УДК 518.5

**В. Я. ИВАНОВ**

*(Новосибирск)*

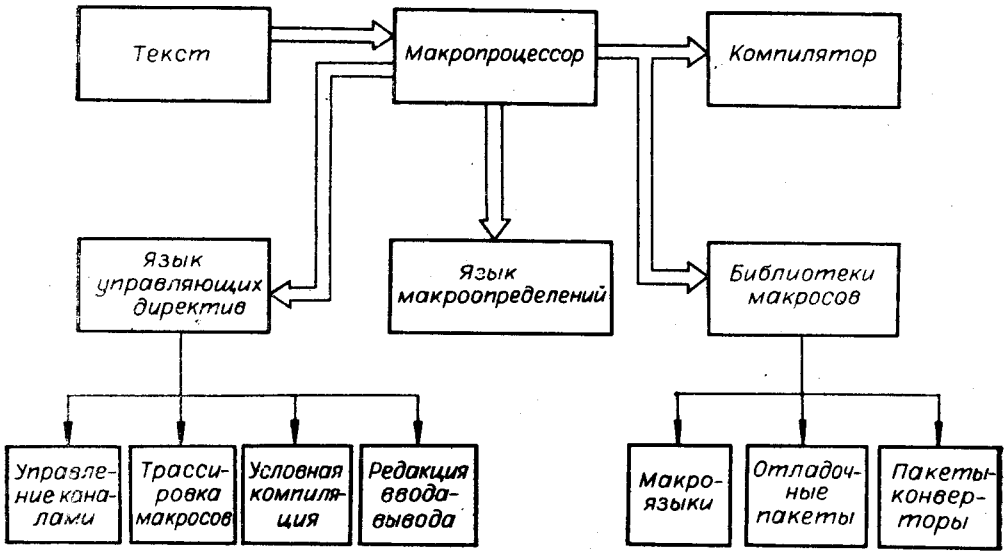
# ГЕНЕРАЦИЯ ПАКЕТОВ ПРОГРАММ С ДИНАМИЧЕСКОЙ СТРУКТУРОЙ НА ОСНОВЕ МАКРОПРОЦЕССОРА

**Введение.** Развитый пакет прикладных программ (ППП) высокого уровня характеризуется весьма сложным межмодульным интерфейсом, т. е. связями по управлению и данным. Как правило, он включает диалоговые средства, машинную графику, систему управления базами данных, средства документирования и т. д. Стремление к универсализации пакета, расширению его функциональных возможностей в известной мере вступает в противоречие с такими его свойствами, как гибкость, простота управления, экономичность использования оперативной памяти и быстродействие алгоритмов пакета.

Одним из наиболее известных средств преодоления этих противоречий является использование аппарата планирования вычислительного процесса. При этом до начала работы численных алгоритмов выполняется фаза генерации с помощью специализированных системных компонент: ФОРМИРОВАТЕЛЬ паспорта, ПРЕОБРАЗОВАТЕЛЬ модуля, АНАЛИЗАТОР управляющей программы и т. д. На этапе исполнения взаимодействие между модулями, выполняющими вычислительные функции, осуществляют другие системные компоненты — МОНИТОРЫ передачи управления и данных, называемые также планировщиками вычислительного процесса. Они анализируют динамическую обстановку, осуществляют распределение необходимых для следующего этапа ресурсов, загружают в оперативную память модули из сформированной перед решением задачи библиотеки и осуществляют передачу управления на начало подготовленной цепочки динамических вызовов. Такая схема работы системных компонент описана в наиболее известных работах [1—4]. Важно отметить, что при таком подходе эффективность работы системы планирования вычислений достаточно высока, но специализация системных компонент, в которых алгоритмы генерации жестко встроены в конкретную реализацию, значительно затрудняет адаптируемость программного обеспечения к иным ППП, новому парку ЭВМ, другому штатному математическому обеспечению.

Преимущества открытой для пользователя системы планирования проще всего реализовать на основе некоторых стандартных средств, обеспечивающих их мобильность при адаптации ППП силами самих разработчиков, зачастую не являющихся системными программистами. В данной работе предлагается решение этой задачи на базе универсального макропроцессора, реализованного автором [5] в мониторной системе «Дубна» для БЭСМ-6 и в ОС ЕС на языке ФОРТРАН.

