ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

На правах рукописи

Штейнберг Роман Борисович

АВТОМАТИЧЕСКОЕ ОТОБРАЖЕНИЕ ПРОГРАММ НА КОНВЕЙЕРНЫЕ И МНОГОКОНВЕЙЕРНЫЕ АРХИТЕКТУРЫ

05.13.11 – МАТЕМАТИЧЕСКОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ВЫЧИСЛИТЕЛЬНЫХ МАШИН, КОМПЛЕКСОВ И КОМПЬЮТЕРНЫХ СЕТЕЙ

ДИССЕРТАЦИЯ
на соискание учёной степени
кандидата физико-математических наук

Научный руководитель:
заслуженный работник высшей школы РФ,
кандидат физико-математических наук,
профессор Ерусалимский Яков Михайлович

Ростов-на-Дону
2012

Оглавление

Список сокращений

АЛУ – арифметико-логическое устройство

БДОП – быстрое дискретное ортогональное преобразование

ДВОР – диалоговый высокоуровневый оптимизирующий распараллеливатель

МВС – многопроцессорная вычислительная система

ОРС – открытая распараллеливающая система

ПЛИС – программируемая логическая интегральная схема

ПО – программное обеспечение

ПЭ – процессорный элемент

РГА – редуцированный граф алгоритма

ЦЛП – целочисленное линейное программирование

DSP – Digital Signal Processor

FPGA – Field Programmable Gate Array

HDL – Hardware Description Language

VBA – Visual Basic for Applications

VLIW – Very Long Instruction Word

Введение

Актуальность темы. Сегодня с использованием высокопроизводительных вычислительных систем решаются задачи прогноза погоды, межмолекулярного взаимодействия, биоинформатики, нефтегазовой разведки, наноинженерии, механики, гидродинамики и т.д. [38, 46]. В мире представлено большое многообразие архитектур высокопроизводительных вычислительных систем, но никакой суперкомпьютер не способен решать разные задачи с одинаковой эффективностью. Более того, известны случаи, когда решение задачи на кластере проигрывало по сравнению с ее решением на одном вычислительном узле этого же кластера. В данной работе рассматривается отображение гнезд циклов на модель многоконвейерной вычислительной системы. Полученные результаты позволяют оценивать эффективность отображения различных программ на суперкомпьютеры, допускающие многоконвейерные вычисления.

В этой работе рассматривается один из вариантов распараллеливания по данным — конвейерные вычисления. В процессе конвейерных вычислений одновременно выполняются независимые этапы одного и того же вычисления.

Конвейеры создавались для вычисления отдельных алгоритмов (уровень функций) [14, 36], для вычисления отдельных операций (уровень АЛУ) [14, 25, 45], для набора операций (VLIW) [14, 47], для инструкций процессора. Несмотря на то, что все эти конвейеры сильно отличаются друг друга по реализации, в их основе лежит идея разбиения вычислений на ступени и одновременного выполнения этих ступеней (для разных данных).

Программные конвейеры [20, 22, 47] используются в работе большинства современных процессоров от Intel, AMD, Apple. Для построения специализированных вычислителей [45, 75], таких как бортовые компьютеры, устройства фильтрации сигналов, активно используется конвейерный подход.

Сегодня в России, как и в других странах мира, ведутся исследования в области и аппаратного [29, 31, 33, 83], и программного обеспечения [34] суперкомпьютеров, в том числе и связанные с компьютерами с *перестраиваемой архитектурой* (reconfigurable architecture). Одним из наиболее крупных проектов по созданию суперкомпьютера с архитектурой перестраиваемого конвейера является проект Merrimac [78]. Многие компьютеры с программируемой архитектурой позволяют создавать перестраиваемые (программируемые) конвейеры [28, 31, 77], показывающие большую эффективность при работе с рекуррентными циклами. В этой архитектуре основной идей является

конфигурирование системы для организации вычислений по заданной программе. Как правило, для этого настраиваются связи между элементарными вычислителями внутри компьютера. При создании компиляторов с языков высокого уровня для таких архитектур, желательно организовать автоматическое формирование связей между вычислителями в соответствии с потоком данных в компилируемой программе. Задачу автоматического формирования связей на этапе компиляции позволяет решить модель графа вычислений программы и построенная на ее основе модель многоконвейерных вычислений.

