ИПМ им.М.В.Келдыша РАН • Электронная библиотека

Препринты ИПМ • Препринт № 132 за 2019 г.

ISSN 2071-2898 (Print)
ISSN 2071-2901 (Online)

Андрианов А.Н., Баранова Т.П., Бугеря А.Б., Гладкова Е.Н., Ефимкин К.Н.

Язык НОРМА

Рекомендуемая форма библиографической ссылки: Язык НОРМА / А.Н.Андрианов [и др.] // Препринты ИПМ им. М.В.Келдыша. 2019. № 132. 48 с. http://doi.org/10.20948/prepr-2019-132
URL: http://library.keldysh.ru/preprint.asp?id=2019-132

О р д е н а Л е н и н а
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
имени М.В.Келдыша
Р о с с и й с к о й а к а д е м и и н а у к

А.Н. Андрианов, Т.П. Баранова,
А.Б. Бугеря, Е.Н. Гладкова, К.Н. Ефимкин

Язык НОРМА

Москва — 2019

Андрианов А.Н., Баранова Т.П., Бугеря А.Б., Гладкова Е.Н., Ефимкин К.Н.

Язык НОРМА

Непроцедурный (декларативный) язык НОРМА предназначен для создания программ вычислительного характера (scientific computing) для параллельных компьютеров различного типа. Язык НОРМА был разработан в ИПМ им. М.В. Келдыша РАН, создана система параллельного программирования НОРМА, язык применялся для решения задач математической физики. Первое строгое описание языка НОРМА было выполнено в 1995 году. С тех пор возможности языка были расширены, в язык были введены новые конструкции, как поддерживающие удобство программирования, так и расширяющие выразительные возможности языка. В настоящем документе описан полный актуальный синтаксис языка НОРМА, приведены семантические комментарии и разъяснения к синтаксическим конструкциям, а также многочисленные примеры программ на языке НОРМА.

Ключевые слова: автоматизация программирования, параллельное программирование, непроцедурный язык, декларативный язык, язык НОРМА

Andrianov Alexander Nikolaevich, Baranova Tat'yana Petrovna, Bugerya Alexander Borisovich, Gladkova Ekaterina Nikolaevna, Efimkin Kirill Nikolaevich

The NORMA language

The non-procedural (declarative) NORMA language is designed to create scientific computing programs for parallel computers of various types. The NORMA language was developed in Keldysh Institute of Applied Mathematics, NORMA parallel programming system was created; the language was used to solve problems of mathematical physics. The first strict description of the NORMA language was made in 1995. Since then, language capabilities have been expanded, and new designs have been introduced to support both the convenience of programming and the expressiveness of the language. This paper describes the full topical syntax of the NORMA language, semantic comments and explanations for syntax, as well as numerous examples of the NORMA language programs.

Key words: programming automation, parallel programming, non-procedural language, declarative language, NORMA language

Оглавление

Введение 5
1. Область применения и основные свойства языка НОРМА 5
2. Нотация синтаксиса 6
3. Основные элементы языка 7
3.1. Лексические правила 7
3.1.1.Основные символы 7
3.1.2. Комментарии 8
3.1.3. Лексемы 8
3.1.4. Идентификаторы 8
3.1.5. Ключевые слова 8
3.1.6. Константы 9
3.1.7. Знаки операций 10
3.1.8. Разделители 10
3.2. Имена 10
3.3. Базовые типы данных 10
3.4. Выражения 11
3.4.1. Арифметические выражения 11
3.4.2. Условные выражения 13
4. Структура программы 14
5. Конструкции языка НОРМА 17
5.1. Описания 17
5.1.1. Именованные константы 17
5.1.2. Параметры области 18
5.1.3. Области 18
5.1.4. Индексы областей 24
5.1.5. Арифметические величины 25
5.1.6. Входные и выходные величины 26
5.1.7. Внешние имена 29
5.1.8. Индексы распределения 30
5.1.9. Область действия описаний 30
5.2. Операторы в языке НОРМА 31
5.2.1. Скалярный оператор 31
5.2.2. Оператор ASSUME 32
5.2.3. Оператор вызова раздела 33
5.3 Функции в языке НОРМА 35
5.3.1. Стандартные арифметические функции 36
5.3.2. Функции редукции 36

5.3.3. Внешние функции пользователя 38
5.4. Итерация 39
5.5. Интерфейс с программами, написанными на целевом языке программирования 42
5.6. Режим последовательного вычисления 43
5.7. Директивы компиляции 44
5.7.1. Директива INCLUDE 44
5.7.2. Директивы условной компиляции 45
Заключение 46
Литература 47

Введение

Одним из подходов к разработке параллельных программ для вычислительных задач является подход с использованием непроцедурных (декларативных) языков. Этот подход развивается в нашей стране и за рубежом достаточно давно.

Идеи, позволяющие автоматически строить программу по спецификации задачи, были сформулированы И.Б. Задыхайло в пионерской работе [1] ещё в 1963 году. Предлагалось в качестве языка программирования для описания решения задачи взять сложившуюся в данной предметной области математическую нотацию и использовать её для автоматического, при помощи компилятора, построения исполняемой программы.

На основе этих идей в ИПМ им. М.В. Келдыша РАН был разработан непроцедурный язык НОРМА [2-4] и создана система параллельного программирования НОРМА [5-6], предназначенные для разработки параллельных программ для расчетных задач математической физики. Оказалось, что для программы на языке НОРМА можно, при определенных условиях, автоматически получить исполняемую параллельную программу, учитывая функцию оптимизации, которая может, в частности, учитывать модель параллелизма и особенности архитектуры компьютера.

Для языка НОРМА определены и теоретически обоснованы методы и алгоритмы распараллеливания, условия и ограничения, при которых разрешима задача распараллеливания [7-9].

Язык НОРМА и система программирования НОРМА успешно использовались для решения задач математической физики [10-15].

Первое строгое описание языка НОРМА было приведено в [16]. К настоящему времени в языке НОРМА были введены новые конструкции, расширяющие его возможности. В данном документе приведено полное описание текущей версии языка.

1. Область применения и основные свойства языка НОРМА

Язык НОРМА является непроцедурным языком, и изначально был ориентирован на решение задач математической физики разностными методами. Однако, как показала практика, язык НОРМА может использоваться и для решения более широкого класса вычислительных задач.

В программе на языке НОРМА не требуется информация о порядке счета, способах организации вычислительных (параллельных) процессов. Порядок предложений языка может быть произвольным, а информационные взаимосвязи выявляются компилятором и учитываются им при организации вычислительного процесса. Известно, что в достаточно общих постановках

решение задачи синтеза выходной параллельной программы приводит к значительным математическим трудностям – она может оказаться NP-полной либо вообще неразрешимой. Язык НОРМА обладает рядом свойств (ограничений), которые позволяют решить эту проблему. Эти свойства можно, неформально, описать следующим образом:

- Декларативность. Программа (и каждый ее оператор) описывает запрос на вычисление. Каким образом этот запрос реализуется, не указывается. Нет понятий памяти, управления (переходов, циклов и тому подобное).
- Однократное присваивание – величины могут принимать значения только один раз, переприсваивание значений невозможно. Язык НОРМА принадлежит к классу SAL-языков (Single Assignment Language). В языке нет понятия глобальных переменных, но есть глобальные декларации.
- Ограничение на вид индексных выражений у величин – индексные выражения имеют вид $(A*i + C_1)/B + C_2$, где i – индексная переменная, A и B – натуральные константы, а C_1 и C_2 – целые константы.
- Области имеют границы, заданные константными выражениями, значения которых известны в момент трансляции. Области также могут иметь переменную верхнюю границу, определяемую целочисленными формальными параметрами раздела.

Язык НОРМА имеет возможности, необходимые для написания сложных расчетных программ достаточно широкого класса, в частности интерфейсы с языками Фортран и Си, позволяющие использовать фрагменты программ на этих языках.

2. Нотация синтаксиса

В нотации синтаксиса, используемой в данном описании, применяется расширенная форма Бэкуса-Наура.

Обозначения $\{\mathbf{A}\}^*$, $\{\mathbf{A}\}^+$, $\{\mathbf{A_1},\ldots,\mathbf{A_n}\}$, $[\mathbf{A}]$ означают

$\{\mathbf{A}\}^*$	:: =	$\varnothing \mid A \mid A\,A$ ...
$\{\mathbf{A}\}^+$	:: =	$\mathbf{A} \mid \mathbf{A}\,\mathbf{A}$ ...
$\{A_1,\ldots,A_n\}$	:: =	$\mathbf{A_1} \mid \ldots \mid \mathbf{A_n}$
$[\mathbf{A}]$	:: =	$\varnothing \mid \mathbf{A}$

где A – некоторый объект языка,

$\varnothing$ – пусто,

$\mid$ – выбор одной из альтернатив,

... – и так далее.

В дальнейшем, при определении правил языка, синтаксические понятия набираются курсивом, а слова и литеры, воспринимаемые буквально, – прямым шрифтом. Альтернативные конструкции перечисляются, как правило, в столбик, каждая альтернатива на отдельной строке. Иногда используются

частично подчёркнутые обозначения синтаксических конструкций, например *name-set*. Синтаксически это обозначение идентично обозначению *name*, а подчеркнутая часть конструкции несет дополнительную семантическую информацию.

Обозначение list-*element* заменяет непустой список *элементов*, перечисленных через запятую:

list-element :
 element { **,** element }*

В каждом конкретном случае определение *элемента* приводится.

Обычно в данном документе сначала приводятся синтаксические правила, а затем комментарии к ним, семантические разъяснения, примеры и т.п.

3. Основные элементы языка

3.1. Лексические правила

Первичными элементами языка, из которых строятся все его конструкции, являются символы. Набор *основных символов* фиксирован и используется для образования любых конструкций языка. Набор *дополнительных символов* не фиксирован, определяется составом оборудования компьютера и используется для образования символьных констант, представления данных на носителях информации.

3.1.1. Основные символы

principal-symbol:
 letter
 digit
 special-symbol
letter:
 { **A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z** }
 { **a,b,c,d,e,f,g,h,i,j k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z**}
digit :
 { **0,1,2,3,4,5,6,7,8,9** }
special-symbol:
 special-symbol-not-apostrophe
 apostrophe
 double-quotes
special-symbol-not-apostrophe:
 { space ,**=**,**>**,**<**,**+**,**-**,*****,**/**,**(**,**)**,**[**,**]**,**!**,**?**,**,**,**.**,**#** , @ }
apostrophe:
 '
double-quotes:
 "

space:
Пробел не имеет графического представления.

3.1.2. Комментарии

Строка, начинающаяся со знака ! или со знаков //, или часть строки, следующая за этими знаками, является комментарием. Последовательность символов в интервале между знаками /* ... */ также является комментарием. В комментарии не могут быть включены другие комментарии. Комментарий – это не лексическая единица, и он рассматривается компилятором как пробел. Например, следующая строка:

VARIABLE u,uu/* переменные*/DEFINED ON DOM DOUBLE.

трактуется как:

VARIABLE u, uu DEFINED ON DOM DOUBLE.

