Российская академия наук
Сибирское отделение
Институт систем информатики
им. А. П. Ершова

В. А. Евстигнеев, И. Л. Мирзуитова

АНАЛИЗ ЦИКЛОВ: ВЫБОР КАНДИДАТОВ НА РАСПАРАЛЛЕЛИВАНИЕ

Препринт
58

Новосибирск 1999

Рассматривается проблема анализа циклов и отбора кандидатов на распараллеливание, осуществляемого распараллеливающим компилятором. Предварительно дано описание с этой точки зрения известных распараллеливающих компиляторов и соответствующих инструментальных систем.

Siberian Division of the Russian Academy of Sciences
A. P. Ershov Institute of Informatics Systems

V. A. Evstigneev, I. L. Mirzuitova

LOOP ANALYSIS: A SELECTION OF CANDIDATES FOR PARALIZATION

Preprint
58

Novosibirsk 1999

The problem of selection of loops for their parallization or vectorization by compilator is considered. At first some known vectorizing compilators and suitable tools are described. For each compilator and tools are pointed how analysis and selection of loops is happened.

1. ВВЕДЕНИЕ

Как известно, циклы являются основными областями распараллеливания, но далеко не каждый цикл может быть распараллелен без предварительного анализа и последующих преобразований, увеличивающих степень параллелизма цикла. В системе ПРОГРЕСС, разрабатываемой в ИСИ СО РАН в качестве инструмента для быстрого прототипирования распараллеливающих компиляторов и учебно-исследовательского средства для манипулирования программами [1–5], предусматривается механизм для отбора среди циклов кандидатов на распараллеливание. Можно, например, указать на следующие причины, затрудняющие векторизацию или даже делающие ее невозможной. (Список не претендует на полноту, играя иллюстративную роль, и относится к 1979 г.):

1. Большое тело цикла.
2. Нетривиальные индексные выражения.
3. Обращения к встроенным функциям.
4. Скалярные промежуточные переменные.
5. Обращения к пользовательским функциям, написанным не на Фортране.
6. Скалярное произведение векторов.
7. Логические условные операторы.
8. Передача управления вовне цикла.
9. Векторные операции редукционного типа.
10. Линейная рекурсия по индексам.
11. Арифметические условные операторы.
12. Некоторые операторы ввода-вывода.
13. Сложные индексные выражения.

Следующие причины делают векторизацию невозможной:

14. Нелинейные по управляющей переменной цикла индексные выражения.
15. Неопределенность зависимостей между индексированными переменными.
16. Передача управления в цикл извне.
17. Обращение к пользовательским подпрограммам.
18. Нелинейная рекурсия по индексам.
19. Некоторые операторы ввода-вывода.

Отбор циклов на векторизацию или распараллеливание представляет собой достаточно трудную задачу, многие компиляторы предлагают

упрощенную схему отбора, предоставляя пользователю ручное исследование нераспараллеливаемых циклов или требуя от пользователя дополнительной информации. Исследование циклов проводится на основе лексического и синтаксического анализа, а также на основе анализа зависимостей по данным.

Не вызывает сомнений, что трансляторы, поддерживающие векторную и параллельную обработку, будут оставаться центральными компонентами сред программирования [6]. Блок векторизации/распараллеливания будет реализовываться преимущественно в виде препроцессора. По мере развития RISC- и VLIW-архитектур все более важным будет распараллеливание на самом нижнем, программно-доступном уровне. Интегрированные среды программирования суперЭВМ будут включать (кроме трансляторов) диалоговые средства статического и динамического анализа программ, используемые в процессе настройки программ для конкретной архитектуры.

Для тщательного анализа невекторизуемых циклов и связанных с этим вопросов разработано большое количество инструментальных систем, к наиболее известным относятся PTOOL, ParaScope, PAT и другие.

Рассмотрен ряд векторизующих компиляторов с точки зрения выполняемых действий по отбору кандидатов и анализу зависимостей, связанных с этим отбором.

Данная работа выполнена при финансовой поддержке РФФИ (грант 98-01-00748).

2. ВЕКТОРИЗАТОРЫ

2.1. PTRAN [6,7]

PTRAN предназначен для автоматического распараллеливания фортрановских программ, выполняемых на параллельных архитектурах. Аналитическая фаза системы PTRAN-A предназначена для нахождения зависимостей по данным и управлению в исходных программах. При этом не уточняется, для какой части программы проводится этот анализ. Указывается только, что анализ выполняется как внутри процедур, так и на межпроцедурном уровне.

При межпроцедурном анализе в точке вызова процедуры определяются области доступа к массиву, косвенные взаимосвязи (наложение) массивов, использование одних и тех же переменных разными процеду-

рами.

Склейка констант выполняется как на меж-, так и на внутрипроцедурном уровне. Поиск индуктивных переменных позволяет обнаружить не только одиночные, но и взаимноиндуктивные переменные.

Построение зависимостей по данным кроме своего основного назначения выявляет наложение переменных и позволяет проверить корректность индексных выражений.

В PTRAN-A рассматриваются только такие индексные выражения, для которых можно построить линейные диофантовы уравнения. Для многомерных массивов предварительно строится отображение на одномерный массив либо уравнения зависимости составляются по каждому измерению.

2.2. KAP [6]

KAP — это семейство препроцессоров-векторизаторов, каждый член которого настроен на конкретную объектную машину.

В версии KAP для системы Sequent исследуются следующие особенности циклов: есть ли в рассматриваемом цикле критический интервал, есть ли операторы IF или CALL, мало ли количество итераций, содержит ли цикл операции редукционного типа. Отметим, что при наличии в теле цикла условных операторов и/или вызовов внешних процедур даже автоматический динамический анализ не всегда дает надежные оценки. В этих случаях остается лишь эмпирическое изучение динамических профилей различных вариантов кода программы, выбор программистом наилучшего варианта и сообщение об этом выборе транслятору при помощи специальных директив.

KAP/205 [10] — векторизатор для Cyber 205 — предназначен для облегчения эффективного исполнения программ на Cyber 205, так как векторизатор универсального типа не решает всех проблем, встающих перед пользователями Cyber 205. Векторизатор воспринимает программы на языке Фортран 200 и векторизует DO-циклы. При этом KAP/205 использует векторные обозначения языка Фортран 200, встроенные векторные функции и специальные вызовы Q8 в ассемблерном коде. Векторизатор выдает сообщения и вопросы, касающиеся тех частей программы, которые в дальнейшем можно будет улучшить с дополнительной помощью пользователя.

Адаптация KAP для Cyber 205 облегчается благодаря модульному принципу построения KAP. На одной из первых стадий работы KAP

запускается **распознаватель кандидатов**. На этой стадии анализа DO-циклов выясняется, являются ли операции внутри них легальными кандидатами на векторизацию. Векторизатор KAP/205 помечает операторы, включающие невекторизуемые операции (например операции типа double precision) как невекторизуемые. Таким же образом помечаются операторы READ и WRITE. Те DO-циклы, которые полностью состоят из невекторизуемых операторов, помечаются как **полностью последовательные** и больше не исследуются.

Многие компоненты KAP, например, **распознаватель индуктивных переменных**, являются машинно-независимыми, и поэтому для их адаптации не требуется никакой дополнительной работы.

В числе других компонент KAP/205 укажем на векторный оптимизатор, который анализирует различные способы исполнения каждого гнезда циклов и выбирает наилучший. При этом основное внимание уделяется стремлению избежать операций сборки и разборки.

Из используемых оптимизаций упоминаются коллапс циклов, разбиение цикла, переупорядочение операторов и разрушение контуров зависимостей по данным.

KAP/ST-100 [6, 11] — векторизующий препроцессор — состоит из сканера, распознавателя кандидатов (на векторизацию), распознавателя процессов и собственно векторизатора. **Распознаватель кандидатов** ищет и отмечает DO-циклы, являющиеся кандидатами для векторизации, например отбрасывается любой оператор, содержащий операнды с двойной точностью. Каждый DO-цикл подвергается проверке; если DO-цикл не имеет операторов-кандидатов, то весь цикл помечается как негодный и к нему не будет применяться векторизация.

С целью улучшения векторизации KAP/ST-100 выполняет такие общие преобразования, как распознавание индуктивных переменных и превращение скаляров в массивы. Кроме того, распознаются некоторые IF-конструкции, которые вычисляют максимум среди компонент вектора или находят индекс этого максимума.

С целью нахождения наилучшего варианта векторизации выполняются такие преобразования, как перестановка и распределение циклов. Для проверки корректности этих преобразований используется граф зависимостей по данным.

KAP/S-1 [6, 12] — векторизующий препроцессор — векторизует программы, написанные на языке Фортран 77, расширенном векторными конструкциями, которые транслируются в векторные команды

отдельного процессора S-1.

В векторизаторе KAP/S-1 используется преобразование "**разбиение цикла по именам**". Суть этого преобразования заключается в том, что DO-цикл разбивается на несколько циклов, каждый из которых ссылается на непересекающийся с другими набор массивов, что значительно повышает эффективность использования кэш-памяти.

Еще более важным преобразованием, реализованным в KAP/S-1, является **сегментация цикла**. Суть его в том, что если число N итераций велико, то предлагается перед векторизацией организовать выполнение тела цикла для части итераций цикла. Размер порций выполнения выбирается таким образом, чтобы не переполнялась кэш-память. Это приблизительно равно размеру кэш-памяти, деленному на число различных ссылок на массивы в теле цикла. В результате исходный цикл преобразуется в гнездо из двух циклов.

2.3. PFC [8]

PFC — разработка университета Райс (начало разработки — 1978 г.), на базе которой был создан векторизующий транслятор IBM VS Fortran Version 2. В 1982 г. появилась законченная версия PFC. На ее основе была создана диалоговая система анализа программ PTOOL.

2.4. CFT [9]

Первая версия CFT для Cray 1 появилась в 1976 г. В течение последующих 15 лет наиболее развивающейся частью этой системы программирования был блок векторизации, непрерывно отслеживающий расширение возможностей аппаратуры. Блок векторизации анализирует самые внутренние циклы в гнездах циклов, выявляя следующие свойства циклов: есть ли возможность выявить совпадение индексов ссылок на переменные, допускается ли расщепление оператора и введение временных переменных, имеются ли нелинейные индексы и условные операторы.

В версии CF77 автоматическое распараллеливание гнезд циклов не делается и требуется предварительная ручная оптимизация для сведения гнезда циклов к одномерному циклу.

2.5. FTN 200

Большая часть программ для Cyber 205 написаны на языке Fortran 200, реализующем расширение языка Fortran 77. Расширения главным образом ориентированы на использование возможностей векторной обработки в Cyber 205. Транслятор с языка Fortran 200 носит название FTN 200. Трансляция в FTN 200 управляется с помощью специальной шкалы режимов, задаваемой пользователем. К таким режимам относятся:

V — векторизация,

O — оптимизация.

Последняя включает такие преобразования, как вынос инвариантов и упрощение индексов в циклах, удаление "мертвых"вычислений, планирование потока команд, вычисление константных выражений.

2.6. Fortran-77/VP [13]

Fortran-77VP — это векторизующий вариант транслятора стандартного языка Фортран 77 для векторных процессоров семейства FACOM VP японской фирмы Fujitsu.

