В. Н. Касьянов, И. Л. Мирзуитова

РЕСТРУКТУРИРУЮЩИЕ ПРЕОБРАЗОВАНИЯ: АЛГОРИТМЫ РАСПАРАЛЛЕЛИВАНИЯ ЦИКЛОВ*

1. ВВЕДЕНИЕ

Существует три основных подхода к обеспечению эффективной эксплуатации суперкомпьютеров как машин с параллельной архитектурой: использование параллельных языков, использование библиотек и автоматическое распараллеливание программ. Все три направления интенсивно развиваются, но наиболее привлекательным остается третий подход, поскольку решает проблему переносимости старых последовательных программ на суперкомпьютеры.

Несмотря на то что автоматическое распараллеливание программ — достаточно трудная задача, существующее ее решение обеспечило коммерческий успех суперкомпьютеров на быстро меняющемся рынке вычислительной техники. В настоящее время по грубой оценке в мире эксплуатируется порядка тысячи суперкомпьютеров различных параллельных архитектур с производительностью, измеряемой в гигафлопах и даже выше. На большинстве из них в качестве языка программирования используется Фортран (Фортран 77, Фортран 90), а в состав программного обеспечения входит распараллеливающий Фортран-компилятор.

Создание успешно работающих распараллеливающих компиляторов для ЭВМ с параллельной архитектурой (векторно-конвейерные ЭВМ, мультипроцессорные системы, машины с VLIW-архитектурой) стало возможным за счет развития методов оптимизирующей трансляции. Произошло существенное расширение класса преобразований, необходимых для достижения приемлемой эффективности при трансляции с языков высокого уровня, за счет включения в него конвейеризующих, векторизующих и других преобразований, ориентированных на те или иные особенности архитектур перспективных ЭВМ [1–5, 7, 8, 10–12, 23, 18–22, 30, 31]. Класс оптимизаций

* Работа выполнена при финансовой поддержке Научной программы «Университеты России» (грант № УР.04.01.027) и Министерства образования РФ (грант № Е02-1.0-42).

расширился не только преобразованиями программ в рамках одной модели вычислений, например некоторой модели последовательных или асинхронных вычислений, но и так называемыми *конструирующими* преобразованиями[1], переводящими программы из одних моделей в другие, например, сопоставляющими определенным последовательным программам эквивалентные им параллельные. При этом процессы оптимизирующей трансляции, так или иначе реализуемые распараллеливающими компиляторами, осуществляются по «типовой» схеме трансформационного конструирования эффективной программы (трансформационного синтеза) [15, 16]. Схема предполагает последовательное выполнение следующих шагов.

1. Применение оптимизирующих преобразований, позволяющих приводить программу в рамках исходной модели вычислений к ориентированному на применение конструирующих преобразований виду.
2. Применение конструирующих преобразований, переводящих программу из исходной модели в результирующую.
3. Применение оптимизирующих преобразований в рамках результирующей модели.

Расширение оптимизаций привело к следующим изменениям в их классификации [10].

Поскольку параллельные машины обладают большим разнообразием архитектур, наряду с разбиением, имеющим место для оптимизаций для последовательных ЭВМ, произошло выделение в классе машинно-зависимых оптимизаций не только машинно-ориентированных, но и архитектурно-ориентированных оптимизаций. Этот подкласс, как и машинно-ориентированные преобразования, содержит оптимизации, в которых зависимости от конкретной машины могут быть вынесены в набор параметров, что позволяет реализовать их не на уровне машинного языка конкретной ЭВМ. Фактически машинно-ориентированные оптимизации можно рассматривать как архитектурно-ориентированные преобразования для класса последовательных машин. Степень зависимости преобразований от конкретной архитектуры может быть различной, например, можно различать мелкозернистые оптимизации, которые ориентированы на архитектуры, эксплуатирующие мелкозернистый параллелизм, и VLIW-оптимизации, реализуемые в рамках языка абстрактной VLIW-машины [31].

В рамках разделения оптимизаций по способу реализации (смешанная стратегия, оптимизирующие преобразования) произошли следующие изме-

[1] Здесь мы используем классификацию оптимизаций, предложенную В.Н. Касьяновым; она является обобщением предложенной им классификации оптимизаций для последовательных компьютеров [15] и подробно описана в [9].

нения. Смешанная стратегия стала частью подкласса конструирующих преобразований, который наряду с преобразованиями для последовательных программ, такими как, например, заменяющие рекурсию на цикл, включает так называемые *распараллеливающие* преобразования, связанные с построением параллельной программы по последовательной: векторизующие преобразования [3]; преобразования компактификации или упаковки [1, 20, 31] и т. п. Что касается класса оптимизирующих преобразований, то он расширился за счет оптимизаций параллельных программ и так называемых *реструктурирующих* преобразований, которые ориентированы на приведение преобразуемой программы к виду, более подходящему для выполнения конструирующих преобразований. Следует отметить, что последнее разделение не является абсолютным: одно и то же преобразование в одном трансляторе (в рамках одной системы преобразований) может быть оптимизирующим, а в другом — реструктурирующим.

Основная задача распараллеливающего компилятора — извлечь как можно больше скрытого параллеллизма из участков повторяемости последовательной программы, определяющих основное время ее выполнения: циклов и рекурсивных процедур. Это требует изменения порядка исполнения операторов в участках повторяемости последовательной программы, которые до выполнения алгоритмов распараллеливания, в случае отсутствия рекурсивных процедур, можно привести к виду гнезд DO-циклов (см., например, [5]). Поэтому гнездо DO-циклов берется в качестве модели программы для рассматриваемых в статье алгоритмов распараллеливания. Условия возможности проведения тех или иных преобразований выражаются в виде так называемых программных зависимостей. Зависимость в общем смысле есть отношение между двумя вычислениями, которое налагает ограничения на порядок их исполнения. Статический анализ зависимостей идентифицирует эти ограничения и дает информацию для проведения реструктурирующих преобразований во время трансляции.

Алгоритмы распараллеливания циклов — это весьма важный класс реструктурирующих преобразований. Они особенно привлекательны тем, что их применение не требует знания целевой архитектуры. Эти алгоритмы можно рассматривать как завершающую стадию машинно-независимой части процесса генерации параллельного кода распараллеливающим компилятором. Распараллеливание циклов позволяет обнаружить параллелизм (преобразуя циклы DO в циклы DOALL) и выявить те зависимости, которые ответственны за изначально «последовательное» состояние некоторых операций исходной программы.

Разумеется, следующая стадия генерации кода, связанная с выполнением конструирующих преобразований и преобразований параллельных программ, должна будет принимать во внимание параметры целевой архитектуры. Нахождение подходящего уровня зернистости является ключом к высокой производительности. Кроме того, необходимо рассматривать такие важные аспекты, как распределение данных и оптимизацию коммуникаций. Все эти проблемы весьма важны, но в данной работе не будут рассматриваться.

Данная работа посвящена изучению различных алгоритмов обнаружения параллелизма (в гнездах циклов), основанных на:

1) простой декомпозиции графа зависимостей на сильно связные компоненты (такие как алгоритм Аллена и Кеннеди [24]);

2) унимодулярных преобразованиях цикла — либо специальных преобразованиях (алгоритм Банержи [27]), либо автоматически генерируемых (алгоритм Вольфа и Лэма [60]);

3) планировании — либо одномерном [37, 39, 46] (особый случай — метод гиперплоскостей [55]), либо многомерном [42, 47].

Существующие алгоритмы распараллеливания циклов различаются по многим направлениям. Во-первых, они используют различные математические инструменты: графовые алгоритмы в случае (1), матричные вычисления в (2), линейное программирование в (3). Во-вторых, они принимают на входе различные описания зависимостей по данным: графовое описание и уровни зависимостей в (1), векторы направления в (2), описание зависимостей в виде многогранников или аффинных выражений в (3). Для каждого из этих алгоритмов мы определим основные понятия, лежащие в их основе, и обсудим их сильные и слабые стороны как на примерах, так и в сравнении «оптимальных» результатов.

Нас будет интересовать характеристика того, какой алгоритм наилучшим образом подходит для того или иного заданного представления зависимостей. Понятно, что нет необходимости использовать усложненный алгоритм анализа зависимостей, если алгоритм распараллеливания не сможет воспользоваться точностью его результата. И наоборот, нет смысла в использовании точнейшего алгоритма распараллеливания, если представление зависимостей не будет достаточно точным.

Оставшаяся часть работы организована следующим образом. Разд. 2 посвящен краткому обзору предмета рассмотрения, а именно — алгоритмов распараллеливания циклов. В разд. 3 мы рассмотрим основные абстракции зависимостей: уровни зависимостей, векторы направлений, многогранники зависимостей. Алгоритм Аллена—Кеннеди [24] приведен в разд. 4, алго-

ритм Вольфа—Лэма [60] — в разд. 5. Показано, что оба алгоритма являются «оптимальными» в классе алгоритмов распараллеливания, использующих ту же самую абстрактную конструкцию в качестве входного представления, а именно — уровни зависимостей у Аллена и Кеннеди и векторы направлений у Вольфа и Лэма. В разделе 6 мы представим алгоритм Дартье—Вивьена [36], охватывающий оба предыдущих. Он основан на обобщении векторов направления — многогранниках зависимостей. В разд. 7 мы приведем краткий обзор алгоритма Фотрие [46, 47], основанного на точных аффинных зависимостях. Наконец, в разд. 8 будут высказаны некоторые заключения.

2. ВХОД И ВЫХОД РАСПАРАЛЛЕЛИВАЮЩИХ АЛГОРИТМОВ

Вложенные DO-циклы в состоянии описывать объем вычислений, размер которого значительно превосходит размер соответствующей программы. Например, рассмотрим n вложенных циклов, счетчики которых описывают n-мерный куб с ребром N: эти циклы вмещают в себя объем вычислений размером N^n. Часто случается, что такие гнезда циклов содержат ненулевую *степень параллелизма*, т. е. множество независимых вычислений размером $\Omega(N^r)$ для $r \le 1$.

Это делает задачу распараллеливания вложенных циклов очень трудной и интересной: распараллеливающий компилятор должен уметь определять по возможности ненулевую степень параллелизма за время, *непропорциональное* по сравнению с последовательным исполнением цикла. Чтобы такое стало возможным, эффективные алгоритмы распараллеливания должны иметь *сложность*, *входной размер* и *выходной размер*, зависящие только от n, но ни в коем случае не от N, т. е. зависящие от размера последовательного кода, а не от количества описываемых им вычислений.

Входом распараллеливающего алгоритма является описание зависимостей, которые связывают отдельные вычисления. Выходом является описание эквивалентного кода с явным параллелизмом.

2.1. Вход: граф зависимостей

Каждый оператор гнезда циклов окружен несколькими циклами. Каждая итерация этих циклов определяет конкретное исполнение оператора, называемое *операцией*. Зависимости между операциями представлены ориенти-

рованным ациклическим графом[2] — *расширенным графом зависимостей* (РГЗ). Количество вершин РГЗ равно количеству операций в гнезде циклов. Исполнение операций гнезда циклов в соответствии с определяемым РГЗ частичным порядком гарантирует сохранение корректности результата вычисления. Распознавание параллелизма в гнезде циклов сводится к нахождению цепей антизависимостей в РГЗ. Понятие «расширенного графа зависимостей» иллюстрирует пример 1. Соответствующий РГЗ изображен на рис. 1.

Пример 1.

```
DO i=1,n
    DO j=1,n
        a(i,j)=a(i-1,j-1)+a(i,j-1)
    ENDDO
ENDDO
```

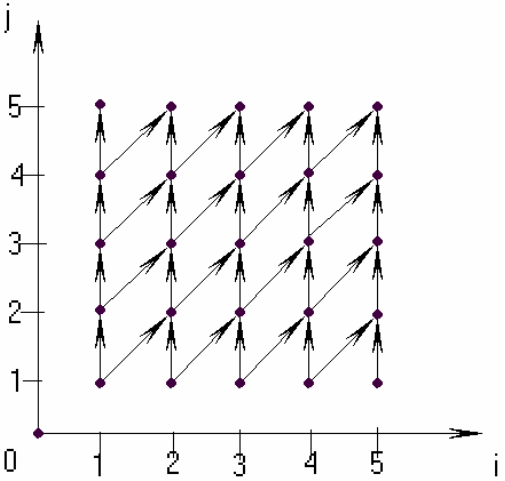


Рис. 1. РГЗ фрагмента примера 1

[2] Здесь и далее мы используем стандартную терминологию теории графов (см., например, [13]).