3.1.3. Лексемы

Существует пять классов лексем*: идентификаторы, ключевые слова, константы, знаки операций, разделители*. Пробелы, символы перевода строки, комментарии рассматриваются как разделители лексем и не влияют на семантику программы.

3.1.4. Идентификаторы

identificator:
letter { { digit, letter } }*

Идентификатором является ряд букв и цифр, начинающийся с буквы. Строчные и прописные буквы различаются. Идентификаторы могут иметь произвольную длину.

3.1.5. Ключевые слова

В языке используются следующие ключевые слова:

MAIN PART	**VARIABLE**	**ITERATION**
PART	**DEFINED ON**	**BOUNDARY**
FUNCTION	**REAL**	**END BOUNDARY**
BEGIN	**INTEGER**	**INITIAL**
END PART	**DOUBLE**	**END INITIAL**
RESULT	**DOMAIN PARAMETERS**	**EXIT WHEN**
EXTERNAL FUNCTION	**INPUT**	**END ITERATION**
EXTERNAL PART	**OUTPUT**	**FOR**
DISTRIBUTION INDEX	**ON**	**ASSUME**
INDEX	**CONSTANT**	**COMPUTE**
CUDA	**PLAIN**	

3.1.6. Константы

Константы (примитивные типы), включаемые непосредственно в текст программы, задаются литералами.

constant:
 arithm-constant
 string
arithm-constant:
 int-constant
 real-constant
 double-constant
int-constant:
 { digit }$^{+}$
real-constant:
 body [**E** [{ +,- }] power]
double-constant:
 body [**D** [{ +,- }] power]
body:
 int-constant **.** int-constant
power:
 int-constant
string:
 ' { { symbol-not-apostrophe, double-quotes } }$^{+}$ '
symbol-not-apostrophe:
 letter
 digit
 special-symbol-not-apostrophe
 supplementary-symbol

Правила записи констант близки к правилам записи констант в языке Фортран. Тип и значение константы определяются способом ее написания (изображения).

Примеры записи арифметических констант:

101 999 0.1 1.0
10.5E-6 1.0E+7 1.0E7 0.1D-8 15.333D4

Следующие записи не являются арифметическими константами:

.1 1. 1E-1 1.E5

Примеры записи строковых констант:

'Hello, world!'
'Об''ем тела = '

Примеры использования строковых констант приведены в описании входных и выходных величин в п. 5.1.6.

3.1.7. Знаки операций

operation:
{ rel-operation, log-operation, arithm-operation }
rel-operation:
{ **=,>,<,<=,>=,<>,><** }
rel-except-not-equal-operation:
{ **=,>,<,>=,<=** }
log-operation:
{ **AND, OR, NOT** }
arithm-operation:
{ **+,-,*,/,**** }

Символы **>=** и **<=** используются для задания отношений ≥ и ≤ соответственно, символы **<>** и **><** – для задания отношения ≠ .

3.1.8. Разделители

delimiter:
{ space,**/**,**(**,**)**,**[**,**]**,**!**,**?**,**'**,**,**,**.**,**#** }

3.2. Имена

name:
identificator

Имя – это идентификатор, имеющий определенный смысл с точки зрения программиста и обозначающий определенный объект программы, например, величину, область, раздел программы. Не может быть двух определений объектов, имеющих одинаковое имя.

Для каждой программной единицы (раздела или функции) справедливо правило локализации имен: имена, описанные в программной единице, локализованы в этом разделе или этой функции. Понятие глобальных переменных в языке отсутствует, но имеются глобальные описания.

В данном описании языка частично подчеркнутые синтаксические обозначения, например *name-set*, синтаксически эквивалентны обозначению *name*, а подчеркнутая часть конструкции несет дополнительную семантическую информацию.

Обозначение list-*name* заменяет непустой список *name*, перечисленных через запятую.

3.3. Базовые типы данных

В языке определены следующие базовые типы данных:
integer, real, double, logical, string

Элемент данных типа *целый* принимает значения из множества целых чисел. Над элементами целого типа определены *арифметические операции* и *операции отношения.*

Элемент данных типа *вещественный* принимает значения из множества вещественных чисел. Над элементами вещественного типа определены *арифметические операции и операции отношения.*

Элемент данных типа *двойной* принимает значения из множества вещественных чисел и позволяет использовать числа с удвоенной, по сравнению с типом *вещественный,* точностью. Над элементами двойного типа определены а*рифметические операции* и *операции отношения.*

В языке могут быть определены переменные типа *целый, вещественный, двойной.*

Элемент данных типа *логический* принимает значения "истина" или "ложь". Этот тип данных используется при определении различных условий. Над элементами логического типа определены *логические операции.* Переменные логического типа в языке не определены.

Элемент данных типа *строка* представляет собой цепочку символов из множества *основных символов и дополнительных символов.* В языке НОРМА допустимы только константы типа *строка,* переменные и операции над этим типом не определены.

3.4. Выражения

3.4.1. Арифметические выражения

arithm-expression:
 [{ +, - }] term { arithm-operation term }*
term:
 arihm-constant
 call-function
 name-index
 name-domain-parameter
 name-scalar
 variable-on-domain
 (arithm-expression**)**
variable-on-domain:
 name-variable-on-domain [**[** list-index-expression **]**]
const-expression:
 [{ +,- }] const-expression-without-sign
const-expression-without-sign:
 integer-term { arithm-operation integer-term }*
integer-term:
 int-constant
 name-domain-parameter
 (const-expression **)**
index-expression:
 name-index [[{ +,- }] const-expression-without-sign]

name-index = const-expression-without-sign
name-index = name-index [[{ +,- }] const-expression-without-sign]
name-index = arithm-index-expression
arithm-index-expression:
a-expression [/ b-expression] [{ +,- } c-expression]
a-expression:
(a-no-brackets-expression **)**
a-no-brackets-expression
a-no-brackets-expression:
b-expression ***** name-index [{ +,- } c-expression]
b-expression:
const-expression-without-sign
c-expression:
const-expression-without-sign

Результатом вычисления индексного выражения index expression у индекса i в фиксированной точке T(i) области (см. п. 5.1.1) является значение индекса i' точки T'(i') области, которое является вычисленным значением constant expression (или arithm-index-expression, если задано такое индексное выражение), с учетом индексного смещения, если оно задано. Например, для точки T с координатами i=9, j=15, k=3, m=37 список индексных выражений для переменной X:

X[i-1, j, k=7, m=i+4]

дает значение точки индексного пространства T' с координатами i=8, j=15, k=7, m=13.

У величин могут быть индексные выражения, имеющие вид: $(A*i + C_1)/B + C_2$, (см. выше определение arithm-index-expression), где i – индекс, A, B, C_1 и C_2 – целые константы. Если при вычислении такого выражения получается дробное значение, то берется только целая часть, а дробная часть отбрасывается. Пример использования индексных выражений такого вида для переменной **FULL**:

FULL[i = i*2+1, j = (j+1)/2]

Порядок выполнения операций при вычислении arithmetical expression следующий:

1) вычисление индексных выражений;
2) определение аргументов (фактических параметров) функций, в том числе определение областей действия редукционных функций (см. п. 5.2.3);
3) выполнение операций внутри скобок **(** и **)**; если таких скобок одного уровня вложенности несколько, то они просматриваются слева направо;
4) выполнение операций возведения в степень **; если таких операций одного уровня несколько, то справа налево;

5) выполнение операций умножения * и деления /; если таких операций одного уровня несколько, то слева направо;
6) выполнение операций сложения и вычитания + и -; если таких операций одного уровня несколько, то слева направо.

Тип результата арифметического выражения определяется типами результатов операций, выполняемых при вычислении арифметического выражения с учетом приведенного выше порядка выполнения операций. Тип результата операции:

1) целый, если оба операнда типа целый;
2) двойной, если хотя бы один операнд типа двойной;
3) вещественный иначе.

Constant expression является частным случаем arithmetical expression.

Примеры арифметических выражений:

AL12[J = 2*M-1] + STEPLP * time
Q +1.0D5/((R[Q-1]-R) * (R[Q-1] - R[Q+1]))
CN * TRAP(STEP, 2*MV+1, MMR*ON*POLL)
Var[i=L * 2 + 1, j=(j+1)/2, k=(2 * k+2)/3 - 1]

3.4.2. Условные выражения

log-expression:
 [**NOT**] logic-term { log-operation logic-term }*
logic-term:
 comparison
 (log-expression)
comparison:
 arithm-expression comp-operation arithm-expression
condition-on-domain:
 log-expression
condition-on-index:
 name-index rel-except-not-equal-operation
 name-index [{ +,- } const-expression-without-sign]
 name-index rel-except-not-equal-operation const-expression-without-sign

Порядок выполнения операций при вычислении *logical expression*:

1) вычисление арифметических выражений;
2) выполнение операций внутри скобок **(** и **)**; если таких скобок одного уровня вложенности несколько, то слева направо;
3) вычисление отношений **=,>,<,>=,<=,<>,><** ; если таких операций одного уровня несколько, то слева направо.
4) выполнение операций логического отрицания **NOT**, если таких операций одного уровня несколько, то справа налево;
5) выполнение операций логического умножения **AND**, если таких операций одного уровня несколько, то слева направо;

6) выполнение операций логического сложения **OR**, если таких операций одного уровня несколько, то слева направо.

Примеры логических выражений:

ABS(MOD - X1 + Y1) <= 1.0D-20
NOT (K = (2 * N + 1) * (2 * NV + 1) AND NU = 0)
T = J + 1

4. Структура программы

program:
{ [global-declaration] part }$^{+}$
global-declaration:
declaration-of-named-constant
declaration-of-domain-parameters
declaration-of-domain
part:
main-part
simple-part
part-function
main-part:
MAIN PART name-part **.** declaration-of-part
simple-part:
[part-specificator] **PART** name-simple-part **.** declaration-of-part
part-function:
[part-specificator] **FUNCTION** name-function [type-function] **.** declaration-of-function
declaration-of-part:
formal-parameters-of-part **BEGIN** body-of-part **END PART**
formal-parameters-of-part:
[list-name] [**RESULT** list-name]
declaration-of-function:
formal-parameters-of-function **BEGIN** body-of-part **END PART**
formal-parameters-of-function:
list-name
body-of-part:
{ element-of-part }*
element-of-part:
declaration**.**
operator**.**
iteration**.**
part-specificator:
PLAIN
CUDA

Программа на языке НОРМА состоит из одного или нескольких разделов. Разделы могут быть трех видов – главный раздел, простой раздел и раздел-функция. Вид раздела определяется ключевыми словами **MAIN PART**, **PART** и **FUNCTION** соответственно.

Разделы могут вызывать друг друга по имени и передавать данные при помощи механизма формальных и фактических параметров.

Разделы могут использовать глобальные декларации: описание области (*declaration-of-domain*), параметры области (*declaration-of-domain-parameters*) и константы (*declaration-of-named-constant*) – см. п.п. 5.1.1, 5.1.2, 5.1.3.