В любом сложном программном комплексе можно выделить два класса модулей: вычислительные, объем которых в задачах математической физики составляет обычно 80—90% от общего объема текстов или программного кода, и управляющие, незначительные по объему. В управляющих модулях сосредоточены логика управления вычислительным процессом, генерация структур данных и обмены с внешней средой. Описываемая в работе технология позволяет обходиться без написания специализированной сложной организующей программы-монитора для реализации динамики взаимодействия модулей. После анализа входных данных на проблемно-ориентированных языках [7, 8] осуществляется генерация рабочих текстов управляющих модулей с фиксацией динамических цепочек вызовов нужных модулей и резервированием необходимых для данной вычислительной фазы элементов данных и опе-




раторов обмена с внешней средой. Эту задачу выполняет универсальный макропроцессор, который не имеет специализированного языка управления генерацией пакетов, а использует стандартные средства условной компиляции. При написании алгоритмов ППП на языках высокого уровня универсальный характер использования средств генерации позволяет без значительных усилий адаптировать их для других ЭВМ и пакетов независимо от языка их реализации.

**Макросредства системы ТОПАЗ.** Общая архитектура макросредств системы автоматизированного проектирования [6] изображена на рисунке. Она включает:

макропроцессор — программу, преобразующую тексты исходного макроязыка в тексты некоторого другого языка, обрабатываемые транслятором, имеющимся в системе штатного математического обеспечения;

язык управления — набор директив, определяющих режимы работы процессора, назначение каналов ввода-вывода, форматы редактирования и размещения выходных текстов;

язык макроопределений — совокупность формальных правил написания текстов макроопределений;

библиотеки макроопределений — наборы макросов, определяющих правила преобразования текстов, поступающих на вход макропроцессора, в том числе и текстов самих макроопределений;

макроязыки. В том случае, когда богатство изобразительных возможностей библиотеки макроопределений позволяет существенным образом изменить основные конструкции базового языка, имеет смысл говорить о возникновении нового языка — макроязыка со своим синтаксисом. Так, например, описанный в [5] язык МОРТРАН вовсе не является простым расширением языка ФОРТРАН. Более всего он напоминает алголоподобный язык с блочной структурой составных операторов, с многочисленными модификациями операторов цикла, условных предложений и операторов обмена.

При разработке данного макропроцессора за основу была взята реализация Кука и Шустека, дополненная средствами взаимодействия с мониторной системой «Дубна», отладочными пакетами и пакетами-конверторами для решения проблем мобильности программного обеспечения. Макропроцессор написан на языке ФОРТРАН и оформлен в виде подпрограммы, которая может запускаться самостоятельно или из программы пользователя.

Для конструирования пользователем собственных макроопределений необходимо знать правила, которым подчиняется их написание. Идентифицирующим признаком макроопределения является символ «%» в первой позиции строки и кавычка или апостроф — во второй. Само макроопределение состоит из левой части, называемой источником или образом, и правой, называемой замещением. Обе части макроса разделены символом «=» и представляют собой некоторые строки, т. е. цепочки символов, заключенные с обеих сторон в апострофы. Макроопределение может быть продолжено на следующую строку; несколько макроопределений может располагаться на одной строке, если это необходимо. При употреблении вложенных строк апострофы внутри них дублируются.

Простейший пример макроопределения имеет вид %'SIZE' = '100'. Если в последующих строках входного текста употребить операторы типа DIMENSION X(SIZE); ... DO J = 1, SIZE<...>..., то после обработки макропроцессором будут сгенерированы строки выходного текста DIMENSION X(100); ... DO J = 1, 100<...>.... Если мы хотим, чтобы макроподстановка происходила правильно для тех контекстов, в которых отдельные фрагменты левой части макроопределения могут быть разделены любым количеством пробелов, то в соответствующем месте макроопределения нужно употребить один пробел. Например, макрос %'DUMP ␣ X;' = 'OUTPUT ␣ X; (F10.2);' будет соответствовать контексту DUMP ␣ X;, а не Y = DUMPX;.

Наиболее интересные свойства и применения макросов возникают при введении понятия «параметры». Формальные параметры макроопределения, обозначаемые символом # в левой части правила, при выполнении макроподстановки ставятся в соответствие фактическим параметрам, в качестве которых могут выступать любые цепочки символов подходящего контекста. Будем считать, что параметры левой части правила нумеруются в естественном порядке. Порядок использования формальных параметров в правой части может не соответствовать порядку их употребления в левой, поэтому параметры в правой части изображаются символами # *i*, где число *i* может принимать значения 1, 2, ..., 9. Рассмотрим пример макроопределения %'PLUS ␣ #;' = '# 1 = # 1 + 1;'. Для него макровызов PLUS A(I, J, K); порождает оператор A(I, J, K)= = A(I, J, K) + 1;. Для цепочек символов, выступающих в качестве фактических параметров, существуют следующие ограничения: не должен включаться символ конца оператора ';'; круглые и угловые скобки <,> должны быть сбалансированы; если в фактическом параметре встретился апостроф, открывающий строку, то вся строка целиком вместе с закрывающим апострофом должна входить в параметр.