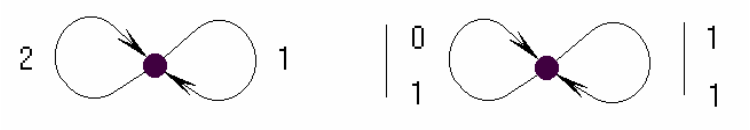
К сожалению, РГЗ не может быть использован в качестве входного представления для распараллеливающих алгоритмов, так как обычно он слишком велик и не может быть точно описан во время компиляции. Вместо него используется так называемый *сокращенный граф зависимостей* (*СГЗ*) — сжатое и приближенное представление РГЗ. Это приближение должно быть надмножеством РГЗ, чтобы сохранить отношения зависимости. В СГЗ имеется одна вершина для каждого оператора гнезда циклов, его дуги помечены в соответствии с выбранным приближением зависимостей (см. более подробное изложение в разделе 3). На рис. 2 представлены два возможных СГЗ для программы из примера 1, соответствующих двум различным приближениям зависимостей.

Поскольку входом является СГЗ, а не РГЗ, распараллеливающий алгоритм не способен отличить два различных РГЗ, имеющих один СГЗ. Следовательно, удастся распознать только параллелизм, содержащийся в СГЗ. Таким образом, качество распараллеливающего алгоритма должно оцениваться относительно анализа зависимостей.

Например, программы из примеров 1 и 2 имеют один и тот же СГЗ с уровнями зависимостей (рис. 2(а)). Таким образом, распараллеливающий алгоритм, принимающий на входе СГЗ с уровнями зависимостей, не в состоянии различить две программы. Однако программа из примера 1 содержит один уровень параллелизма, тогда как программа из примера 2 является существенно последовательной.

Пример 2.

```
DO i = 1, n
    DO j = 1, n
        a(i, j) = 1 + a(i-1, n) + a(i, j-1)
    ENDDO
ENDDO
```



а *б*

Рис. 2. Сокращенный граф зависимостей: (а) с уровнями зависимостей; (б) с векторами направлений

2.2. Выход: вложенные циклы

Размер распараллеленного кода, как упоминалось ранее, не должен зависеть от количества описываемых им операций. В этом причина того, что результат работы распараллеливающего алгоритма всегда должен описываться множеством циклов[3].

Имеются по крайней мере три способа определить новый порядок на операциях заданного гнезда циклов (то есть три способа определить выход распараллеливающего алгоритма) в терминах вложенных циклов.

1. Использование элементарных преобразований циклов в качестве базовых шагов алгоритма, таких как распределение цикла (в алгоритме Аллена и Кеннеди), перестановка или перекос циклов (как в алгоритме Банержи).
2. Линейное изменение базы области итераций, то есть применение унимодулярного преобразования на векторах итераций (как в алгоритме Вольфа и Лэма).
3. Определение d-мерного плана, т. е. применение аффинного преобразования из Z^n в Z^d и интерпретация преобразования как многомерной временной функции. Каждая компонента соответствует последовательному циклу, а недостающие (n – d) размерностей соответствуют циклам DOALL (как в алгоритмах Фотрие и Дарте—Вивьен).

Результат работы этих трех схем преобразований, разумеется, может быть описан в терминах гнезд циклов после более или менее сложного процесса переписывания (см. [32, 35, 39, 60, 65]). Здесь мы не будем говорить о переписывании. Вместо этого мы сосредоточимся на связи между представлением зависимостей (входом) и преобразованиями циклов, используемыми распараллеливающим алгоритмом (выходом). Нашей целью является формулировка того, какой алгоритм окажется оптимальным для заданного представления зависимостей. «Оптимальный» означает здесь, что алгоритм в состоянии выявить максимальное количество параллельных циклов.

[3] Эти циклы могут быть произвольно сложными, при условии, что их сложность зависит только от размера исходного кода. Очевидно, что чем проще результат, тем лучше. Но в этом контексте значение слова «проще» не совсем тривиально, поскольку все зависит от оптимизаций, которые могут последовать далее. Обычно полагается, что структурная простота предпочтительна.

3. АБСТРАКЦИИ ЗАВИСИМОСТЕЙ

Для ясности мы ограничим изложение случаем совершенного гнезда DO-циклов с аффинными границами. Это ограничение позволяет описать итерации n вложенных циклов (n называется *глубиной* гнезда циклов) с векторами из Z^n (называемыми *векторами итераций*), содержащимися в конечном выпуклом многограннике (называемом *областью итераций*), ограниченном границами циклов. i-я компонента вектора итерации является значением счетчика i-го цикла в гнезде, считая от самого внешнего к самому внутреннему. Таким образом, в последовательной программе итерации исполняются в лексикографическом порядке их векторов итераций.

Введем следующие обозначения:

D — область итераций в виде многогранника; **I** и **J** — n-мерные векторы итераций в **D**; S_i — i-й оператор в гнезде циклов, где $1 \le i \le s$. Мы будем писать $\mathbf{I} >_l \mathbf{J}$, если **I** лексикографически больше **J**, и $\mathbf{I} \ge_l \mathbf{J}$, если $\mathbf{I} >_l \mathbf{J}$ или $\mathbf{I} = \mathbf{J}$.

В разд. 3.1 описываются различные концепции графов зависимостей, неформально введенные в разд. 2.1: расширенные графы зависимостей (РГЗ), сокращенные графы зависимостей (СГЗ), графы истинных зависимостей (ГИЗ), а также понятие множеств расстояний. В разд. 3.2 мы формально определим понятие многогранного сокращенного графа зависимостей (МСГЗ), то есть сокращенного графа зависимостей, дуги которого помечены многогранниками. Наконец в разд. 3.3 мы покажем, как модель МСГЗ обобщает классические абстракции множеств расстояний, такие как уровни зависимостей и векторы направления.

3.1. Графы зависимостей и множества расстояний

Отношения зависимостей между операциями определяются условиями Бернштейна [30]. Вкратце, две операции связаны зависимостью, если обе они обращаются к одной и той же области памяти и по меньшей мере одно из обращений производит запись в эту область. Зависимость имеет направление в соответствии с последовательным порядком, от первой исполненной операции к последней. В зависимости от порядка операций записи и/или чтения, зависимость называется *потоковой зависимостью, антизависимостью* или *выходной зависимостью*. Мы будем писать $S_i(I) \Rightarrow S_j(I)$, если оператор S_j на итерации J зависит от оператора S_i на итерации I. Частичный порядок, определяемый отношением $\Rightarrow$, описывает расширенный граф за-

висимостей (РГЗ). Заметим, что (J - I) всегда является лексикографически ненулевым, когда $S_i(I) \Rightarrow S_j(J)$.

В общем случае РГЗ не может быть вычислен во время компиляции, либо в силу недостатка информации (такой как значения параметров границ цикла или даже точное число обращений к памяти), либо из-за того, что генерация полного графа обходится слишком дорого (см. в [61, 66] обзор алгоритмов проверки зависимостей, таких как проверка наибольших общих делителей [24, 28], тест Банержи [27–29], ω-тест [58], λ-тест [56], I-тест [48]; в [45] — более детальное изложение точного анализа зависимостей). Вместо этого рассматривается представление зависимостей в виде меньшего по размеру ориентированного графа с циклами, имеющего s вершин (по количеству операторов), который называется *сокращенным графом зависимостей* (*СГЗ*) (или *графом зависимостей на уровне операторов*).

СГЗ представляет собой сжатие РГЗ. В СГЗ два оператора S_i и S_j являются зависимыми (мы будем обозначать это как e: $S_i \to S_j$), если существует по меньшей мере одна пара (I, J) такая, что $S_i(I) \Rightarrow S_j(J)$. Дуга[4] e от S_i к S_j в СГЗ помечена множеством $\{(I, J) \in D^2 \mid S_i(I) \Rightarrow S_j(J)\}$ или аппроксимацией D_e, содержащей это множество. Точность этой аппроксимации и ее представление и составляют силу алгоритма анализа зависимостей.

Другими словами, СГЗ описывает в сокращенной форме граф зависимостей на уровне итераций, называемый (*максимальным*) *графом истинных зависимостей* (*ГИЗ*), представляющий собой подмножество РГЗ. ГИЗ и РГЗ имеют одни и те же вершины, но ГИЗ имеет большее количество дуг, определяемых следующим образом:

$$(S_i,I) \Rightarrow (S_j,J) \text{ (в ГИЗ)} \Leftrightarrow$$
$$\exists\, e = (S_i, S_j) \text{ (в РГЗ), такая, что } (I, J) \in D_e.$$

Для определенного класса вложенных циклов существует возможность выразить в точности это множество пар (I, J) (см. [45]): I задается как аффинная функция (в некоторых конкретных случаях, включающих функции нахождения ближайшего целого снизу или сверху) $f_{i,j}$ от J, где J изменяется в пределах многогранника $P_{i,j}$:

$$\{(I, J) \in D^2 \mid S_i(I) \Rightarrow S_j(J)\} = \{(f_{i,j}(J),J) \mid J \in P_{i,j} \subset D\}. \qquad (1)$$

[4] Такая дуга существует для каждой пары обращений к памяти, вызывающих зависимость между S_i и S_j.

Однако большинство алгоритмов анализа зависимостей вместо множества пар (I, J) вычисляет множество $E_{i,j}$ всех возможных значений (J - I). $E_{i,j}$ называется множеством *векторов расстояния*, или *множеством расстояний*:

$$E_{i,j} = \{(J - I) \mid S_i(I) \Rightarrow S_j(J)\}.$$

В случаях, когда возможно провести точный анализ зависимостей, равенство 1 показывает, что множество векторов расстояния является проекцией целочисленных точек многогранника. Это множество может быть аппроксимировано его выпуклой оболочкой или более или менее точным многогранником большего размера (или конечным объединением многогранников). Когда множество векторов расстояния представлено конечным объединением многогранников, соответствующая дуга зависимостей в СГЗ разлагается на кратные дуги.

Заметим, что представление с помощью векторов расстояния не эквивалентно представлению с помощью пар (как в равенстве 1), поскольку теряется информация о *расположении* в РГЗ этого вектора. Это может вызвать даже некоторое уменьшение количества параллелизма, как будет показано в примере 9. Однако такое представление все же остается полезным, особенно в случаях, когда точный анализ зависимостей слишком накладен или вообще неосуществим.

Классическими представлениями множеств расстояний (в порядке уменьшения точности) являются:

- *уровни зависимостей*, введенные и использованные в распараллеливающем алгоритме Аллена и Кеннеди [24, 25];
- *векторы направления*, введенные Лампортом [55] и Вольфом [62, 63], а затем использованные в распараллеливающем алгоритме Вольфа и Лэма [60];
- *многогранники зависимостей*, введенные в работе [50] и использованные в алгоритме с разделением супервершин Иригойна и Триоле [51]. Более подробное описание многогранников зависимостей можно найти в работах по проекту PIPS [49].

Теперь мы дадим формальное определение сокращенных графов зависимостей, дуги которых помечены многогранниками зависимостей. Затем мы покажем, что это представление объединяет два других представления, а именно — уровни зависимостей и векторы направления.

3.2. Многогранные сокращенные графы зависимостей

Вспомним математическое определение многогранника и то, как можно провести его декомпозицию на вершины, лучи и линии.

Множество P векторов в пространстве Q^n называется (*выпуклым*) *многогранником,* если существуют целочисленная матрица A и целочисленный вектор b такие, что

$$P = \{x \mid x \in Q^n, Ax \le b\}.$$

Политопом называется ограниченный многогранник.

Справедливы следующие свойства. Любой многогранник может быть разложен на сумму политопа и многогранного конуса. Политоп определяется его вершинами, и каждая точка политопа является неотрицательной барицентрической комбинацией вершин политопа. Многогранный конус может быть определен с помощью его лучей и линий. Любая точка многогранного конуса есть сумма неотрицательной комбинации его лучей и произвольной комбинации его линий.