В данной работе рассматриваются конвейерные вычисления и автоматизация разработки программ, использующих конвейерные вычисления. Рассматривается отображение гнезд циклов на модель многоконвейерной вычислительной системы. Тем самым описывается класс программ, которые могут быть эффективно выполнены на суперкомпьютерах, допускающих многоконвейерные вычисления, и основы разработки распараллеливающих компиляторов на такие архитектуры.

Обзор литературы

Начнем рассмотрение публикаций по конвейерным вычислениям с работ, посвященных *программным конвейерам* (software pipelining). В серии работ [20], [21], [22] рассматриваются гнезда циклов, содержащие только операторы присваивания. Рассматривается развертка внешних циклов этих гнезд как средство повышения эффективности программной конвейеризации. В обзоре [20] рассматриваются алгоритмы планирования программных конвейеров, в основном VLIW-подобные архитектуры. Проводится сравнение различных вариантов алгоритмов планирования по модулю, а также методы глобального планирования (иерархическая редукция, if-конверсия, метод суперблоков, усовершенствованное планирование по модулю). Рассматривается генерация конвейерного кода, в частности оптимизация его объема. В работе [22] говорится о том, что она является развитием метода гиперплоскостей Лампорта [81] и использует ЦЛП-подход к решению задачи планирования конвейера. Предлагается в итоге свести задачу вычисления коэффициентов развертки к задаче ЦЛП над векторами расстояния зависимостей (distance vector). Алгоритм приведен в общих чертах.

В работе [86] предлагается итеративный алгоритм для нахождения минимального интервала инициализации итераций, применительно к программным конвейерам.

В области ПО для суперкомпьютерного рынка также происходят существенные изменения. Например, 1 сентября 2010 г. журнал HPCWire опубликовал пресс-релиз программного продукта C-to-FPGA, который разработан компанией Stone Ridge

Technology в рамках проекта ImpulseC (см. [91]). Этот продукт предназначен для генерации HDL-описаний в RTL-форме на основе алгоритма, написанного на языке высокого уровня. По оценкам экспертов, это позволит сэкономить до 2/3 времени разработки проектов на ПЛИС. В России также существуют группы исследователей, занимающиеся изучением новых языковых средств параллельного программирования (см. [29], [37], [57]), а также предлагаются сервисы по исследованию программы на параллельность [43, 89 (см. web-ops)].

В главе 5 работы [45] рассматривается оптимизация проектирования конвейерных вычислителей на базе ПЛИС. Для этого создано полуавтоматическое ПО Paredit, в котором необходимо задать РГА, для последующей оптимизации *конфигурации алгоритма* и отображения его на микросхему. В данной диссертации рассматривается автоматическое создание графа вычислений (или РГА), что может существенно помочь в цикле проектирования конвейерных вычислителей, описанном в [45].

В работах [8, 9] рассматривается комплексный подход по созданию программно-аппаратных систем. В основе лежит язык описания макроархитектуры процессора – PADLA (Processor Architecture Description Language), из которого автоматически генерируются все необходимые средства разработки, а также VHDL-описание процессора.

В работе [25] рассматривается новая автоматическая система проектирования, специализированная на автоматизации ранних этапов проектирования устройств трехмерной голографической памяти.

Наиболее распространенными языками для описания электронных схем на сегодняшний день являются VHDL, Verilog. Тем не менее можно встретить работы, например [7], в которых авторы пытаются развить существующие языки описаний. В частности, это может повлиять и на способы описания конвейерных вычислений.

В работе [49] рассматривается абстрактная модель, предназначенная для оптимизации проектирования конвейерных вычислителей на базе DSP. Пользователю программы Simulink, входящей в состав пакета MatLab, предлагается вручную задать архитектурную модель системного уровня. Тестирование такой модели представляется более трудоемким, чем тестирование программы на языке С. В настоящей диссертации рассматриваются вопросы генерации HDL-описаний по программе на языке С.

