ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ
1970 год Выпуск 42

УДК 681.142.2

СИСТЕМНЫЙ ПОДХОД К ПОСТРОЕНИЮ ТРАНСЛЯТОРОВ ДЛЯ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

И.В.Вельбицкий, Ю.Г.Косарев

В связи с разработкой вычислительных систем (ВС) высокой производительности [1], предназначенных для решения сложных задач, возникает необходимость создания системы автоматизации программирования, которая должна удовлетворять двум основным требованиям:

обладать высокой эффективностью - рабочая программа должна быть сравнима по времени счета с аналогичной программой, составленной "вручную" на внутреннем языке системы;

учитывать специфику подготовки задач для вычислительных систем - процесс параллельного программирования должен приближать его по трудоемкости к процессу программирования для одной машины.

Для выполнения этих требований необходимо учитывать более тонкие семантические свойства исходных алгоритмов, чем те, на которых основываются существующие системы автоматизации программирования.

В данной работе обсуждаются подходы к решению этой сложнейшей проблемы.

Практически наиболее реально решать данную проблему по-

этапно, уменьшая от этапа к этапу объем и сложность работ, выполняемых при подготовке исходной программы.

I. На первом этапе целесообразно использовать существующие языки и трансляторы. Принципиальная возможность такого подхода была показана и практически подтверждена на системе "Минск-222" на примере трансляторов с языков АЛГОЛ, Автокод АКИ и ЛЯПАС [2-4]. Суть этого подхода заключается в следующем. Программа для ВС пишется на соответствующем входном языке обычным образом. Исключение составляют места программы, связанные с взаимодействием различных ветвей вычислений. Эти места записываются как специальные процедуры в машинных командах. Опыт показал, что такие команды не оказывают существенного влияния на объем программы и на время ее выполнения.

На этом этапе распределение частей задачи между параллельными ветвями не автоматизируется и полностью определяется программистом.

Кроме того, на программиста ложится непростая задача - добиться высокого качества результирующей программы. Непосредственное использование существующих трансляторов дает, как правило, неудовлетворительное для решения сложных задач время счета*). Для увеличения эффективности результирующей программы на первом этапе участки программы, критичные по времени выполнения, записываются так же, как и специальные системные процедуры, в машинных командах, что, конечно, усложняет программирование задач.

2. На втором этапе в языке предусматриваются специальные средства, облегчающие программисту запись взаимодействия между различными ветвями вычислений. Для этой цели в современных языках программирования получили распространение специальные операторы, с помощью которых можно выделить параллельную ветвь вычислений, синхронизировать ее с работой основной программы, сделать некоторые указания транслятору для повышения качества результирующей программы и т.п.

Эти операторы, как правило, ориентированы на определенный

*) Подчеркнем, что это связано, в первую очередь, не с качеством распараллеливания, а со свойствами существующих трансляторов.

тип системы. Их использование делает программу более однородной по структуре, однако не вносит существенного облегчения в процесс программирования. Поэтому при создании универсального языка программирования требуется либо определить независимые (от типа ВС) операторы распараллеливания вычислительного процесса, либо возложить распараллеливание вычислений на систему математического обеспечения, сводя роль программиста к определению и записи самых общих указаний в исходной программе.

Для облегчения программирования участков программы, критичных по времени счета, можно применять языки, объединяющие несколько языков различных уровней (например, АЛГОЛ и Автокод и т.п.). Такие языки в последнее время начинают получать распространение.

3. Успешная реализация последующих этапов вряд ли возможна без привлечения качественно новых подходов к созданию трансляторов. Рассмотрим некоторые принципы, которые могут быть положены в основу их построения.

3.1. П р и н ц и п с и н т а к с и ч е с к о г о у п р а в л е н и я. Согласно этому принципу [5, 6], процесс трансляции входной программы, записанной на соответствующем алгоритмическом языке L_0, осуществляется под управлением грамматики $G(L_0)$ – набора синтаксических правил языка (рис. 1).

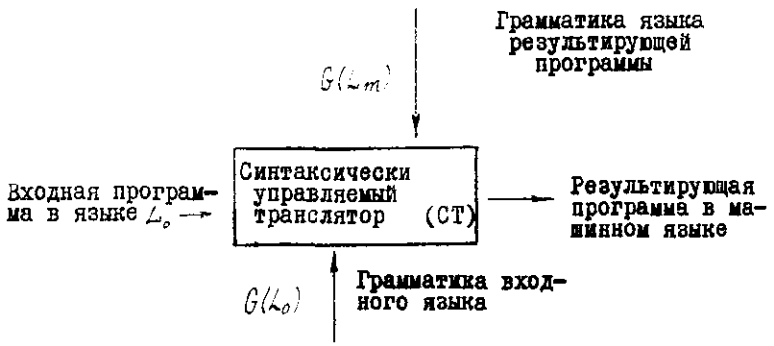

Рис. 1.