Таким образом, многогранник зависимостей P может быть эквивалентно определен с помощью множества *вершин* (обозначаемых $\{v_1, ..., v_\omega\}$), множества *лучей* (обозначаемых $\{r_1, ..., r_\rho\}$) и множества *линий* (обозначаемых $\{l_1, ..., l_\lambda\}$). Тогда P есть множество всех векторов p таких, что

$$p = \sum_{i=1}^{w} \mu_i \upsilon_i \quad + \quad \sum_{i=1}^{\rho} \upsilon_i r_i \quad + \quad \sum_{i=1}^{\lambda} \xi_i l_i \ , \tag{2}$$

где $\mu_i \in Q^+$, $\nu_i \in Q^+$, $\xi_i \in Q$ и $\sum_{i=1}^{w} \mu_i = 1$.

Теперь мы определим то, что называется многогранным сокращенным графом зависимостей (или МСГЗ), а именно — сокращенный граф зависимостей, помеченный многогранником зависимостей. На самом деле нам нужны только целочисленные векторы, принадлежащие к многограннику зависимостей, поскольку расстояния зависимостей в сущности являются целочисленными векторами.

Многогранный сокращенный граф зависимостей (МСГЗ) есть СГЗ, в котором каждая дуга e: $S_i \to S_j$ помечена многогранником P(e), аппроксимирующим множество векторов расстояний: ассоциированный истинный граф зависимостей содержит дугу из экземпляра I вершины S_i к экземпляру J вершины S_j в том и только том случае, когда $(J - I) \in P(e)$.

В разд. 6 это представление зависимостей рассматривается более подробно; пока будем воспринимать многогранник зависимостей как некоторое обобщение векторов направления.

3.3. Векторы направлений

Когда множество векторов расстояния содержит только один элемент, зависимость называется униформной, и единственный вектор расстояния называется *униформным вектором зависимости.*

В противном случае множество векторов расстояния может быть представлено n-мерным вектором (*вектором направления*), компоненты которого принадлежат к $Z \cup \{*\} \cup (Z \times \{+, -\})$. Его i-я компонента представляет собой аппроксимацию i-х компонент всех возможных векторов расстояния: она равна z+ (соответственно z-), если i-я компонента больше (соответственно меньше) или равна z. Она равна *, если i-я компонента может принимать любое значение, и z, если зависимость является униформной по этой размерности с единственным значением z. Обычно + (соответственно -) используется как сокращение для 1+ (соответственно для (-1)-).

Мы обозначаем как e_i i-й канонический вектор, то есть n-мерный вектор, все компоненты которого являются нулевыми, за исключением i-й компоненты, равной 1. Тогда вектор направления — не что иное, как аппроксимация многогранником, имеющим одну вершину, все лучи и линии которого, если они имеются, суть канонические векторы.

Действительно, рассмотрим дугу e, помеченную вектором направления d, и обозначим через I^+, I^- и I^* множества компонент d, которые равны z+ (для некоторого целого z), z- и *, соответственно. Наконец, обозначим через d_z n-мерный вектор, чья i-я компонента равна z, если i-я компонента d равна z, z+ или z-, и равна 0 в противном случае.

Теперь, в силу определения символов +, - и *, вектор направления d представляет в точности все n-мерные векторы p, для которых существуют такие целые (v, v', ξ) в $N^{|I+|} \times N^{|I-|} \times Z^{|I*|}$, что

$$p = d_z + \sum_{i \in I^+} v_i e_i - \sum_{i \in I^-} v_i' e_i + \sum_{i \in I^*} \xi_i e_i \ . \qquad (3)$$

Другими словами, вектор направления d представляет все целые точки, которые принадлежат многограннику, определяемому единственной вершиной d_z, лучами e_i для $i \in I^+$, лучами $-e_i$ для $i \in I^-$ и линиями e_i для $i \in I^*$.

Например: вектор направления (2+, *, -, 3) определяет многогранник с одной вершиной (2, 0, -1, 3), двумя лучами (1, 0, 0, 0) и (0, 0, -1, 0) и одной линией (0, 1, 0, 0).

3.4. Уровни зависимостей

Представление с помощью уровня является менее точной абстракцией зависимостей. В гнезде из n вложенных циклов множество векторов расстояния аппроксимируется целым числом l из интервала $[1, n] \cup \{\infty\}$, которое определяется как наибольшее целое, такое, что первые $l - 1$ компонент векторов расстояний оказываются нулевыми.

Зависимость на уровне $l \le n$ означает, что зависимость обнаруживается на уровне l гнезда циклов, т. е. на заданной итерации $l - 1$ внешних циклов. В этом случае говорят, что зависимость является *циклически порождаемой зависимостью* на уровне l. Если $l = \infty$, то зависимость происходит внутри тела цикла, между двумя различными операторами, и называется *циклически независимой зависимостью*. Сокращенный граф зависимостей, дуги которого помечены уровнями зависимостей, называется *сокращенным уровневым графом зависимостей* (*СУГЗ*).

Рассмотрим дугу e уровня l. Согласно определению уровня, первой ненулевой компонентой вектора расстояния является l-я компонента, и теоретически она может принимать любое положительное целое значение. Об оставшихся компонентах у нас нет никакой информации. Следовательно, дуга уровня $l < \infty$ эквивалентна вектору направления (0, … , 0, 1+, ∗, … ,∗), начинающемуся с (l-1) нулевой компоненты, а дуга уровня ∞ соответствует нулевому вектору зависимости. Подобно тому как любой вектор направления допускает построение эквивалентного многогранника, то же можно проделать и с представлением с помощью уровня. Например, зависимость уровня 2 в трехмерном гнезде циклов дает вектор направления (0, 1+, ∗), который соответствует многограннику с одной вершиной (0, 1, 0), одним лучом (0, 1, 0) и одной линией (0, 0, 1).

4. АЛГОРИТМ АЛЛЕНА—КЕННЕДИ

Алгоритм Аллена—Кеннеди [24] был первоначально разработан для векторизации циклов. Затем он был расширен с тем, чтобы максимизировать количество параллельных циклов и минимизировать количество синхронизаций в преобразованном коде. Алгоритм принимает на входе сокращенный уровневый граф зависимостей.

Алгоритм базируется на следующих фактах.

1. Цикл является параллельным, если он не содержит циклически порождаемых зависимостей, т.е. если не существует зависимости, уро-

вень которой равен глубине цикла, которая относится к оператору, окруженному циклом.

2. Все итерации оператора S_1 могут быть осуществлены перед любой итерацией оператора S_2, если в СУГЗ не существует зависимости из S_2 в S_1.

Свойство (1) позволяет пометить цикл как DOALL или DOSEQ, тогда как свойство (2) указывает на то, что нахождение параллелизма может быть проведено независимо в каждой сильно связной компоненте СУГЗ. Извлечение параллелизма производится посредством разделения циклов.

4.1. Алгоритм

Для графа зависимостей G через G(k) обозначается такой его подграф, который получается из G удалением всех зависимостей на уровнях, строго меньших k. Ниже приведен набросок алгоритма Аллена—Кеннеди в его базовой формулировке. Работа алгоритма начинается с вызова ALLEN-KENNEDY(СУГЗ, 1).

```
проц ALLEN-KENNEDY(G: граф, k: целое)=
    если k≤n то
        Произвести декомпозицию G(k) на сильно связные компоненты
      Gi и топологически отсортировать их;
        Переписать код таким образом, чтобы каждая Gi принадлежала
      своему гнезду циклов (на уровне k) и сохранялся порядок на Gi
      (распределение циклов на уровне больше или равном k);
        для всех Gi цикл
                если Gi не имеет дуг на уровне k то
                        пометить цикл на уровне k как цикл DOALL
                иначе пометить его как цикл DOSEQ
                все
        все;
        для всех Gi цикл ALLEN-KENNEDY(Gi, k+1) все
    все
все.
```

Проиллюстрируем работу алгоритма на следующем фрагменте кода:

Пример 3.

```
DO i = 1, n
DO j = 1, n
      DO k = 1, n
      S1: a(i, j, k) = a(i-1, j+i, k) + a(i, j, k-1) + b(i, j-1, k)
      S2: b(i, j, k) = b(i, j-1, k+j) + a(i-1, j, k)
  ENDDO
ENDDO
ENDDO
```

Граф зависимостей G = G(1) для данного гнезда циклов приведен на рис. 3. Он имеет только одну сильно связную компоненту и по меньшей мере одну дугу на уровне 1, в силу чего первый вызов процедуры обнаружит, что самый внешний цикл является последовательным. Тем не менее, на уровне 2 (дуга на уровне 1 более не рассматривается) G(2) имеет две сильно связных компоненты: все итерации оператора S_2 могут быть вынесены перед итерациями оператора S_1.

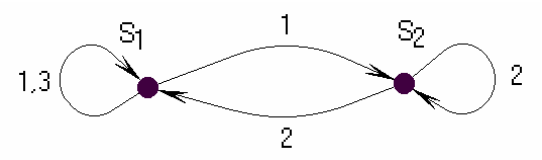


Рис. 3. СУГЗ для программы примера 3

Проведем разделение цикла. Сильно связная компонента, включающая S_1, не содержит дуг на уровне 2 и содержит одну дугу на уровне 3. Таким образом, второй цикл, окружающий S_1, помечается как DOSEQ, а третий — как DOALL. Сильно связная компонента, включающая S_2, содержит одну дугу на уровне 2 и не содержит дуг на уровне 3. Следовательно, второй цикл, окружающий S_2, помечается как DOALL, а третий — как DOSEQ. В итоге мы получаем:

```
DOSEQ i = 1, n
DOSEQ j = 1, n
      DOALL k = 1, n
      S2: b(i, j, k) = b(i, j-1, k+j) + a(i-1, j, k)
  ENDDO
ENDDO
DOALL j = 1, n
  DOSEQ k = 1, n
      S1: a(i, j, k) = a(i-1, j+i, k) + a(i, j, k-1) + b(i, j-1, k)
  ENDDO
ENDDO
ENDDO
```

4.2. Сильные и слабые стороны алгоритма

В работе [32] было показано, что для каждого оператора исходного кода алгоритм Аллена—Кеннеди обнаруживает столько объемлющих параллельных циклов, сколько вообще возможно. Более точно, рассмотрим оператор S исходного кода, и пусть L_i — один из объемлющих циклов. Тогда L_i будет помечен как параллельный только в том случае, когда не существует зависимости на уровне i между двумя экземплярами S. Однако этот результат показывает только то, что алгоритм Аллена—Кеннеди является оптимальным среди тех распараллеливающих алгоритмов, которые рассматривают в преобразованном коде экземпляры S в точно тех же циклах, что и в исходном коде. В действительности можно доказать намного более сильное следующее утверждение [42].

Теорема 1. *Алгоритм Аллена—Кеннеди является оптимальным среди всех алгоритмов нахождения параллелизма, которые принимают СУГЗ в качестве входа.*

В работе [43] доказано, что для любого гнезда циклов N_1 существует гнездо циклов N_2, имеющее тот же СУГЗ, такое, что для любого оператора S из N_1, окруженного после распараллеливания d_S последовательными циклами, в точном графе зависимостей N_2 существует путь зависимости, который включает $\Omega(N^{d_S})$ экземпляров оператора *S*. Другими словами, алгоритм Аллена—Кеннеди не различает гнезд N_1 и N_2, которые имеют один и тот же СУГЗ, и распараллеливающий алгоритм является оптимальным в сильном смысле на гнезде N_2, так как на каждом операторе он достигает верхней

границы параллелизма, определенного самыми длинными путями зависимостей в расширенном графе зависимостей.

Таким образом, доказано, что в случаях, когда вся доступная информация — это та, которая содержится в СУГЗ, не представляется возможным обнаружить больше параллелизма, чем находится алгоритмом Аллена—Кеннеди. Другими словами, алгоритм Аллена—Кеннеди хорошо адаптирован к представлению зависимостей в виде уровней. Следовательно, чтобы обнаружить большее количество параллелизма, чем этот алгоритм, требуется иметь больше информации о зависимостях. Классическими примерами того, как можно превзойти алгоритм Аллена—Кеннеди, являются гнездо циклов примера 4, в котором параллелизм выявляется простой перестановкой циклов (рис. 4), и гнездо циклов примера 5, в котором для выявления параллелизма достаточно осуществить преобразования перекоса и перестановки (рис. 5).