В работе [35] рассматривается проектирование однородных вычислительных сред. Такие вычислительные среды рассматривались и ранее в качестве архитектур, позволяющих эффективное параллельное вычисление широкого класса задач [27, 31]. Работа [28] является закономерным развитием работы [27] и содержит описание

архитектуры суперкомпьютеров со структурно-процедурной организацией вычислений, получившей практическое применение. В основе этой архитектуры лежит идея настройки компонент компьютера под задачу, особенно эффективно это применимо для организации конвейерных вычислений. В других источниках такие суперкомпьютеры называются компьютерами с перестраиваемой архитектурой (см. [32, 77, 78]). В работе [28] подробно рассматриваются идеология такой архитектуры и принципы взаимодействия компонент компьютера. Также рассматриваются вопросы выполнения программ на компьютерах этой архитектуры, в частности, отображение графового представления программы на вычислительную систему этой архитектуры и разбиение программы на кадры. Задача автоматического разбиения программы на кадры рассматривается в [2].

В последнее время все чаще можно встретить упоминания об архитектурах суперкомпьютеров, допускающих многоконвейерные вычисления. Одной из наиболее ранних работ, в которой предлагалось рассматривать семейства конвейеров, является [44]. В ней рассматривается возможность организации мультиконвейерных вычислений для задач быстрых дискретных ортогональных преобразований (БДОП) на базе двумерного линейно-циклического процессора (см. [44, §5.5]). В работе [75] рассматриваются разные алгоритмы решения одномерных, двумерных и трехмерных задач цифровой фильтрации сигналов. Утверждается, что для одномерной задачи, параллельно-конвейерный (многоконвейерный) алгоритм является наиболее эффективным [75, с. 58]. В работе [29] рассматриваются однородные вычислительные среды, модульно-наращиваемая реализация реконфигурируемых мультиконвейерных архитектур, а также инструменты разработки программ для таких архитектур. Такими программными инструментами являются среда разработки параллельных программ Argus IDE и среда разработки вычислительных структур Fire!Constructor, разработанные в НИИ МВС (г. Таганрог). Fire!Constructor позволяет создавать граф-схемы, описывающие алгоритмы решения задач, и пытается отобразить их на аппаратный ресурс реконфигурируемых архитектур. Для этого используются алгоритмы полного перебора, имеющие экспоненциальную сложность. Если отображение выполнено успешно, пользователь среды может получить файлы VHDL-описаний и файлы временных и топологических ограничений.

Во многих работах рассматриваются приложения методов анализа информационных зависимостей к проектированию электронных схем. Так, например, в работе [6] описывается новый подход к проектированию микросхем. Целью этого подхода является улучшение качественных характеристик разрабатываемых микросхем при увеличении времени разработки не более чем в 2 раза. Результатом явились более

20 реализованных микросхем объемом от нескольких сот тысяч до десятков миллионов транзисторов, при этом параметры (частота микросхемы) были улучшены в 3 раза, а в некоторых случаях и более.

В работах В.В. Воеводина [14, 15, 16, 17, 19] рассматриваются различные графовые модели применительно к параллельным вычислениям. Особо надо выделить работу [14], которая содержит описания моделей и архитектур для организации параллельных вычислений. Также представлены несколько классификаций архитектур, использующих параллельные вычисления. Дан широкий обзор графовых моделей программ: граф информационных связей, история вычислений, решетчатый граф и т.д. Описываются возможные применения графовых моделей в аппаратно-программных комплексах. Рассматриваются эквивалентные преобразования программ и их применение в параллельных вычислениях. Описываются способы хранения графов, среди которых стоит особенно выделить хранение решетчатого графа в виде кусочно-аффинных функций.

Целью диссертации является разработка методов автоматизации отображения программ на конвейерные и многоконвейерные вычислительные архитектуры.