Благодаря этому транслятор становится универсальным, ориентированным на широкий класс практически используемых языков.

3.2. П р и н ц и п в ы д е л е н н о г о м е т а я з ы к а. Эффективность синтаксически управляемого транслятора в сильной степени определяется метаязыком, используемым для описания грамматик. Попытаемся перечислить основные требования к метаязыку:

метаязык должен быть достаточно мощным, чтобы задавать широкий класс языков (по крайней мере, класс бесконтекстных языков) и тем самым ориентировать транслятор на многоцелевое применение;

обеспечивать компактную запись грамматик существующих языков;

обеспечивать ускоренную синтаксически управляемую обработку исходной последовательности символов. Синтаксический анализ не должен содержать просмотров "вперед-назад" при обработке любого символа исходной последовательности;

обеспечивать синтаксический анализ при минимальных требованиях к дополнительному объему оперативной памяти.

Этим требованиям удовлетворяет, например, метаязык R – грамматик [7-9]. Суть данного метаязыка состоит в том, что в некоторой компактной форме описываются все возможные разложения языка. Синтаксический анализ с помощью R –грамматик эквивалентен синтаксическому разбору по готовому дереву соответствующего предложения и выполняется с большой скоростью.

3.3. И т е р а т и в н ы й п р и н ц и п п о с т р о е н и я т р а н с л я т о р а. Существующие трансляторы, как правило, состоят из большого числа блоков, каждый из которых реализуется соответствующей ему программой. Это приводит к тому, что построение транслятора превращается в большую системную задачу, которая, как правило, решается с ущербом для качества реализуемой программы.

Итеративный принцип [9] заключается в том, что все символы входной программы обрабатываются одной и той же последовательностью блоков (рис. 2) и все блоки работают по одной, единой для них программе (рис. 3). Эта программа перенастраивается в зависимости от типа символа и этапа работы. Параметрами каждого этапа служат грамматики входного и выходного язы-

ков, соответствующих реализуемому на данном этапе блоку.

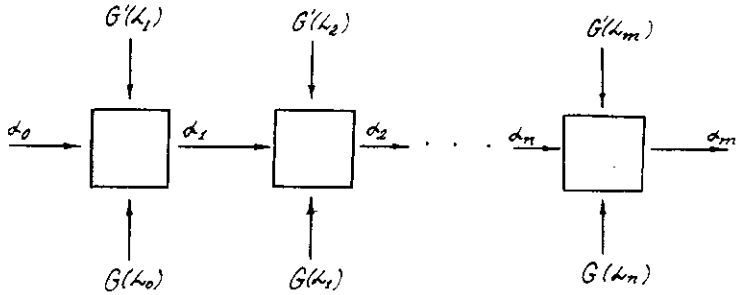

Рис. 2.

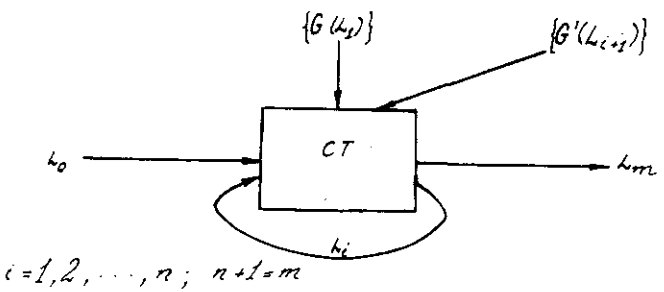

Рис. 3.

Итеративный принцип работы хорошо согласуется с параллельным принципом обработки на однородных вычислительных системах [I] . В этом случае каждая машина системы имеет свои, отличные от других грамматики входного и результирующего языков и одну общую для всех машин программу синтаксического анализа. В процессе работы, выполняемой параллельно и независимо, каждая машина обменивается результатами с машиной справа и получает текущий символ для обработки от машины слева.