Пример 4.

```
DO i=1,n
    DO j=1,n
        a(i,j)=a(i-1,j-1)+a(i,j-1)
    ENDDO
ENDDO
```

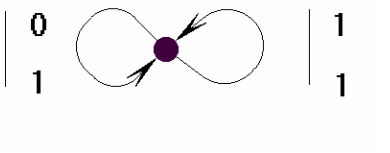


Рис. 4. СГЗ программы примера 4

Пример 5.

```
DO i=1,n
    DO j=1,n
        a(i,j)=a(i-1,j)+a(i,j-1)
    ENDDO
ENDDO
```

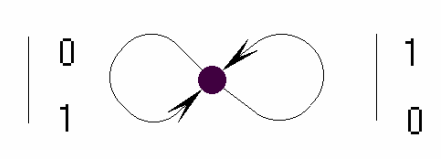


Рис. 5. СГЗ программа примера 5

5. АЛГОРИТМ ВОЛЬФА—ЛЭМА

Фрагменты программ примеров 4 и 5 содержат параллелизм, который не может быть обнаружен с помощью алгоритма Аллена—Кеннеди. Следовательно, как показывает теорема 1, этот параллелизм не может быть извлечен, если зависимости представлены в виде уровней. Чтобы обойти это ограничение, Вольф и Лэм [60] предложили алгоритм, использующий в качестве входа векторы направления. Их работа объединяет все предыдущие алгоритмы, основанные на операциях с элементарными матрицами, такие как перекос цикла, перестановка циклов, обращение цикла, в единую структуру: структуру допустимых *унимодулярных преобразований.*

5.1. Цель

Цель, которую преследовали Вольф и Лэм, состояла в построении множеств полностью переставляемых гнезд циклов. Полностью переставляемые гнезда являются основой для всех техник мозаичного размещения [51, 59, 60]. Мозаичное размещение используется для выявления среднезернистого и крупнозернистого параллелизма. Более того, множество d полностью переставляемых циклов может быть переписано в виде одного последовательного цикла и d-1 параллельных циклов. Следовательно, этот метод также может быть использован для выявления мелкозернистого параллелизма.

Алгоритм Вольфа—Лэма строит максимальное множество самых внешних полностью переставляемых[5] циклов. Затем он рекурсивно просматривает оставшиеся размерности и зависимости, не задействованные этими цик-

[5] i-й и i+1-й циклы являются переставляемыми в том и только том случае, когда i-я и i+1-я компоненты любого вектора расстояния глубины ≥ i неотрицательны.

лами. Версия, представленная в [60], строит множество циклов с помощью анализа более простых случаев, полагаясь на эвристики для гнезд циклов с глубиной 6 и более. В оставшейся части данного раздела мы рассмотрим этот алгоритм с теоретической точки зрения и дадим его наиболее общий вариант.

5.2. Теоретическое обоснование

Унимодулярные преобразования обладают двумя важными свойствами: линейностью и обратимостью. Для данного унимодулярного преобразования T свойство линейности позволяет с легкостью определить, является ли T допустимым. Действительно, T является допустимым в том и только том случае, когда $Td >_l 0$ для всех ненулевых векторов расстояния d. Обратимость позволяет легко переписывать код, так как преобразование представляет собой простое изменение базиса в Z^n.

В общем случае справедливость выражения $Td >_l 0$ не может быть проверена для всех векторов *расстояния*, число которых может быть слишком большим. Следовательно, нужно попытаться гарантировать, что $Td >_l 0$ для всех ненулевых векторов *направления*, с обычными математическими соглашениями, в $Z \cup \{*\} \cup (Z \times \{+, -\})$. В дальнейшем мы ограничимся рассмотрением только ненулевых векторов направления, известных как *лексикографически положительные* векторы направлений.

Обозначим через t(1), ..., t(n) строки T. Пусть Γ — замыкание конуса, порождаемого всеми векторами направления. Для вектора направления d:

$$Td >_l 0 \Leftrightarrow \exists k_d,\ 1 \le k_d \le n \mid\ \forall i,\ 1 \le i \le k_d,\ t(i).d = 0 \text{ и } t(k_d).d > 0.$$

Это означает, что зависимости, представленные вектором d, порождаются в цикле уровня k_d. Если $k_d = 1$ для всех векторов расстояния d, тогда все зависимости порождаются в первом цикле, а все внутренние циклы являются циклами DOALL. Тогда t(1) называется *вектором синхронизации* или *разделяющей гиперплоскостью*. Такой вектор синхронизации существует только в том случае, когда Γ является направленным, т. е. тогда и только тогда, когда Γ не содержит линейного пространства. Это также эквивалентно тому факту, что конус

$$\Gamma^+ = \{y \mid \forall x \in \Gamma,\ y.x \le 0\}$$

является полномерным. Построение T по n линейно независимым векторам из Γ^+ позволяет преобразовывать циклы в n полностью переставляемых циклов.

Понятие вектора синхронизации лежит в основе метода гиперплоскостей и его вариаций (см., например, [37, 55]), особенно полезных при выявлении мелкозернистого параллелизма, тогда как понятие полностью переставляемых циклов является базисом всех техник мозаичного размещения. Как было сказано ранее, обе формулировки эквивалентны для случая Γ^+.

Когда конус Γ не является направленным, Γ^+ имеет размерность r, $1 \le r \le n$, $r = n - s$, где s — размерность пространства линейности Γ. При наличии r линейно независимых векторов в Γ^+, можно преобразовать гнездо циклов таким образом, что r самых внешних циклов будут полностью переставляемыми. Затем можно рекурсивно применять ту же технику для преобразования n-r внутренних циклов, рассматривая векторы направления, не задействованные еще ни одним из r внешних циклов, т. е. рассматривая векторы направления, входящие в пространство линейности Γ. Это основная идея алгоритма Вольфа—Лэма [60], представленного ниже.

5.3. Обобщенный алгоритм

Алгоритм Вольфа—Лэма воспринимает на входе множество векторов направления D и последовательность линейно независимых векторов E (первоначально пустую), из которых он строит матрицу преобразования с помощью следующей процедуры:

проц WOLF-LAM (D, E : **множество векторов**)=
Определить Γ как замыкание конуса, генерируемого векторами направления из D;
Определить $\Gamma^+ = \{y \mid \forall x \in \Gamma, y.x \le 0\}$, и пусть r — размерность Γ^+;
Дополнить E до множества E' r линейно независимых векторов из Γ^+ (по построению $E \subset \Gamma^+$);
Пусть D' — подмножество D, определяемое по правилу: $d \in D \Leftrightarrow \forall v \in E'$, $v.d = 0$ (то есть $D' = D \cap E'^{\perp} = D \cap$ lin.space(Γ));
Вызвать WOLF-LAM(D', E')
все.

Поскольку процесс, описываемый данной процедурой, может дать в итоге неунимодулярную матрицу, построение желаемой унимодулярной

матрицы T может быть проведено выполнением следующей последовательности шагов.

1. Взяв в качестве D множество векторов направления, а в качестве E пустое множество ∅, вызвать WOLF-LAM(D, E).
2. Построить невырожденную матрицу T_1, первые строки которой являются векторами из построенного множества E (в том же порядке). Пусть $T_2 = pT_1^{-1}$, где p выбирается таким образом, что T_2 будет целочисленной матрицей.
3. Вычислить левую эрмитову форму T2, T2 = QH, где H — неотрицательная нижняя треугольная матрица, а Q — унимодулярная матрица.
4. Q^{-1} есть искомая матрица преобразования (поскольку $pQ^{-1}D = HT_1D$).

Проиллюстрируем работу описанного алгоритма на следующем гнезде циклов.

Пример 6.

```
DO i = 1, n
DO j = 1, n
    DO k = 1, n
        a(i, j, k) = a(i-1, j+i, k) + a(i, j, k-1) + a(i, j-1, k+1)
    ENDDO
ENDDO
ENDDO
```

Множество векторов расстояния для данного гнезда равно D = {(1, –, 0), (0, 0, 1), (0, 1, -1)} (см. рис. 6).

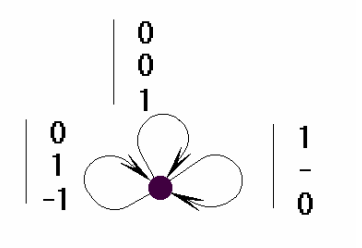


Рис. 6. СГЗ программы примера 6

Нетрудно видеть, что для примера 6 пространство линейности Γ(D) является двумерным (оно порождается векторами (0, 1, 0) и (0, 0, 1)). Следовательно, $\Gamma^+(D)$ является одномерным и порождается $E_1 = \{(1, 0, 0)\}$. Тогда $D' = \{(0, 0, 1), (0, 1, -1)\}$, и Γ(D') является направленным. E_1 пополняется двумя векторами из $\Gamma^+(D')$, например, $E_2 = \{(0, 1, 0\}, (0, 1, 1)\}$. В данном конкретном случае матрица преобразования, имеющая строки E_1 и E_2, уже является унимодулярной и соответствует простому перекосу цикла. Для выявления DOALL-циклов мы выбираем первый вектор E_2 в относительно внутренней части Γ^+, например, $E_2 = \{(0,2,1), (0,1,0)\}$. В терминах преобразований циклов это сводится к перекосу цикла k со множителем 2 и последующей перестановке циклов j и k:

```
DOSEQ i = 1, n
DOSEQ k = 3, 3×n
    DOALL j = max(1,⌈(k-n)/2⌉), min (n,⌊(k-1)/2⌋)
        a(i, j, k-2×j)=a(i-1, j+i, k-2×j)+a(i, j, k-2×j-1)+a(i, j-1, k-2×j+1)
    ENDDO
ENDDO
ENDDO
```

5.4. Сильные и слабые стороны алгоритма

Вольф и Лэм показали оптимальность этой методологии (теорема B.6 из [60]): «алгоритм, который обнаруживает максимальный крупнозернистый параллелизм, а затем рекурсивно вызывает себя на внутренних циклах, производит параллелизм максимально возможной степени». Однако и в этом случае оптимальность следует понимать в контексте используемого анализа: а именно — наличие векторов расстояния при полном отсутствии какой-

либо информации о структуре графа зависимостей. Поэтому корректной является следующая формулировка данного утверждения [36].

Теорема 2. *Алгоритм Вольфа—Лэма является оптимальным среди всех алгоритмов нахождения параллелизма, принимающих на входе множество векторов расстояния (неявно предполагается, что гнездо циклов имеет только один оператор или все операторы образуют атомарный блок).*

Таким образом, как и для алгоритма Аллена—Кеннеди, частичная оптимальность алгоритма Вольфа—Лэма в общем случае имеет причиной не методологию алгоритма, но слабость его входного представления: то, что структура СГЗ не принимается во внимание, может повлечь потерю параллелизма. Например, в противоположность алгоритму Аллена—Кеннеди, алгоритм Вольфа—Лэма не находит параллелизма в примере 3 (СГЗ которого приведен на рис. 7) из-за типичной структуры векторов направления (1, –, 0), (0, 1, –), (0, 0, 1).

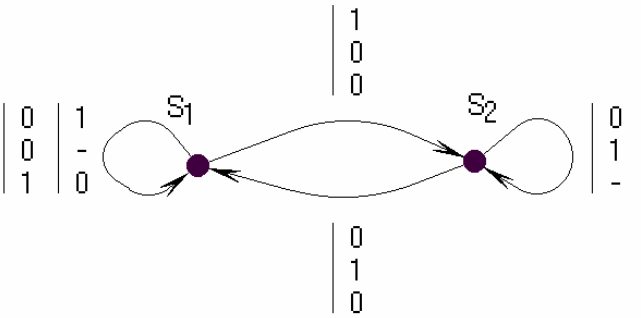


Рис. 7. СГЗ с векторами направлений для программы из примера 3

6. АЛГОРИТМ ДАРТЕ—ВИВЬЕНА

В этом разделе рассматривается третий распараллеливающий алгоритм, который принимает на входе многогранные сокращенные графы зависимостей. Вначале мы изложим некоторую мотивацию необходимости нового алгоритма (разд. 6.1), а затем приведем пошаговое описание алгоритма и обсудим его работу на примерах.

6.1. Потребность в новом алгоритме

Рассмотренные два распараллеливающих алгоритма таковы, что каждый из них может дать на выходе чисто последовательный код для той же самой программы, в которой другой алгоритм обнаружит некоторый параллелизм. Это побуждает к поискам нового алгоритма, объединяющего достоинства алгоритмов Вольфа—Лэма и Аллена—Кеннеди. Чтобы достичь такой цели, можно представить себе комбинацию алгоритмов, которая одновременно использует структуру СГЗ и структуру векторов направления, осуществляя следующую последовательность шагов.