К достижению этой цели приводит решение следующих **задач**:

1) автоматическое определение характеристик конвейера, таких как интервал инициализации итераций, буферные задержки, обратные дуги (см. 2.6);
2) автоматическое составление расписания работы каждого конвейера в рамках многоконвейерной модели вычислений (см. 2.6);
3) разработка алгоритмов синхронизации конвейера в рамках многоконвейерной модели вычислений (см. 3.3-3.4);
4) разработка методов и алгоритмов эффективного автоматического отображения программ на многоконвейерную модель вычислений (см. 3.3-3.5);
5) разработка программных средств автоматически выполняющих расчеты характеристик конвейера, в частности, задержек в стартах конвейеров, позволяющих синхронизировать эти конвейеры.

Методы исследований

В процессе решения рассматриваемых задач использовались методы теории преобразований программ, теории графов, линейной алгебры. При создании программного обеспечения использовался объектно-ориентированный подход к программированию.

Научная новизна

1. Впервые рассмотрена задача отображения многомерных гнезд циклов, как тесных, так и не тесных, на многоконвейерную архитектуру с использованием информации о зависимостях, описанной с помощью решетчатых графов. Кроме того, этот подход видится наиболее общим, так как решетчатые графы более полно описывают зависимости в многомерных циклах, по сравнению с другими графовыми моделями.

2. Предложен упрощенный алгоритм отображения на многоконвейерную архитектуру двумерных гнезд циклов. Указанный алгоритм применим к меньшему классу программ, чем общий, но при этом решает рассматриваемую задачу эффективнее. Этот алгоритм отображения, в частности, применим к программам, решающим задачи цифровой фильтрации сигналов, рассмотренные в работе М.С. Яджака [75], и позволяет вычислять характеристики конвейера автоматически.

3. Предложен алгоритм линеаризации выражений на этапе компиляции, в отличие от известных, учитывающий типы данных.

Практическая значимость

Описанные алгоритмы и методы могут быть использованы при разработке распараллеливающих компиляторов.

Эти алгоритмы уже используются в реализации проекта C2HDL [87], разрабатываемого группой ОРС. Этот конвертер предполагает по программе на языке С генерацию семейства алгоритмически эквивалентных VHDL-описаний микросхем с разными параметрами синтеза. Итоговое решение по выбору наиболее качественного описания должен осуществлять специалист-схемотехник, при этом конвертер будет выдавать семейство описаний, в котором будут исключены заведомо неэффективные варианты.

Также модель многоконвейерных вычислений может использоваться при создании прототипов специализированных конвейерных и многоконвейерных вычислителей.

В процессе программной реализации алгоритмов, описанных в диссертации, в рамках проектов ОРС (Открытая распараллеливающая система) и ДВОР (Диалоговый высокоуровневый оптимизирующий распараллеливатель), автору потребовалось реализовать несколько вспомогательных подпроектов. Так, например, была реализована библиотека линейных выражений, включающая алгоритм линеаризации. Эта библиотека активно используется при построении графа информационных связей, генерации MPI и других подпроектов ДВОР.

Система тестирования, описываемая в диссертации, использовалась для тестирования преобразований в OPC, а также при тестировании высокопроизводительного программного комплекса HPC-NASIS.

Внедрение результатов работы

Результаты работы реализованы в виде программных модулей в рамках проектов OPC и ДВОР.

Программа построения и визуализации графа вычислений и его характеристик используются в учебном процессе.

Достоверность полученных результатов подтверждается строгими математическими формулировками и доказательствами, программно реализованными методами и вычислительными экспериментами.

Личный вклад

Постановка задачи создания многоконвейерной модели вычислений, расчета параметров конвейеризации, а также отображения многомерных гнезд циклов на многоконвейерную архитектуру с помощью информации о зависимостях описанной решетчатыми графами принадлежит Б.Я. Штейнбергу. Все теоретические результаты получены автором диссертации лично. Программная реализация графа вычислений и алгоритмов, описанных в работе, также выполнена лично и велась в рамках проектов OPC и ДВОР, и была бы невозможна без соответствующего вклада разработчиков группы OPC, выполнявших смежные проекты, особенно В.В. Петренко и Р.И. Морылева.

Все рисунки в данной диссертации выполнены с помощью программных средств, в разработке которых автор принимал дейстельное участие. Часть из них являются экранными формами, созданными с помощью OPSDemo 3, OPSDemo 5 — программных продуктов сделанных в рамках проекта OPC. Другие рисунки — с помощью макросов, написанных на VBA под MS Word лично автором.

