	11	3	6	9
3	2	4			
4	1	3	5	12	14
5	1	4	8	13	10
6	2	7	12	16	14
7	6	8	13	10	
8	5	7	16	9	
9	2	8	13		
10	5	7	14		

Рис. 2. Массив grafnoded для первого процессора

Это означает, что под номером 11 в grafnoded указан узел vneig(11)=25 (в глобальной нумерации), т.е. это узел из процессора с номером 2 и его локальный номер в этом процессоре равен 5. Под номером 15 указан узел vneig(15)=32 (в глобальной нумерации) из процессора 3, локальный номер этого узла равен 2.

2.4.3. Структуры данных для хранения расчетных переменных в узлах. Массив grafnoded, предназначенный для хранения локального графа, является принципиально важной базовой структурой, обеспечивающей доступ к значениям расчетных величин, используемых в расчете. Будем в дальнейшем обозначать векторы для хранения значений расчетных величин именами R и Z (в программе этих величин обычно больше — скорости, температура, давление и т.п., но метод доступа к значениям всех расчетных переменных одинаков).

Типичный пример обработки значений представлен ниже (цикл по локальным узлам и чтение значения в соседнем узле i):

```
DO J = 1,NNODED
    DO I = 1,VSND(J)
        R(J ) = F(Z(GRAFNODED(J,I)))
    ENDDO
ENDDO
```

Значение grafnoded(J,I) является указателем на значение расчетной переменной Z. Так как не все соседние узлы j-го узла могут находиться в том же процессоре, что и узел j (а следовательно, и значения переменной Z(grafnoded(J,I)) в этих узлах), для правильного выполнения этого цикла в параллельной программе надо обеспечить подкачку значений переменной Z, размещенных в других процессорах. При этом необходимо обеспечить хранение не только “своих” значений (значений в локальных узлах своего процес-

сора), но и “чужих” значений (принимаемых из других процессоров). Для этой цели векторы значений переменных организованы следующим образом:

“свои” узловые значения					“чужие” узловые значения				
1	2	. . .	. . .	mm	mm+1	mm+2	. . .	. . .	2*mm

Первые mm компонент вектора отводятся для хранения значений, которые вычисляются в данном процессоре, в последующих компонентах размещаются значения из других процессоров. Эти значения требуются в качестве аргументов в текущем процессоре.