Аналогично другим системам векторизация здесь основана на построении графа зависимостей внутри итеративных циклов. Транслятор может частично векторизовать цикл, т. е. расщепить его тело на векторизуемую и невекторизуемую части, подставить текст внешней Фортран-процедуры на место обращения к ней из тела цикла. Возможен диалог с пользователем в процессе отладки программы. При помощи специальных директив, вставляемых в текст программы, пользователь может сообщить транслятору:

- об отсутствии зависимости между элементами определенного массива;
- о количестве итераций цикла;
- об относительном числе срабатываний альтернатив логического условного оператора IF.

Другими словами, с помощью операторов управления оптимизацией, оформленных как комментарии, может указываться, например, коэффициент истинности условия; рекомендуемая длина векторов при исполнении оператора цикла, позволяющая лучшим образом использовать возможности регистров; способ принудительной векторизации циклов, которые носят очевидный рекуррентный характер из-за при-

сутствия неизвестных величин; и, наоборот, принудительный переход к скалярной форме, когда заранее известно, что длина цикла мала.

С помощью интерактивного векторизатора возможна перестройка программы в диалоговом режиме. Программист может ввести в систему сведения о том, какие операторы цикла относятся к векторизуемым, как повысить скорость счета и т. д. Кроме того, работая в интерактивном режиме, программист может изменять текст исходной программы, вводить операторы управления векторизацией и др., контролируя процесс изменения программы на экране.

2.7. Фортран-компилятор для NEC SX [14, 16]

В системе NEC SX имеются две Фортран-системы: Fortran 77 (F77) и Fortran 77/SX (F77/SX). F77 генерирует объектный код для управляющего процессора и используется при отладке; F77/SX генерирует векторизованный объектный код для арифметического процессора и используется для реальных вычислений.

F77/SX имеет развитые функции автоматической векторизации. Они позволяют векторизовать полностью или частично циклы, которые очень трудно или почти невозможно векторизовать в том виде, как они есть. Компилятор рассматривает циклы, включающие невекторизуемые части; вложенные циклы; циклы, включающие вызовы пользовательских функций; циклы, включающие обращения к массивам, у которых есть зависимости, препятствующие векторизации.

В дополнение к функциям автоматической векторизации компилятор F77/SX использует все виды оптимизационной техники для достижения максимальной производительности системы SX. Некоторые из условных оптимизаций, таких как **удаление общих подвыражений**, **перемещение кода** и **понижение силы операций**, были расширены для применения к векторизуемому коду.

Ключевым фактором для получения высокой производительности от системы SX является как можно большее увеличение векторизуемой части. Другой важный фактор — сделать генерируемый код более эффективным, например путем увеличения длины вектора, устранения конфликтов банков памяти и т. д. Чтобы облегчить такую настройку, система SX снабжена двумя утилитами: ANALYZER/SX и VECTORIZER/SX.

ANALYZER/SX анализирует и документирует динамические характеристики исходных программ: коэффициент векторизации программы,

общее число обрабатываемых DO-циклов, число векторизованных DO-циклов и т. д.

VECTORIZER/SX представляет собой сервисную программу помощи пользователям в настройке своих программ для получения высокого коэффициента векторизации. Нужная для настройки информация о динамических характеристиках программы поступает от ANALYZER/SX через файл, который пользователь может просматривать на экране. VECTORIZER/SX имеет экранный редактор, что позволяет пользователям модифицировать свои программы непосредственно, обращаясь к необходимой информации.

2.8. Компилятор FORT 77/HAP для Hitachi S-810

Векторизующий компилятор FORT 77/HAP [15] переводит программу, написанную на стандартном Фортране 77, в объектный код, включающий векторные команды. Компилятор способен автоматически векторизовать:

- циклы, содержащие арифметические выражения, в том числе содержащие один или несколько вложенных операторов IF;
- циклы, содержащие встроенные функции;
- циклы, содержащие списочные векторы;
- циклы, содержащие скалярное произведение векторов, сумму элементов вектора, итерацию первого порядка.

Последнее осуществляется с помощью специальных векторных команд.

Компилятор FORT 77/HAP позволяет также осуществлять частичную векторизацию, т.е. такую, которая производится под управлением непосредственно пользователя и касается определенных участков программ. При этом возможны следующие варианты управления:

а) принудительная векторизация, когда пользователь может представить в векторной форме те участки программ, которые компилятор отнес к числу неподдающихся векторизации;

б) запрет векторизации, когда пользователь может отменить векторизацию тех или иных участков программы, которые могли бы быть векторизованными, и получить соответствующие объектные коды, содержащие только скалярные команды.

Несмотря на большие возможности компилятора, для наилучшего использования особенностей системы S-810 необходимо проводить настройку программ на оптимальный режим исполнения. Для этой цели служит векторный программный анализатор VECTIZER.

VECTIZER позволяет получить следующие данные об исследуемой программе:

а) коэффициент векторизации;

б) коэффициент нагрузки для каждого цикла, т.е. какой процент от общего числа тактов исполнения программы приходится на тот или иной цикл;

в) диагностические сообщения о результатах векторизации каждого цикла.

Таким образом, для каждого цикла VECTIZER выдает на печать коэффициент нагрузки и диагностическое сообщение о результатах векторизации. Коэффициент нагрузки для цикла определяется как отношение числа скалярных команд, которое пришлось бы исполнить при реализации цикла в скалярном режиме, к общему числу скалярных команд, которое пришлось бы затратить на всю программу, если бы она также исполнялась в скалярном режиме. Диагностическое сообщение о результатах векторизации содержит информацию о том, оказалась ли попытка векторизовать данный цикл успешной, и если нет, то указывается причина неудачи.

Цикл, который не поддается векторизации, относится к одной из следующих категорий:

- циклы, которые могут быть векторизованы путем незначительной модификации на уровне кодирования;
- циклы, которые компилятор не смог распознать как векторизуемые;
- циклы, для векторизации которых необходимы значительные преобразования, в том числе изменение алгоритма;
- циклы, которые в принципе не могут быть векторизованы в силу своей внутренней природы.

Проблемы, возникающие с циклами первых двух типов, легко могут быть решены при настройке. Что же касается циклов третьего типа, то с ними сопряжены значительные трудности, иногда бывает выгоднее целиком перестроить программу.

2.9. Фортран-компилятор для IBM 3090 VF [17]

Особенностью данного компилятора является то, что векторизация была включена в оптимизатор, образовав новую фазу между оптимизацией текста и оптимизацией регистров. Это было связано с тем, что, во-первых, уменьшается объем кусков кода, с которыми работает векто-

ризатор (удалены общие подвыражения, вынесены некоторые операторы из циклов при чистке последних); во-вторых, процесс оптимизации в компиляторе почти дублирует процесс нормализации индексов для подготовки к векторизации; в-третьих, векторизация не портит оптимизацию тех частей программы, которые не поддавались векторизации.

Кратко остановимся на основном алгоритме векторизации, являющемся развитием подхода, принятого в PFC. После нормализации циклов векторизатор формирует граф зависимостей по данным, разбивает его на сильно связные области и потом генерирует либо векторный код для операторов вне сильно связных областей, либо скалярный код для DO-цикла на уровне, определяющем область.

Алгоритм, определяющий, может ли сильно связная область все же исполняться векторно, имеет две фазы. На первой фазе алгоритм проверяет каждую зависимость из сильно связной области и пытается переставить соответствующий DO-цикл последовательно с каждым более внутренним циклом, пока рассматриваемый цикл не станет самым внутренним. Если перестановка невозможна из-за мешающих зависимостей, то область не может быть исполнена в векторном стиле на уровне, определяющем область, и алгоритм заканчивается.

На второй фазе алгоритм строит граф, используя только циклически независимые зависимости в области и самые внутренние "чувствительные" зависимости на уровне DO-цикла, определяющего область. Другие зависимости, существенные для других уровней DO-цикла, исключаются. Если этот граф имеет контур, то область не может быть исполняема в векторном режиме на уровне, определяющем область. В противном случае область пригодна для векторного исполнения даже в случае, когда другие циклы внутри цикла, определяющего область, могут иметь рекуррентности.

На всем описанном выше этапе анализа компилятор не делал никаких необратимых решений о конечном виде объектной программы. В частности, он не писал скалярные DO-циклы и не разбивал исходные гнезда DO-циклов на отдельные скалярные и векторные циклы. Теперь векторизатор будет разыскивать наибыстрейшее возможное исполнение программы, используя или векторную, или скалярную аппаратуру.

Главные факторы, принимаемые во внимание при оценке стоимости исполнения, суть стоимость накладных расходов в цикле, стоимость векторных команд и стоимость считывания и сохранения операндов. Стоимости будут варьироваться, помимо всего прочего, из-за того, что

операторы будут скалярными на некоторых уровнях, но векторными на других уровнях и из-за того, что массивы-операнды будут считываться и сохраняться в различных размерностях, когда рассматриваются в целях векторизаации различные уровни.

2.10. Компиляторы фирмы Convex [18]

Компиляторы для суперкомпьютера Convex C-1, близкого по своей архитектуре к Cray-1, имеют две общие части: оптимизатор, осуществляющий также векторизацию, и кодогенератор. Каждый компилятор имеет, однако, свой собственный препроцессор для языков Фортран, Си и Ада. Компилятор выполняет оптимизацию первой; он делает все подходящие скалярные оптимизации такие, как подстановка присваивания, сворачивание констант и протягивание, понижение силы операций, удаление мертвого кода и избыточных подвыражений, вынос из цикла инвариантных вычислений.

Одно из первых действий по настройке прикладной программы — ее профилировка, т. е. прогон программы на реальных данных и сбор информации о расходуемом времени. Очевидно, что попытка оптимизировать части программы, которые используют относительно мало времени, — бесплодное занятие.

Профилировка может дать два вида информации: число исполнений каждой подпрограммы и количество времени, затрачиваемого каждой подпрограммой. Компилятор порождает экстракод для подсчета числа исполнений и оценки времени исполнения. Специальные программы могут тогда строить гистограммы, а программы для более изощренного анализа могут протягивать эту информацию сквозь дерево вызова подпрограмм для настройки собираемых данных о времени, затрачиваемом вызываемыми подпрограммами.

На основании этих данных подпрограммы разбиваются на четыре класса, и процесс оптимизации начинается с первого класса — подпрограмм, вызываемых много раз с высоким временем исполнения во время каждого вызова. Рассмотрим идентифицированные, наиболее подходящие для оптимизации кандидаты. Оптимизация некоторых из них может быть осуществлена подстановкой библиотечной подпрограммы. С другими компилятор может делать все, что только возможно, после чего снова проводится профилировка и отбор кандидатов для дальнейшей работы.

Что касается векторизации, то большинство векторизующих компи-

ляторов рассматривает в качестве кандидатов на векторизацию циклы с одним входом и одним выходом. Самые внутренние фортрановские DO-циклы без GOTO или арифметических IF-операторов удовлетворяют этому требованию. В противоположность им векторизующие компиляторы Convex достаточно изощренны для векторизации одного, двух или трех самых внутренних циклов в гнезде. Закодированные вручную циклы (т.е. имеющие возможность закончить исполнение цикла с помощью IF-оператора), однако, не векторизуются, если объект, отличный от проверки цикла, ссылается на метку оператора в начале цикла.