1. Вначале он строит конус, генерируемый векторами направления, и преобразует гнездо циклов таким образом, чтобы выявить максимальное внешнее полностью переставляемое гнездо циклов.
2. Затем рассматривает подграф СГЗ, образованный векторами направления, которые не включены в самые внешние циклы, и вычисляет его сильно связные компоненты.
3. Наконец, применяет распределение цикла для разделения этих компонент и рекурсивно применяет эту же технику к каждой компоненте.

Такая стратегия позволяет выявить большее количество параллелизма, комбинируя унимодулярные преобразования и распределение циклов. Однако она не является оптимальной, что демонстрирует пример 7. Действительно, в программе этого примера вышеописанное сочетание алгоритмов Аллена—Кеннеди и Вольфа—Лэма позволяет определить только одну ступень параллелизма, так как на второй фазе СГЗ остается сильно связным. Результат не лучше, чем у базового алгоритма Аллена—Кеннеди. Однако можно обнаружить две ступени параллелизма, назначив $S_1(i, j, k)$ на момент времени $4i - 2k$ и $S_2(i, j, k)$ на момент времени $4i - 2k + 3$.

Пример 7.

```
DO i = 1, n
DO j = 1, n
   DO k = 1, n
      S1: a(i, j, k) = b(i-1, j+i, k) + b(i, j, k+2)
      S2: b(i, j, k) = a(i, j-i, k+j) + a(i, j, k-1)
   ENDDO
ENDDO
ENDDO
```

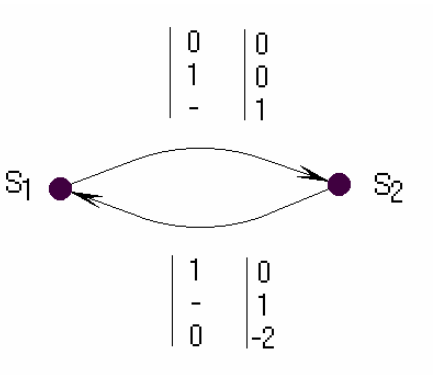


Рис. 8. СГЗ программы примера 7

Нам хотелось иметь простой распараллеливающий алгоритм, который бы находил некоторый параллелизм хотя бы во всех тех случаях, в которых это может сделать алгоритм Аллена—Кеннеди или алгоритм Вольфа—Лэма. Естественным решением было бы применить сначала алгоритм Аллена—Кеннеди, а затем алгоритм Вольфа—Лэма (или комбинацию обоих алгоритмов) и выдать наилучший результат. Но такой наивный подход не будет достаточно мощным, поскольку он использует либо структуру графа зависимостей, либо векторы направления, но не в состоянии воспользоваться знанием обоих структур одновременно. К примеру, предложенная комбинация двух алгоритмов могла бы использовать структуру графа зависимостей до или после вычисления максимального множества полностью переставляемых циклов, но никогда — во время этого вычисления. Утверждается [36], что и графовая структура, и векторы направления *должны* быть использованы одновременно — по той причине, что ключевым понятием при планировании СГЗ является не конус, генерируемый векторами направления (т. е. весами дуг СГЗ), а конус, генерируемый *весами циклов* СГЗ.

Все вышеописанное является мотивацией создания многомерного планирующего алгоритма, представленного ниже. Его можно рассматривать как комбинацию унимодулярных преобразований, распределения цикла и метода сдвига индексов. Это алгоритм, объединяющий достоинства алгоритмов Аллена—Кеннеди и Вольфа—Лэма, был предложен Дарте и Вивьеном [42]. Прежде чем переходить к его описанию, поясним выбор представления зависимостей, используемый алгоритмом.

6.2. Многогранные зависимости: пример

В данном разделе будет приведен пример гнезда циклов, содержащего параллелизм, который не может быть обнаружен, если зависимости представлены в виде уровней или векторов направления. Вместе с тем, пример таков, что для нахождения параллелизма в этом гнезде циклов нет необходимости использовать точное представление зависимостей, а достаточно иметь представление зависимостей в виде многогранника.

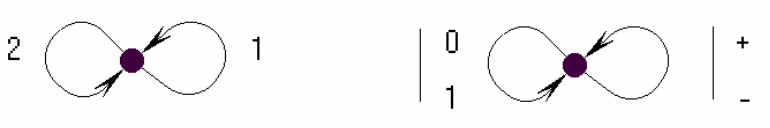
Рассмотрим пример 8. На рис. 9 представлены точные зависимости для этого фрагмента кода, а на рис. 10 изображены соответствующие (сокращенные) графы, дуги зависимостей которых помечены уровнями зависимостей и векторами направления, соответственно. Рассмотрим, что получается в результате применения к данному гнезду циклов рассмотренных ранее алгоритмов.

Пример 8.

```
DO i = 1, n
DO j = 1, n
    S : a(i, j) = a(j, i) + a(i, j-1)
ENDDO
ENDDO
```

$$S(i, j) \xrightarrow{flow} S(i, j+1) \text{ для } n \geq i \geq 1,\ n \geq j \geq 1,$$
$$S(i, j) \xrightarrow{flow} S(j, i) \text{ для } n \geq j > i \geq 1,$$
$$S(j, i) \xrightarrow{anti} S(i, j) \text{ для } n \geq i > j \geq 1,$$

Рис. 9. Точные отношения зависимостей для гнезда циклов из примера 8



а *б*

Рис. 10. СГЗ для фрагмента программы примера 8, в котором дуги помечены уровнями зависимостей (а) и векторами направления (б)

Алгоритм Аллена—Кеннеди базируется на рассмотрении уровней зависимостей. В программе примера 8 уровни трех зависимостей равны соответственно 2, 1 и 1. Имеется цикл зависимостей на уровне 1 и на уровне 2. Таким образом, параллелизм не обнаруживается.

Алгоритм Вольфа—Лэма рассматривает векторы зависимостей. В программе примера 8 векторы зависимостей выглядят как (0, 1), (+, –) и (+, –). Во втором измерении «1» и «–« не позволяют обнаружить два полностью переставляемых цикла. Следовательно, код остается без изменения и параллелизм не обнаруживается.

Для сравнения рассмотрим также результат применения алгоритма Фотрие, который будет описан в разд. 7. Алгоритм воспринимает в качестве входа точные зависимости и на выходе дает допустимый план $T(i, j) = 2i + j - 3$. Таким образом, алгоритм обнаруживает один уровень параллелизма.

Для данного конкретного гнезда циклов представление зависимостей в виде уровней или векторов направления не является достаточно точным для обнаружения параллелизма. В этом и состоит причина неудачи первых двух алгоритмов. Точный анализ зависимостей, связанный с методами линейного программирования, требующими решения больших[6] параметрических линейных программ (как в алгоритме Фотрие), позволяет обнаружить одну степень параллелизма. Соответствующий распараллеленный код будет выглядеть следующим образом:

```
DO j = 3, 3n
  DOPAR i = max (1, ⌈(j-n)/2⌉), min (n, ⌊(j-1)/2⌋)
        a(i, j-2i) = a(j-2i, i) + a(i, j-2i-1)
  ENDDO
ENDDO
```

Однако для гнезда циклов примера 8 точное представление зависимостей вовсе не является необходимым условием для обнаружения параллелизма. Действительно, можно заметить, что имеется униформная зависимость $u = (0, 1)$ и множество векторов расстояний $\{(j - i, i - j) = (j - i)(1, -1) \mid 1 \le j - i \le n - 1\}$, которое может быть приближено (сверху) множеством $P = \{(1, -1) + \lambda(1, -1) \mid \lambda \ge 0\}$. P является многогранником с одной вершиной $v = (1, -1)$ и одним лучом $r = (1, -1)$. Предположим теперь, что мы ищем линей-

[6] Количество неравенств и переменных связано с количеством граничных условий, которые определяют область допустимых значений каждого отношения зависимости.

ный план $T(i, j) = x_1 i + x_2 j$. Пусть $X = (x_1, x_2)$. Чтобы план T был допустимым, нужен такой X, чтобы выполнялось $Xd \geq 1$ для любого вектора зависимости d. Следовательно, $X(0, 1) \geq 1$ и $Xp \geq 1$ для всех $p \in P$. Последнее неравенство эквивалентно следующему: $X(1, -1) + \lambda X(1, -1) \geq 1$ при $\lambda \geq 0$, что эквивалентно $X(1, -1) \geq 1$ и $X(1, -1) \geq 0$, т. е. $Xv \geq 1$ и $Xr \geq 0$. Следовательно, достаточно решить следующие три неравенства:

$Xu \geq 1$ $\quad$ $Xv \geq 1$ $\quad$ $Xr \geq 0$, т.е.

$$X\begin{pmatrix}0\\1\end{pmatrix} \geq 1 \qquad X\begin{pmatrix}0\\-1\end{pmatrix} \geq 1 \qquad X\begin{pmatrix}-0\\1\end{pmatrix} \geq 0$$

что ведет, как у Фотрие, к $X = (2, 1)$. Таким образом, в данном примере приближение зависимостей с помощью уровней или даже векторов направления не является достаточным для определения параллелизма. Однако с помощью приближенного представления зависимостей в виде многогранника можно обнаружить тот же параллелизм, что и с помощью точного анализа зависимостей, но при этом решая более простое множество уравнений.

Особенно важна здесь «униформизация», которая позволяет нам перейти от неравенства на многограннике P к униформным неравенствам на v и r. Благодаря ей исчезают аффинные ограничения, и нам нет необходимости использовать аффинную форму леммы Фаркаша, как в алгоритме Фотрие (см. разд. 7). Чтобы лучше понять принцип «униформизации», будем рассуждать в терминах путей зависимостей. Рассмотрим дугу e, ведущую из оператора S к оператору T и помеченную вектором расстояния $p = v + \lambda r$, как путь φ, который использует один раз «униформный» вектор зависимости v и λ раз «униформный» вектор зависимости r. Такой способ рассмотрения представлен на рис. 11, где введена новая вершина S', которая позволяет моделировать φ и дугу нулевого веса, ведущую из S' обратно в начальную вершину T. Принцип «униформизации» является основной идеей алгоритма распараллеливания циклов, описанного в данном разделе.

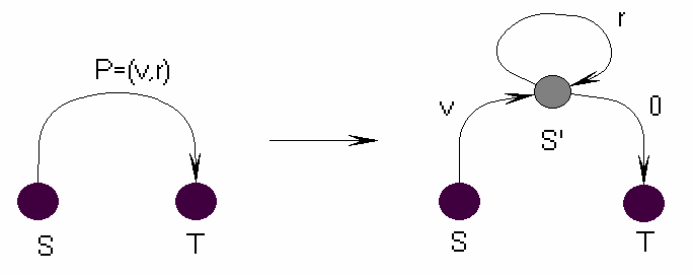


Рис. 11. Моделирование дуги, помеченной многогранником, с одной вершиной и одним лучом

«Униформизуя» зависимости, мы фактически «униформизовали» граничные условия и свели исходную проблему планирования с аффинными граничными условиями к простой проблеме планирования, где все зависимости являются униформными (u, v и r). Однако есть два фундаментальных различия между этой структурой и классической структурой униформного гнезда циклов.

- Униформные векторы зависимостей не обязательно являются лексически положительными (например, луч может быть равен (0, -1)). Поэтому задача планирования будет более сложной. Однако она может быть решена с помощью техник, подобных используемым при решении задачи нахождения решений систем униформных рекуррентных уравнений [52].
- Ограничения, налагаемые на луч r, являются более слабыми, чем в классическом случае, поскольку вместо $Xr \geq 1$ требуется, чтобы $Xr \geq 0$. Это ослабление должно приниматься во внимание распараллеливающим алгоритмом.

6.3. Работа алгоритма: пример

Рассмотрим пример, иллюстрирующий работу алгоритма. Будем считать, что в сокращенном графе зависимостей дуги помечены векторами направления. Граф зависимостей, изображенный на рис. 12, был построен с помощью анализатора зависимостей Tiny [62].