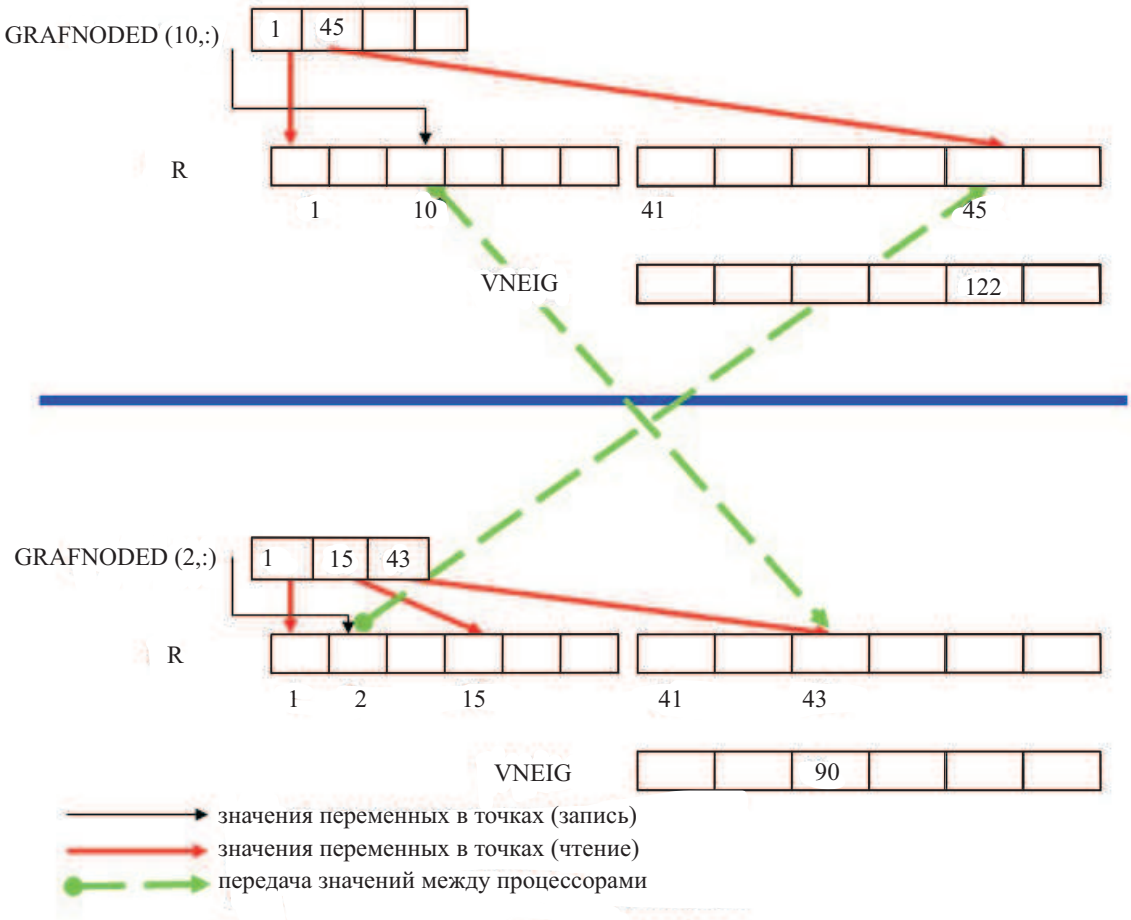

Рис. 3. Пример структур данных для хранения значений расчетной переменной R в узлах

Введенные выше структуры иллюстрируются на рис. 3. Рассматривается взаимодействие двух процессоров (с номером 2 — верхняя часть рисунка и с номером 3 — нижняя часть рисунка). Параметр mm равен 40. Для каждого процессора представлено по одной строке из матрицы grafnoded. Для второго процессора указана строка, соответствующая узлу с номером 10. Вычисление в этом узле зависит от значений в узлах с номером 1 и с номером 45. Поскольку номер $1 < \mathrm{mm}$, то требуемые значения вычисляются в этом же процессоре. Значения в узле с номером 45 вычисляются в другом процессоре (это следует из того, что $45 > \mathrm{mm}$). Следовательно, прежде чем использовать значение в этом узле, необходимо провести подкачку требуемого значения в 45-ю компоненту вектора R. Для этого надо определить:

1) в каком процессоре проводятся необходимые вычисления,

2) номер компоненты в векторе (т.е. локальный номер).

Соответствующая компонента вектора $\mathrm{vneig}(45) = 122$. Это глобальный номер; из его представления (1) однозначно определяется номер процессора, равный 3, и локальный номер узла в этом процессоре, равный 2.

Для третьего процессора указана строка, соответствующая узлу с номером 2. Вычисление в этом узле зависит от значений в узлах с номерами 1, 15 и с номером 43. Поскольку номера $1 < \mathrm{mm}$ и $15 < \mathrm{mm}$, то

требуемые значения вычисляются в этом же процессоре. Значения в узле с номером 43 вычисляются в другом процессоре (это следует из того, что 43 > mm). Следовательно, прежде чем использовать значение в этом узле, необходимо выполнить подкачку требуемого значения в 43-ю компоненту вектора R. Для этого надо определить

1) в каком процессоре проводятся необходимые вычисления,

2) номер компоненты в векторе (т.е. локальный номер).

Соответствующая компонента вектора vneig(43) = 90. Это глобальный номер, из его представления (1) однозначно определяется номер процессора, который равен 2, и локальный номер узла в этом процессоре, который равен 10.

2.4.4. Распределенные структуры данных для представления ячеек. Локальные ячейки и их соседство задаются двумя массивами, представляющими собой локальный граф ячеек. Соседство узлов и ячеек представляется в виде массива integer grafcelld(mm, maxnode), где параметр mm равен максимально возможному числу узлов, распределенных на любой процессор, maxnode — максимальное число возможных соседей и i-я строка массива содержит номера ячеек, в которые входит i-й локальный узел. Кроме того, задан массив vscd(mm), i-й элемент которого равен числу ячеек, в которые входит i-й узел.