Для успешного использования макроопределений, придумываемых пользователем, необходимо хотя бы в общих чертах понимать, как работает макропроцессор. Мы приведем краткое описание этой процедуры. Подаваемая на вход строка «чистится»: из нее удаляются комментарии, последовательность пробелов заменяется на один пробел всюду, за исключением строк, ограниченных апострофами. Очищенная строка подается на буфер макрорасширений. Предположим, что стек макроопределений процессора не пуст. Тогда, начиная с первого символа буфера макрорасширений, инициализируется процесс сравнения контекста с левыми частями набора имеющихся макросов. Если сравнение прошло успешно, проводится замена текста на правую часть подходящего макроопределения и процесс сравнения начинается заново с первого символа расширенного текста. Если же подходящей замены не нашлось после проверки всех макросов, первый символ буфера макрорасширений помещается в выходной буфер и процесс продолжается со следующего символа. Если встретился символ '″', весь текст до следующего такого же символа попадает в выходной буфер и не просматривается на предмет замен. Если в левой части макроса присутствует пробел, то соответствующий пробел должен присутствовать в контексте буфера макрорасширений

для успешной замены, Если же в левой части макроса нет пробела, то пробелы во входном тексте при проверках будут игнорироваться, что ускоряет работу процессора.

Так как макроопределения по мере их ввода поступают в стек, немаловажное значение имеет порядок их следования. Пользователь может учитывать это обстоятельство, размещая свои макроопределения раньше макросов стандартного библиотечного набора, чтобы блокировать нежелательные для его программы конфликтные ситуации.

Проиллюстрируем механизм генерации макросов на примере использования отладочных средств. Примерно таким же образом, как это описано ниже, был реализован отладочный пакет. Пусть для отладки необходимо выдавать дамп переменной X, которая может в различных фрагментах программы встречаться в левой части оператора присваивания. Скорее всего, мы напишем макрос

%';X = #;' = '"; X" = #1; OUTPUT ⌴ X; ("X ⌴ ASSIGNED", E 12.5);'.

Точка с запятой употреблена, чтобы не спутать контекст «X=» с «PX=» или каким-либо другим. Второй символ ';' — ограничитель параметров, кавычки '"' употребляются во избежание зацикливания, а двойные апострофы — для кодирования символа апострофа, расположенного внутри строки-замещения.

Хотя этот макрос и может быть полезным, при необходимости трассировки какой-либо другой переменной, например Y, нужно писать другой макрос. Естественно попытаться сделать имя переменной параметром макроса, а для этого употребляемый нами макрос должен генерировать собственный макрос трассировки:

%'; TRACE #; '=' %"; # 1 = ##; "="""; # 1" = ## 1; OUTPUT # 1;
(""" # 1 ⌴ ASSIGNED"", E 12.5);"'.

Теперь можно писать TRACE X или TRACE Y. Единственный новый для нас элемент — это указание параметра порождаемого нами макроса в виде пары символов ##.

Часто возникает необходимость получения различных версий одной и той же программы, отличающихся только некоторыми вполне определенными свойствами. Здесь наиболее полезным может оказаться механизм условной компиляции. Альтернативные фрагменты программных текстов нужно заключить в уникальные ограничители, т. е. цепочки символов, не пересекающиеся с другими цепочками, встречающимися в программе, например,

/MULTI/ DIMENSION A(10); /ENDMUL/
/DOUBLE/ DOUBLE PRECISION A; /ENDDOUB/

Если мы не хотим использовать вариант программы с двойной точностью, необходимо произвести замену соответствующих ограничителей фрагментов по всей программе на ключевое слово NOGENERATE в начале программы и ENDGENERATE — в конце, а для нужных фрагментов — начальный ограничитель на слово GENERATE. В этом нам помогут макросы

%'/MULTI/' = 'GENERATE' %'/DOUBLE/' = 'NOGENERATE'
%'/ENDMUL/' = 'ENDGENERATE' %'/ENDDOUB/' = 'ENDGENERATE'

Сегменты условной компиляции могут быть вложенными, что позволяет реализовать условия выбора сегментов, не являющихся взаимно исключающими. В этом случае, если не генерируется объемлющий сегмент, не будут генерироваться и все сегменты, вложенные в него.