Компиляторы Convex могут векторизовать циклы, которые имеют индуктивные переменные следующих типов: целые, вещественные с плавающей точкой, указатели (согласованные с определением языка), такие как приращения циклов на итерационо-независимое количество во время каждого прохода по циклу.

Компиляторы Convex автоматически используют скалярную обработку даже векторизуемых циклов, когда компилятор может определить во время компиляции, что число итераций мало (меньше 5). Предположим, что цикл с одним входом и одним выходом действует с массивами, имеет приемлемую итерационую переменную цикла и достаточно большое число итераций — использование векторной обработки может оказаться невозможным, если цикл содержит определенные языковые конструкции, например. Операторы ввода/вывода, фортрановские операторы PAUSE, STOP и вычисляемый GOTO, оператор переключения в Си, операторы CASE и RAISE в Аде делают векторизацию невозможной или, по крайней мере, увеличивают трудности анализа. Символьные выражения, битовые операции и структурные присваивания также препятствуют векторизации из-за большого числа требуемых скалярных операций.

Фортран-компилятор для Convex C-1 делает возможным использование внутренних фортрановских функций внутри векторизуемых циклов. Компилятор частично векторизует циклы, содержащие вызовы подпрограмм, помещая вызовы в скалярную часть цикла.

Компилятор может не векторизовать цикл или векторизовать его частично, если самый внутренний цикл содержит рекуррентность или если компилятор не может надежно определить, что рекуррентности нет. Некоторые реуррентности фактически являются операциями редукции, распознаются и обрабатываются в векторном режиме командами векторной редукции. Вообще говоря, векторизация невозможна,

когда две ссылки на массив так завязаны, что компилятор не может выяснить, какую брать первой.

2.11. Компилятор Pascal-XT [19]

Векторизующий компилятор Pascal-XT принадлежит к семейству компиляторов для различных машин фирмы Siemens. Члены этого семейства состоят из общего препроцессора, который проводит машинно-независимый анализ программ, и (при необходимости) из собственного пост-процессора для машинно-ориентированной генерации кода. Частью этого пост-процессора является векторизатор, который анализирует параллелизм программ и управляет генерацией кода, состоящего из векторных команд.

Как известно, в векторизующем компиляторе анализ зависимостей по данным имеет решающее значение. В векторизующем компиляторе Pascal-XT применяется анализ зависимостей, основанный на методе координат Лэмпорта (L. Lamport).

Если зависимости по даннным достаточно сложны и простое переупорядочение операторов уже не помогает, то компилятор может разбить операторы на части и переупорядочить их, принимая во внимание зависимости по данным. При разбиении присваиваний нужно учитывать еще одну (тривиальную) зависимость: правая часть присваивания должна выполняться перед присваиванием результата левой части. Для векторизации нужно так переупорядочить переменные, чтобы все зависимости по данным были направлены вперед. Это возможно, когда вхождения переменных в зависимости между ними представляются бесконтурным орграфом, вершины которого упорядочиваются с помощью топологической сортировки. В топологически отсортированном орграфе вхождения переменных находятся в той последовательности, в которой они должны стоять в оттранслированной программе.

Метод, описанный выше, не работает, если граф зависимостей по данным содержит контуры. Компилятор Pascal-XT изолирует эти контуры и генерирует минимально необходимый скалярный код, а остальная часть векторизуется, как описано выше.

Операции над массивами целых или вещественных чисел в FOR-циклах (после анализа зависимостей по данным и возможных переупорядочиваний) можно преобразовывать непосредственно в векторные команды. Соответствующие операции существуют и в Фортране и поэтому поддерживаются аппаратно. Условные операторы (операторы

IF) также поддерживаются аппаратно различными способами (маски, сборка-рассылка, списочные методы). Символьные типы, перечислимые типы и множества представляются в ЭВМ целыми числами как скалярные типы. Массивы этих типов обрабатываются как массивы целых чисел. Интересно отметить, что в компиляторе векторизуются такие конструкции языка Паскаль, как данные типа запись (массивы типа запись), связанный с помощью указателей список, множества, файлы, и частично векторизуются такие управляющие структуры, как WHILE- и REPEAT-циклы.

2.12. Векторизующий компилятор UFTN для Unisys ISP [20]

Компилятор UFTN переводит Фортран-программы в некоторый промежуточный код, причем после фазы векторизации следует фаза оптимизации, где осуществляется полный набор традиционных оптимизирующих преобразований, таких как удаление общих подвыражений, понижение силы операций и удаление "мертвого"кода.

Фаза векторизации состоит из следующих подфаз.

Сбор информации. Эта подфаза разделяет программу на линейные участки с одним входом и одним выходом, определяет, какие участки формируют циклы, и собирает информацию об определениях и использованиях переменных. На этой подфазе определяются циклы — кандидаты на векторизацию. UFTN векторизует только DO-циклы. Как и большинство векторизаторов, он не пытается векторизовать циклы, содержащие пользовательские подпрограммы или вызовы функций, передачи управления к операторам вне цикла, неявные циклы, т. е. передачи управления назад, а также такие невекторизуемые операторы, как операторы ввода/вывода, STOP или RETURN.

Преобразование текста к стандартному виду. На этой подфазе для облегчения проведения анализа зависимостей преобразуется текст программ; к таким преобразованиям относятся *нормализация циклов*, *подстановка индуктивных переменных*, *сжатие цикла* (loop rerolling — операция, обратная раскрутке (unwinding)).

Построение графа зависимостей.

Отделение векторизуемых циклов от невекторизуемых. На этой подфазе происходит выделение сильно связных областей графа зависимостей, генерирование для них минимально необходимого скалярного кода с разделением, если нужно, цикла на несколько циклов, и преобразование остатка цикла в векторную форму. Кроме того, на

этой подфазе происходит переупорядочение операторов, если это способствует векторизации.

Специальные случаи векторизации. Некоторые операторы, которые часто встречаются на практике и могут быть векторизованы, помечаются на предыдущей подфазе как невекторизуемые, если они входят в контур зависимости. UFTN распознает некоторые из них и генерирует специальный код, который может быть исполнен с помощью аппаратной поддержки векторных вычислений, имеющейся в ISP. К таким случаям относятся: редукции суммы и произведения, редукции минимума и максимума, линейные рекуррентности первого порядка, охватывающие рекуррентности, а также различные случаи частичной векторизации.

Генерация векторного кода. На этой подфазе осуществляется упорядочение π-блоков графа зависимостей, подготовка операндов, реконструкция циклов в соответствии с упорядочением π-блоков и собственно генерация векторного кода.

Существенной особенностью компилятора является способ обработки операторов IF с помощью управляющих выражений, отличающийся от IF-конверсии тем, что управляющие выражения не изменяют поток управления в цикле и ссылаются только на блоки, непосредственно предшествующие рассматриваемому. Это позволяет значительно ослабить ограничения на возможность векторизации операторов IF.

2.13. Прочие разработки

Следует указать на некоторые разработки, выполненные в СССР и имеющие отношение к рассматриваемому вопросу.

Система программирования для ПС-3000 [22–24]. Система программирования с языка Фортран 77ВП для МВК ПС-3000 состоит из

— языка программирования Фортран 77ВП, представляющего расширение стандарта Фортрана 77 средствами организации векторных и параллельных вычислений;

— компилятора, переводящего программу с входного языка на язык загрузки МВК ПС-3000;

— препроцессора-векторизатора циклов;

— препроцесора-анализатора параллельных программ.

Векторизатор производит эквивалентные преобразования циклических участков входной программы в блоки векторных величин так, как

это описано в [21]. Распараллеливанию подвергаются гнезда DO-циклов вложенностью от 1 до 7 циклов, содержащие операторы присваивания с индексными выражениями в виде линейной комбинации параметров цикла. Распараллеливание осуществляется согласно методу линейного преобразования, являющемуся более мощным инструментом, нежели методы гиперплоскостей или координат Лэмпорта.

В результате распараллеливания образуется конструкция, задающая векторные вычисления с использованием операторов расширения языка Фортран 77.

Система М-10 [24,25] представляет собой векторно-параллельную, ориентированную на выполнение сложных задач в реальном масштабе времени систему . Для нее разработан специальный векторный язык для программ, не являющихся критичными по времени исполнения. Транслятор оптимизирует векторное распараллеливание и совмещение операций во времени. Основная особенность транслятора — явное указание на возможность векторного распараллеливания циклов. На каждом повторении параллельного цикла одновременно обрабатываются восемь последовательных значений индексной переменной цикла. Для более длинных векторов применяется прием, аналогичный сегментации циклов.

Параллельный цикл записывается с помощью цикла PARDO. Цикл PARDO, встречающийся внутри другого цикла PARDO, считается недопустимым. Однако в циклах PARDO могут встречаться обычные DO-циклы, последние могут охватывать циклы PARDO. Помимо DO-циклов внутри циклов PARDO допустим лишь следующий ограниченный набор операторов: арифметический оператор присваивания, оператор CALL, условные конструкции, оператор CONTINUE и оператор JUMP, похожий на оператор GOTO и предназначенный в основном для выхода из параллельного цикла.

Конвертер-векторизатор Фора-ЕС [26], разработанный в Институте прикладной математики АН СССР, предназначен для замены циклов языка Фортран IV на векторные конструкции в терминах программного обеспечения специализированного векторного процессора СПЕС.

Алгоритм распараллеливания циклов основан на методе координат, но данная реализация содержит ряд характерных особенностей:

1) операторы тела цикла, для которых распараллеливание невозможно, объединяются в супероператоры, что позволяет упростить про-

ведение анализа на возможность векторного исполнения остальных операторов;

2) операторы, входящие в состав супероператоров, оформляются в виде циклов, осттальные операторы передаются блоку векторного программирования.

Векторизатор выполнен в виде конвертора, реализованного на ЕС ЭВМ в ОС СВМ на языке Рефал.

Векторизующий ПЛ/1-компилятор ИПК АН СССР [27] для векторно-конвейерной ЭВМ типа Cray разработан в Институте проблем кибернетики АН СССР. Векторные команды генерируются по операторам "массивного"присваивания и таким итеративным циклам (не содержащим вложенных циклов), для выполнения которых векторными командами не требуется предварительного преобразования циклов.

Циклы для векторизации должны удовлетворять следующим условиям:

— заголовок цикла не содержит конструкцию WHILE, но содержит конструкцию TO, т.е. имеет вид

DO I = <expr1> TO <expr2> [BY <expr3>]

или

DO I = <expr1> BY <expr3> TO <expr2>,

где I — скалярная целочисленная неиндексированная переменная, expr1, expr2, expr3 — скалярные выражения;

— операторы в теле цикла — непомеченные операторы присваивания;

— в правых частях операторов стоят арифметические выражения, операндами которых могут быть константы, скалярные переменные, отличные от управляющей переменной цикла, индексированные переменные и вызовы встроенных функций, аргументы которых удовлетворяют требованиям для операторов;

— в левых частях операторов присваивания стоят индексированные переменные с индексом, зависящим от управляющей переменной цикла;

— каждый индекс задается линейным относительно управляющей переменной цикла;

— в цикле нет векторной зависимости, т.е. выполнение цикла векторными командами не приводит к конфликтам по памяти.