Простые разделы и разделы-функции могут иметь спецификатор (*part-specificator*). Спецификатор влияет на организацию вызова данного раздела из других разделов и способ передачи параметров, поэтому в других разделах, вызывающих данный, данный раздел должен быть описан в описании внешних имён **EXTERNAL** с точно таким же спецификатором, см. п. 5.1.6.

Спецификатор **PLAIN** означает, что данный раздел является «плоским», т.е. не содержит внутри себя никаких распараллеливающих действий и может быть вызван из любого места программы без каких-либо синхронизаций, из последовательного или параллельного кода, как стандартная арифметическая функция (п. 5.3.1).

С точки зрения пользователя, это указание позволяет рассматривать раздел как последовательный код и не предполагает его распараллеливания.

Пример «плоского» раздела:

```
PLAIN FUNCTION One. X
BEGIN
VARIABLE X.
One = sin(X)**2 + cos(X)**2.
END PART.
```

Спецификатор **CUDA** означает, что при трансляции данного раздела в выходную программу с использованием технологии NVIDIA CUDA будет использовано соглашение о передаче фактических параметров раздела, при котором, помимо самой переменной, будет передаваться также адрес в памяти графического процессора, соответствующий данной переменной. При трансляции раздела в выходную программу без использования технологии NVIDIA CUDA спецификатор **CUDA** будет проигнорирован.

С точки зрения пользователя, это указание означает, что он предполагает использовать распараллеливание на графические процессоры (GPU, технологию NVIDIA CUDA).

Пример раздела со спецификатором **CUDA**:

```
Oi:(i=1..1000).
CUDA PART One. X RESULT Y
BEGIN
VARIABLE X, Y DEFINED ON Oi.
FOR Oi ASSUME Y = sin(X)**2.
```

END PART.

Для каждого раздела справедливо правило локализации: имена, описанные в разделе, локализованы в этом разделе; понятие глобальных переменных в языке отсутствует.

Главный раздел обязательно должен присутствовать в программе на языке НОРМА и быть единственным; формальных параметров он не имеет. Вызовы главного раздела, а также рекурсивные вызовы разделов запрещены.

В заголовке раздела указываются имя раздела и список формальных параметров. Формальные параметры должны быть описаны в теле раздела при помощи *declaration-of-scalar-variables, declaration-of-variables-on-domains* или *declaration-of-external*.

Параметры-величины, указанные до ключевого слова **RESULT** в списке формальных параметров, являются исходными данными для вычислений, описываемых в разделе; параметры, перечисленные после, являются результатами вычислений. Один и тот же параметр не может быть одновременно исходным и результатом: это приводит к переприсваиванию значений переменным (повторному присваиванию), что запрещено в языке НОРМА. В разделе-функции ключевое слово **RESULT** не используется: результат вычисления функции связывается с именем и типом функции.

В теле раздела могут быть заданы описания, операторы и итерации (порядок их расположения, вообще говоря, произвольный).

Ниже приведен фрагмент программы, в котором присутствует главный раздел (**Linear**), простой раздел (**IntFKP**), а также определены глобальные объекты (**DPAR_G**, **C_GLOB**, **oN_R**). (Область действия описаний объектов – см. п.5.1.9).

```
//описание глобальных объектов DPAR_G, C_GLOB, oN_R
DOMAIN PARAMETERS DPAR_G = 100.
CONSTANT C_GLOB = 5.0E+5.
oN_R:(i=1..10; j=1..10; k=1..10).
//главный раздел Linear
MAIN PART  Linear.   // ==== linear approximation ====
BEGIN
. . . . . . //здесь располагается тело раздела Linear
END PART.

//простой раздел IntFKP
PART IntFKP .  //Compute of integral from FKP function
//формальные параметры раздела IntFKP
  BETA, ALPHA,     //node points
  ALL, ALC, ALR,   //points for interpolation
  STEP,            //integration step
  FKP              //name of external integral function
  RESULT    IntResult
```

BEGIN
//описания переменных раздела IntFKP
EXTERNAL FUNCTION FKP DOUBLE .
VARIABLE ALPHA, BETA, ALL, ALC, ALR, STEP, IntResult DOUBLE.
. . . . . //здесь располагается тело раздела IntFKP
END PART.

5. Конструкции языка НОРМА

Конструкции языка НОРМА делятся на *описания и спецификации правил вычисления.* К описаниям относятся следующие объекты: области, скалярные величины (скаляры), величины, определенные на области, индексы распределения, параметры области, входные и выходные величины, имена внешних функций и разделов, именованные константы. *Спецификации правил вычисления* (операторы, функции, итерации и т.д.) описывают решение задачи.

5.1. Описания

declaration:
 declaration-of-named-constant
 declaration-of-domain
 declaration-of-domain-indexes
 declaration-of-scalar-variables
 declaration-of-variables-on-domains
 declaration-of-distribution-indexes
 declaration-of-domain-parameters
 declaration-of-input
 declaration-of-output
 declaration-of-external

5.1.1. Именованные константы

declaration-of-named-constant:
 CONSTANT name-constant = value-constant.
value-constant:
 arithm-constant

Именованная константа представляется переменной с модификатором **CONSTANT**. С точки зрения программы такая переменная сохраняется как постоянная. Такой переменной нельзя присваивать значения в процессе исполнения программы. Единственное место, где ей присваивается значение, – это ее описание.

Пример именованной константы:
CONSTANT Velocity = 5.0E+5.

5.1.2. Параметры области

declaration-of-domain-parameters:
 DOMAIN PARAMETERS list-prescription
prescription:
 name-domain-parameter = int-const

Границы диапазонов при описании областей могут задаваться неявно – при помощи параметров области. Значения этих параметров должны быть определены в описании параметров области, например:

DOMAIN PARAMETERS M=3, J=90.

Параметры области могут входить в *арифметические выражения* и *константные выражения.*

Например:

DOMAIN PARAMETERS DPAR_G = 100.
CONSTANT Velocity = 5.0E+5 + DPAR_G.
Grid_oNR:(i=1..10; j=1.. DPAR_G; k=1..DPAR_G).

5.1.3. Области

declaration-of-domain:
 declaration-of-unconditional-domain
 declaration-of-conditional-domain
domain:
 new-domain-without-name
 name-domain
unconditional-domain:
 new-domain-without-name
 name-unconditional-domain
name-domain:
 name-unconditional-domain
 name-conditional-domain
name-unconditional-domain:
 name-rectangular-domain
 name-diagonal-domain

Понятие области введено в языке НОРМА для представления понятия индексного пространства. Область – это совокупность целочисленных наборов $\{i_1,....,i_n\}$, $n>0$, $i_j>0$, $j=1,...,n$, каждый из которых задает координаты точки n-мерного индексного пространства. С каждым направлением (осью координат) n-мерного пространства задачи связывается уникальное имя – *имя индекса* (имя оси координат индексного пространства).

Следует отметить, что область определяет значения координат точек индексного пространства, а не значения расчетных величин в этих точках. Например, если требуется вычислить значения величины $Y_{i,j}$, $i.j=1,...,n$ на некоторой сетке $X_{i,j}$, $i,j=1,...n$, введенной при решении задачи (например,

формулой вида $X_{i,j} = F(h,i,j)$, F – заданная функция, h – заданный параметр), то следует:

1) описать область, состоящую из точек $i,j = 1,...n$;
2) описать на этой области величины X и Y;
3) задать на этой области правило вычисления значений сетки $X_{i,j}$: $X_{i,j} = F(h,i,j)$ и правило вычисления значений $Y_{i,j}$: $Y_{i,j} = G(X_{i,j})$ (считается, что F,G,h также некоторым способом заданы).

В языке НОРМА область может иметь имя, над областями определены операции модификации и произведения. Индексы областей специально не описываются – они вводятся при определении областей. Область может быть *условной* и *безусловной*. Условная область состоит из точек индексного пространства, число и координаты которых могут меняться в зависимости от выполнения (или невыполнения) *условий на область*. Безусловная область состоит из точек индексного пространства, число и координаты которых могут быть определены на этапе трансляции.

В языке различается *описание области* – это именованная условная или безусловная область – и *использование области* – синтаксически это *область*, *безусловная область, имя-области*, *имя-безусловной-области* или *новая-область-без-имени* (п. 5.1.3.1). *Использование области* возможно в описаниях величин, определенных на области, при задании области вычисления в операторах **ASSUME** (п. 5.2.2.), в описаниях входных или выходных величин, при задании областей фактических параметров в вызовах разделов или функций, в функциях редукции.

5.1.3.1. Безусловная область

declaration-of-unconditional-domain:
 declaration-of-rectangular-domain
 declaration-of-diagonal-domain
declaration-of-rectangular-domain:
 multidimensional-domain
 new-domain
multidimensional-domain:
 onedimensional-domain
 [name-<u>multidimensional-domain</u> **:**] **(** domain-product **)**
domain-product:
 component-domain { **;** component-domain }$^{+}$
component-domain:
 multidimensional-domain
 name-<u>unconditional-domain</u>
onedimensional-domain:
 [name-<u>onedimensional-domain</u> **:**] **(** name-<u>index</u> = value **)**
value:

range
const-expression
range:
const-expression **..** const-expression
const-expression .. formal-in-parameters-of-part
formal-in-parameters-of-part:
name
new-domain:
[name-new-domain **:**] new-domain-without-name
new-domain-without-name:
name-unconditional-domain / list-modification
modification:
name-index = value
name-onedimensional-domain { { +,- } boundary-function }$^{+}$
boundary-function:
LEFT(const-expression **)**
RIGHT(const-expression **)**
name-rectangular-domain:
name-onedimensional-domain
name-multidimensional-domain
name-new-domain
declaration-of-diagonal-domain:
name-diagonal-domain**:**name-unconditional-domain /list-condition-on-index

Ключевым понятием при описании прямоугольных областей является понятие *одномерной области*. Одномерная область служит для задания диапазона точек на некоторой оси координат индексного пространства. В простейшем случае при описании одномерной области указывается *имя одномерной области*, *имя индекса* и границы изменения значений индекса:

RegionK: (k =1..15).

Имя-одномерной-области может использоваться для ссылки на эту область.

Имя-индекса есть индексная переменная, множество значений которой определяется диапазоном:

const-expression **..** *const-expression*

или

const-expression **..** *formal-in-parameters-of-part*

Границами диапазона являются целые положительные константные выражения, построенные из *целых констант, параметров области* и *арифметических операций*. Верхняя граница может задаваться входным формальным параметром целого типа.

Если всем параметрам области присвоены конкретные значения в *описании параметров области,* то значение нижней границы диапазона должно быть не больше значения верхней границы. Если верхняя граница задается

входным формальным параметром, то проверка на корректность границ не производится.