**Реализация динамической структуры пакетов прикладных программ.** Для иллюстрации механизма работы средств генерации ППП остановимся на рассмотрении двух характерных примеров решения этих проблем в системе автоматизированного проектирования ТОПАЗ [6]. Первый

Таблица 1

**Варианты структур данных пакета POISSON-2**

| Назначение структуры, имя | Альтернативные варианты | | | |
|---|---|---|---|---|
| Ионизационные процессы COMMON /BUFION/, /STRION/ | Нет | Модель дерева реакций | Каскадная модель | Комбинированная модель |
| Объемный заряд частиц COMMON /Q/ | Нет | | Есть | |
| Осевое магнитное поле COMMON /B/ | Нет | Массив | Константа | Функция |
| Магнитное поле соленоидов COMMON /H/ | Нет | Массив | Функция | |
| Магнитное поле пучка частиц COMMON /BEAM/ | Нет | Компонента азимутальная BF | Все компоненты BR, BZ, BF | |
| Вторичная эмиссия COMMON /BTOP/ | Нет | | Есть | |
| Расчет эквипотенциалей COMMON /EKV/ | » | | » | |
| Машинная графика COMMON /GRAPH/ | » | | » | |
| Представление границы сплайном COMMON /SPLINE/ | » | | » | |
| Задача Робэна COMMON /ROB/ | » | | » | |

пример отражает способы генерации структур данных. Возможные варианты этих структур приведены в табл. 1.

«По умолчанию» файл генерации для всех этих структур содержит параметр NOGENERATE, а на основе анализа входных данных для некоторых из них в файл будут занесены макросы замены меток этих структур на слово GENERATE. Поскольку сами макросы представляют собой текстовые константы, модуль генерации состоит из примитивных операторов присваивания и может быть написан даже неквалифицированным пользователем, знающим основы языков ФОРТРАН или АЛГОЛ. Таким образом, в простейшем случае решается вопрос о том, оставить ли структуру данных или исключить ее полностью. В более сложных случаях структура присутствует в модуле всегда, но содержит объекты переменной размерности. Пусть, например, размерность некоторых массивов описывается идентификатором COMMON/A/B(N), C(N), ..., а конкретное значение этой размерности, определяемое входными данными, равно 8500. Новым элементом здесь является то, что в текст генерируемого макроопределения должно быть вставлено текстовое представление значения целой переменной, которое выдаст подпрограмма-функция TEXT(N). Занесение в стек макроопределений макроса %'N' = '§' (§ обозначает текстовое представление N) с позиции K на ФОРТРАНе реализуется так:

```
STEK(K) = 6H%'N' = '
STEK(K + 1) = TEXT(N)
STEK(K + 2) = 1H'
K = K + 3.
```

Таблица 2

**Варианты моделей уравнений движения частиц**

| Признак классификации | Возможные варианты | |
|---|---|---|
| Система координат | $x, y$; $x, y, z$; $r, z$; $r, z, \theta$ | |
| Наличие релятивизма | $\gamma \approx 1$, $\gamma \gg 1$ | |
| Вид магнитного поля | $\mathbf{B} = 0$, $\mathbf{B} = \mathbf{B}_0$, $\mathbf{B}$ — функция | |
| Наличие электрического поля | $\mathbf{E} = 0$, $\mathbf{E} \neq 0$ | |
| Наличие гравитации | $\mathbf{g} = 0$, $\mathbf{g} = (0, g_z)$, $\mathbf{g} = (g_x, g_y, g_z)$ | |
| Интегрирование азимутальной координаты $\theta$ | Закон Буша | Численное |
| Алгоритм интегрирования траекторий | Метод Эйлера | Схема Рунге — Кутта |
| Диффузионное приближение в магнитном поле | Отсутствует | Инвариант $p_\perp^2/B$ |
| Диффузионное приближение в электрическом поле с учетом столкновений | Отсутствует | $\vartheta \sim \sqrt{Ez/Mn\sigma e}$ |
| Наличие стоксового трения | $\eta = 0$ | $\eta \neq 0$ |

Следующий пример отражает способ генерации структур управления. Одной из типовых задач пакета POISSON-2 [9] является расчет траекторий заряженных частиц в комбинированных полях электромагнитных, гравитационных и гидродинамических сил. Наиболее общая запись уравнений движения имеет вид

$$d\mathbf{p}/dt = Z\mathbf{E} + (Z/c)[\mathbf{vB}] + M\mathbf{g} - \eta(\mathbf{v} - \mathbf{v}_0)^\alpha,$$