Режим векторизации задается пользователем при помощи директивы VECTORIZE на всю ПЛ/1-программу. Кроме того, имеются средства отменить векторизацию некоторого массивного присваивания или

итеративного цикла, игнорируя предполагаемую векторную зависимость, задать на всю программу режим, при котором операнды — формальные параметры — считаются препятствием для векторизации.

3. ИНСТРУМЕНТАЛЬНЫЕ СИСТЕМЫ

3.1. PTOOL [28, 6]

Система PTOOL разработана в университете Райс по заказу фирмы IBM и представляет собой шаг в направлении создания развитых диалоговых систем программирования, основанных на анализе зависимостей. Разработка и отладка программ выполняется обычными методами на модели последовательной машины, а потом пользователь ведет диалог с PTOOL по поводу распараллеливания итеративных циклов.

PTOOL позволяет пользователю выбрать цикл в последовательной Фортран-программе и спросить, могут или нет его итерации исполняться параллельно. Если ответ — "нет", то PTOOL отображает все зависимости, препятствующие распараллеливанию. При обработке цикла PTOOL определяет множество переменных, которые могут быть объявлены частными (private) для каждой итерации. Это — ключевая особенность в уменьшении числа зависимостей, показываемых пользователю, так как именно переменные, не относящиеся к частным, могут порождать зависимости. Кроме того, PTOOL может работать с вызовами внтури циклов.

PTOOL состоит из двух основных частей: анализатора PSERVE, выполняющего построение графа зависимостей и межпроцедурный анализ, и диалогового блока PQUERY, использующего построенный граф в процессе диалога с пользователем в терминах исходной программы. При помощи графа зависимостей PTOOL обнаруживает потенциальные ошибки организации доступа к данным и предупреждает о них пользователя, но не обнаруживает ошибок синхронизации.

Анализатор PSERVE — это модифицированная версия предыдущей распараллеливающей системы PFC. В PFC перед построением графа зависимостей делаются предварительные преобразования (нормализация цикла, устранение индуктивных переменных), которые могут существенно изменить структуру программы. Поскольку целью PTOOL является точное отображение зависимостей, существующих в исходном тексте, в ней эти предварительные преобразования исключаются. Например, PTOOL не удаляет индуктивные переменные, но соответствую-

щие дуги графа зависимостей помечаются как избыточные. Построение избыточных дуг требует дополнительных просмотров текста.

В процессе межпроцедурного анализа PTOOL, как и PFC, анализирует граф вызова процедур на предмет выявления побочных эффектов. Помимо того, что межпроцедурный анализ позволяет векторизовать некоторые циклы, содержащие обращения к внешним процедурам, он уменьшает количество дуг в графе зависимостей.

PQUERY — диалоговый компонент PTOOL — выдает текст мсходной программы, отмечая прямо на экране те переменные, которые образуют зависимости. Как побочный эффект возможно обнаружение некоторых семантических ошибок, вносимых при распараллеливании.

Опыт эксплуатации PTOOL сразу после ее создания позволил выделить три категории недостатков.

1. PTOOL не полностью интерактивен.
2. PTOOL не может использоваться как отладчик.
3. PTOOL отображает слишком много ложных зависимостей.

3.2. Интегрированная среда программирования R^n [29, 6]

Существенным недостатком системы PTOOL является ее неспособность поддерживать все фазы разработки программ. После внесения изменений в программу необходимо повторить ее анализ с самого начала. PTOOL не поддерживает ни фазу редактирования, ни фазу трансляции, ни фазу выполнения программы. Для решения задач такого рода служит интегрированная среда программирования R^n, также разработанная в университете Райса.

Исходной целью создания R^n было обеспечение процесса разработки и сопровождения опытными программистами больших программных систем, написанных на Фортране (не обязательно для суперЭВМ), поэтому система поддерживает межпроцедурный анализ и оптимизацию, сохраняя в то же время удобства независимой трансляции.

Транслятор и редактор, входящие в состав R^n, подразделяются каждый на две части: композиционную и модульную. Композиционные части обрабатывают целиком текст программы, состоящий из модулей. Модульные части обрабатывают каждый модуль отдельно, но при этом используют информацию, созданную композиционной частью.

Все модули представлены в виде абстрактных синтаксических деревьев и хранятся в базе данных R^n. Деревья порождаются в процессе редактирования модулей. Поэтому и сам редактор является структур-

ным, т.е. процесс ввода и изменения программ состоит в использовании заготовок (шаблонов) языковых конструкций, в который пользователь вставляет необходимые выражения, метки и т.д. Синтаксический разбор выполняется сразу же при вводе каждой конструкции, и обычный синтаксический анализатор всей программы применяется только для "чужих" программ, поступающих в R^n извне.

3.3. ParaScope [30, 31, 6]

Среда ParaScope предназначена для поддержки процесса разработки, анализа, трансляции и отладки больших параллельных программ научного характера, написанных на языке Фортран. При разработке этой системы предполагалось, что ParaScope будет включать следующие новые возможности:

— специальный редактор, объединяющий функции PTOOL и R^n и способный дать программисту советы о том, как следует преобразовать программу для повышения степени ее параллелизма; в частности, он должен уметь пошагово реконструировать зависимости после любого вида редакторской правки и быть в состоянии анализировать общие параллельные конструкции;

— глобальный анализ программы на предмет увеличения параллелизма за счет открытой подстановки процедур; программный компилятор R^n должен быть расширен для осуществления более мощных форм межпроцедурного анализа и оптимизации;

— отладка программ с использованием результатов статического анализа, подобно выполняемому в PFC и PTOOL. Отладчик должен помогать программисту искать ошибки, возникающие в оптимизированной программе при доступе к разделяемым данным.

Реализация этих возможностей привела к созданию входного редактора, компилирующей системы и системы отладки. Рассмотрим их подробнее.

Редактор PED. Редактор в ParaScope включает новую версию входного компилятора R^n, который объединяет все возможности PTOOL, устраняя его слабые места — он может, в частности, пошагово реконструировать зависимости после любой редакторской правки и анализировать общие параллельные конструкции.

Ключевой системой, автоматически выявляющей параллелизм, является *граф зависимостей*. Чтобы достичь поставленных целей, PED должен быть способен быстро перестраивать граф зависимостей в ответ

на редактирующие изменения и применение предварительных преобразований, что реализуется в виде пошагового анализатора зависимостей. В этом анализаторе имеются три принципиальных компонента: пошаговый анализатор потока данных, эффективная реализация одного или более стандартных тестов на зависимость и эффективный механизм оповещений, который позволяет каждому компоненту, включенному в анализ, информировать других об имеющихсхя изменениях. Для понимания этой организации следует смотреть на проверку зависимостей как на двухфазный процесс. Первая фаза выявляет, какие определяющие точки в программе могут воздействовать на каждое использование или определение в программе (*достигающие определения* в анализе потока данных). На второй фазе применяется тест на зависимость между определением и каждым использованием или определением, которые достигаются им.

Кроме того, планируется поддерживать в PED богатый набор преобразований, включая перестановку, разбиение и слияние циклов, расширение скаляров и др. Каждое преобразование будет реализовываться как опция меню. При выборе конкретного преобразования PED будет сообщать пользователю, можно ли установить корректность преобразования, и будет оценивать его полезность. Если пользователь решит продолжать, то PED будет применять преобразование и перестраивать граф зависимостей.

Компилирующая система. Программный компилятор есть устройство, предназначенное для оптимизации программ на глобальном уровне. Кроме того, он должен явно управлять всеми межпроцедурными оптимизациями.

Программный компилятор в R^n первоначально был сконструирован как поддержка оптимизации скалярных операций. Проблема компиляции для параллельных машин значительно более сложная и требует более сложных инструментов. Поэтому программный компилятор нуждается в доработке по следующим трем направлениям. Во-первых, требуется увеличение точности межпроцедурного анализа для обеспечения более строгого анализа в PED. Во-вторых, нужно расширить номенклатуру оптимизаций программы в целом, особенно для помощи в управлении иерархической памятью. В-третьих, требуется повысить статус модульного компилятора до генератора кода хотя бы для одной целевой машины.

После завершения разработки программный компилятор будет реа-

лизовывать следующую последовательность операций:

1. Строить граф вызовов процедур.
2. Проводить межпроцедурный анализ.
3. Собирать информацию для поддержки глобальных оптимизаций таких, как подстановка тела процедуры.
4. Проводить межпроцедурную оптимизацию.
5. Вызывать компиляцию каждого модуля, посылая информацию об окружении вызова и сторонних эффектах компилятору модулей.

Чтобы завершить задачу подготовки программы для исполнения в ParaScope, там должен быть компилятор, уделяющий внимание всем деталям генерации кода. С этой целью оптимизирующий модульный компилятор в R^n будет расширен для поддержки генерации кода для параллельных суперкомпьютеров.

3.4. Инструментальная система PAT [32]

В данной системе используется модель параллелизма SPMD в многозадачном режиме с общей памятью. В силу этого программа обрабатывается как отдельный процесс (нить, thread) управления, который создает другие процессы по мере надобности для исполнения параллельно выделенного кода. Основной техникой распараллеливания, использованной здесь, служит исполнение итераций цикла в последовательной программе как отдельных параллельных заданий.

Многое в PAT имеет параллель с системой PTOOL, однако в следующих позициях они отличаются.

- PAT предлагает фактические преобразования программы и модифицирует начальный код в дополнение к выявлению зависимостей.
- PAT распознает параллельные примитивы и обрабатывает частично параллельные программы со способностью предложения модификации существующих параллельных структур.
- "Крупнозернистые циклы", которые были вложены в критические секции, распараллеливаются.
- PAT, построенный на базе модифицированного компилятора 4.2 BSD Unix Fortran 77, переносим с использованием адаптируемого front-end, который может допускать различные примитивы и X-окна для графического отображения исходного кода и программных зависимостей.

В системе различаются две фазы в процессе распараллеливания: фаза анализа зависимостей и фаза интерактивного распараллеливания.

Анализ зависимостей. При анализе программы строится управляющий граф CFG. Вызовы подпрограмм в CFG для простоты расширяются подстановкой (т.е. каждый вызов анализируется индивидуально в контексте).

Замечание. PAT обрабатывает каждый вызов процедуры как отдельный фрагмент кода для целей распараллеливающего анализа, вынуждая пользователя гарантировать, что распараллеливающие модификации для одного вызова не будут конфликтовать с модификациями для другого вызова. Это может потребовать дублирования тела процедуры. PAT помогает в такого рода проверках.

Желаемая информация о зависимостях извлекается из прохождения путей сквозь программу с использованием списков ссылок для построения глобального графа зависимостей. Кроме обычных типов зависимостей, PAT идентифицирует зависимости, возникающие из параллельных операций. Каждый доступ к глобальным данным в фрагменте параллельного кода зависит от каждой *записи* этих данных в параллельных фрагментах. При выявлении параллельных зависимостей PAT игнорирует ссылки, защищенные *замком*, и выполняет анализ индексов.

Просмотр зависимостей обеспечивается интерактивным графическим интерфейсом. Зависимости могут рассматриваться последовательно или выборочно по номеру строки или по имени переменной.