Локальные ячейки представляются массивом integer celld(mmj,3), где параметр mmj равен максимально возможному числу ячеек, распределенных на любой процессор (в общем случае значение этого параметра отличается от mm), и i-я строка массива содержит номера трех узлов, составляющих i-ю локальную ячейку.

Информация о связи ячеек из различных процессоров задается по аналогии с информацией о связи узлов из различных процессоров. В массиве grafcelld, представляющем информацию о вхождении локальных узлов в ячейки, могут указываться ссылки на номера ячеек, размещенных в других процессорах, если ячейка является “разорванной” (одна вершина — в одном процессоре, две вершины в — другом). Ячейка, которая располагается в другом процессоре, задается номером, являющимся ссылкой на компоненту вектора vneigc(mmj+1:2*mmj) (аналогично вектору vneig для массива grafnoded). Компонента вектора vneigc содержит глобальный номер ячейки Nc, по которому, учитывая правило нумерации глобальных и локальных ячеек и представление (2), можно определить номер процессора i, в котором расположена эта ячейка, и ее локальный номер n(i) в этом процессоре.

Информация о локальных ячейках, представляемая массивом celld, может содержать ссылки на номера узлов, размещенных в других процессорах. Как и в случае массива grafnoded для хранения локальных узлов, соседний узел, который располагается в другом процессоре, задается номером, являющимся ссылкой на компоненту вектора vneig(mm+1:2*mm).

Организация структур данных для хранения расчетных переменных в ячейках аналогична организации структур данных для хранения расчетных переменных в узлах. Отличие состоит в размерах векторов для хранения значений в ячейках — они имеют длину 2*mmj, а не 2*mm, как в случае векторов для хранения значений в узлах. Например, распределенный вектор V для хранения скоростей в локальных ячейках должен быть описан как real V(2*mmj).

2.4.5. Пересылка данных между процессорами. Карты приема и отправки. Обмен данными между процессорами осуществляется на основании *карт отправки и приема*. Для узлов и для ячеек строятся различные карты отправки и приема. Карта приема должна содержать *количество* принимаемых значений от других процессоров. Кроме того, данная карта должна содержать информацию о том, в *какие компоненты* расчетных векторов должны быть размещены требуемые значения, получаемые от других процессоров. Карта отправки должна содержать *количество* отправляемых к другим процессорам значений и *локальные номера* компонент отправляемых значений. В общем случае число отправляемых и принимаемых значений в одном процессоре могут не совпадать.

Замечание 6. Очевидно, что карта приема процессора с номером q от процессора с номером p должна быть согласована с картой отправки из процессора с номером p в процессор с номером q. Согласованы должны быть как число передаваемых значений, так и порядок этих значений в картах.

Рассмотрим метод организации и построения карт приема и отправки для узлов (для ячеек все делается аналогично). Карта приема для узлов представляется массивом

integer howrec(iproc),

где iproc — параметр, равный числу процессоров, и i-я компонента этого вектора равна числу значений, которые будут приниматься от i-го процессора, и массивом

integer recind(iproc, mm),

где i-я строка массива содержит индексы (значения которых лежат в диапазоне от mm+1 до 2*mm — области принимаемых значений), определяющие номера компонент расчетных величин, в которые следует разместить значения, принимаемые от i-го процессора.

При построении *карты приема* для узлов используется вектор vneig, в котором хранятся номера узлов в глобальной нумерации, не принадлежащих данному процессору, но значения в которых требуются при вычислениях в данном процессоре. По этим глобальным номерам с использованием правила (1) перехода от глобальной нумерации узлов к локальной определяется, из какого процессора требуется принять значение в локальном узле, и индекс этого узла в массиве vneig (для занесения в массив recind), а также подсчитывается, сколько узлов надо принять из некоторого процессора (для занесения в массив howrec).