Пример описания области **oLCmin**, в котором нижняя граница задается константой 1, а верхняя граница – арифметическим выражением над параметрами области:

```
DOMAIN PARAMETERS FIRSTCOL = 0.
DOMAIN PARAMETERS LASTCOL = NA - 1.
DOMAIN PARAMETERS FIRSTROW = 0.
DOMAIN PARAMETERS LASTROW = NA - 1.
oLCmin:(i=1 .. LASTCOL - FIRSTCOL + 1).
```

Пример описания и использования области **oN**, для которой верхняя граница задается формальным параметром **N** целого типа:

```
PART Test.
//входные параметры раздела:
//u – переменная, определённая на области, размер которой
//     задается целочисленным параметром N
u, N
BEGIN
oN:(i=1..N; j=1..N; k=1..N).  //описание области
VARIABLE N INTEGER.
VARIABLE u DEFINED ON oN DOUBLE.
FOR oN/(i=2..N-1, j=2..N-1, k=1)
        ASSUME u = u[k=N-1]. //использование области
END PART.
```

Многомерная область строится при помощи операции "**;**" произведения прямоугольных областей.

Пример описания двумерной области, полученной с помощью операции произведения одномерных областей **AxisK** и **AxisL**:

```
Square: (AxisK: (k=1..15) ; AxisL: (l=1..5) ).
```

Область **Square** может быть описана также другими способами (например, с учетом описания, приведенного выше):

```
Square: (RegionK ; AxisL: (l=1..5) ).
```

Операция "**;**" произведения прямоугольных областей обладает следующим свойством: если **A** и **B** – области, то **A;B = B;A**.

Это означает, что порядок направлений индексного пространства при описании области не фиксируется (или, что с точки зрения пользователя то же самое, фиксируется некоторым произвольным образом). Если порядок направлений индексного пространства существенен (например, необходимо согласование направлений при использовании величин на одних и тех же областях в различных разделах), то он может быть задан при помощи описания **INDEX** (см. п. 5.1.2).

Области, являющиеся операндами операции произведения областей, должны иметь различные имена индексов.

Модификация прямоугольной области может состоять в добавлении некоторого числа точек, в удалении точек или изменении диапазона. Модификация двух первых типов описывается при помощи *функций границ* **LEFT(n)** и **RIGHT(n).** Функция **LEFT** применяется к левой границе диапазона, функция **RIGHT** – к правой границе диапазона. Знак "+" перед функцией означает, что к одномерной области добавляются точки, "-" – что из одномерной области удаляются точки. Обе эти функции имеют один параметр n, определяющий число точек, которые необходимо удалить или добавить к области. В качестве фактического параметра функций **LEFT** и **RIGHT** может быть задана только целая положительная константа. Обращение к функциям допустимо лишь в контексте с именем одномерной области, задающей модифицируемый диапазон:

name-onedimensional-domain {+,-} **LEFT(n)**

или

name-onedimensional-domain {+,-} **RIGHT(n)**

Например, описание

FlushK : Square / AxisK - LEFT(2) + RIGHT(2).

определяет область **FlushK**, состоящую из точек k=3 .. 17 (если учитывать описание области **Square**, приведенное выше).

Границы диапазона, получаемого в результате применения функций **LEFT** или **RIGHT**, должны быть положительными целыми, причем левая граница не должна превосходить правую. Описание

Square2: Square / AxisK + LEFT (1) - RIGHT (2).

неверно, так как левая граница получаемого в результате диапазона k=0..13 не является положительной.

Область, модифицируемая при помощи функций **LEFT** или **RIGHT,** должна быть многомерной областью.

Можно изменить составляющую одномерную область и путем явного переопределения диапазонов. Для этого в модификации надо указать *имя индекса* направления и его новое *значение*. Например:

Newsquare: Square / AxiisK+RIGHT(3), L=50...80.

Если список модификаций содержит более одного элемента, то заданные в этом списке модификации действуют на модифицируемую область в порядке написания слева направо.

Кроме определения прямоугольных областей, в языке НОРМА имеется возможность задания *диагональных* областей путем наложения условий на уже определенную область (прямоугольную или диагональную).

Диагональная область определяется при помощи условий на индексы некоторой ранее определенной области **D** и состоит из тех точек области **D**, в которых эти условия принимают значение "истина". Например, описания

KL: ((k=1..10) ; (l=1..10)).

Diagonal: KL/ k=l.

задают область **Diagonal** , состоящую из точек (k=1, l=1), (k=2, l=2), ..., (k=10, l=10).

Индексы величин, используемых при записи условия на индексы, необязательно должны входить в множество индексов области **D**, на которую накладываются условия – это могут быть внешние индексы (см. п. 5.2.3.2).

Допустимый вид условий на индексы позволяет статически определить точки, из которых состоит диагональная область (это возможно, конечно, и для прямоугольной области). Поэтому прямоугольные и диагональные области являются *статическими* объектами, в отличие от *условных* областей, рассматриваемых ниже.

5.1.3.2. Условная область

declaration-of-conditional-domain:

name-conditional-domain**,**name-conditional-domain**:**name-domain/condition-on-domain

Помимо задания статических областей, в языке НОРМА имеется возможность задания условной области путем наложения условий на ранее определенную область. Условная область состоит из точек индексного пространства, число и координаты которых могут меняться в зависимости от выполнения (или невыполнения) *условий на область.*

Идея задания условной области заключается в следующем. Ранее определенная область **D** разбивается на две непересекающиеся подобласти **D1** и **D2**. Первая подобласть состоит из точек области **D**, в которых заданное *условие на область* принимает значение "истина", вторая – из точек, в которых оно принимает значение "ложь". При этом **D1** ∪ **D2** = **D**, **D1** ∩ **D2** = ∅.

Например, пусть заданы области:

Domain: ((i=2..n)) ; (j=1..m)) .

Domain1, Domain2: Domain / x+y[i-1, j] - z[j+1] > 0 .

Это описание определяет разбиение исходной области **Domain** на области **Domain1** и **Domain2**, такие, что:

Domain1 ∪ Domain2 = Domain, Domain1 ∩ Domain2 = ∅ .

Область **Domain1** состоит из точек **Domain**, в которых условие $x+y[i-1, j] - z[j+1] > 0$ принимает значение "истина", область **Domain2** – из точек, в которых это условие принимает значение "ложь".

Индексы величин, используемых при записи условия, необязательно должны входить в множество индексов области **D**, на которую накладываются условия – это могут быть внешние индексы (см. п. 5.2.3.2).

Ниже приводится фрагмент программы, описывающий условные области **Bf2Pl** и **BfNot2Pl**. Область **BfNot2Pl** разбивается на условные области **Nodes** и **NotNodes**, а область **NotNodes**, в свою очередь, разбивается на условные области **DomainTrue, DomainFalse**:

DOMAIN PARAMETERS N=3, NV=3.

Bnu: (Nu=0..2*N).

```
Bf: ( K=0..(2*N+1) * (2*NV+1 ).
//описание безусловной области BfNu
BfNu: (Bf ; Nu).
//описание условных областей
//Bf2PI , BfNot2PI на безусловной области BfNu
Bf2PI , BfNot2PI : BfNu/ K=(2*N+1)*(2*NV+1 ) AND NU=0.
//описание условных областей
//Nodes, NotNodes на условной области BfNot2PI
Nodes , NotNodes : BfNot2PI/ ABS (BTNodes - BT ) < 0.0001.
//описание условных областей
// DomainTrue, DomainFalse на условной области Nodes
DomainTrue, DomainFalse: Nodes/ ABS(BTNu - BTNodes ) < 0.0001.
//описание переменных
VARIABLE BT, BTNodes DEFINED ON Bf DOUBLE.
VARIABLE BTNu DEFINED ON Bnu DOUBLE.
```

5.1.4. Индексы областей

declaration-of-domain-indexes:

INDEX list-name-index

При описании области порядок направлений индексного пространства не фиксируется (или, что с точки зрения автора программы то же самое, фиксируется некоторым произвольным образом). Если порядок направлений индексного пространства существенен (например, необходимо согласование направлений при использовании величин на одних и тех же областях в различных разделах), то он задается при помощи *описания индексов областей*. Порядок направлений индексного пространства совпадает с порядком перечисления имен индексов в описании **INDEX** (слева направо).

Рассмотрим вызов раздела **B** из раздела **A** с передачей из раздела **A** в раздел **B** значений величины **X**, определенной на области **SquareInA**:

```
PART A.
BEGIN
INDEX k, m .
SquareInA: (AxisK: (k=1..5) ; AxisL: (m=1..10) ).
VARIABLE X DEFINED ON SquareInA.
COMPUTE B( X ON SquareInA).
END PART.

PART B.
 X.
BEGIN
INDEX i, j.
SquareInB: (AxisK:  (i=1..5) ; AxisL:  (j=1..10) ) .
VARIABLE X DEFINED ON SquareInB.
```

END PART.

Описания **INDEX k,l** и **INDEX i,j** устанавливают соответствие между индексами: **i~k, j~l** , то есть значения величины **X** трактуются в обоих разделах одинаково. Если описание **INDEX i, j** из раздела **B** заменить на **INDEX j, i**, то значения по соответствующим направлениям оказываются несогласованными: 5 $\not\sim$ 10, 10 $\not\sim$ 5.

5.1.5. Арифметические величины

declaration-of-scalar-variables:

VARIABLE list-name-scalar [type]

declaration-of-variables-on-domains:

VARIABLE list-name-variable-on-domain **DEFINED ON** name-unconditional-domain [type]

type:

{**REAL**, **INTEGER**, **DOUBLE**}

Скалярные величины (*скаляры*) и *величины на области* относятся к *арифметическим величинам*. Описание ставит в соответствие каждой арифметической величине уникальное в текущем разделе *имя величины*, а также задает тип величины: **REAL, INTEGER** или **DOUBLE** (по умолчанию – тип **REAL**).

Пример описания скаляров:

VARIABLE Alpha, X, H DOUBLE.

VARIABLE IJK, Z INTEGER.

Каждая *величина на области* связывается с указанной в описании областью. Эта область определяет *имена индексов*, которые могут использоваться в *индексных выражениях* при обращении к данной величине, причем порядок указания индексных выражений несущественен. Для индексов не требуется специального описания – они вводятся при описании областей.

Square: (AxisK: (k=1..5) ; AxisL: (m=1..5)).

VARIABLE First, Last DEFINED ON Square.

VARIABLE SingleK DEFINED ON AxisK DOUBLE.

VARIABLE SingleL DEFINED ON AxisL INTEGER.

Приведенные описания определяют величины **First**, **Last** на области **Square**; это означает, что эти величины могут иметь в индексных выражениях индексы **k** и **m**. Величины **SingleK, SingleL** определены, соответственно, на областях **AxisK** и **AxisL**, то есть могут иметь в индексных выражениях индексы **k** и **m** соответственно. При этом обращения **First[k-1, m+1]** и **First[m+1, k-1]** эквивалентны (подробнее см п. 5.1.2).

При обращении к *величине на области* действует правило задания индексов по умолчанию: *индексные выражения*, совпадающие с *именем индекса*, могут быть опущены. Например, обращения **First[k, m]**, **First[k]**, **First[m]**, **First** к величине **First** эквивалентны.

В случае если какой-либо индекс задается константой (константным выражением), необходимо указать, к какому индексному направлению относится данная константа, например **First[k=5, m-1].** Если надо одно индексное направление связать зависимостью с другим, то это также необходимо делать явно, например, диагональные элементы матрицы **First** можно определить как **First[k, m=k]** или **First[k=m]**.