Распараллеливание. Система PAT обеспечивает выбор циклов на распараллеливание. Имеются три распараллеливающие операции: модификация параллельных примитивов в существующем задании, конвертирование последовательного цикла к параллельному исполнению, назначение сегментов последовательного кода в качестве индивидуальных заданий.

Размещение больших блоков кода без циклов, которые могли бы исполняться параллельно, является более трудной задачей. Тесно сцепленные сегменты последовательного кода, которые мало или вообще не зависят от окружаающего кода, могут быть организованы в отдельные задания. Программист может предложить системе рассмотреть тот или иной фрагмент кода, и она будет помогать распараллеливать его, отображая зависимости в самом фрагменте и в окружающем коде.

Существуют плохо структурированные программы, которые не под-

даются анализу никакими инструментамми. В таких ситуациях PAT показывает зависимости и предлагает альтернативные примитивы. Последнее дает повод программисту реструктурировать программу.

Анализ циклов. После того как пользователь выбрал цикл, PAT определяет, какие переменные, на которые имеются ссылки внутри цикла, могут стать *локальными* по отношению к исполняющим тело цикла заданиям, а какие лучше оставить *глобальными*, или *общеиспользуемыми* (shared). Подмножество общеиспользуемых переменных должно быть также *упорядочено*, используя *события* (events), для гарантии упорядоченнности операторов присваивания по отношению к глобальным данным. На момент написания статьи [32] PAT различал переменные следующих типов: local, shared, shared locked, shared ordered. Особо следует выделить *индуктивные переменные*. Немодифицированные, они представляют собой глобальные переменные, требующие упорядочения, но они могут быть заменены локальными переменными.

PAT должен также решать, будет ли требоваться *барьер* в конце параллельного цикла; этот барьер необходим, если любые значения, порождаемые в цикле, будут позднее читаться. Например, если цикл реализует суммирование по массиву, то в каждом задании могут использоваться и локальная сумма, и глобальная сумма, увеличиваемая каждым глобально вычисленным значением. Барьер требуется для гарантии того, что все задания добавляют свою локальную сумму к конечному значению, прежде чем будет разрешено исполнение кода, следующего за циклом.

Преобразования. PAT обладает рядом преобразований кода, в частности преобразованиями, устраняющими необходимость синхронизации. К ним относятся такие преобразования, как **выравнивание индексов** (subscript alignment), **дублирование кода** (code replication), **сдвиг кода** (code shifting) **расщепление вершины** (node splitting). Каждое из них включает изменение тела цикла для минимизации или устранения доступа к конфликтующим переменным.

Еще одна забота при выборе кандидата на распараллеливание — оптимизация видимости (контекста) параллельной области. Например, PAT может помочь выделить часть задания для включения в параллельный DO-цикл. Строки, которые не требуются для каждой итерации, но требуются для исполнения всего цикла, могут быть сдвинуты за пределы области параллельного DO-цикла.

Структура циклов может быть упрощена с помощью **нормализа-**

ции цикла (loop normalization).

Имеется группа преобразований, нацеленных на улучшение функций путем учета специфики компьютера, включающая **перестановку, слияние, коллапсирование циклов** и снова **сдвиг кода**. Такие преобразования работают на структуре цикла и окружающего его кода для оптимизации структуры для конкретных целей, таких как привязка задания к определенному числу процессоров или подгонка размера массива к размеру кэш-памяти.

3.5. Система SUIF [33]

Система SUIF (Stanford University Intermediate Format), разработанная в Стэнфордском университете, представляет собой инфраструктуру для исследований в области распараллеливающих и оптимизирующих компиляторов.

SUIF была разработана как платформа для исследования методов компиляции для высокопроизводительных машин. Эта мощная, модульная, гибкая, ясно документированная и достаточно полная для компиляции больших тестовых задач система была успешно использована для проведения исследований в таких областях, как скалярная оптимизация, анализ зависимостей в массивах данных, преобразования циклов с точки зрения локальности и параллелизма, программное считывание с упреждением, планирование команд. Исследовательские проекты, использующие SUIF, включают в себя декомпозицию глобальных данных и вычислений как для машин с общей, так и с распределенной памятью, приватизацию массивов, межпроцедурное распараллеливание, эффективный анализ указателей.

Исходя из целей создания структура системы представляет собой небольшое ядро (kernel) плюс инструментарий (toolkit), состоящий из различных компиляторных анализов и оптимизаций. Ядро определяет промежуточное представление и интерфейс между проходами компилятора. Этот интерфейс всегда один и тот же, так что проходы в инструментарии могут быть легко включены, заменены и переупорядочены.

Вся информация о программах, необходимая для реализации скалярных и параллельных компиляторных оптимизаций, легко доступна из ядра SUIF. Промежуточное представление сохраняет почти всю информацию высокого уровня из исходного кода. Доступ к структурам данных и манипулирование ими прямолинейны благодаря модульной конструкции ядра.

Инструментарий SUIF содержит большое число проходов. Препроцессоры для Фортрана и ANSI Си доступны для трансляции исходных программ в SUIF, который включает параллелизатор, способный автоматически находить параллельные циклы и порождать распараллеливаемый код. SUIF-to-C-транслятор позволяет компоновать распараллеливаемый код на любой платформе, на которую перенесена параллельная библиотека времени исполнения. Система SUIF обеспечивает много возможностей для поддержки распараллеливания: анализ зависимостей по данным, распознавание редукции, множество символьных приемов для улучшения выявления параллелизма и унимодулярные преобразования для увеличения параллелизма и локальности. Включены также такие скалярные оптимизации, как частичное устранение избыточности и распределение регистров.

Строение и функции ядра. Ядро системы SUIF реализует три главные функции:

- определяет промежуточное представление программ, поддерживающее как реструктурирующие преобразования программ высокого уровня, так и анализ программ и оптимизацию низкого уровня;
- обеспечивает функции для доступа и манипулирования с промежуточным представлением;
- структурирует интерфейс между проходами компилятора. Проходы в SUIF — это отдельные программы, которые осуществляют связь через файлы. Формат этих файлов — один и тот же для всех стадий компиляции. Система поддерживает эксперименты, допуская в аннотациях определенные пользователями данные.

Программный промежуточный формат. Уровень представления программ в распараллеливающем компиляторе есть критический элемент конструкции компилятора. Традиционные компиляторы для унипроцессоров в общем случае используют представления программ, которые являются нижним уровнем для распараллеливания. Другая крайность состоит в том, что многие распараллеливающие компиляторы являются трансляторами типа текст-в-текст и их анализ и оптимизация осуществляются непосредственно на абстрактных синтаксических деревьях, поскольку эти деревья полностью сохраняют семантику языков высокого уровня; они являются языково-специфическими и не могут быть быстро адаптированы к другим языкам.

Используемый промежуточный формат — представление смешан-

ного уровня. Кроме условных операций нижнего уровня он включает в себя три конструкции высокого уровня: циклы, условные операторы и доступы к массивам. Циклические и условные представления аналогичны абстрактным синтаксическим деревьям, но в отличие от них языковонезависимы. Эти конструкты накапливают всю высокоуровневую информацию, необходимую для распараллеливания.

Когда компиляция доходит до стадии оптимизации и кодогенерации, высокоуровневые конструкты больше не нужны и расширяются до операций нижнего уровня. Результат такого расширения — простой список команд для каждой процедуры. Информация из высокоуровневого анализа может быть легко проведена до низкоуровневого представления с помощью аннотаций. Например, информация о зависимостях по данным может быть доведена до команд планирования с помощью аннотирования команд загрузки и записи в память.

Таблицы символов в SUIF-программе хранят детальную информацию о символах и типах. Информации вполне хватает для перевода выхода SUIF в высокоуровневый Си-код. Система хранит достаточно информации о фортрановских массивах и общих блоках для проведения межпроцедурного анализа. Поддержка для межпроцедурного анализа и оптимизации также встраивается в промежуточное представление.

Компиляторная инструментальная система в SUIF. Компиляторная часть системы SUIF включает Си- и Фортран-препроцессоры, оптимизатор параллелизма и локальности на уровне циклов, оптимизирующий MIPS-постпроцессор и множество инструментов для разработки компиляторов.

Чтобы достичь хорошей производительности на современных архитектурах, программы должны эффективно использовать иерархию памяти компьютера, а также его способность исполнять операции параллельно. Ключевым моментом компиляторного инстсрументария SUIF является, таким образом, библиотека и драйвер для осуществления оптимизаации уровня циклов и локальности.

SUIF-параллелизатор переводит последовательные программы в параллельный код для машин с общей памятью. Компилятор порождает SPMD-программу, которая содержит вызовы переносимой библиотеки времени исполнения. Потом используется SUIF-to-C-транслятор для конвертирования SUIF в ANSI Си.

SUIF-параллелизатор сделан из многих различных проходов компилятора. Во-первых, некоторое число скалярных оптимизаций помо-

гает выявлять параллелизм. Последние включают **протягивание констант** (constant propagation), **протягивание вперед** (forward propagation), **выявление индуктивных переменных**, **свертывание констант** (constant folding) и анализ **приватизации скаляров** (scalar privatization). Далее применяются **унимодулярные преобразования циклов** для реструктуризации кода в целях оптимизации и увеличения степени параллелизма. Наконец, генератор параллельного кода порождает код с вызовами из параллельной библиотеки времени исполнения.

В системе имеется анализатор, который распознает редукции суммы, произведения, минимума и максимума. Это достаточно трудная задача — распознать редукции, которые охватывают кратные произвольно вложенные гнезда циклов.

Параллелизатор исполняет анализ зависимостей по данным, используя SUIF-библиотеку зависимостей. Анализатор зависимостей базируется на алгоритме Майдана и состоит из серии быстрых точных тестов, каждый из которых применим в ограниченной области. Его последний тест — алгоритм исключения Моцкина-Фурье, расширенного для решения целочисленных задач. Анализатор также способен обрабатывать некоторые простые нелинейные доступы к массивам.

Преобразователь циклов основан на алгоритме Вульфа — Лэм. Он работает с циклами на базе векторов расстояний или векторов направлений. Механизмы реализации преобразований циклов запасаются в библиотеке, которую могут использовать другие пользователи для реализации различных стратегий преобразования циклов.

3.6. Система Faust [34]

Система Faust, разработанная в Иллинойском университете, — это интегрированное окружение для параллельного программирования, базирующееся на рабочих станциях и ориентированное на научные приложения. При разработке системы преследовались следующие цели:

- Конструирование и реализация множества новых инструментов, специально спроектированных для оказания помощи при разработке эффективных параллельных программ: средства интерактивной компиляции и оптимизации, а также средства для отладки и анализа производительности в параллельных окружениях.
- Интеграция новых инструментов с существующими, такими как текстовые редакторы и компиляторы, без модификации. Чтобы быть эффективным, окружение должно представлять собой ин-

тегрированное множество функций и унифицированный пользовательский интерфейс.

- Обеспечение переносимости. Хотя основной платформой для Faust'а является растровая рабочая станция, работающая под Unix, ожидается, что она будет реализована на разной аппаратуре.

В системе Faust всякая прикладная работа осуществляется в контексте проекта. Проект есть исполняемая программа. Faust выполняет функциональную интеграцию с помощью операций над общими множествами данных, поддерживаемых в каждом проекте.