Пусть mm=20 и в процессоре с номером 1 фрагмент массива vneig имеет вид, приведенный ниже; тогда в массивах карты приема по этой информации будут определены следующие значения:

Вектор vneig

...	...
25	21
26	23
27	26
28	31
29	42
...	...

Массив recind

1	...	...	...	...
2	25	26	27	
3	28			
4	29			
5				
6				
...	...	...	...	...

Вектор howrec

1	0
2	3
3	1
4	1
5	0
6	0
...	...

Во второй строке массива recind размещены три значения — 25, 26, 27 (число этих значений задается второй компонентой вектора howrec: howrec(2) = 3). Это означает, что в первом процессоре требуются три значения от второго процессора и эти значения должны размещаться в компонентах с номерами 25, 26, 27. От третьего процессора требуется одно значение (howrec(3) = 1), и оно должно быть размещено в 28-й компоненте расчетного вектора. И наконец, от четвертого процессора также требуется одно значение, которое будет размещено в 29-й компоненте расчетного вектора.

Таким образом, мы определили для каждого процессора число требуемых значений и место их размещения.

Для построения карты отправки применяется следующий алгоритм, использующий рабочий массив:

integer recinda(iproc,mm),

где i-я строка массива содержит номера узлов (в локальной нумерации), требуемых в текущем процессоре от i-го процессора. Фактически, i-я строка массива задает компоненты расчетных векторов, которые необходимо отправить из процессора с номером i в текущий процессор.

1	...	...	...	...
2	1	3	6	
3	1			
4	2			
5				
6				
...	...	...	...	...

Рис. 4. Массив recinda

Для приведенного выше примера массив recinda имеет вид, представленный на рис. 4. Это означает, что в процессоре 1 требуются значения из трех узлов, находящихся в процессоре 2 и имеющих локальные номера 1, 3, 6, а также значения из узла с локальным номером, равным 1, находящегося в третьем процессоре, и значение из узла с локальным номером 2, находящегося в четвертом процессоре.

Карта отправки для узлов представляется вектором

integer howsend(iproc),

где i-я компонента равна числу значений, которые надо отправить в i-й процессор, и массивом

integer sendind(iproc,mm),

где i-я строка содержит номера узлов в локальной нумерации, значения которых требуется отправлять в i-й процессор.

Для процессоров с номерами p и q справедливо следующее утверждение: если в процессоре q выполнено howrec(p) $\neq 0$, то в процессоре p выполнено howrec(q) $\neq 0$. Это свойство следует из понятия связи между узлами: если узел x связан с узлом y, то узел y связан с узлом x. Следует заметить, что в общем случае эти значения не совпадают.

Эти факты позволяют получить в каждом процессоре q значения вектора howsen(p) по следующей схеме:

Процессор p		Процессор q
howrec(q) $\neq 0$,		howrec(p) $\neq 0$
howrec(q)	$\Longrightarrow$	howsen(p)
howsen(q)	$\Longleftarrow$	howrec(p)

После этого пересылаются значения требуемых номеров из recinda, и они становятся значениями соответствующих строк sendind.

Построение карты приема для ячеек аналогично построению карты приема для узлов, при этом вместо массива vneig используется массив vneigc, а вместо правила (1) нумерации для узлов — правило (2) нумерации для ячеек.

Обмен соответствующими значениями расчетных переменных проводится на основании карт приема и отправки.

2.5. Поддерживающий набор подпрограмм. Подпрограммы, входящие в набор, поддерживающий рассматриваемую методику распараллеливания, можно разделить на подпрограммы, *обеспечивающие обмен данными* (значениями переменных) между процессорами, и *подпрограммы инициализации*, обеспечивающие первоначальное заполнение структур данных, поддерживающих параллельное исполнение программы. При этом все рассматриваемые подпрограммы поддержки являются распределенными и выполняются параллельно в каждом из процессоров.

Подпрограммы инициализации должны быть вызваны до начала основной расчетной части программы.

Основные *подпрограммы инициализации* выполняют следующие подготовительные действия в каждом из процессоров структур по узлам (ячейкам):

— заполнение значениями распределенных массивов grafnoded(mm,neig) и vneig(mm+1:2*mm),

— тоже самое для распределенных массивов grafcelld(mm,neig), celld(mmj,3) и vneigc(mmj+1:2*mmj),

— подготовку в каждом из процессоров карт приема и отправки по ячейкам и узлам.