5.1.6. Входные и выходные величины

declaration-of-input:
 INPUT list-input-scalar
 INPUT input-on-domain
input-scalar:
 name-scalar [attributes]
input-on-domain:
 list-input-output-variable **ON** unconditional domain
input-output-variable :
name-variable-on-domain [attributes]
declaration-of-output:
 OUTPUT list-output-scalar
 OUTPUT output-on-domain
output-scalar:
 name-scalar [attributes]
output-on-domain:
 list-input-output-variable **ON** domain
attributes:
 (list-attribute **)**
attribute:
 STR(int-constant **)**
 string
 ORDER(list-name-index **)**
 FILE = ' file-name '
 ALL
 format
format:
 I int-constant
 {**F**,**E**,**D**} int-constant **.** int-constant

Описание входных (выходных) величин означает, что подлежат вводу (выводу) значения всех величин, указанных в списке ввода (вывода). Например, описания:

B1 , B2 : B / Z<Eps.
INPUT Velocity ON A.
OUTPUT Tau ON B1.
INPUT X, ALPHA.

являются запросами на ввод значений скаляров **X, ALPHA**, величины **Velocity** во всех точках области **A**, а также на вывод значений величины **Tau** во всех точках области **B1**, в которых выполнено условие **Z<Eps**.

Порядок, в котором будут вводиться (выводиться) величины, при этом не задается – он определяется в процессе трансляции. Минимальная единица информации, которая вводится (выводится) как единое целое – *скаляр*, или *величина на области*. Управление вводом (выводом) этих единиц может осуществляться при помощи *атрибутов*.

Атрибуты действуют на величину (список величин), после которой они указаны. Например, описание:

INPUT R1(FILE ='norma'), R2 (FILE ='myfile') ON Grid

является запросом на ввод из файла myfile.dat значений величины **R2** в области **Grid**, то есть для всех значений индексов этой области, и величины **R1** в области **Grid** из файла **norma.dat.**

Атрибуты позволяют управлять формой входных и выходных данных и связывать данные с входными и выходными файлами:

STR(n) задает пропуск n-1 строки;

'string' задает вывод текстовой константы *string*;

ORDER указывает порядок изменения индексов при выводе: самым старшим индексом является самый правый, самым младшим – самый левый; для очередного значения каждого старшего индекса младшие принимают все свои значения;

FILE задает имя входного или выходного файла; способ задания имени файла определяется при реализации (обычно входной файл по умолчанию имеет расширение .dat; при задании имени файла это расширение может быть опущено; если атрибут **FILE** не задан, то ввод осуществляется из файла **norma.dat**, а вывод – в стандартный поток вывода stdout);

ALL приводит к выводу имени величины и ее координат (значений индексов) для каждого значения *величины на области*.

format задает формат для числовых значений при вводе и выводе и соответствует спецификаторам формата **I, F, E, D** в языке Фортран. По умолчанию установлены форматы:

E15.8 – для величин типа **REAL**
D15.8 – для величин типа **DOUBLE**
I5 – для величин типа **INTEGER**

Для входных величин разрешены только атрибуты *format* и **FILE**.

Пример описания входных величин:

INPUT One (FILE=‘file1’,F10.2), Two(FILE=‘file2.dat’) ON Grid2.

Примеры описания выходных величин:

OUTPUT Velocity(' Velocity = ', F9.1) .
OUTPUT X(FILE = 'FILE17' , ORDER(J,K,I), ALL, F5.1) ON Grid.
OUTPUT Y('Matrix of values Y par',FILE='OT5', ALL, F15.2) ON ABC.

Для вывода данных в более сложной форме (таблицы, графики и т.п.) следует использовать либо средства стандартных библиотек и пакетов, либо собственные программы, написанные на других языках (см. п. 5.2.5).

Синтаксис файла исходных данных:

file:
 input-element { *input-element* }*
input-element:
 name-scalar = *arithm-constant;*
 name-variable-on-domain **(** list-*index-range* **)** = *data* **;**
index-range:
 name-index = *int-constant* **..** *int-constant*
data:
 list-*data-element*
data-element:
 int-constant
 body
 int-constant **(** *data* **)**

При задании файла исходных данных можно:

- располагать *элементы ввода* в произвольном порядке,
- изменением порядка следования индексов управлять порядком расположения числовых значений в файле,
- сокращенно записывать повторяющиеся данные.

Пример содержимого файла исходных данных:

```
C(K=1..10)=5(-10.1) ,5(1.01);
ALPHA=3.17; BETA=-0.12; GAMMA=0.000000001;
C(K=11..20)=5(10.1),5(-1.01);
```

Пример программы Gauss и содержимого файла исходных данных data.dat для этой программы:

```
MAIN PART Gauss.
//Solution of linear equations by Gauss-Jourdan method.
BEGIN
Ot:(t=0..n). Oi:(i=1..n). Oj:(j=1..n).
Oij:(Oi;Oj). Otij:(Ot;Oij).
Oti:(Ot;Oi). Otij1:Otij / t=1..n.  Oti1:Oti / t=1..n.
DOMAIN PARAMETERS n=5.
VARIABLE a ON Oij. VARIABLE m ON Otij.
VARIABLE b, ON Oi.  VARIABLE r ON Oti.
INPUT a(FILE='data') ON Oij, b(FILE='data') ON Oi.
OUTPUT x(FILE='results',ALL) ON Oi.
FOR  Otij/ t=0 ASSUME m=a.
FOR  Oti / t=0 ASSUME r=b.
OtiEQtij1,OtiNEtj1:Otij1 / i=t. OiEQti1,OiNEti1:Oti1 / i=t.
FOR  OtiEQtj1 ASSUME m = m[t-1,i=t]/m[t-1,i=t,j=t].
```

FOR OiEQj1 ASSUME r = r[t-1,i=t]/m[t-1,i=t,j=t].
FOR OiNEtj1 ASSUME m = m[t-1]-m[t-1,j=t]*m[i=t].
FOR OiNEti1 ASSUME r = r[t-1]-m[t-1,j=t]*m[i=t].
FOR Oi ASSUME x = r [t=n].
END PART.

Входной файл *data.dat*:

B(I=1..3)=5.0, 13.0, 3.0;
A(I=1..5,J=1..5)=2.0, 3.0, -4.0, 5.0, -1.0,
3.0, 4.0, -1.0, 6.0, 1.0,
2.0, 0.0, -3.0, 0.0, 4.0,
0.0, 2.0, 0.0, 0.3, 0.0,
3.0, -1.0, 2(0.0), 1.0;
B(I=4..5)= 5.0, 3.0;

Значения многомерных массивов данных располагаются в файле в соответствии с указанными индексами, причем первыми меняются значения более правого индекса – в приведенном примере матрица **A(I,J)** задана по строкам, то есть *i* =1, j = 1,2,3,4,5, затем *i* = 2, *j* = 1,2,3,4,5, и так далее до *i*=5, *j* = 1,2,3,4,5.

Способ задания входного файла data.dat, конечно, не является единственным.

5.1.7. Внешние имена

declaration-of-external-names:
declaration-of-external-functions
declaration-of-external-parts
declaration-of-external-functions:
[part-specificator] **EXTERNAL FUNCTION** list-name-external-function [type]
declaration-of-external-parts:
[part-specificator] **EXTERNAL PART** list-name-external-simple-part

В *описании внешних имен* обязательно указываются имена функций или разделов, которые вызываются из данного раздела. По умолчанию внешняя функция имеет тип **REAL.** Спецификатор внешнего раздела имеет точно такое же значение, как и при описании самого внешнего раздела (см. п. 4). При описании раздела и при описании раздела во внешних именах другого раздела спецификатор, если он имеется, должен быть одним и тем же.

Также во внешних именах раздела должны быть описаны все функции и процедуры, написанные непосредственно на целевом языке программирования и вызываемые из данного раздела. Если необходимо, при этом может быть использован спецификатор раздела, соответствующий такой написанной непосредственно на целевом языке программирования функции или процедуры.

Примеры описания внешних функции:

PLAIN EXTERNAL FUNCTION Last, First DOUBLE.

5.1.8. Индексы распределения

declaration-of-distribution-indexes:
 DISTRIBUTION INDEX list-distribution-index
distribution-index:
 name-index=int-constant
 name-index=simple-range
simple-range:
 int-constant **..** int-constant

Описание индексов распределения служит для отображения одного или более индексных направлений индексного пространства области задачи на массив процессорных элементов (ПЭ) распределенной системы. Разделы или функции, в которых присутствует это описание, называются *распределенными*, если оно отсутствует – *нераспределенными*. В распределенном разделе или функции это описание должно встречаться не более одного раза.

Данное описание приводит к распределению между ПЭ системы как данных, так и управления и автоматической генерации операторов обмена данными между ПЭ, если это необходимо. Распределению подлежат величины, участвующие в расчетах и имеющие индексы, совпадающие с указанными в описании **DISTRIBUTION INDEX**.

Например, если задано описание **DISTRIBUTION INDEX i=8**, то все переменные, определенные на областях с индексом i, будут распределены на 8 ПЭ с виртуальными номерами от 1 до 8.

Вычисления, описанные в нераспределенном разделе или в нераспределенной функции, выполняются целиком в одном ПЭ (хотя таких ПЭ может быть и несколько).

Описание **DISTRIBUTION INDEX i=2...8, j=0..0** неверно: считается, что элементы матрицы ПЭ нумеруются начиная с 1.

5.1.9. Область действия описаний

В языке НОРМА с каждым объектом (константы, переменные, описания и параметры областей, функции) связывается область действия – часть программы, в которой этот объект доступен. Область действия зависит от местоположения описания объекта в программе. Если описание расположено внутри раздела, то оно является локальным и доступно только внутри этого раздела. Если описание расположено вне всех разделов, то оно является глобальным и доступно всем разделам в рамках одного программного модуля.

Глобальными могут быть следующие объекты:

- описание области,
- описание параметров (**DOMAIN PARAMETERS**),
- именованные константы (**CONSTANT**).

В примере ниже два раздела (**test1** и **test2**) используют глобальные объекты (**DPAR_G, C_G, O_G**), кроме того, в разделе **test1** имеется локальный объект **O_L**, недоступный второму разделу.

```
DOMAIN PARAMETERS DPAR_G = 100.
CONSTANT C_G = 5.0E+5.
O_G:(i=1..10; j=1..3).

PART test1.
O_L:(i=1..10; j=1.. DPAR_G).
VARIABLE VAR1 DEFINED ON O_G DOUBLE.
VARIABLE VAR2 DEFINED ON O_L DOUBLE.
FOR O_G  ASSUME VAR1 = C_G..
FOR O_L  ASSUME VAR2 = DPAR_G..
END PART.

PART test2.
VARIABLE VAR1 DEFINED ON O_G DOUBLE.
FOR O_G  ASSUME VAR1 = C_G.
END PART.
```

5.2. Операторы в языке НОРМА

operator:
 scalar-operator
 operator-**ASSUME**
 call-part

В языке НОРМА определены три вида операторов: скалярный оператор, оператор **ASSUME** и вызов раздела. Операторы предназначаются для описания вычислительных действий, необходимых для решения задачи.