Пример 9.

```
DO i = 1, n
DO j = 1, n
    DO k = 1, n
        a(i, j, k) = c(i, j, k-1) + 1
        b(i, j, k) = a(i-1, j+i, k) + b(i, j-1, k)
        c(i, j, k+1) = c(i, j, k) + b(i, j-1, k+i) + a(i, j-k, k+1)
    ENDDO
ENDDO
ENDDO
```

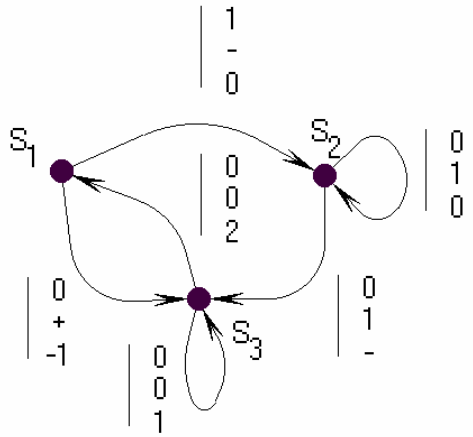


Рис. 12. СГЗ программы примера 9

Нетрудно убедиться, что алгоритмы Аллена—Кеннеди и Вольфа—Лэма не способны найти весь содержащийся в этой программе параллелизм: третий оператор кажется полностью последовательным. Однако алгоритм обнаружения параллелизма, который будет представлен в следующем разделе, способен построить следующий многомерный план: (2i + 1, 2k) для первого оператора, (2i, j) для второго и (2i + 1, 2k + 3) для третьего. Этот план соответствует коду с явным параллелизмом, приведенному ниже (в котором,

однако, не было проведено никаких модификаций наподобие отшелушивания цикла с целью устранения операторов IF). Таким образом, для каждого оператора может быть обнаружен один уровень параллелизма.

```
DOSEQ i = 1, n
DOSEQ j = 1, n
  DOPAR k = 1, j
       b(i, j, k) = a(i-1, j+i, k) + b(i, j-1, k)
  ENDDO
ENDDO
DOSEQ k = 1, n+1
  IF (k ≤ n) THEN
       DOPAR j = k, n
        a(i, j, k) = c(i, j, k-1) + 1
       ENDDO
  ENDIF
  IF (k ≤ 2) THEN
       DOPAR j = k-1, n
        c(i, j, k) = c(i, j, k-1) + b(i, j-1, k+i+1) + a(i, j-k+1, k)
       ENDDO
  ENDIF
ENDDO
ENDDO
```

Этот код был сгенерирован из вышеприведенного плана с помощью процедуры “codegen” программы Omega Calculator[7], поставляемого в комплекте Petit [53]. Следует напомнить, что вышеприведенный код является «виртуальным» в том смысле, что алгоритм только выявляет скрытый параллелизм. Реальный код не обязательно должен быть именно таким.

6.4. Шаг униформизации

Вначале мы покажем, как МСГЗ (многогранные сокращенные графы зависимостей) могут быть изображены с помощью эквивалентной, но более простой в обращении структуры — униформных графов зависимостей, то есть графов, дуги которых помечены константными векторами зависимо-

[7] Omega Calculator служит для вычисления зависимостей, проверки допустимости программных преобразований и преобразования программ заданным образом.

стей. Такая униформизация достигается с помощью приведенного ниже *алгоритма трансляции*.

Чтобы избежать путаницы между вершинами графа зависимостей и вершинами многогранника зависимостей, мы будем называть первые *узлами*, а вторые — *вершинами*. Будут использоваться также следующие соглашения. Исходный МСГЗ, описывающий зависимости в распараллеливаемом коде, называется *исходным графом* и обозначается как $G_0 = (V, E)$. Униформный СГЗ, эквивалентный G_0 и построенный с помощью алгоритма трансляции, называется *униформным графом* или *трансляцией* G_0 и обозначается как $G_u = (W, F)$.

Алгоритм трансляции строит G_u путем сканирования всех дуг G_0. Он начинает с $G_u = (W, F) = (V, \varnothing)$, и для каждой дуги e из E добавляет в G_u новые узлы и новые дуги в зависимости от вида многогранника P(e). Будем называть вновь созданные узлы *виртуальными узлами* в отличие от *реальных узлов*, соответствующих узлам G_0.

Пусть e — дуга из E. Обозначим через x_e и y_e соответственно начало и конец дуги e, т. е. узел, из которого e исходит, и узел, в который она заходит. Это определение обобщается до путей: начало (соответственно конец) пути — это начало (соответственно конец) его первой (соответственно последней) дуги.

Будем следовать нотациям, введенным в разделе 3.2: ω, ρ и λ обозначают количество вершин v_i, лучей r_i и линий l_i многогранника P(e).

Трансляцию осуществляет следующий алгоритм:

начало
Пусть W = V и F = $\varnothing$;
для всех e: $x_e \rightarrow y_e \in E$ **цикл**
Добавить к W новый виртуальный узел n_e;
Добавить к F ω дуг с весами $v_1, v_2, ..., v_\omega$, соединяющих x_e с n_e;
Добавить к F ρ петель, соединяющих n_e, с весами $r_1, r_2, ..., r_\rho$;
Добавить к F λ петель, соединяющих n_e, с весами $l_1, l_2, ..., l_\lambda$;
Добавить к F λ петель, соединяющих n_e, с весами $-l_1, -l_2, ..., -l_\lambda$;
Добавить к F дугу с нулевым весом, ведущую из n_e в y_e.
все
конец.

Вернемся к программе примера 9. Граф МСГЗ данной программы изображен на рис. 12. Рис. 13 представляет связанный с ней униформный граф зависимостей. Граф имеет три виртуальных узла, соответствующих символу

«+» и двум символам «–» в исходных векторах направления; это узлы, помеченные символами S'_1, S'_2 и S'_3.

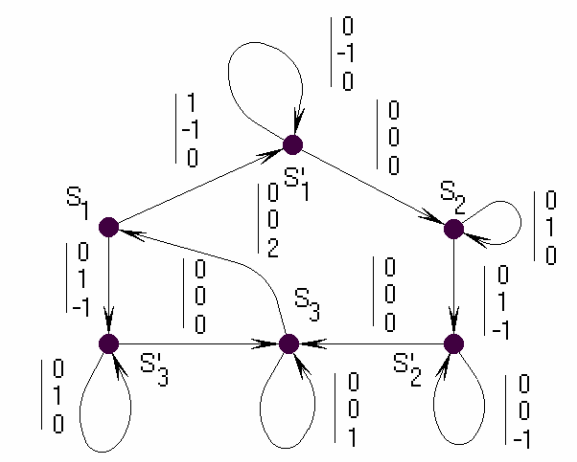


Рис. 13. Транслированный униформный сокращенный граф зависимостей

6.5. Шаг планирования

Шаг планирования принимает на входе транслированный граф зависимостей G_u и строит многомерный план для каждого реального узла, т. е. для каждого узла G_u, который соответствует узлу G_0. G_u предполагается сильно связным (в противном случае нужно вызывать алгоритм для каждой сильно связной компоненты G_u).

Алгоритм является рекурсивным. Каждый шаг рекурсии строит определенный подграф G' текущего обрабатываемого графа G. Как только G' построен, выводится множество линейных ограничений и может быть вычислен допустимый план, охватывающий все дуги зависимостей, не входящие в G'. Затем алгоритм продолжает работу с оставшимися дугами, то есть дугами из G' (более точно, с дугами из G' и некоторыми добавочными дугами — см. ниже).

G' определяется как подграф G, генерируемый всеми дугами G, принадлежащими по меньшей мере к одному мультициклу нулевого веса. Мультицикл есть объединение циклов, не обязательно связанных, а его вес есть

сумма весов составляющих его циклов. G' строится на основе решения линейной программы (см. разд. 6.6).

Шаг планирования описывается нижеприведенным рекурсивным алгоритмом. Первым вызовом является DARTE-VIVIEN(G_u, 1). Для каждого реального узла S из G_u алгоритм строит последовательность векторов X^1_S, ..., X^{dS}_S и последовательность констант ρ^1_S, ..., ρ^{dS}_S, которые определяют допустимый многомерный план.

проц DARTE-VIVIEN(G: **граф**, k: **целое**)=

1. Построить G' — такой подграф G, который порожден всеми дугами, принадлежащими по меньшей мере одному мультициклу нулевого веса в G;
2. Добавить к G' все дуги, ведущие из реального узла x_e к виртуальному узлу y_e, и все петли, соединяющие y_e, если дуга $e = (x_e, y_e)$ уже входит в G';
3. Выбрать вектор X и константу ρ_S для каждого узла S из G таким образом, чтобы выполнялось два условия:

$$e = (x_e, y_e) \in G' \text{ или } x_e \text{ — виртуальный узел} \Rightarrow$$
$$X\omega(e) + \rho_{y_e} - \rho_{x_e} \geq 0$$

$$e = (x_e, y_e) \notin G' \text{ или } x_e \text{ — реальный узел} \Rightarrow$$
$$X\omega(e) + \rho_{y_e} - \rho_{x_e} \geq 1,$$

   и для всех реальных узлов S из G положить $\rho^k_S = S$ и $X^k_S = X$.
4. **если** G' — пустой граф или содержит только виртуальные узлы
   то возврат все;
5. **если** G' — сильно связный граф и содержит реальные узлы
   то G является невычислимым (а исходный МСГЗ
   G_0 — несогласованным); **возврат**
   все;.
   Провести декомпозицию G' на сильно связные компоненты G_i;
   для всех G_i, имеющих реальные узлы, **цикл**
   DARTE-VIVIEN(G_i, k+1)
   все.

все.

К данному описанию алгоритма можно сделать следующие два замечания.

1. Шаг (2) является необходимым только для МСГЗ общего вида: его можно опустить, к примеру, для СГЗ, помеченных векторами направления (см. [44] для более подробного изложения). В этом случае решение простой линейной программы может заменить одновременно шаг (1) и шаг (3).
2. На шаге (3) мы сознательно не указываем, каким образом выбираются вектор X и константы ρ, что позволяет применять различные критерии выбора. Например, можно выбрать максимальное множество линейно независимых векторов X, если целью является построение полностью переставляемых циклов (см. [46]).

Вернемся к примеру 9. Рассмотрим униформный граф зависимостей на рис. 13. Имеются два элементарных цикла с весами (1, 0, 1) и (0, 1, 1) и пять петель с весами (0, 0, 1), (0, 0, –1), (0, 1, 0) (дважды) и (0, –1, 0). Следовательно, все дуги (кроме дуг, принадлежащих только к циклу с весом (1, 0, 1)) принадлежат к мультициклу нулевого веса. Подграф G' показан на рис. 14.

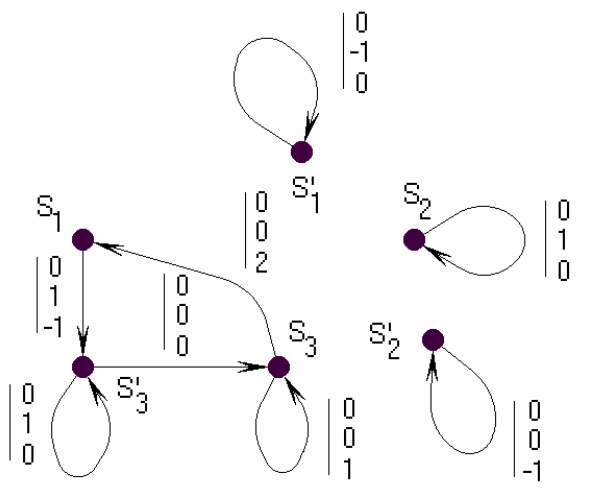


Рис. 14. Подграф мультициклов нулевого веса для примера 9

Ограничения, следующие из дуг подграфа G', устанавливают, что X = (x, y, z) должен быть ортогонален к весам всех циклов G'. Следовательно, y = z = 0. Наконец, рассматривая остальные ограничения, мы находим решение: X = (2, 0, 0), $\rho_{S1} = \rho_{S3} = 1$ и $\rho_{S2} = 0$. В G' остаются четыре сильно связных компоненты, две из которых не рассматриваются, так как имеют только виртуальные узлы. Две оставшиеся компоненты не содержат мультициклов нулевого веса. Сильно связная компонента с единственным узлом S_2 может быть спланирована с вектором X = (0, 1, 0), тогда как исследование второй сильно связной компоненты приводит к одному из возможных решений: X = (0, 0, 2), $\rho_{S1} = 0$ и $\rho_{S3} = 3$.

Суммируя все результаты, мы находим уже рассмотренные в разд. 6.3. двумерные планы: (2i, j) для S_2, (2i + 1, 2k) для S_1 и (2i + 1, 2k + 3) для S_3.