где $\mathbf{p}$ — импульс частицы; $\mathbf{v}$ — ее скорость; $Z$ — заряд; $M$ — масса; $c$ — скорость света; $\mathbf{E}$ — напряженность электрического поля; $\mathbf{B}$ — индукция магнитного поля; $\mathbf{g}$ — ускорение свободного падения; $\eta$ — коэффициент трения Стокса; $\mathbf{v}_0$ — скорость движения среды, в которой распространяется пучок макрочастиц; $\alpha$ — показатель, определяющий параметр нелинейности модели сил трения; $t$ — время как параметр движения частицы. При движении элементарных частиц определяющими являются электромагнитные силы, а в случае распространения пучка мелкодисперсных макрочастиц в струе газа — электрические, гравитационные и гидродинамические. Альтернативные варианты уравнений движения отражены в табл. 2.

Их вид определяется характером приближения физической модели, выбором системы координат (плоскопараллельная, осесимметричная или трехмерная), наличием тех или иных видов полей и схемой численного интегрирования. В осесимметричном случае при движении в магнитном поле существует интеграл движения, описываемый теоремой Буша, и азимутальная компонента импульса не интегрируется, а вычисляется по формуле

$$p_\theta = [Z(\Psi - \Psi_0)/2\pi + r_0 p_{\theta_0}]/r,$$

где $\Psi$ — магнитный поток через площадь круга радиусом $r$; $\Psi_0$, $r_0$, $p_{\theta_0}$ — начальные данные траектории.

При движении частицы в электрическом поле с учетом столкновений, если длина свободного пробега много меньше характерных размеров системы, используется диффузионное приближение, в котором усредненная по столкновениям скорость движения определяется по формуле

$$\mathbf{v} = \mathbf{E}\sqrt{Z/EMn\sigma}.$$

Здесь $n$ — плотность среды, а $\sigma$ — суммарное сечение столкновительных процессов. Если частица движется в сильном магнитном поле, где $|cp_\perp \nabla B/ZB| \ll B$, то используется диффузионное приближение для вычисления координат ведущего центра с использованием интеграла движения — адиабатического инварианта $p_\perp^2/B = \text{const}$, где $p_\perp$ — поперечная к направлению магнитного поля компонента импульса.

При решении одной из характерных задач с учетом объемного заряда проводилось около 10 итераций по объемному заряду, на каждой из которых рассчитывалось около 100 траекторий, содержащих по 50 точек. Для расчета параметров движения частицы на один шаг вызывается модуль STEP, реализующий метод Эйлера 2-го порядка точности или модуль РК3 — трехточечная схема Рунге — Кутта 4-го порядка точности. Таким образом, эти модули являются наиболее часто используемыми, а каждый их вызов влечет за собой около 20 проверок различного рода условий, которые не меняются динамически, а определяются входными данными. Многолетняя практика эксплуатации пакета POISSON-2 показала, что наиболее сложный вариант, включающий учет всех возможных членов в уравнениях движения, не встретился ни разу. Следовательно, в среднем ППП со статическим текстом модуля работает неэффективно как по быстродействию, так и по использованию оперативной памяти. Иметь же все возможные варианты уравнений движения в виде отдельных модулей практически немыслимо. Полный текст модуля STEP составляет 108 выполняемых операторов языка ФОРТРАН, а его программный код занимает 386 машинных слов. При расчете одной из наиболее характерных задач — движении нерелятивистской частицы в электрическом поле — на вход макропроцессора подается единственная директива %'EFIELD' = 'GENERATE'. Окончательный текст генерируемого модуля содержит 23 оператора, а программный код составляет 57 слов. Время расчета одного шага траектории уменьшается на 15%, а суммарное время работы макропроцессора и транслятора с ФОРТРАНа составило 1,2 с.

Общий итог всех вышеприведенных рассуждений состоит в следующем. Разработка и эксплуатация больших и сложноорганизованных программных комплексов неэффективна без квалифицированного решения проблемы компоновки или генерации управляющих модулей. В то же время из всех возможных инструментальных средств наименее трудоемким является использование макропроцессоров. Так, например, разработка макросредств автором настоящей работы заняла 3 человеко-месяца.