5.2.1. Скалярный оператор

scalar-operator:
 name-scalar = scalar-arithm-expression
scalar-arithm-expression:
 arithm-expression

Скалярный оператор предназначен для вычисления арифметических значений скаляров. По существу, это аналог оператора присваивания традиционных языков программирования, в левой части которого указывается имя скаляра, а в правой части – скалярное арифметическое выражение, построенное обычным образом из скаляров, арифметических констант, параметров области, обращений к функциям, величин на области с индексами-константами.

В скалярное арифметическое выражение не могут входить величины, определенные на области, у которых индексные выражения не являются

константами (исключением из этого правила являются аргументы функций редукции, см. п. 5.2.3).

IJ: ((i=1..MaxI) ; (j=1..MaxJ)).
DOMAIN PARAMETERS MaxI = 3, MaxJ = 90.
VARIABLE ScalarV INTEGER.
VARIABLE Pi DOUBLE.
VARIABLE ArrayV DEFINED ON IJ DOUBLE.
ScalarV = MaxI*(MaxJ-1)/2+SQRT(PI)/SIN(ArrayV[i=1,j=55]).

Переприсваивание в скалярном операторе запрещено, поэтому следующий оператор по определению является неверным: **ALPHA = ALPHA-1.**

5.2.2. Оператор ASSUME

operator-**ASSUME**:
FOR domain **ASSUME** relation { **;** relation }*
relation:
name-variable-on-domain = arithm-expression
call-part

Оператор **ASSUME** используется для вычисления арифметических значений величин, определенных на областях.

Случай *вызова раздела* в теле оператора **ASSUME** описан в п. 5.2.4.

Термами арифметического выражения в правой части соотношения могут быть величины, определенные на области, скаляры, арифметические константы, параметры области, обращения к функциям, индексы.

Неформально семантика оператора **ASSUME** определяется следующим образом. Рассмотрим соотношение, записанное в виде:

FOR D$(i_1,...,i_n)$ **ASSUME**

$$X^1_{Indl(i_1,...,i_n)} = F(X^{j_1}_{IndR^{j_1}(i_1,...,i_n)},...,X^{j_k}_{IndR^{j_k}(i_1,...,i_n)},Other)$$

где

- **D**$(i_1,...,i_n)$ – область оператора **ASSUME**,
- $(i_1,...,i_n)$ – индексы области **D**,
- X^{j_q}, $1 \le q \le k$ – имена величин, определенных на области,
- $IndL(i_1,...,i_p)$, $1 \le p \le n$ – индексные выражения левой части,
- $IndR^{j_q}(i_1,...,i_n)$ – индексные выражения правой части,
- F – функция, вычисляемая в правой части,
- *Other* – другие термы правой части.

Каждое соотношение задает правило F вычисления значений величины X^1 из левой части по значениям величин $X^{j_1},...,X^{j_k}$ и термов *Other* из правой части:

1) определяются все точки $(a_1,...,a_n) \in \mathbf{D}(i_1,...,i_n)$;
2) для каждой точки $(a_1,...,a_n) \in \mathbf{D}$ требуется вычислить значение величины X^1 из левой части в точке $IndL(a_1,...,a_n)$;

3) для каждой точки $(a_1,...,a_n) \in \mathbf{D}$ вычисляются значения индексных выражений $IndR^{j_q}(a_1,...,a_n)$ всех величин X^{j_q}, $1 \le q \le k$, входящих в правую часть соотношения, и определяется множество аргументов правой части $\mathbf{X}(a_1,...,a_n) = \{X^{j_q}_{IndR^{j_q}(a_1,...,a_n)}, Other\}$, $1 \le q \le k$;

4) если в некоторый момент времени для точки $(a_1,...,a_n) \in \mathbf{D}$ все аргументы из **X** вычислены, то возможно вычисление значения $X^1_{IndL(a_1,...,a_n)}$ величины из левой части, если не все аргументы определены, то вычисление в данной точке в данный момент невозможно (это не означает, что оно вообще невозможно).

Имя **ASSUME**, которое дано оператору, подчеркивает то обстоятельство, что он однозначно определяет правило для вычисления значений величины, но не требует немедленного выполнения вычисления в данном месте программы и не задает порядка или способа (параллельно, последовательно и т.п.) вычисления.

Например, если величина **X** описана как

Matrix: ((I=1..5) ; (j=1..5)).

VARIABLE X DEFINED ON Matrix.

то оператор

FOR Matrix ASSUME X = 0.

является запросом на обнуление 25 элементов величины X. Способ реализации этого запроса в языке не фиксируется.

Язык НОРМА является языком с одним присваиванием. Переприсваивание значений величинам запрещено. Поэтому следующие операторы по определению являются некорректными:

FOR Matrix ASSUME X = Y; X = Z.

FOR Matrix ASSUME X = X+1.

Ограничение на вид индексов левой части соотношения (это индексы без смещений) не является принципиальным, так как область **D** из заголовка оператора позволяет описывать достаточно сложные соотношения. Пусть необходимо описать вычисление вида

$X_{i,+1,i} = F(Y_i)$, $i=1,...,n$, на области **Ox: ((I=1..N) ; (J=1..N))**

Это можно сделать следующим образом:

Ox: ((I=1..N) ; (J=1..N)).

OxII: Ox/ J=I+1.

VARIABLE X DEFINED ON Ox.

FOR OxII ASSUME X = F(Y) // F - некоторая функция

Как указано выше, в скалярное арифметическое выражение и в арифметическое выражение могут входить функции.

5.2.3. Оператор вызова раздела

call-part:

COMPUTE name-part [**(** actual-parameters **)**]

actual-parameters:
[list-in-parameter] [**RESULT** list-out-parameter]
out-parameter:
name-scalar
name-variable-on-domain **ON** domain-of-parameter
iterated-variable-on-domain **ON** domain-of-parameter

Фактическими параметрами раздела могут быть арифметические выражения, имена разделов и функций либо величины на областях (массивы).

Фактические параметры раздела могут быть объявлены исходными или результатами при помощи ключевого слова **RESULT**. Например:

COMPUTE Velocity (Delta+0.5, Fi ON Oijk RESULT V ON Oij).

Первые два параметра являются исходными, третий – результат. Побочный эффект невозможен – если множества исходных параметров и параметров-результатов пересекаются, то это приводит к повторному присваиванию значений величинам, что в языке НОРМА запрещено.

Следует отметить, что вызов раздела, по существу, является развитием понятия соотношения, используемого в операторе **ASSUME**, так как дает возможность получать несколько результатов (значений различных величин) сразу. Вызов раздела вне оператора **ASSUME** является развитием понятия скалярного оператора.

Способы задания исходных фактических параметров раздела совпадают со способами задания фактических параметров функции, описанными в п. 5.2.3.3.

Фактическими параметрами-результатами могут быть скаляры, величины с индексами (быть может, заданными правилом умолчания индексов), величины на областях (массивы).

Если вызов раздела находится в теле оператора **ASSUME**, то параметрами-результатами не могут быть скаляры и величины на статических областях (которые не меняются при различных значениях индексов из заголовка оператора **ASSUME**) – это заведомо приводит к переприсваиванию.

Если вызов раздела находится вне тела оператора **ASSUME**, то параметрами-результатами могут быть только имена скаляров, величины с индексами-константами и величины на статических областях.

Примеры различных способов задания фактических параметров при вызове раздела.

VARIABLE A, B DEFINED ON Grid: (Oi: (I=1..N); Oj: (J=1..M)).
VARIABLE X, Y DEFINED ON Oi.
VARIABLE T DEFINED ON Oj.
VARIABLE Gamma DOUBLE.
INPUT A ON Grid ,Y ON Oi, T ON OJ.
INPUT Gamma.
OUTPUT X ON Oi, B ON Grid /J=2..12.
DOMAIN PARAMETERS N =10, M =20.
EXTERNAL PART DzDy.

FOR Oi ASSUME
COMPUTE TEST (DzDy, T ON Oj, Y RESULT X, B ON Grid /J=I+2).
COMPUTE SCALAR(A ON Grid RESULT Gamma, B ON Grid /J=2).

В операторе **ASSUME** описано правило вычисления вектора X_i, $i=1,...,10$ и части матрицы **B**: $B_{i,j}$, $i=,...,10$, $j=3,...,12$.

Скалярный вызов раздела **SCALAR** определяет скаляр **Gamma** и второй столбец матрицы **B**.

Правила соответствия формальных и фактических исходных параметров раздела совпадает с правилами соответствия для функции, описанными в п. 5.3.

Правила соответствия формальных и фактических параметров-результатов приведены в таблице 1 и являются частным случаем правил п. 5.3.

Таблица 1.

Соответствие формальных и фактических параметров-результатов

Формальный параметр	**Фактический параметр**
скаляр	скаляр
скаляр	величина с индексами-константами
величина на области	величина на области

5.3 Функции в языке НОРМА

call-function:
 call-reduction-function
 call-standard-function
 call-external-function
call-reduction-function:
 name-reduction-function **(** **(** name-domain **)** arithm-expression **)**
call-standard-function:
 name-standard-function **(** arithm-expression **)**
name-reduction-function:
 { **SUM,MULT,MAX,MIN** }
call-external-function:
 name-external-function [**(** list-in-parameter **)**]
in-parameter:
 arithm-expression
 name-variable-on-domain **ON** domain-of -parameter
 iterated-variable-on-domain **ON** domain-of-parameter
 name-external-simple-part
 name-external-function
domain-of-parameter:
 name-unconditional-domain
 name-unconditional-domain / index-expression

name-unconditional-domain / **(** list-index-expression **)**
iterated-variable-on-domain :
name-variable-on-domain **[** name-iteration-index [-1] **]**

В языке определены стандартные арифметические функции, функции редукции и внешние функции пользователя.

5.3.1. Стандартные арифметические функции

В таблице 2 приведены стандартные арифметические функции.

Таблица 2

Стандартные арифметические функции

abs(x)	**alog(x)**	**alog10(x)**	**asin(x)**	**atan(x)**
amax1(x,y,...)	**amin1(x,y,...)**	**cos(x)**	**dsqrt(x)**	**dalog(x)**
dalog10(x)	**dsin(x)**	**dcos(x)**	**dabs(x)**	**dsign(x)**
dexp(x)	**datan(x)**	**dmin1(x,y,...)**	**dmax1(x,y,...)**	**entier(x)**
exp(x)	**sin(x)**	**sign(x)**	**sqrt(x)**	

Эти функции аналогичны соответствующим стандартным функциям языка Фортран.

5.3.2. Функции редукции

К функциям редукции относятся функции:
SUM (сумма),
MULT (произведение),
MAX (максимум),
MIN (минимум).
Результат вычисления этих функций имеет тип **DOUBLE.**
Обращение к этим функциям имеет вид:
name-function **(** **(** *name-domain* **)** *arithm-expression* **)**.
Область определяет множество точек области, в которых необходимо вычислить функцию, *арифметическое выражение* – множество значений, к которым применяется функция.