6.6. Схематические пояснения

Граф G_u не всегда соответствует СГЗ гнезда циклов, так как его векторы зависимостей не обязательно лексически неотрицательны. Фактически, если забыть о том, что некоторые узлы виртуальны, то G_u — это сокращенный граф зависимостей *Системы Униформных Рекуррентных Уравнений*, введенной Карпом, Миллером и Виноградом в [52]. Карп, Миллер и Виноград исследовали проблему вычислимости таких систем. Они показали, что ее вычислимость связана с проблемой нахождения циклов нулевого веса в ее СГЗ G, что может быть проделано посредством рекурсивной декомпозиции графа, основанной на нахождении мультициклов нулевого веса. Ключевой структурой их алгоритма является G', подграф G, генерируемый дугами, принадлежащими к мультициклу нулевого веса.

G' может быть успешно построен с помощью решения простой линейной программы (программа примера 7 или двойственная ей программа примера 8). Это решение позволяет построить алгоритм распараллеливания, принцип которого двойственен алгоритму Карпа, Миллера и Винограда:

$$\min\left\{\sum_e v_e \quad : \quad q \ge 0,\ v \ge 0,\ w \ge 0,\ q + v = 1 + w,\ Bq = 0\right\}, \tag{4}$$

$$\max\left\{\sum_e z_e \quad : \quad z \ge 0,\ 0 \le z_e \le 1,\ Xw(e) + \rho_{y_e} - \rho_{x_e} \ge z_e\right\}, \tag{5}$$

где w(e) — вектор зависимостей, связанный с дугой e, B = $[CW]^t$, C — матрица связей, а W — матрица векторов зависимостей.

Если не вдаваться в детали, X — это n-мерный вектор, имеется одна переменная ρ для каждой вершины СГЗ и одна переменная z для каждой дуги СГЗ. Дугами G' (или G\G') являются дуги $e = (x_e, y_e)$, для которых $z_e = 0$ (соответственно $z_e = 1$) в оптимальном решении двойственной программы 8, и, эквивалентно этому, для которых $v_e = 0$ (соответственно $v_e = 1$) в изначальной программе 7. Суммируя неравенства $Xw(e) + \rho_{ye} - \rho_{xe} \geq z_e$ в цикле C в G, мы находим, что Xw(C) = 0, если C — цикл из G', и $Xw(C) \geq l(C) > 0$ в противном случае (l(C) — количество дуг C, не принадлежащих G').

Чтобы увидеть связь с алгоритмом Вольфа—Лэма, можно рассмотреть конус Γ, генерируемый *весами циклов* (а не весами дуг); тогда G' — подграф, веса циклов которого генерируют пространство линейности Γ, а X — вектор в относительно внутренней части Γ^+. Однако нет необходимости для построения G' реально строить Γ. Это просто одно из любопытных свойств программ примеров 7 и 8.

Мы обрисовали основные идеи алгоритма Дарте—Вивьена [44]. Требуются некоторые технические модификации для распознавания виртуальных и реальных узлов и для учета природы дуг (помеченных вершинами, лучами или линиями многогранника зависимостей). Отсылаем читателя к работе [42] за полным изложением алгоритма.

6.7. Сильные и слабые стороны алгоритма

Теперь, когда у нас есть многомерный план T, мы можем доказать его оптимальность в терминах степени параллелизма. Можно показать [41, 42], что для каждого оператора S (то есть для каждого узла G_0) количество экземпляров S, исполняемых последовательно в результате применения T, имеет тот же порядок, что и количество экземпляров S, являющихся врожденно последовательными в силу действия зависимостей.

Теорема 3. *Планирующий алгоритм является почти оптимальным, если область итерации содержит полномерный куб размера $\Omega(N)$ (соответственно содержится в кубе размерности $O(N)$) и если d — глубина (количество вложенных рекурсивных вызовов) алгоритма, то время ожидания плана равно $O(N^d)$ и длина самого длинного пути зависимостей равна $\Omega(N^d)$. Более точно, после генерации кода каждый оператор S окружен в точности d_S последовательными циклами, и эти циклы считаются врожденно последовательными также после анализа зависимостей.*

Этот алгоритм также является оптимальным относительно анализа зависимостей. Рассмотрим следующий пример.

Пример 10.

```
DO i = 1, n
DO j = 1, n
      S1: a(i, j) = b(i-1, j+i) + a(i, j-1)
      S2: b(i, j) = a(i-1, j-i) + b(i, j-1)
ENDDO
ENDDO
```

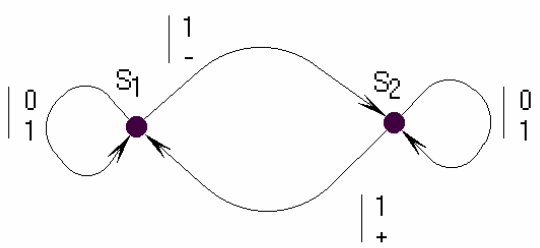


Рис. 15. СГЗ программы примера 10

Если зависимости описываются с помощью векторов расстояния, СГЗ имеет две автозависимости (0, 1) и две дуги, помеченные многогранниками, обе имеющие по одной вершине и одному лучу — (0, 1) и (0, –1), соответственно (см. рис. 15). Следовательно, существует мультицикл нулевого веса. Далее, две реальных вершины принадлежат к G'. Таким образом, глубина алгоритма Дарте—Вивьена равна 2, и никакого параллелизма не обнаруживается.

Однако вычисление итерации (i, j) первым (вторым) оператором на шагу 2i + j (соответственно i + j) приводит к допустимому плану, который обнаруживает одну степень параллелизма[8]. Алгоритм Дарте—Вивьена не способен найти параллелизм в этом примере, так как аппроксимация зависимостей уже потеряла весь параллелизм.

Используемая здесь нами техника нахождения параллельных циклов заключается в поиске многомерных планов, линейные части которых (векто-

[8] Планы $\lfloor(3/2)i + j + 1/2\rfloor$ и $\lfloor(1/2)i + j\rfloor$ минимизируют время ожидания, но написание кода становится более сложным.

ры X) могут быть разными для разных операторов, даже если они принадлежат к одной и той же сильно связной компоненте. Это является основой алгоритма Фотрие [47], математической базой которого является аффинная форма леммы Фаркаша. Однако теорема 3 показывает, что нет необходимости искать различные линейные части (построение которых будет более дорогим и приведет к усложнению процесса переписывания) в заданной сильно связной компоненте текущего подграфа G', если зависимости заданы с помощью векторов расстояния. С другой стороны, пример 10 показывает, что такое усовершенствование может быть полезным только тогда, когда доступен более точный анализ зависимостей.

7. АЛГОРИТМ ФОТРИЕ

В работе [47] Пол Фотрие предложил алгоритм планирования статических управляющих программ с аффинными зависимостями. Этот алгоритм использует точный анализ зависимостей, который всегда является осуществимым для таких типов программ [45]. В этом его отличие от трех ранее рассмотренных алгоритмов (Аллена—Кеннеди, Вольфа—Лэма, Дарте—Вивьена), которые работают с теми или другими аппроксимациями зависимостей.

Алгоритм Фотрие принимает на входе сокращенный граф зависимостей G, в котором дуга e: $S_i \to S_j$ помечена множеством пар (I, J) таких, что $S_j(J)$ зависит от $S_i(I)$. Далее он рекурсивно строит многомерный аффинный план для каждого оператора гнезда циклов.

проц FEAUTRIER(G : **граф**) =
    Провести декомпозицию G на сильно связные компоненты G_i и топологически отсортировать их.
    для всех сильно связных компонент G_i **цикл**
        Найти аффинный план с участием оператора, который налагает неотрицательную задержку на все зависимости и задействован в наибольшем возможном количестве зависимостей;
        Построить множество G'_i незадействованных дуг;
            если $G'_i \neq \varnothing$ **то** FEAUTRIER(G'_i) **все**
    все
все

Данный алгоритм похож на алгоритм Дарте—Вивьена по своей структуре и по выдаваемому результату, а также тем, что оба алгоритма используют линейные программы для построения аффинных планов. Приведем основные свойства, связанные со сравнением данных двух алгоритмов (см. [36]).

1. Алгоритм Дарте—Вивьена способен планировать программы, даже если точный анализ зависимостей не является осуществимым, но дан СГЗ с многогранником зависимостей. Алгоритм Фотрие способен обрабатывать только статические управляющие программы с аффинными зависимостями. В этом смысле первый алгоритм является более мощным. Однако укажет на попытки обобщить подход, реализованный в алгоритме Фотрие, путем ослабления ограничений на вход за счет использования анализа потока данных на нечетких массивах (Fuzzy Array Dataflow Analysis) [33].
2. Когда доступен анализ зависимостей, алгоритм Фотрие становится намного более мощным. Этот алгоритм способен обрабатывать любое множество циклов, описывающих многогранник, даже если циклы не образуют совершенное гнездо. Алгоритм Дарте—Вивьена также может обрабатывать не являющиеся совершенными вложенные гнезда циклов, либо рассматривая каждый блок совершенно вложенных циклов по отдельности, либо искусственно сливая несовершенно вложенные гнезда. Однако согласно теории, этот подход будет менее естественным и менее мощным.
3. Алгоритм Дарте—Вивьена основан на решении линейных программ, подобных тем, что решает алгоритм Фотрие. Единственное (но фундаментальное) различие состоит в том, что первый ищет менее общие аффинные преобразования. Следовательно, на статических управляющих программах с аффинными зависимостями, алгоритм Фотрие всегда находит больше параллелизма, чем алгоритм Дарте—Вивьена (см. пример 10). Однако, несмотря на это различие, результат оптимальности для алгоритма Дарте—Вивьена дает некоторые указания на случаи оптимальности для Фотрие, которые были впервые введены как «жадные эвристики».
4. Алгоритм Фотрие должен использовать аффинную форму леммы Фаркаша для получения линейных программ, чего алгоритм Дарте—Вивьена избегает благодаря своей схеме униформизации. Следовательно, линейные программы алгоритма Фотрие более сложны.
5. Оба алгоритма расширяют действие от мелкозернистого параллелизма до среднезернистого путем поиска полностью переставляемых циклов. Дарте и др. [40] предложили расширение алгоритма Дарте—Вивьена,

которое представляет собой простое обобщение Вольфа—Лэма. Лим и Лэм [57] предложили расширение алгоритма Фотрие, которое находит максимальные множества полностью переставляемых циклов, минимизируя при этом количество синхронизаций, необходимых в распараллеленном коде.

6. Алгоритм Дарте—Вивьена производит настолько регулярные планы, насколько это возможно, чтобы получить возможно более простой код. Действительно, этот алгоритм переписывает код с использованием аффинных планов, но, в отличие от алгоритма Фотрие, эти аффинные планы выбираются таким образом, чтобы возможно большее количество операторов имели одну и ту же линейную часть: после этого генерация кода может рассматриваться как последовательность частичных унимодулярных преобразований и распределений циклов. В результате полученные программы выходят гарантированно более простыми, чем программы, получаемые с помощью алгоритма Фотрие.

В работе [36] приводятся результаты небольшого сравнительного исследования алгоритмов, которое проводилось на четырех примерах. Как и ожидалось авторами, в этих исследованиях сложность алгоритма Дарте—Вивьена была намного ниже, чем у алгоритма Фотрие. Более удивительным оказался тот факт, что оба алгоритма дали один и тот же результат на всех четырех рассмотренных примерах. Разумеется, для получения окончательного заключения о сравнительных характеристиках алгоритмов данного исследования явно не достаточно: необходимо сравнительное исследование алгоритмов на существенно большем числе реальных программ.

На рис. 16 приведен пример кода, который с очевидностью содержит некоторый параллелизм, но не поддается распараллеливанию ни одним из четырех рассмотренных в данной статье алгоритмов.

```
                        DOPAR i = 1, ⌊n/2⌋
                        a(i) = 1 + a(n-i)
DO i = 1, n             ENDDO
   a(i) = 1 + a(n-i)    DOPAR i = ⌊n/2⌋ + 1, n
ENDDO                   a(i) = 1 + a(n-i)
                    ENDDO
```

Рис. 16. Пример исходного (слева) и распараллеленного (справа) кода

8. ЗАКЛЮЧЕНИЕ

В статье рассмотрены основные существующие алгоритмы распараллеливания циклов — одного из наиболее важных реструктурирующих преобразований. Суммируя их свойства, следует отметить следующее: алгоритм Аллена—Кеннеди является оптимальным для представления зависимостей в виде уровней, а алгоритм Вольфа—Лэма — для представления зависимостей в виде векторов направления (но для гнезда циклов с единственным оператором). Ни один из двух алгоритмов не охватывает другой, поскольку каждый использует информацию, которая не может быть использована вторым (графовая структура для первого алгоритма, векторы направления — для второго). Однако оба алгоритма охватываются алгоритмом Дарте—Вивьена, который является оптимальным для любого многогранного представления векторов расстояния. Алгоритм Фотрие охватывает алгоритм Дарте—Вивьена в случаях, когда зависимости могут быть представлены как аффинные, но вопрос об его оптимальности остается открытым.