## ЛИТЕРАТУРА


1. **Загацкий Б. А., Савочкина О. А., Темноева Т. А.** Планирование вычислительного процесса в системе ФИХАР.— В кн.: Системное программирование.— Новосибирск: изд. ВЦ СО АН СССР, 1973.
2. **Кахро М. И., Мяннисалу М. А., Саан Ю. П., Тыугу Э. Х.** Система программирования ПРИЗ.— Программирование, 1976, № 1.
3. **Воеводин В. В., Гайсарян С. С., Кабанов М. И.** Автоматизированная генерация программ.— В кн.: Численный анализ на ФОРТРАНе.— М.: изд. ВЦ МГУ, 1973.
4. **Воронков А. В. и др.** Система обеспечения комплексов программ математической физики.— М.: изд. ИПМ АН СССР, 1979. (Препринт/АН СССР, ИМП; № 49).
5. **Иванов В. Я.** Универсальный макропроцессор.— Новосибирск: изд. ВЦ СО АН СССР, 1981. (Препринт/АН СССР, Сиб. отд-ние, ВЦ; № 277).

6. **Иванов В. Я.** Информационное обеспечение системы «ТОПАЗ».— В кн.: Численные методы решения задач электронной оптики.— Новосибирск: изд. ВЦ СО АН СССР, 1979.
7. **Иванов В. Я.** Входные языки системы «ТОПАЗ».— Новосибирск: изд. ВЦ СО АН СССР, 1979. (Препринт/АН СССР, Сиб. отд-ние, ВЦ; № 154).
8. **Иванов В. Я.** Проблемно-ориентированный язык описания данных для экстремальных задач электронной оптики.— Автометрия, 1980, № 3.
9. **Астрелин В. Т., Иванов В. Я.** Пакет программ для расчета характеристик интенсивных пучков релятивистских заряженных частиц.— Автометрия, 1980, № 3.

*Поступила в редакцию 10 августа 1981 г.*

---

УДК 621.384 : 663.0015

**В. Т. АСТРЕЛИН, В. Я. ИВАНОВ**

*(Новосибирск)*

# ЧИСЛЕННОЕ МОДЕЛИРОВАНИЕ СТОЛКНОВИТЕЛЬНЫХ ПРОЦЕССОВ В УСКОРИТЕЛЯХ РЭП

**1. Введение.** При построении математических моделей сильноточных релятивистских электронных пучков (РЭП) в системах с газовым наполнением можно выделить ряд самостоятельных этапов: расчет электростатических полей в системах с произвольной конфигурацией границы, модель учета собственных электрических и магнитных полей движущихся частиц и механизм учета упругих и, главным образом, неупругих столкновений различного типа. Используемые в настоящей работе алгоритмы расчета внешних и собственных электромагнитных полей рассмотрены в [1], и поэтому будут описаны достаточно кратко, а основное внимание будет уделено вопросам моделирования столкновительных процессов.

Газовое наполнение пучковых систем приводит к полной или частичной компенсации объемного заряда пучка ионами или плазмой. Теоретическому рассмотрению проблем, связанных с поведением РЭП в плазме, и исследованию различного рода неустойчивостей посвящены монографии [2, 3]. Вопросы устойчивости частично или полностью нейтрализованных ионами пучков рассмотрены в работе [4]. Однако в этих случаях коэффициент компенсации объемного заряда пучка считается заданной величиной и механизм его формирования не рассматривается.

Учет ряда неупругих процессов в ускорителях проводился и ранее [5—7]. Так, в [5] в рамках одномерной теории исследовалось влияние ионизации нейтрального газа электронным пучком в диоде на его характеристики. Численное решение одномерного кинетического уравнения с учетом ионизации и резонансной перезарядки ионов позволило провести сравнение модели с экспериментом [6]. Одной из последних работ, наиболее близких по проблематике к данной, является [7], где детально рассматриваются одномерные и двумерные модели процессов ионизации и перезарядки в приближении «сильного» электрического поля.

В настоящей работе описываются двумерные математические модели и численные алгоритмы учета столкновительных процессов, свободные от ряда ограничений, присущих предшествующим работам. Учитывается передача энергии налетающих частиц, рожденных в актах неупругих столкновений, проведено существенное расширение используемого ассортимента столкновительных процессов. Принципиально новым моментом является комплексное использование нескольких различных моделей, позволяющее учитывать специфику физических процессов самых различных задач.

Другой важный аспект предлагаемой вниманию работы — высокая степень автоматизации проведения расчетов, позволяющая пользователю, практически не знающему программирования, описывать физические па-