Подпрограммы, обеспечивающие обмен данными, в качестве параметров требуют задания имен подкачиваемых переменных. Подпрограммы отличаются видом (узловые или ячеечные) подкачиваемых одновременно значений. Например, CALL MkNODE(X) — подкачка одной узловой переменной X или CALL MkCELL(X,Y) — подкачка двух ячеечных переменных X и Y.

2.6. Модификация текста последовательной программы. Одной из главных целей работы по распараллеливанию рассматриваемой задачи было стремление к минимальному изменению исходного текста последовательной программы. Некоторые типы модификаций текста приведены ниже.

1. Необходимо написать стартовую часть параллельной программы, которая осуществляет ввод исходных данных из файлов, рассылку соответствующих частей обрабатываемых массивов по процессорам и прием результатов расчета в конце исполнения программы.

Что касается применения данной методики с “минимальными” дополнительными усилиями, то можно разработать шаблон стартовой программы, в фиксированные места которого пользователь вносит информацию в терминах своей исходной программы.

2. Модификация собственно текста программы:

а) привязка имен переменных, используемых в исходной программе, к именам структур, используемых в наборе поддерживающих программ для обозначения тех же, с содержательной точки зрения, объектов (узлов, ячеек и т.п.); эта проблема может быть решена с помощью объявления эквивалентности имен, если данные имеют “близкие” структуры;

б) должны быть модифицированы описания массивов расчетных переменных;

в) модификация текста, связанная с включением вызовов подпрограмм подкачки значений расчетных переменных.

Основная проблема — определение мест в последовательной программе, где необходимо вставить вызов подпрограммы подкачки. Анализ рассматриваемой программы позволяет описать случаи, когда необходима модификация текста, например, при использовании в правой части оператора присваивания “соседних” элементов (узлов или ячеек); формальным признаком является косвенная индексация массивов

расчетных переменных по массиву grafnoded(grafcelld) или celld.

С другой стороны, такие операции могут быть избыточны: например, в том случае, когда ранее был вставлен оператор подкачки значения и до текущего места это значение не изменялось. Избыточность таких операторов, конечно же, приводит к снижению эффективности вычислений, но зато гарантирует правильность.

Пример 6.

```
C Необходимо вставить подкачку значений FI, V в ячейках
      CALL MkCELL(FI,V)
C поскольку в цикле по ячейкам
      DO J = 1,NNODED
         S1 = 0.D0
         S2 = 0.D0
         DO K = 1,VSCD(J)
C используются значения FI, V в соседних ячейках
            I = GRAFCELLD(J,K)
            S1 = S1+FI(I)*V(I)
            S2 = S2+V(I)
         ENDDO
         FJR3(J) = S1/S2
      ENDDO
```

Пример 7.

```
C Необходимо вставить подкачку значений FIR1, FIZ1 в узлах
      CALL MkNODE (FIR1,FIZ1)
C поскольку в цикле по ячейкам
      DO I = 1,NCELLD
         FI(I) = 0.0D0
         DO K = 1,3
C используются значения FIR1, FIZ1 в узлах ячейки
            J = CELLD(I,K)
            FI(I) = FI(I)+ANR(I,K)*FIR1(J)+ANZ(I,K)*FIZ1(J)
         ENDDO
         FI(I) = FI(I)/(3.0D0*ZRV(I))
      ENDDO
```

Следующий пример на первый взгляд кажется достаточно простым. Его сложность определяется тем, что в цикле последовательно просматриваются ячейки, а присваивание проводится в узловую переменную. Фактически в узловой переменной FJR2 вычисляется сумма. При этом каждый элемент суммирования (выражение (ANR(I,K)*SK(I))*FI1(I) +ANZ(I,K)* FI2(I)) зависит от значений ячеечных переменных. Для узла с номером J рассматриваются все ячейки, в которые входит этот узел. Каждый процессор просматривает ячейки, принадлежащие текущему процессору. Однако ячейка может иметь узел, принадлежащий другому процессору, поэтому в данном случае необходима более существенная перестройка этого фрагмента вычислений, так как *параллельная программа должна проводить вычисление* (*запись*) *только в узлах и ячейках, принадлежащих текущему процессору.*