Управление проектом (УП) организует компоненты проекта и манипулирует ими. Такие компоненты, называемые объектами, обычно представляют собой Unix-файлы. Простой проект состоит из единственной исполняемой программы; сложный может включать много исполняемых программ, библиотеки и хранилища данных.

УП организует объекты проекта путем построения орграфа, вершины которого суть объекты, а дуги — именованные отношения между объектами. Согласованность объектов поддерживается отношениями, основанными на времени зависимости. Объект согласован, если время его последней модификации много раньше, чем времена последней модификации всех файлов, от которых он зависит. Если обнаружен несогласованный объект, то он может стать согласованным путем исполнения сценария команд, сопоставленных с ним.

УП поддерживает восемь типов файлов, составляющих компоненты БД системы.

- **Исполняемость**. Конечный целевой объект.
- **Источник**. Исходный программный текст, написанный на Фортрани или Си.
- **Объект**. Промежуточные файлы, генерируемые системными компиляторами, когда они порождают исполняемую программу.
- **Ассемблер**. Версии входного текста на языке ассемблера, порожденные системными компиляторами.
- **Зависимость**. Информация о символьных таблицах и зависимостях по данным, собранная компиляторами Faust'а для обращения со стороны реструктурирующего окружения.
- **Программный граф**. Статический граф вызова процедур, используемый графическим просмотрщиком Faust'а.

- **Маршрут исполнения**. Собранный во время прогона файл как результат мониторинга, осуществляемого инструментами оценки производительности. Эти маршрутные файлы используются инструментами анализа производительности и визуализации.
- **Аннотации**. Детальная информация о модификациях, полученных при применении инструментов Faust'a. Например, каждый маршрут исполнения, выдаваемый инструментами анализа эффективности, имеет файл аннотаций, который содержит детальное описание собранных данных о производительности и основание для сбора их.

Инструменты уровня пользователя. Для того чтобы помочь пользователю параллельных суперкомпьютеров настроить и оптимизировать прикладную программу, в составе системы Faust имеется подсистема Sigma.

На самом нижнем уровне Sigma представляет собой работающий с мышью многооконный текстовый редактор с интерфейсом-оболочкой. Мощность системы Sigma лежит в ее интерфейсе с программной БД Faust'a.

База данных прикладных проектов содержит:

- полный анализ зависимостей по данным (внутрипроцедурный и межпроцедурный) входных файлов приложений;
- управляющий граф с достаточной информацией для восстановления исходного входного файла (включая комментарии);
- сборку и анализ объектного кода, сгенерированного компилятором.

После построения прикладной программы УП использует специальные лексические анализаторы для генерации БД проектов, к которой можно обратиться с запросом из редактора Sigma. Например, когда исходный файл с прикладной программой вызывается на экран, с помощью мыши можно выбрать элемент данных, такой как переменная или имя функции, сделать запросы и задать команды типа:

- Где была инициализирована или последний раз модифицирована данная переменная?
- Какие подпрограммы модифицируют или используют эту переменную?
- Какие сторонние эффекты имеет вызов даной процедуры или функции и какие сегменты массивов-параметров используются или модифицируются?

- Может ли этот цикл быть распараллелен или векторизован? Если нет, то какие переменные мешают распараллеливанию?
- Если эта переменная является указателем на Си-структуру или объектом в Си++ классе, то что представляют собой поля (операторы) в такой структуре (объекте)?
- Сгенерировать оценку отношений совпадений кэша для массивных объектов в этом сегменте кода.
- Сгенерировать оценку эффективности кода (измеряемой числом операций с плавающей точкой в секунду) для этого сегмента кода.
- Нарисовать статический граф вызовов функций.
- Нарисовать граф зависимостей по данным для заданного сегмента кода.

Faust предоставляет пользователю меню заранее определенных трансформаций программ, включая векторизацию цикла, распараллеливание, перестановку циклов, разбиение на блоки и некоторые машинно-ориентированные преобразования. К ним добавляются другие меню опций, включающие подпрограммы расширения, инкапсуляции и локализации переменных. Если пользователь попытается преобразовать программу с нарушением исходной семантики, то он будет предупрежден, что преобразование изменит смысл программы.

При переносе программ на параллельные компьютеры применяется двухшаговый процесс. Первый шаг — использование любого доступного автоматического реструктуризатора; второй шаг — начало реструктурирования программы путем преобразования сегментов кода для извлечения большего количества параллелизма.

Еще одна подсистема — Impact — объединяет все инструменты, которые собирают данные о производительности, с инструментами, которые анализируют и отображают результаты, касающиеся производительности. В составе Impact'а есть также инструмент для трассировки многозадачных событий, который отображает их на линейную шкалу времени. В главном окне Impact'а пользователь может выбрать каталог трасс, который хранит файлы трассы: файл определения событий, который описывает типы событий,встретившихся в трассе; файл событий трассы, который содержит упорядоченный во времени список всех порожденных событий.

3.7. Близкие работы

3.7.1. Система FALCON [35]

Прототипное окружение FALCON включает средства для быстрого прототипирования и для интерактивных и/или автоматических преобразований программ на уровне операторов, функциональном или алгоритмическом уровнях для получения эффективной программы. FALCON поддерживает разработку и переиспользование численных программ и библиотек и комбинирует технику преобразований и анализа, используемых в реструктурирующем компиляторе, с алгебраическими методами, используемыми разработчиками для выражения и манипулирования алгоритмами.

Основное усилие разработчиков в этом исследовании было направлено на создание интерактивной реструктурирующей системы, которая с помощью сохраняющих корректность преобразований приведет к эффективному коду из исходного прототипа. Разработана однородная система преобразований, основанная на расширяемом семействе переписывающих правил. Некоторые преобразования будут действовать на отдельный оператор, в то время как другие, например выбор структуры данных, оказывают глобальное влияние на код. Система FALCON существенно опирается на библиотеку MATLAB, которая предоставляет инструменты (язык, библиотеку модулей, графику, ввод/вывод и др.) в качестве первичного объекта прототипирования.

Процесс построения эффективной программы начинается с выбора прототипа в форме MATLAB-программы. Прототип анализируется для сбора информации о типе и виде каждой переменной, а также о зависимостях. Эта информация необходима для обработки в следующей фазе — интерактивной реструктуризации.

На фазе анализа программ MATLAB-программа преобразуется в абстрактное синтаксическое дерево (AST). Во время этого процесса проводится анализ программы с целью выявления параллелизма. На стадии анализа система использует некоторое число подходящих алгоритмов преобразований, таких как алгоритм распознавания индуктивныых переменных для вычисления их верхних и нижних границ этих и втягивания интервалов изменения значений скаляров и массивов, используемых как индексы.

На фазе реструктуризации пользователь может выбрать для реструктуризации любой фрагмент программы, включая составные опе-

раторы, подвыражения и вызовы функций. Система будет потом определять, какие из имеющихся в ее БД преобразований применить к указанному фрагменту. Система также позволяет пользователюю выбирать, какие применимые преобразования использовать.

3.7.2. Система PTOPP [36]

Система PTOPP служит поддержкой достаточно прямолинейного подхода к оптимизации программ, при котором вначале определяются те части программы, которые потребляют много времени, а затем эти части последовательно улучшаются. Система состоит из ряда ИС, рассчитанных на программистов-профессионалов с хорошим пониманием Unix и Fortran. Предполагается, что пользователь свободно владеет понятиями параллельных архитектур и типов преобразований, полезных для программ для этих архитектур. Развитие этой системы направлено на удовлетворение потребностей широкого круга пользователей — от прикладных программистов, которые не имеют возможности тратить время на оптимизацию параллельных программ, до экспертов-программистов, которые хотят эксплуатировать все особенности параллельных машин.

В качестве первого интерактивного интерфейса был выбран Emacs, который предлагает полноценный текстовый редактор, средства управления файлами, интерактивную подсказку (help) и хорошо известный язык Elisp, на котором можно программировать. Замена Emacs на альтернативную систему управления файлами с аналогичными возможностями производится непосредственно.

Большей частью анализ проводится на базе изучения циклов. С этой целью осуществляется инструментовка всех гнезд циклов в последовательном входном коде. Все варианты оптимизированной программы порождаются из инструментованного входного кода.

PTOPP обеспечивает такие средства, что инструментовка программы может быть произведена неявно с помощью команды cfmake в сочетании с командой instrument или с помощью редактора, просматривающего входной код.

За шагом инструментовки программы следует генерация и исполнение многочисленных вариантов программы. Примерами вариантов программ, интересных в исследовании эффективности программы, служат последовательные, векторизованные и векторно-параллельные программы.

Для того чтобы найти преобразования программ, которые могли бы улучшить производительность, данные о результирующей программе собираются и анализируются относительно некоторого числа **оптимизационных факторов**. Последние обеспечивают информацию о производительности программы на основе знания структуры циклов и делают намеки для оптимизирующих преобразований. Примерами таких факторов могут служить ускорение цикла (loop speedup) и штраф за глобализацию (globalization penalty) (т.е. производительность уменьшается, когда данные помещаются не в локальную, а в глобальную память).

Редактирование преобразований — достаточно простая задача, хотя некоторые из них требуют аккуратного межпроцедурного анализа программ. Примерами таких преобразований могут служить **приватизация массивов** и превращение цикла в параллельный цикл. Редактирующее действие приватизации массивов состоит в перемещении объявлений данных из исходной позиции в заголовок цикла и создании параллельного цикла заменой DO на DOALL. Однако анализ приватизируемых массивов требует внимательного определения связей def-use, а превращение цикла в полностью параллельный цикл требует анализа зависимостей по данным.

Другие преобразования легче анализируются, но более утомительны для редактирования. Они могут быть кандидатами на поддержку со стороны ИС. Примерами служат преобразования **сегментирование** (strip mining) и **сжатие цикла** (loop coalescing).

3.7.3. Система PEPP [37]

PEPP — инструментальная система, нацеленная на создание и оценку стохастических граф-моделей параллельных и распределенных программ. Указанные модели изображают исполняемый порядок частей программы, распределение времен прогона и вероятности ветвления.

PEPP может также использоваться для управляемого событиями мониторинга. И в моделировании, и в мониторинге важные части параллельной программы представлены как события, а полное динамическое поведение — как трассы событий. Использование одного и того же набора событий для моделирования и мониторинга имеет следующие преимущества:

- Предсказание производительности может быть выполнено с реальными параметрами. Использование параметров, полученных

от мониторинга, делает предсказание производительности более соответствующим решаемой задаче.

- Спецификация событий для мониторинга может быть построена из моделей, получающихся в систематической инструментовке.

Интеграция моделирования и мониторинга называется *модельно-управляемым мониторингом*. Модель, на которой базируется инструментовка, называется *мониторинговой моделью*. Основа модельно-управляемого мониторинга — *модельно-управляемая инструментовка*.

Следующие средства модельно-управляемого мониторинга поддерживаются системой PEPP.

Создание модели — первый шаг, состоящий в создании модели мониторинга. В качестве ее берется стохастическая граф-модель, которая может быть создана интерактивно с графической поддержкой.

Инструментовка. Модельно-управляемая инструментовка имеет перед интуитивной инструментовкой ряд преимуществ: она может выполняться систематически и инструментально-поддерживаемым способом. Все процессы в программе, которые подлежат исследованию и представлены в мониторинговой модели, будут инструментованы.