3.4. Принцип привнесения семантики в синтаксические определения грамматик языков. Итеративный принцип построения синтаксически управляемого транслятора позволяет эффективно свести синтаксический анализ контекстно-зависимых языков к анализу бесконтекстных языков. Действительно, на каждом i - м этапе синтаксического анализа текущего символа по i - й грамматике каждый символ семантически доопределяется для упрощения работы последующих блоков. Например, текущий символ "идентификатор" на i - м этапе передается для синтаксического анализа на этапе i + I как "идентификатор типа *real*", либо "идентификатор типа *integer*", либо "идентификатор типа *real* индексного выражения", либо "идентификатор параметра цикла типа *integer*" и т.д. Таким образом, одному символу - - "идентификатор" на i -м этапе соответствует n символов на i + I- м. Благодаря этому, в словари грамматик каждого этапа можно включать только часть символов. Это сокращает число правил в грамматике этапов и открывает принципиальную возможность синтаксического управления автоматическим преобразованием алгоритма, например, с целью выделения параллельно выполняемых блоков программы.

3.5. Принцип анализа по логической структуре программы. В существующих трансляторах синтаксический анализ выполняется последовательно при посимвольном просмотре слева направо каждого символа исходной программы. Однако для синтаксически управляемого анализа сложного языка важно порой осуществить просмотр исходной последовательности по логической схеме алгоритма, синтаксически определяемой операторами условного и безусловного переходов . Для реализации данного принципа достаточно написать для одного из последних блоков итеративного синтаксически управляемого транслятора соответствующую грамматику синтаксического анализа Один из примеров использования данного принципа - обнаружение "неявных" циклов, то есть циклов, организованных без операторов цикла. Отсутствие этого принципа нарушает общую структуру синтаксически управляемого анализа входной программы и обусловливает необходимость прибегать к нестандартным, специализированным приемам трансляции, например, к распараллеливанию вычислительного процесса.

3.6. Матричный принцип описания семантики языка. Существующие трансляторы даже для такого сравнительно простого языка, как АЛГОЛ, нередко насчитывают десятки тысяч машинных слов. Наметившаяся тенденция к увеличению сложности языков и особенно необходимость выявления тонких семантических особенностей алгоритмов для построения эффективных параллельных программ могут, если принципы построения трансляторов оставить неизменными, сделать создание транслятора практически неосуществимым. Одним из эффективных способов сокращения семантических описаний языка и самого процесса трансляции служит применение матричных структур. Например, запись R -грамматик в виде матриц позволяет единым образом различать все, вплоть до самых сложных, конструкции языка. Благодаря этому все строки матриц, в которых заключены R-грамматики для различных этапов трансляции, анализируются с помощью одной и той же программы. Интересно отметить, что эта программа оказалась несложной (менее 100 команд "Минск-22").

Матричные методы могут быть полезны на всех этапах трансляции. Так, например, при матричной форме записи таблиц приоритете с учетом символов операции типа операнд ("*real*", "*integer*", "*boolean*") и их местонахождения (в таблице соответствия, в таблице констант или в сумматоре) так же, как и в случае 2 -грамматик, удается "упрятать" семантические различия в содержание элементов матриц, оставляя неизменной их форму. Это позволяет обрабатывать все случаи единообразно и сравнительно простыми средствами с помощью транслятора итеративного типа.

Матричный способ записи удобен своей компактностью и простотой нахождения нужной величины, благодаря аналитической связи между наименованием величины и ее местом. Это позволяет избавиться от поиска её путем перебора и существенно ускоряет трансляцию. Другое важное достоинство матриц – наглядность представления информации, отсюда простота обнаружения и исправления ошибок при отладке транслятора, а также простота внесения в него дополнений.

3.7. Адаптивный принцип анализа. При разработке сложных систем, как правило, возникает противоречие между универсализацией и специализацией. Универсализация уменьшает общий объем программирования. Специализация позволяет создавать более эффективные рабочие программы. Для проблемы, рассматриваемой в данной работе, возникает насущная потребность сочетать достоинства обоих направлений.

Один из эффективных путей разрешения данного противоречия – применение трансляторов, структура которых может программно перестраиваться в зависимости от ситуации. Простейшим примером применения такого адаптивного подхода может служить введение различных типов объединения термов в синтермы, что в сочетании с R -грамматиками позволило разработать эффективный и простой в реализации метод синтаксического контроля и анализа исходной программы [7-9].

Смена типа означает перестройку транслятора на исследование других свойств исходной программы. Введение даже сравнительно небольшого числа состояний (типов) трансляции уже позволяет получить значительный эффект – анализировать достаточно тонкие свойства исследуемых конструкций языка, используя для этой цели сравнительно простые средства.