Пример 8.

```
C цикл по ячейкам
      DO I = 1,NCELLD
         DO K = 1,3
            J = CELLD(I,K)
C J — номер узла
            FJR2(J) = FJR2(J)+(ANR(I,K)*SK(I))*FI1(I) +ANZ(I,K)*FI2(I)
         ENDDO
      ENDDO
```

Можно разбить вычисление переменной FJR2 на два этапа. На первом этапе вычисляется часть выражения, зависящая от значений ячеечных переменных (ANR, SK, FI1, ANZ и FI2). Присваивание проводится в рабочий массив, определенный в ячейках.

```
      DO I = 1,NCELLD
         DO K = 1,3
```

```
            FJR2rab(i,k) = (ANR(I,K)-SK(I))*FI1(I)+ANZ(I,K)*FI2(I)
        ENDDO
    ENDDO
```

Конечно, после этого вычисления необходимо провести подкачку вычисленных значений:

```
CALL MkNODE (FJR2rab).
```

Наконец, выполняем операцию суммирования всех вычисленных значений, относящихся к рассматриваемому узлу:

```
      DO I = 1,NNODED
          DO j = 1,VSC(i)
              ij = GRAFCELLD(i,j)
              DO k = 1,3
                  IF(CELLD(ij,k).eq.i)THEN
                      FJR2(i) = FJR2(i)+FJR2rab(ij,k)
                      GOTO 100
                  ENDIF
              ENDDO
100       CONTINUE
          ENDDO
      ENDDO
```

Подпрограммы вычисления редукционных функций включены в набор поддерживающих подпрограмм, чтобы упростить правила их вызова и не использовать термины библиотеки MPI. Интерфейс вызова такой же, как и для подпрограмм подкачки: аргументом является имя переменной, редукционную операцию над которой надо выполнить, например, CALL Summa(X) — суммирование значений X. Формальное определение места в последовательной программе, где необходимо вставить вызов редукционной подпрограммы (сумма, максимум и т.п.), в общем случае затруднительно, хотя часто фрагмент программы, реализующий редукционную функцию, и имя переменной-результата определить несложно.

Пример 9.

```
C Вычисление суммы SL2 и максимума S2
    SL2 = 0.0
    S2 = 1.D-10
    DO J = 1,NNODED
        R1 = FJR2(J)-FJR3(J)
        SL2 = SL2+R1*R1*RV(J)/1.D5
        FPR = DABS(FJR(J))
        FJR6(J) = FJR1(J)
        FJR1(J) = R1
        SS2 = DABS(R1/(FPR+1.D0))
        S2 = DMAX1(S2,SS2)
    ENDDO
C требует вставки после вычисления вызовов редукционных подпрограмм
    CALL SUMMA(SL2)
    CALL MAXIMA(S2)
```

СПИСОК ЛИТЕРАТУРЫ

1. *Арделян Н.В., Космачевский К.В.* Неявный свободно-лагранжев метод расчета двумерных магнитогазодинамических течений // Математическое моделирование. М.: Изд-во Моск. ун-та, 1993. 25–44.
2. METIS — family of multilevel partitioning algorithms (http://www-users.cs.umn.edu/∼karypis/metis/).
3. *Якобовский М.В.* Вычислительная среда для моделирования задач механики сплошной среды на высокопроизводительных системах: Автореферат дисс. ... уч. степ. д.ф.-м.н. М., 2006.
4. *Андрианов А.Н., Жохова А.В., Четверушкин Б.Н.* Использование параллельных алгоритмов для расчетов газодинамических течений на нерегулярных сетках // Прикладная математика и информатика. М.: Диалог-МГУ, 2000. 68–76.
5. *Андрианов А.Н.* Система Норма. Разработка, реализация и использование для решения задач математической физики на параллельных ЭВМ: Дисс. ... уч. степ. д.ф.-м.н. М., 2001.

Поступила в редакцию
08.02.07