Проверка правильности. Полученная после инструментовки программа исполняется и наблюдается, давая в результате трассу событий. Мониторинговая модель может быть использована для проверки правильности динамического поведения программы: отслеженное поведение, представленное в трассе событий, автоматически проверяется относительно функционального поведения модели. Такой способ проверки обеспечивает краткие указания для нахождения программных ошибок. Если записанная трасса событий отображает мониторинговую модель, то трасса может быть использована как основа для анализа программ.

Анализ трасс. В PEPP имеются средства для проведения анализа трасс модельно-управляемым способом.

Модельная оценка. Результаты анализа трасс событий могут быть также использованы для создания модели производительности с реалистичными параметрами. Результирующая модель производительности есть самый необходимый инструмент для предсказания производительности еще не реализованной версии программы, отображения процессов или других компьютерных конфигураций.

3.7.4. Система CHIEF [38]

Известно, что производительность суперкомпьютеров достаточно широко варьируется — при переходе от одного приложения к другому, от одного алгоритма к другому и даже от одного размера задачи к другому. Для понимания поведения существующих систем может быть использовано хорошо продуманное средство моделирования, которое также может быть применено для оценки вариантов существующих систем и в процессе проектирования сверху вниз новых суперкомпьютеров и параллельных систем.

Проект CHIEF обеспечивает интегрированное множество инструментов для создания, отладки, прогона и анализа моделей перспективных параллельных вычислительных систем. Имеется два входа в систему: множество тестовых программ и архитектурная спецификация. Тестовые программы обрабатываются реструктурирующим компилятором согласно архитектуре целевой системы.

Архитектура изучаемой системы определяется как иерархическое множество компонент. Препроцессоры системы CHIEF конвертируют спецификации во входной код, который может быть использован для построения симуляторов вокруг некоторых ядер, основываясь на трех различных парадигмах: гибридной модели, управляемой временем и событиями, консервативной распределенной модели событий и оптимистичной распределенной модели событий.

В целом проект CHIEF представляет собой мощное окружение для изучения параллельных систем. Окружение включает инструмент для извлечения параллелизма, устройства генерации трасс, язык моделирования, мощный пользовательский интерфейс и инструменты для сбора и визуализации исполняемых данных.

3.7.5. Система Delta [39]

Delta представлляет собой систему, работающую с Фортран-программами, так как этот язык реализован в большинстве суперкомпьютеров. Она обращается с программами как система символьной алгебры и включает в себя большую коллекцию функций анализа и преобразований, которые могут комбинироваться, используя язык высокого уровня. Алгоритмы символьной алгебры нужны в процессе распараллеливания для достижения ряда целей, включая символьное тестирование зависимостей и работу с индуктивными переменными.

Главная цель проекта Delta — создать инструмент для быстрого прототипирования реструктурирующих алгоритмов. Программы, намеченные к преобразованию, представляются в виде абстрактных синтаксических деревьев.

3.8. Инструментальные системы, ориентированные на межпроцедурный анализ

3.8.1. FIAT/SUIF [40, 41]

Для того чтобы система SUIF могла распараллеливлать циклы, содержащие вызовы процедур, она дополняется системой FIAT, ориентированной на межпроцедурный анализ. Более точно, FIAT представляет собой инструмент для построения компиляторов, дающий возможность быстро прототипировать межпроцедурный анализ и компилирующие системы [40]. Первоначально FIAT был интегрирован в систему ParaScope. При работе с FIAT программист должен только обеспечить функции инициализации, оператор объединения, функцию передачи и направление анализа. FIAT — система, управляемая спросом: программист не должен беспокоиться о порядке межпроцедурного анализа. В FIAT/SUIF доступны несколько видов межпроцедурного анализа. Вначале обратный анализ вычисляет передаточную функцию, затем некоторые виды предусловий, которые дают каждое значение переменной в терминах инвариантов цикла и индексов вложенных циклов. Дополнительная фаза протягивает ограничения в виде неравенств из тестовых условий. Наконец, реализуются некоторые виды обратного межпроцедурного анализа массивов.

3.8.2. Parascope и система D [42, 43]

Parascope и система D, разработанные в университете Райса, используют для межпроцедурного анализа систему FIAT так же, как это сделано в FIAT/SUIF. Parascope обеспечивает проведение некоторых видов межпроцедурного анализа, таких как MOD/REF анализ, анализ массивных областей, основанный на *дескрипторах регулярных секций* [44], анализ смешивания, протягивание констант и анализ символьных величин. Система D построена в контексте Parascope и имеет целью трансляцию Фортран-D-программ. Фортран D — диалект Фортрана, подобный HPF — требует проведения специальных видов анализа, таких как *достигающая декомпозиция* и *перекрытие*.

3.8.3. Система Polaris [45]

Polaris разработан в Иллинойском университете. Большая часть фаз Polaris'а интрапроцедуральны и требуют подстановки тел вызываемых подпрограмм для эффективного распараллеливания циклов. В системе поддерживается автоматическая подстановка, а некоторые межпроцедурные фазы, такие как протягивание констант, либо реализованы, либо такая реализация планируется (например анализ массивных областей, основанный на списках *дескрипторов регулярных секций*).

3.8.4. Система Panorama [46]

Распараллеливающий компилятор Panorama, разработанный в университете Миннесоты для поддержки систем с иерархической памятью, реализует решение ряда задач межпроцедурного анализа, таких как построение def-use-цепочек и исследование потоко-чувствительных секций массивов на основе дескрипторов защищенных секций массивов. Межпроцедурный анализ проводится на программном иерархическом суперграфе — расширении суперграфа Майера, что дает возможность создать необходимые условия для проведения потоково- и контекстно-чувствительного анализа. При этом не используется никакая подсистема типа FIAT и при решении каждой задаи анализа нужно переопределять свой собственный обход суперграфа.

3.8.5. Система PIPS [47]

PIPS представляет собой текст-в-текст Фортран-компиляторное инструментальное средство, служащее инфраструктурой общего назначения для целей межпроцедурного анализа. PIPS использует *неявный* граф вызовов процедур и позволяет проводить потоково- и контекстно-чувствительный анализ, так как представление программы содержит индивидуальные управляющие графы всех процедур. Суммирование используется для сохранения линейной сложности всех видов межпроцедурного анализа при росте размера программы, что достигается путем исключения управляющей информации и устранения списочных структур данных, размер которых может расти вместе с высотой процедуры в графе вызовов. Трансляция сквозь границы процедур реализуется на каждой дуге графа вызовов, используя соответствия между формальными и фактическими параметрами и между объявлениями типа common.

PIPS — пока единственная система, сравнимая с другими подходами. Во-первых, она обеспечивает управляемое спросом средство, которое уникально, так как PIPS был специально сконструирован для исследования межпроцедурных связей. Во-вторых, PIPS предлагает представительный набор межпроцедурных анализов и преобразований, эксплуатирующих собранную символьную информацию, например частичное вычисление.

4. ВЫБОР КАНДИДАТОВ НА РАСПАРАЛЛЕЛИВАНИЕ

Как ясно из сказанного выше, большинство векторизующих компиляторов содержат в явном или неявном виде средства для оценки целесообразности распараллеливания того или иного цикла (гнезда циклов). Критерием отбора, чаще всего индивидуальным, служит отношение стоимости распараллеливания к получаемому ускорению исполнения программы. Очевидно, что этот критерий скрытый и не поддается непосредственному измерению. Поэтому он заменяется совокупностью других, поддающихся измерению или качественной оценке, критериев, называемых *индикаторами*. К ним относятся, например, объем дополнительно вводимого в программу скалярного кода, уменьшение размера тела цикла за счет выноса инвариантных вычислений, удаления общих подвыражений, преобразований цикла типа сегментации, разбиения или, наоборот, слияния циклов, выноса за пределы цикла условных операторов, ликвидации передач управления извне цикла в тело цикла и др.

Основой для многих решений служит граф зависимостей по данным, возможно дополненный вспомогательной информацией типа глубины зависимости для гнезда циклов или векторов направлений зависимости. Исследование графа выявлет области нераспараллеливания или последовательного исполнения, а также структуру таких областей. В частности, изучение контуров зависимости на предмет однородности дуг контура может показать, что контур может быть разрушен. Здесь возможно применение таких преобразований, как переименование переменных (или перевод программы в SSA-форму).

Заметим, что наличие большого количества включенных в компилятор преобразований не всегда облегчает задачу распараллеливания, так как делает компилятор громоздким. Для устранения этого недостатка разрабатываются вспомогательные инструментальные системы, которые позволяют оценить целесообразность применения того или иного

преобразования, не затрагивая всей программы в целом. Информация о полученных результатах может стать доступной компилятору путем включения в текст программы директив пользователя.

Важно учитывать также непосредственные характеристики целевой машины. Так, информация о структуре конвейеров позволяет определить наименьшее число итераций, начиная с которого векторизация становится выгодной. Кроме того, в машине могут быть векторные команды для распараллеливания линейной рекуррентности, а также задач редукционного типа (скалярное произведение векторов, отыскание наибольшего элемента вектора, нахождение суммы элементов вектора и т.д.). Еще одна важная характеристика машины — длина векторных регистров; несоответствие между длиной регистров и числом итераций цикла требует проведения сегментации циклов.

Проект ПРОГРЕСС предусматривает создание системы манипулирования программами, на базе которой должен создаваться прототип распараллеливающего компилятора в соответствии со спецификациями целевой машины. Предполагается, что прототип будет иметь модульную структуру, включающую в себя препроцессор с соответствующего языка, модуль анализа зависимостей по данным, модуль промежуточного представления, модуль общей оптимизации, модуль реструктуризации, модуль архитектурно зависимой оптимизации, генератор кода. Сказанное выше позволяет сделать следующее уточнение. Модуль анализа зависимостей дополняется подмодулем отбора кандидатов на распараллеливание, отбрасывающим все циклы, которые не могут обрабатываться с помощью средств, включенных в данный прототип.

Данный подмодуль дает ответ на ряд вопросов типа: может ли он быть разбит на меньшие циклы, содержит ли операторы ввода/вывода, вызова процедур, функций, подпрограмм, линейную рекурсию, оператор IF, возможна ли передача управления извне цикла в тело цикла и т.д. Положительный ответ на первый вопрос влечет за собой немедленное преобразование цикла на меньшие (преобразование fission-by-name [48]), что существенно облегчает получение ответов на последующие перечисленные выше вопросы. Цель этих вопросов — выявить заведомо невекторизуемые циклы. Кроме того, на этом этапе решается вопрос о целесообразности векторного исполнения цикла из-за малого числа итераций, а также сегментации циклов.

Другая цель — выявить заведомо векторизуемые циклы — может быть достигнута на основе анализа зависимостей по данным. Заметим,

что этот анализ, как правило, проводится для нормализованных циклов (индексные переменные изменяются от 1 до верхней границы с шагом 1, и все индуктивные переменные удалены), хотя встречаются ситуации, когда анализ должен проводиться для ненормализованных циклов. В таком случае эти ситуации должны быть формализованы, а соответствующий цикл должен быть особо помечен.