Пусть необходимо вычислить: $V_i = W_i + \sum_{j=1}^{m} A_{i,j} * X_j, \quad i = 1,...,n.$

Фрагмент записи этого вычисления на языке НОРМА:
Grid: (Oi: (I1..N) ; Oj: (J=1..M)).
VARIABLE A DEFINED ON Grid.
VARIABLE V,W DEFINED ON Oi. VARIABLE X DEFINED ON Oj.
FOR Oi ASSUME V = W + SUM((OJ) A*X).
Пример использования вложенных функций редукции:
SurfaceMax:
(Left: (Three:(IV=1..3) ; A:(J=0..2*M); B:(I=0..2*N));

```
   Right: ( ( IK=1..9) ; (W=0..2*M) ; BNU:(NU=0..2*N) ) )
SUMPOL:( (W=0..2*M) ;BNU).
ThreeAB:(Three;A;B).   OLINE:(LINE=1..3).
VARIABLE VV,V DEFINED ON ThreeAB DOUBLE.
VARIABLE S      DEFINED ON SurfaceMax DOUBLE.
FOR ThreeAB ASSUME
        VV=SUM( (SUMPOL)
          SUM((OLINE)V[IV=LINE]*
          (V[IV=1]*S[IK=LINE]+
          V[IV=2]*S[IK=LINE+3]+
          V[IV=3]*S[IK=LINE+6]) ) ).
```

Для редукционных функций справедливо правило локализации индексов: область, указанная в качестве аргумента функции редукции, вводит свою систему индексов, область действия которой – арифметическое выражение, заданное в качестве второго аргумента функции редукции. Это означает, что индексы, используемые в арифметическом выражении, можно разделить на два типа: "внутренние", то есть совпадающие с индексами области редукции и определяемые ими, и "внешние", то есть не совпадающие с индексами области редукции и определяемые внешними областями (например, областями других функций редукции или оператора **ASSUME**). Множество значений "внутренних" индексов полностью определяется областью, заданной в качестве аргумента функции редукции. Множество значений "внешних" индексов определяется внешними по отношению к функции редукции областями.

Например, оператор **FOR Oik ASSUME X=(i+k) / SUM((Oij5) B*C)** описывает запрос на вычисление:

$$X_{i,k} = \frac{i+k}{\sum\limits_{i,j=1}^{5}(B_{i,j,k} * C_{i,j})} \quad i,k = 1,...,10$$

при наличии описаний:

```
Oij5 : ((i=1..5) ; (j=1..5)). Oik:  (i=1..10) ; (k=1..10)).
Oijk:  (Oij:((i=1..15) ; (j=1..15)) ;(k=1..15)).
VARIABLE B  DEFINED ON Oijk.
VARIABLE C DEFINED ON Oij.
VARIABLE X  DEFINED ON Oik.
```

Если внешняя область условная, то условие действует на все внешние индексы арифметического выражения, стоящего под знаком функции редукции. Например, формула

$$X_{i,j} = \sum_{i=1}^{100} B_{i,j} Y_j \quad i = 1,...,10, \quad j = 1,...,10, \quad i < j, \quad Y_i < 0,$$

задающая вычисление

$$Y_2 < 0: \quad X_{1,2} = \sum_{i=1}^{100} B_{i,2} Y_2 \quad \ldots \quad Y_{10} < 0: \quad X_{1,10} = \sum_{i=1}^{100} B_{i,10} Y_{10}$$

$$Y_3 < 0: \quad X_{2,3} = \sum_{i=1}^{100} B_{i,3} Y_3 \quad \ldots \quad Y_{10} < 0: \quad X_{2,10} = \sum_{i=1}^{100} B_{i,10} Y_{10}$$

. . . . . .

$$Y_{10} < 0: \quad X_{9,10} = \sum_{i=1}^{100} B_{i,10} Y_{10}$$

может быть записана следующим образом:

Oij: (Oi: (i=1..10); Oj: (1..10)) . OiLTj:Oij / i<j.
OiLTjYLTO, OiLTjYGEO:OiLTj / Y<0.
Oi100: (i=1..100). Oij100: (Oi100;Oj).
VARIABLE B DEFINED ON Oij100.
VARIABLE X DEFINED ON Oij.
VARIABLE Y DEFINED ON Oj.
FOR OiLTjYLTO ASSUME X=SUM((Oi100) B*Y).

В языке НОРМА запрещены рекурсивные вычисления переменных, используемых в левой части оператора **ASSUME**, и в то же время – в арифметическом выражении в функции редукции. Например, формула:

$$X_{i,j} = \sum_{\substack{i=1 \\ i<j}}^{10} X_{i,j} \quad i = 2,\ldots,10, \quad j = 1,\ldots,5$$

не может быть записана одним оператором **ASSUME** вида:

FOR Oij ASSUME X = SUM ((Oij1)X).

5.3.3. Внешние функции пользователя

Внешняя функция описывается пользователем. Обращение к ней представляется именем функции, за которым в круглых скобках следует список фактических параметров, разделенных запятыми. Результат вычисления функции представляется именем функции. Вычисление возможно, если все фактические параметры приняли соответствующие значения – все фактические параметры рассматриваются как исходные данные для вычисления функции, побочный эффект запрещен.

Фактическими параметрами внешней функции могут быть арифметические выражения, имена внешних разделов и внешних функций либо величины на областях (массивы).

Примеры использования внешней функции с различными способами задания фактических параметров:

Grid: (Oi: (I=1..N) ; Oj: (J=1..M)).
VARIABLE A, B DEFINED ON Grid.
VARIABLE X, Y DEFINED ON Oi.
VARIABLE T DEFINED ON Oj.

VARIABLE Gamma DOUBLE.
INPUT A,B ON Grid, Y ON Oi, T ON Oj. INPUT Gamma.
OUTPUT X ON Oi.
DOMAIN PARAMETERS N = 10, M = 20.
EXTERNAL FUNCTION DxDy.
FOR Oi ASSUME
X = F(DxDy, Gamma+0.5*Y, T ON Oj, A ON Grid / (i=2..8,J=M-2), B ON Grid /J=I+2).

Оператор **ASSUME** задает правила вычисления вектора X_i, i=1,10, при этом для каждого i необходимо вычислить значение функции пользователя F с параметрами:

1-ый параметр – имя *DxDy* внешней функции пользователя;

2-ой параметр – арифметическое выражение $Gamma+0.5*Y_i$;

3-ий параметр – значения $T_{j,j}=1,...,20$ – передается статический массив;

4-ый параметр – значения $A_{i,j}$, $i=2,...,8$, $j=18$ – передается статическая секция массива – вектор, состоящий из элементов 2-й–8-й строк 18-го столбца матрицы **A**;

5-ый параметр – значения $B_{i,j}$, $i=1,...,10, j=i+2$ – передается динамическая секция массива – i+2-й столбец матрицы **B**.

В таблице 3 показаны правила соответствия формальных и фактических параметров функции.

Таблица 3

Соответствие формальных и фактических параметров функции

Формальный параметр	Фактический параметр
имя функции	имя функции
имя раздела	имя раздела
скаляр	арифметическое выражение
величина на области	величина на области

Число формальных и фактических параметров должно быть одинаковым; соответствие устанавливается слева направо в порядке написания. Если и формальный и фактический параметры являются величинами на областях, то должны совпадать число индексов областей и число точек в диапазонах по соответствующим индексам (см. п. 5.1.2).

5.4. Итерация

iteration:
head-of-iteration
[boundary-value]

initial-value
body-of-iteration
exit-condition
end-iteration
head-of-iteration:
ITERATION list-iterated-element **ON** name-iteration-index
iterated-element:
name-variable [**(** list-name-result **)**]
boundary-value:
BOUNDARY { operator **.**}$^{+}$ **END BOUNDARY**
initial-value:
INITIAL name-iteration-index = 0 **:** { element-of-initial **.**}$^{+}$
END INITIAL
element-of-initial:
operator
declaration-of-input
declaration-of-output
body-of-iteration:
{ element-of-iteration-body **.** }$^{+}$
element-of-iteration-body:
operator
iteration
declaration-of-output
exit-condition:
EXIT WHEN log-expression
end-iteration:
END ITERATION name-iteration-index

При решении задач математической физики разностными методами процесс вычислений часто носит итеративный характер. Такие процессы можно задавать с помощью уже описанных средств языка НОРМА. Для этого необходимо изменить (расширить) описание областей, определенных в задаче, добавив дополнительное направление, соответствующее индексу итерации. Однако такое расширение не всегда оправдано, так как это направление, по существу, является фиктивным, оно отражает способ вычисления, а не пространственно-временную сетку. Кроме этого, для такого фиктивного направления не всегда известны границы диапазона.

Специальная конструкция **ITERATION** позволяет задать итеративный вычислительный процесс и обойти при этом отмеченные выше трудности.

Неформально: итерация задает итеративные вычисления с индексом итерации, меняющимся от 0 до некоторого целого положительного значения, которое определяется условием завершения итерации.

Рассмотрим итеративный вычислительный процесс, связанный с решением системы уравнений:

$$\sum_{j=1}^{m} A_{i,j} X_j = F_i, \quad i = 1,\ldots,m$$

и задаваемый формулами:

$$X_i^{n+1} = \frac{1}{A_{i,i}}(F_i - \sum_{i \neq j} A_{i,j} X_j^n). \quad X_i^0 = X0, \quad i = 1,\ldots,m$$

условие завершения:

$$\left\| X^n - X^{n-1} \right\| < \varepsilon$$

На языке НОРМА этот процесс описывается следующим образом:

```
// Описание областей и переменных на областях
Array :  (Oi:  (I=1..M) ;  Oj :  (J=1..M)) . O1,O2:Oj / I<>J.
VARIABLE X0,X,F DEFINED ON Oi.
VARIABLE A DEFINED ON Array.
VARIABLE Epsilon.
DOMAIN PARAMETERS M = 100.
// Ввод исходных данных для величин X0, A, Epsilon
INPUT X0 ON Oi, A ON Array. INPUT Epsilon.
// Вывод результатов X, Xpred
OUTPUT X, Xpred ON Oi.
// Итерация
ITERATION X (Xpred) ON N.
INITIAL N=0 :
FOR Oi ASSUME X = X0.
END INITIAL
FOR Oi ASSUME  X = 1/A[J=I]*(F-SUM( (O1)A*X[I=J,N-1]).
EXIT WHEN MAX( (Oi)  ABS(X[N] - X[N-1] ) ) < Epsilon.
END ITERATION N.
```

Собственно итерация представлена последними 7 строками примера. В заголовке итерации указывается индекс итерации – в данном случае **N**, и перечисляются величины, участвующие в итеративных вычислениях и представляющие результат этих вычислений – в данном случае это **X** (значение с последнего шага итерации) и **Xpred** (значение с предпоследнего шага итерации). Величина **Xpred** не требует дополнительного описания: считается, что она описана так же, как и **X**. Значения **X**, как и **Xpred,** могут использоваться для вычислений вне итерации; в данном примере эти значения просто объявлены выходными.

При описании итерации можно использовать возможность задания граничных значений итерируемых величин (вообще говоря, необязательных). Граничные значения итерируемой переменной задаются при помощи обычных операторов языка НОРМА внутри блока **BOUNDARY ... END BOUNDARY**. Эти значения считаются неизменными на протяжении всей итерации и определены на каждом шаге итерации.