3.8. Принцип локализации изменений. Выше было отмечено (3.3), что итеративный принцип синтаксического управления позволяет представить процесс трансляции в виде последовательно выполняемых этапов. Принцип локализации изменений устанавливает, как наилучшим образом использовать данную возможность для уменьшения чувствительности транслятора к изменению способов кодирования исходной программы, системы внесения исправлений и дополнений в нее, тех или иных средств языка, параметров машины или системы и т.п. Необходимо так разбить процесс трансляции на этапы, чтобы при любом из указанных изменений в трансляторе сохранилось в неизменном виде наибольшее число блоков. Такой подход [7, 8] облегчает разработку математического обеспечения, так как при создании нового транслятора можно полностью использовать многие блоки уже имеющихся трансляторов.

Другим немаловажным свойством такого подхода служит легкость обнаружения степени сложности трансляции той или иной конструкции языка, способа кодирования и т.п., что способствует их совершенствованию. Кроме того, принцип локализации исправлений облегчает обнаружение и устранение ошибок или неудачных решений в трансляторе в процессе его отладки.

3.9. Принцип распараллеливания по циклам. Основой для реализации указанной системы автоматизации распараллеливания вычислительного процесса может служить принцип, описанный в [10]. Суть его заключается в следующем. По исходному алгоритму выделяются независимые циклы, охватывающие основные по времени операторы. Затем все повторения циклов и соответствующие им исходные данные равномерно распределяются между ветвями вычислений. До и после цикла вставляются системные операторы, которые компилируют полученные результаты и организуют взаимодействие ветвей.

Указанные выше принципы не исключают друг друга и могут использоваться одновременно. Более того, многие из них обладают свойствами, которые облегчают практическое осуществление других принципов. Так, например, матричные методы хорошо согласуются с $\mathcal{L}$-грамматиками, принципами адаптации и локализации. $\mathcal{L}$-таблицы являются хорошим средством для реализации принципа распараллеливания по циклам и т.д.

Можно надеяться, что применение совокупности указанных принципов позволит решить проблему создания эффективной системы автоматизации программирования для вычислительных систем.

ЛИТЕРАТУРА

1. Э.В. ЕВРЕИНОВ, Ю.Г. КОСАРЕВ. Однородные универсальные вычислительные системы высокой производительности. Новосибирск, Изд-во "Наука" СО, 1966.
2. Ю.Г. КОСАРЕВ, Е.Л. ЮЩЕНКО. Некоторые проблемы организации вычислительного процесса на системах ЭВМ. – Труды семинара "Автоматизация программирования", Киев, Изд-во ИК АН УССР, 1968, вып.1, стр.3-7.
3. Ю.Г. КОСАРЕВ. Об автоматизации программирования для однородных вычислительных систем. – Вычислительные системы. Труды I Всесоюзной конференции по вычислительным системам. Новосибирск, 1968, вып.4, стр.75-79.
4. Л.В. ГОЛОВАШКИНА, Ю.И.КОЛОСОВА, Ю.Г. КОСАРЕВ, Н.Н.МИРЕНКОВ. Автоматизация программирования для систем на основе существующих трансляторов. – Вычислительные системы, Изд-во "Наука" СО, Новосибирск, 1968, вып.30, стр.63-69.

5. T.E. IRONS. A syntax - directed compiler for ALGOL 60, Comm. ACM, vol.4, N 1, 1961.
6. П. ИНГЕРМАН. Синтаксически ориентированый транслятор. Изд-во "Мир", М., 1968.
7. И.В. ВЕЛЬБИЦКИЙ, Е.Л. ЮЩЕНКО. Метаязык, ориентированный для синтаксического анализа и контроля. – Кибернетика, № 2, 1970, стр.50-53.
8. И.В. ВЕЛЬБИЦКИЙ. Первичные средства подготовки синтаксически правильных программ и данных для вычислительных систем. – Вычислительные системы. Труды II Всесоюзной конференции по вычислительным системам. (В печати).
9. И.В. ВЕЛЬБИЦКИЙ. О метаязыке синтаксически управляемого транслятора. – Настоящий сборник, стр.22-33.
10. Ю.Г. КОСАРЕВ. Распараллеливание по циклам. – Вычислительные системы. Новосибирск, Изд-во "Наука" СО, 1967, вып.24, стр.3-20.

Поступила в редакцию
20 июня 1970 г.