Рассмотренная классификация алгоритмов распараллеливания циклов имеет практический интерес. Она дает возможность распараллеливающему компилятору выбрать наиболее подходящий алгоритм: при условии, что задан определенный анализ зависимостей, должен выбираться самый простой и дешевый распараллеливающий алгоритм, который остается оптимальным. Это будет именно тот алгоритм, который лучше всего подходит к доступному представлению зависимостей.

СПИСОК ЛИТЕРАТУРЫ

1. Булышева Л.А. Методы и средства оптимизации вычислений для процессоров с VLIW-архитектурой. — Новосибирск, 1993. — (Препр. / РАН. Сиб.отд-ние. ВЦ; N 976).
2. Вальковский В. А. Распараллеливание алгоритмов и программ. Структурный подход. — М.: Радио и связь, 1989.
3. Векторизация программ: теория, методы, реализация: Сб. статей / Под ред. Г.Д. Чинина. — М.: Мир, 1991.
4. Воеводин В. В. Математические основы параллельных вычислений. — М.: МГУ, 1991.
5. Воеводин Вл. В. Теория и практика исследования параллелизма последовательных программ // Программирование. — 1992. — N 3. — С. 38–54.
6. Глушков В.М., Капитонова Ю.В., Летичевский А.А. Алгебра алгоритмов и динамическое распараллеливание последовательных программ // Кибернетика. — 1982. — N 5. — С. 4–10.

7. Дорошенко А.Е. Математические модели и методы организации высокопроизводительных параллельных вычислений. — Киев: Наукова думка, 2000.
8. Дорошенко А.Е. О преобразованиях циклических операторов к параллельному виду // Кибернетика. — 1984. — N 4. — С. 22–28.
9. Евстигнеев В.А., Касьянов В.Н. Оптимизирующие преобразования в распараллеливающих компиляторах // Программирование. — 1996. — N 6. — С. 12–26.
10. Евстигнеев В.А., Мирзуитова И.Л. Анализ циклов: выбор кандидатов на распараллеливание. — Новосибирск, 1999. — (Препр. / ИСИ СО РАН; N 58).
11. Евстигнеев В. Анализ зависимостей: состояние проблемы // Системная информатика. — Новосибирск: Наука, 2000. — Вып. 7. — С.112–173.
12. Иванников В.П., Гайсарян С.С. Особенности систем программирования для векторно-конвейерной ЭВМ // Кибернетика и вычислительная техника. — М.: Наука, 1986. — Вып. 2. — С. 3–17.
13. Касьянов В.Н., Евстигнеев В.А. Графы в программировании: обработка, визуализация и применение. — СПб.: БХВ-Петербург, 2003.
14. Касьянов В.Н. Оптимизация программ // Прикладная информатика. — М.: Статистика, 1983. — Вып. 2. — С. 38–76.
15. Касьянов В.Н. Оптимизирующие преобразования программ. — М.: Наука , 1988.
16. Касьянов В.Н. Трансформационный подход к конструированию и оптимизации программ // Смешанные вычисления и преобразование программ. — Новосибирск, 1991. — С. 30–43.
17. Касьянов В.Н. Трансформационные методы и средства конструирования эффективных и надежных программ // Кибернетика и системный анализ. — 1993. — N 2. — С. 30–39.
18. Системное и математическое обеспечение многопроцессорного вычислительного комплекса ЕС / Под ред. Ю.В. Капитоновой. — М.: ВВИА им. Н.Е. Жуковского, 1985.
19. Трантенгерц Э. А. Программное обеспечение параллельных процессов. — М.: Наука, 1987.
20. Французов Ю. А. Обзор методов распараллеливания кода и программной конвейеризации // Программирование. — 1992. — N 3. — С. 16–37.
21. Черняев А.П. Системы программирования для высокопроизводительных ЭВМ. — Т.3. Вычислительные науки. — М., 1990. — С. 1–141. — (Итоги науки и техн. ВИНИТИ АН СССР).
22. Черняев А.П. Программные системы векторизации и распараллеливания Фортран-программ для некоторых векторно-конвейерных ЭВМ (обзор) // Программирование. — 1991. — N 2. — С. 53–68.
23. Allan V.H., Jones R.B., Lee R.M., Allan S.J. Software pipelining // ACM Computing Surveys. — 1996. — Vol. 27, N 3. — P.367–432.
24. 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.

25. Allen J.R., Kennedy K. PFC: a program to convert programs to parallel form. — Houston, 1982. — (Tech. Rep. / Dept. of Math. Sciences, Rice University; MASC-TR82-6).
26. Bacon D. F., Graham S. L., Sharp O. J. Compiler transformations for high-performance computing // ACM Computing Surveys. — 1994. — Vol. 26, N 4. — P.345–420.
27. Banerjee U. A theory of loop permutations // Languages and Compilers for Parallel Computing. — MIT Press, 1990. — P. 54–74.
28. Banerjee U. Data dependence in ordinary programs. — Urbana, 1976. — (Tech. Rep. / Univ. Ill., 76-837).
29. Banerjee U. Dependence analysis for supercomputing. —Norwell: Kluwer Academic Publishers, 1988.
30. Bernstein A. J. Analysis of programs for parallel processing. // IEEE Trans. on Electronic Computers. — 1966. — Vol. 15, N 5. — P. 757–763.
31. Bockle G. Exploitation of fine-grain parallelism. — Berlin: Springer-Verlag, 1995. — (Lect. Notes Comput. Sci.; Vol. 942).
32. Callahan D. A Global Approach to Detection of Parallelism: PhD thes. — Dept. of Computer Science, Rice University. — Houston, 1987.
33. Collard J.-F., Barthou D., and Feautrier P. Fuzzy Array Dataflow Analysis // Proc. of 5th ACM SIGPLAN Symp. on Principles and practice of Parallel Programming. — Santa Barbara, CA, 1995.
34. Collard J.-F., Feautrier P., Risset T. Construction of DO loops from systems of affine constraints // Parallel Processing Letters. — 1995. — Vol. 5, N 3. — P. 421–436.
35. Collard J-F. Code generation in automatic parallelizers // Proc. Int. Conf. on Application in Parallel and Distributed Computing. — North Holland, 1994. — P. 185–194.
36. Computer optimizations for scalable parallel systems: languages, compilation techniques, and run time systems / Santosh Pande; Dharma P. Agrawal (ed.). — Berlin: Springer-Verlag, 2001. — (Lect. Notes Comput. Sci.; Vol. 1808).
37. Darte A., Khachiyan L., Robert Y. Linear scheduling is nearly optimal // Parallel Processing Letters. — 1991. — Vol. 1, N 2. — P. 73–81.
38. Darte A., Robert Y. Affine-by-statement scheduling of uniform and affine loop nests over parametric domains // J. Parallel and Distributed Computing. — 1995. — Vol. 29. — P. 43–59.
39. Darte A., Robert Y. Mapping uniform loop nests onto distributed memory architectures // Parallel Computing. — 1994. — Vol. 20. — P. 679–710.
40. Darte A., Silber J.-A., Vivien F. Combining retiming and scheduling techniques for loop parallelization and loop tiling. — ENS-Lyon, 1996. — (Tech. Rep. / LIP; 96-34).
41. Darte A., Vivien F. Automatic parallelization based on multi-dimensional scheduling. — Lyon, 1994. — (Tech. Rep. / LIP; 94-24).
42. Darte A., Vivien F. On the optimality of Allen and Kennedy's algorithm for parallelism extraction in nested loops //. J. of Parallel Algorithms and Application. — 1996. — (Special issue on Optimizing Compilers for Parallel Languages).

43. Darte A., Vivien F. Optimal fine and medium grain parallelism detection in polyhedral reduced dependence graphs. — Lyon, 1996. — (Tech. Rep. / LIP; 96-06).
44. Darte A., Vivien F. Optimal fine and medium grain parallelism detection in polyhedral reduced dependence graphs // Proc. of PACT'96. — Boston: IEEE Computer Society Press, 1996.
45. Feautrier P. Dataflow analysis of array and scalar references // Int. J. Parallel Programming. — 1991. — Vol. 20, N 1. — P. 23–51.
46. Feautrier P. Some efficient solutions to the affine scheduling problem, part I, one-dimensional time // Int. J. Parallel Programming. — 1992. — Vol. 21, N 5. — P. 313–348.
47. Feautrier P. Some efficient solutions to the affine scheduling problem, part II, multi-dimensional time // Int. J. Parallel Programming. — 1992. — Vol. 21, N 6. — P. 389–420.
48. Goff G., Kennedy K., Tseng C.-W. Practical dependence testing // Proc. of the SIGPLAN Conf. on Programming Language Design and Implementation. — SIGPLAN Not. — 1991. —Vol. 26, N 6. — P. 15–29.
49. Irigoin F., Jouvelot P., Triolet R. Semantical interprocedural parallelization: an overview of the PIPS project // Proc. of the 1991 ACM Internat. Conf. on Supercomputing. —Cologne, 1991.
50. Irigoin F., Triolet R. Computing dependence direction vectors and dependence cones with linear systems. — Fontainebleau, 1987. — (Tech. Rep. / Ecole des Mines de Paris; ENSMP-CAI-87-E94).
51. Irigoin F., Triolet R. Supernode partitioning // Proc. 15th Annual ACM Symp. Principles of Programming Languages. — San Diego: CA, 1988. — P. 319–329.
52. Karp R.M., Miller R.E., Winograd S. The organization of computations for uniform recurrence equations // J. of the ACM. — 1967. — Vol. 14, N 3. — P. 563–590.
53. Kelly W., Maslov V., Pugh W., Rosser E., Shpeisman T., and Wonnacott D. New user interface for Petit and other interfaces: user guide. — University of Maryland, 1995.
54. Kong X., Klappholz D., Psarris K. The I test: an improved dependence test for automatic parallelization and vectorization // IEEE Trans. Par. Distr. Syst. — 1991. — Vol. 2, N 3. — P. 342–349.
55. Lamport L. The parallel execution of DO loops // Commun. of the ACM. — 1974. — Vol. 17, N 2. — P. 83–93.
56. Li Z. Array privatization for parallel execution of loops // Proc. of the 1992 Intern. Conf. on Supercomputing. — 1992. — P. 313—322.
57. Lim A.W., Lam M.S. Maximizing parallelism and minimizing synchronization with affine partitions // Parallel Computing. — 1998. — Vol. 24, Iss. 3-4. — P. 445–475.
58. Pugh W. A practical algorithm for exact array dependence analysis // Commun. ACM. — 1992. — Vol. 35, N 8. — P. 102–115.
59. Schreiber R., Dongarra J.J. Automatic blocking of nested loops. — Knoxville, 1990. — (Tech. Rep. / The University of Tennessee; 90-38).

60. Wolf M.E., Lam M.S. A loop transformation theory and an algorithm to maximize parallelism // IEEE Trans. Parallel Distributed Systems. — 1991. — Vol. 2, N 4. — P. 452–471.
61. Wolfe M. High Performance Compilers For Parallel Computing. — Addison-Wesley Publishing Company, 1996.
62. Wolfe M. Optimizing Supercompilers for Supercomputers: PhD thes. — Dept. of Computer Science, University of Illinois. — Urbana-Champaign, 1982.
63. Wolfe M. Optimizing Supercompilers for Supercomputers. —Cambridge: MIT Press, 1989.
64. Wolfe M. TINY, a loop restructuring research tool. — Oregon Graduate Institute of Science and Technology, 1990.
65. Xue J. Automatic non-unimodular transformations of loop nests // Parallel Computing, 1994. — Vol. 20, N 5. — P.711–728.
66. Zima H. and Chapman B. Supercompilers for Parallel and Vector Computers. — ACM Press, 1990.