Начальные значения для итерируемых величин задаются при помощи блока **INITIAL...END INITIAL**.

Тело итерации представляет собой часть программы, написанной на языке НОРМА. В частности, внутри итерации можно определить новую итерацию по другому направлению. В примере тело итерации состоит из единственного оператора **ASSUME**.

Индекс итерации может использоваться в списке индексов, указываемом для итерируемой величины, т.е. фактически можно считать, что итерируемая величина в пределах итерации имеет дополнительный итерационный индекс (по фиктивному направлению). Индекс итерации без смещения можно не указывать.

Индекс итерации может быть указан только у итерируемых величин.

В теле конструкции **ITERATION** могут вычисляться значения величин, не указанных в списке итерируемых величин. Такие величины могут потребоваться, например, для представления промежуточных результатов, вычисляемых на каждом из витков итерации. Использование таких величин не приводит к нарушению правила об однократном присваивании – считается, что на каждом витке итерации используется новая копия величины.

Итерационный процесс завершается, если логическое выражение, заданное в условии выхода, принимает значение "истина".

5.5. Интерфейс с программами, написанными на целевом языке программирования

Подпрограммы и функции, написанные на языках Си или Фортран, могут быть вызваны из программы, написанной на языке НОРМА, при помощи обычных средств вызова разделов и обращения к функциям. При этом осуществляется контроль переприсваивания при анализе фактических параметров. Например, подпрограмма на языке Фортран:

```
SUBROUTINE SINXY(Y,N,X)
REAL X(N), Y(N)
DO 1 I=1,N
X(I) = SIN(Y(I))
1  CONTINUE
RETURN
END
```

может быть вызвана следующим образом:

```
Oi  : ( I=1..N ).
VARIABLE X, Y DEFINED ON Oi.
INPUT Y ON Oi.
EXTERNAL PART SINXY.
DOMAIN PARAMETERS N = 10.
COMPUTE SINXY(Y ON Oi, N RESULT X ON Oi).
```

5.6. Режим последовательного вычисления

При программной реализации вычислительных алгоритмов часто порядок выполнения операций (операторов) бывает существенным: от этого порядка может зависеть сходимость, устойчивость метода решения задачи и т.п. В языке НОРМА определена возможность задания режима последовательного вычисления, что дает возможность пользователю фиксировать необходимый ему порядок выполнения операторов. Для этой цели используются конструкция блока # ... #. Операторы, заключенные в блоке между # ... #, выполняются в порядке их следования в программе, при этом проверяется корректность последовательности операторов с точки зрения языка НОРМА: использование неопределенных значений и т.п.

Например, следующая запись:

```
#
X=5.0 .
Z=SIN(X+0.5) .
Y=COS(X)-Z*X.
#
```

является корректной,

а запись:

```
#
X=5.0.
Y=COS(X)-Z*X .
Z=SIN(X+0.5) .
#
```

некорректна, так как во втором операторе используется неопределенное значение величины **Z**.

Если указатели режима последовательного вычисления убрать, то обе записи эквивалентны и корректны, так как компилятор сам определит корректный порядок выполнения операторов, что в данном случае возможно.

Важно отметить, что внутри блока # ... # допускается переприсваивание значений переменным, что не является ошибочным в данном случае. Эта возможность может оказаться полезной при написании программы, когда программист сам оценивает последствия переприсваивания и сознательно его осуществляет. Наличие переприсваивания часто не влияет на распараллеливание, но может, например, повышать эффективность программы или экономить память. Такая возможность и предоставляется программисту.

Например, в следующем примере в операторе **ASSUME** производится переприсваивание переменной **X:**

```
Matrix: ( (I=1..5) ; (j=1..5) ).
VARIABLE X,Y,Z DEFINED ON Matrix.
FOR Matrix ASSUME Y = 1; Z = 2.
#
```

```
//ниже – корректное переприсваивание
FOR Matrix ASSUME X = Y; X = X + Z.
//ниже – корректное переприсваивание
FOR Matrix ASSUME X = Y+1.
#
```

Режим параллельного вычисления в последовательном блоке

В языке НОРМА определена возможность внутри блока # ... # задавать режим параллельных вычислений. Для этой цели используются блок @ ... @. Операторы, заключенные в этом блоке (в блоке @ ...@ и только в этом блоке), трактуются как параллельные, и компилятор может реализовать их параллельно. Внутри блока @ ... @ задавать режим последовательных вычислений запрещается.

Пример:

```
#
@
FOR oN/(i=2..N-1, j=2..N-1, k=1) ASSUME u = u[k=N-1].
FOR oN/(i=2..N-1, j=2..N-1, k=N) ASSUME u = u[k=2].
@
@
FOR oN/(i=2..N-1, j=1) ASSUME u = u[j=N-1].
FOR oN/(i=2..N-1, j=N) ASSUME u = u[j=2].
@
@
FOR oN/i=1 ASSUME u = u[i=N-1].
FOR oN/i=N ASSUME u = u[i=2].
@
#
```

В этом примере в рамках последовательного блока (#.....#) с помощью @...@ выделены три участка программы. Сами участки выполняются последовательно, но внутри каждого участка операторы между собой могут исполняться параллельно.

5.7. Директивы компиляции

5.7.1. Директива INCLUDE

include:
 INCLUDE "file-name"
file-name:
 { symbol-not-apostrophe }$^{+}$

Директива **INCLUDE** является указанием компилятору для включения в текст программы (непосредственно вместо этой директивы) исходного файла, имя которого указывается в кавычках после директивы.

Например:

INCLUDE "npbparams.hin".

5.7.2. Директивы условной компиляции

Директивы условной компиляции **#if**, **#else, #endif** дают возможность в зависимости от значения выражения включать или исключать те или иные участки кода программы. В общем виде директива **#if** выглядит таким образом:

#if(<формула>) … #else … #endif.

В <формула> поддерживаются термы:

__SEQ
__OPENMP
__CUDA
__MPI
__FORTRAN
__C

и логические операции с термами: AND, OR, NOT.

Значение терма (истина или ложь) определяется заданным при компиляции типом выходной программы для различных параллельных архитектур и используемого выходного языка.

Пример использования директив:

```
#if(__CUDA)
COMPUTE conj_grad_HF1_CUDA(rowstr ON oNA1, a ON oNZ,
        z ON oNA, colidx ON oNZ RESULT q ON oLRminFRp1).
#else
FOR oLRminFRp1 ASSUME q = conj_grad_HF1(rowstr[i = i],
        rowstr[i = i+1], a ON oNZ, z ON oNA, colidx ON oNZ).
#endif
```

Если программа компилируется для архитектуры технологии NVIDIA CUDA, то формируется код, где производится вызов раздела **conj_grad_HF1_CUDA,** для архитектур других типов формируется код, где производится выполнение оператора присваивания **ASSUME.**

Заключение

Активные исследования и новые результаты в области архитектуры компьютерных систем непрерывно повышают вычислительную мощность этих систем. Однако использование новых возможностей, связанное с созданием эффективных параллельных программ, до сих пор остается сложным делом, требующим высокой квалификации программиста. Попытки автоматизации разработки параллельных программ сталкиваются с трудностями, часто наталкиваясь, в достаточно общих постановках, на математически неразрешимые проблемы.

Язык НОРМА и подход к разработке параллельных программ, основанный на его применении, являются исследованием в этой сложной области автоматизации. Авторы отдают себе отчет в том, какие трудности возникают при решении этой фундаментальной задачи. Тем не менее авторы считают данное направление исследований важным, перспективным и приносящим пользу уже сейчас, о чем свидетельствуют уже полученные результаты решения ряда вычислительных задач с помощью системы НОРМА.

Литература

1. **Задыхайло И.Б.** Организация циклического процесса счета по параметрической записи специального вида // Журн. выч. мат. и мат. физ., №2, 1963, с.337-357.

2. **Андрианов А.Н., Ефимкин К.Н., Задыхайло И.Б.** Непроцедурный язык для решения задач математической физики // Программирование, №2, 1991, с.80-95.

3 **Ефимкин К.Н., Задыхайло И.Б.** Содержательные обозначения и языки нового поколения // Информационные технологии и вычислительные системы, №2, 1996, с.46-58.

4. **Андрианов А.Н., Бугеря А.Б., Гладкова Е.Н., Ефимкин К.Н., Колударов П.И.** Простые вещи // Суперкомпьютеры, № 2(18), 2014, с.58–61.

5. Система НОРМА URL: http://www.keldysh.ru/pages/norma

6. **Андрианов А.Н., Бугеря А.Б., Ефимкин К.Н., Колударов П.И.** Модульная архитектура компилятора языка Норма+ // Препринты ИПМ им. М.В. Келдыша РАН. М., 2011. № 64. 16 с.

7. **Андрианов А.Н., Андрианова Е.А.** Организация циклического процесса по непроцедурной записи // Программирование, №4, 1996, с.62-72.

8. **Андрианов А.Н.** Синтез параллельных и векторных программ по непроцедурной записи в языке Норма. Дис. канд. физ.-мат. наук, М., 1990. 144 с.

9. **Андрианов А.Н.** Система Норма. Разработка, реализация и использование для решения задач математической физики на параллельных ЭВМ. Автореф. дис. докт. физ.-мат. наук. М., 2001.

10. **Васильев М.М., Ефимкин К.Н., Иванова В.Н.** О применении метода гидродинамических потенциалов к задаче обтекания тела вязкой жидкостью // Математическое моделирование, №10, Т.6, 1994, с.57–65.

11. **Андрианов А.Н., Гусева Г.Н., Задыхайло И.Б.** Применение языка Норма для расчета дозвукового течения вязкого газа // Математическое моделирование, №9, Т.11, 1999, с.45-53.

12. **Андрианов А.Н., Жохова А.В., Четверушкин Б.Н.** Использование параллельных алгоритмов для расчетов газодинамических течений на нерегулярных сетках. В сб. Прикладная математика и информатика, М.: Изд. факультета ВМиК МГУ, №4, 2000, с.68-76.

13. **Andrianov A.N., Efimkin K.N., Levashov V.Y., Shishkova I.N.** The Norma language application to solution of strong nonequilibrium transfer process problem with condensation of mixtures on the multiprocessor system. Computational Science - ICCS 2001, Lecture Notes in Computer Science, v.2073, p.502-510, May, 2001.

14. **Андрианов А.Н.** Применение языка Норма для решения задач на вложенных сетках // Вычислительные методы и программирование, №1, Т.3, 2002, с.103-112.

15. **Андрианов А.Н., Березин А.В., Воронцов А.С., Ефимкин К.Н., Марков М.Б.** Моделирование электромагнитных полей радиационного происхождения на многопроцессорных вычислительных системах // Математическое моделирование, №3, Т.20, 2008, с.98–114.

16. **Андрианов А.Н., Бугеря А.Б., Ефимкин К.Н., Задыхайло И.Б.** НОРМА. Описание языка. Рабочий стандарт // Препринт ИПМ им. М.В. Келдыша РАН. М., 1995. №120. 52 с.