Модуль анализа зависимостей по данным включается после модуля оптимизационных преобразований общего назначения (классических преобразований в терминологии работы [49]). Анализ выявленных зависимостей позволяет выделить безусловно векторизуемые циклы и циклы, подлежащие дополнительному исследованию на базе подсистемы реструктуризации системы ПРОГРЕСС. Результаты исследований оформляются либо в виде директив, либо в виде библиотеки, доступной прототипу компилятора.

СПИСОК ЛИТЕРАТУРЫ

1. **Evstigneev V., Kasyanov V.** The PROGRESS program manipulation system // Proc. PACT-93, 1993. — Vol. 3. — P. 651–656.
2. **Evstigneev V., Kasyanov V.** A rapid compiler prototyping system for fine-grained concurrence architectures // Proc. 2nd Int. Conf. on Software for Multim. and Supercomp. Theory, Practice, Exp. (SMS TRE 94). — Moscow, 1994. — P. 32–38.
3. **Евстигнеев В.А., Касьянов В.Н.** Инструментальная система для изучения преобразований программ // Интеллектуализация и качество программного обеспечения. — Новосибирск, 1994. — С. 90–99.
4. **Evstigneev V., Kasyanov V.** A program manipulation system for fine-grained architectures // Proc. First Int. Conf. EURO-PAR, Stockholm. — Berlin a.o.: Springer Verlag, 1995. — (Lect. Notes Comp. Sci.: vol. 966). — P. 719–722.
5. **Evstigneev V., Kasyanov V.** A program manipulation system for fine-grained architectures // Proc. 3rd Int. Conf. PaCT-95. — Berlin a.o.: Springer Verlag, 1995. — (Lect. Notes Comp. Sci.: vol. 964). — P. 163–168.
6. **Черняев А.П.** Программные системы векторизации и распараллеливания Фортран-программ для некоторых векторно-конвейерных ЭВМ (обзор) // Программирование. — 1991. — N 2. — С. 53–68.
7. **Allen F., Burke M., Charles Ph. a.o.** An overview of the PTRAN analysis system for multiprocessing // J. Parallel and Distrib. Comput. — 1988. — V. 5, N 5. — P. 617–640.
8. **Allen J.R., Kennedy K.** Automatic translation of Fortran programs to vector form // ACM Trans. Program Lang. Syst. — 1987. — Vol. 9, N 4. — P. 491–542. — Рус. перевод см. Векторизация программ: теория, методы, реализация: Сб. статей/ Под. ред. Г.Д.Чинина. — М.: Мир, 1991. — С. 77–140.
9. **Черняев А.П.** Системы программирования для высокопроизводительных ЭВМ. — Т.3. Вычислительные науки. — М., 1990. — С. 1–141. — (Итоги науки и техн. ВИНИТИ АН СССР).

10. **Хьюзон К., Мак Т., Дейвис Д. и др.** KAP/205: усовершенствованный векторизатор типа текст-текст для суперкомпьютера Cyber 205 // Векторизация программ: теория, методы, реализация: Сб. статей/ Под. ред. Г.Д.Чинина. — М.: Мир, 1991. — С. 217–235.
11. **Мак Т., Хьюзон К., Дейвис Д. и др.** KAP/ST-100: Фортран-транслятор для присоединенного процессора ST-100 // Там же. — С. 202–216.
12. **Дейвис Д., Хьюзон К., Мак Т. и др.** KAP/S-1: усовершенствованный векторизатор типа текст-текст для суперкомпьютера S-1 Mark IIa // Там же. — С. 236–245.
13. **Миура К.** СуперЭВМ фирмы Fujitsu: векторная система FACOM // СуперЭВМ. Аппаратная и программная организация. — М.: Радио и связь, 1991. — С. 166–183.
14. **Ватанабе Т., Катияма Х., Ивая А.** Супер ЭВМ фирмы NEC: семейство SX // Там же. — С. 184–200.
15. **Одака Т., Нагасима С., Канабэ С.** Векторная суперсистема S-810 фирмы Hitachi // Там же. — С. 140–165.
16. **Katayama H., Tsukagoshi M.** Fortran and tuning utilities aiming at ease of use of a supercomputer // Fall Joint Comput. Conf., Dallas, Tex., May 2-6, 1986: Proc., 1986. — P. 1034–1040.
17. **Scarborough R.G., Kolsky H.G.** A vectorizing Fortran compiler // IBM J Res. Develop. — 1986. — Vol. 30, N 2. — P. 163–171.
18. **Dodson D.S., Metzger R.C., Smith P.E.** Optimize supercomputer code with vectorizing compilers // Electronic Design. — 1988. — Vol. 36, N 5. — P. 79–84.
19. **Хаммер К.** Паскаль-компилятор для векторного процессора // Векторизация программ: теория, методы, реализация. Сб. статей / Под ред. Г.Д.Чинина. — М.: Мир, 1991. — С. 192–201.
20. **Coleman H.B.** The vectorizing compiler for the Unisys ISP // Proc. Int. Conf. Parallel Process., Aug. 17–21, 1987. — University Park, PA, 1987. — P. 567–576.
21. **Бабичев А.В., Лебедев В.Г.** Распараллеливание программных циклов // Программирование. — 1983. — N 5. — С. 52–63.
22. **Разработка** системного и прикладного программного обеспечения МВК ПС-2000/2100, ПС 3000/3100: Всесоюзн. научно-технич. семинар. Тез. докл. — М., 1990.
23. **Бабичев А.В., Лебедев В.Г., Трахтенгерц Э.А.** Построение транслятора для многопроцессорных вычислительных систем // Кибернетика. — 1985. — N 1. — С. 38–44.
24. **Высокопроизводительные** вычислительные системы: III Всесоюзн. совещ. / Тез. докл. — Таллин, 1988.
25. **Варченко В.С., Натансон Л.Г.** Разработка и реализация векторного Фортрана для М-10 // Программирование. — 1986. — N 4. — С. 47–58.
26. **Задыхайло И.Б., Зеленецкий С.Д., Платонова Л.Н. и др.** Фора-ЕС: система программирования Фортран-IV для многопроцессорного вычислительного комплекса ПС-3000. — М., 1987. — (Препр. / Ин-т прикладной математ,; N 17).
27. **Иванников В,П., Нестеров С.В., Черняев А.П.** Сравнительное исследование трансляторов ПЛ/1 для векторно-конвейерной ЭВМ. — М., 1986. — (Препр. / АН СССР. Научн. совет по комплекс. проблеме "Кибернетика").
28. **Carle A., Cooper K.D., Hood R.T. a.o.** A practical environment for scientific programming // Computer. — 1987. — Nov. — P. 75–89.

29. **Cooper K.D., Kennedy K., Torczon L.** The impact of interprocedural analysis and optimization in the R^n programming environment // ACM TOPLAS. — 1986. — Vol. 8, N 4. — P. 491–523.
30. **Callahan D., Cooper K., Hood R.T. a.o.** Parallel programming support in ParaScope // Lect. Notes Comput. Sci. — 1987. — Vol. 297. — P. 91–105.
31. **Callahan D., Cooper K., Hood R.T. a.o.** ParaScope: a parallel programming environment // The Int. J. of Supercomputer Appl. — 1988. — Vol. 2, N 4. — P.84–99.
32. **Smith K., Appelbe W.F.** PAT - An interactive Fortran parallelizing assistant tool // Proc. 1988 Int. Conf. on Parallel Proc. University Park, Pa. Aug. 15–19. 1988. — P. 58–62.
33. **Wilson R.P., French R.S., Wilson C.S. a.o.** SUIF: An infrastructure for research on parallelizing and optimizing compilers // SIGPLAN Not. — 1994. — Vol. 29, N 12. — P. 31–37.
34. **Guarna V.A., Jr., Gannon D., Jablonowski D. a.o.** Faust: An integrated environment for parallel programming // IEEE Software. — July 1989. — P. 20–26.
35. **De Rose L., Gallivan K., Gallopoulos E. a.o.** FALCON: an environment for the development of scientific libraries and applications. — Univ. Ill.: CSRD Rep. 1437, Sept. 1995.
36. **Eigenmann R., McClaughry P.** Practical tools for optimizing parallel programs. — Univ. Ill.: CSRD Rep. 1276, Jan. 1993.
37. **Quick A.** PEPP: performance evaluation of parallel programs // 6th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation, Edinburgh, 16–18 Sept. 1992.
38. **Bruner J., Cheong H., Veidenbaum A., Yew P.-C.** CHIFF: a parallel simulation environment for parallel systems. — Univ. Ill.:CSRD. Rep. 1050, April 1991.
39. **Padua D.** The Delta program manipulation system. Preliminary design. — Univ. Ill.: CSRD Rep. 880, June 1989.
40. **Hall M., Meller-Crummey J., Carle A., Rodriguez R.** FIAT: A framework for interprocedural analysis and transformation // Lect. Notes Comput. Sci. — 1993. — Vol. 768. — P. 522–545
41. **Hall M., Murphy B., Amarasinghe S. a.o.** Interprocedural analysis for parallelization // Lect. Notes Comput. Sci, — 1996. — Vol. 1033. — P. 61–80.
42. **Cooper K., Hall M., Hood R. a.o.** The Parascope parallel programming environment // Proc. IEEE. — 1993. — Vol. 81, N 2. — P. 224–263.
43. **Hall M., Hiranandani S., Kennedy K., Tseng C.-W.** Interprocedural compilation of Fortran D // J. of Parallel and Distrib. Comp. — 1996. — Vol. 38, N 2. — 114–129.
44. **Blume W., Eigenmann R.** Performance analysis of parallelizing compilers on the Perfect Benchmarks programs // IEEE Trans. on Parallel and Distrib. Syst. — 1992. — Vol. 3, N 6. — P. 643–656.
45. **Blume W., Doallo R., Eigenmann R. a. o.** Parallel programming with Polaris // Computer. — 1992. — Vol.29, N 12. — P. 78–82.
46. **Nguen T., Gu J., Li Z.** An interprocedural parallelizing compiler and its support for memory hierarchy research // Lect. Notes Comput. Sci. — 1995. — Vol. 1033. — P. 96–110.
47. **Creusillet B., Irigoin F.** Interprocedural analyses of Fortran programs. — Ecole des Mines de Paris: Tech. Rep., 1997.

48. **Падуа Д., Вольф М.** Оптимизация в компиляторах для суперкомпьютеров // Векторизация программ: теория, методы, реализация: Сб. статей / Под. ред. Г.Д.Чинина. — М.: Мир, 1991. — С. 7–47.
49. **Евстигнеев В.А., Касьянов В.Н.** Оптимизирующие преобразования в распараллеливающих компиляторах // Программирование. — 1996. — N 6. — С. 12–26.

В. А. Евстигнеев, И. Л. Мирзуитова

АНАЛИЗ ЦИКЛОВ: ВЫБОР КАНДИДАТОВ НА РАСПАРАЛЛЕЛИВАНИЕ

Препринт
58

Рукопись поступила в редакцию 13.11.1998
Рецензент И. Б. Вирбицкайте
Редактор Л. А. Карева

Подписано в печать 13.10.1999
Формат бумаги 60×84 1/16 Объем 2,8 уч.-изд.л., 3 п.л.
Тираж 50 экз.

ЗАО РИЦ "Прайс-курьер"630090, г. Новосибирск, пр. Акад. Лаврентьева, 6