В совместных работах [26, 56, 57, 59, 60, 62, 63] личным вкладом автора является разработка многоконвейерной модели вычислений, алгоритмов построения и анализа графа вычислений. В совместных работах [11, 62, 63, 64] также описывается и используется линеаризация выражений, выполненная лично автором.

В совместных работах [26, 61] описывается тестирующая система, в разработке которой автор принимал участие вместе с соавторами. В частности, формат представления файлов с данными разработан лично автором.

Гранты, поддерживавшие исследования диссертации.

- НИОКР «Разработка системной поддержки высокопроизводительного программного комплекса для квантово-механических расчетов и моделирования наноразмерных атомно-молекулярных систем и комплексов», госконтракт 02.524.11.4005, шифр: 2008-04-2.4-15-02-003.
- ФЦП «Научные и научно-педагогические кадры инновационной России» по лоту «Проведение научных исследований коллективами научно-образовательных центров в области информатики» по теме: «Диалоговый высокоуровневый оптимизирующий распараллеливатель программ и его приложения», госконтракт 02.740.11.0208 от 07 июля 2009 г., шифр: 2009-1.1-113-050-043.
- ФЦП «Научные и научно-педагогические кадры инновационной России» по лоту «Проведение научных исследований коллективами научно-образовательных центров в области: геномных и постгеномных технологий создания лекарственных средств; клеточных технологий; биоинженерии; биоинформационных технологий» по теме: «Создание биоинформационной технологии поиска взаимосвязанных сценариев организации в геномах животных и человека некодирующей ДНК и кодирующей белок ДНК», госконтракт 14.740.11.0006 от 01 сентября 2010 г.
- Грант Российско-Белорусского сотрудничества «ТРИАДА», НИЦЭВТ, 2005г.
- Внутренний грант Южного федерального университета «Учебно-научная лаборатория оптимизации и распараллеливания программ», 2007г.
- ФЦП «Научные и научно-педагогические кадры инновационной России» на 2009–2013 гг., в рамках реализации мероприятия № 1.4 «Развитие внутрироссийской мобильности научных и научно-педагогических кадров путем выполнения научных исследований молодыми учеными и преподавателями в научно-образовательных центрах», госконтракт 14.740.11.0442 от 30 сентября 2010 г.

Апробация работы

Изложенные в диссертации результаты обсуждались на многих научно-технических конференциях:

- Интеллектуальные и многопроцессорные системы ИМС-2003, международная научно-техническая конференция (Россия, пос. Дивноморское, 2003 г)
- VI международной конференции памяти академика А.П. Ершова “Перспективы систем информатики”, рабочий семинар “Наукоемкое программное обеспечение” (Новосибирск, 2006 г.)
- международная конференция РАСО-2010 (Москва, 2010 г.)
- международная конференция РАСО-2006 (Москва, 2006 г.)
- международная конференция IEEE East & West Design and Test (Москва, 2009 г.)
- международная конференция «Математика. Экономика. Образование» (Ростов н/Д, 2005 г.)
- международная суперкомпьютерная конференция “Научный сервис в сети Интернет: суперкомпьютерные центры и задачи” (Новороссийск, 2010 г.)
- всероссийская научно-техническая конференция «Параллельные вычисления в задачах математической физики» (Ростов н/Д, 2004 г.)
- всероссийская научная конференция «Научный сервис в сети Интернет: технологии распределенных вычислений» (Новороссийск, 2005 г.).
- региональная научно-методическая конференция «Современные информационные технологии в образовании: Южный федеральный округ», Ростов-на-Дону, 12-15 мая 2004 г.

Изложенные в диссертации результаты обсуждались на научно-технических семинарах:

- семинар группы ОРС, мехмат, ЮФУ, г. Ростов-на-Дону, 2003–2010 гг.
- семинар факультета компьютерной инженерии и управления, ХНУРЭ, г. Харьков, Украина (руководитель проф. Хаханов В.И.). 2008 г.
- семинар научно-технического совета «НКБ ВС», г. Таганрог (руководитель директор ОАО «НКБ ВС», к.т.н. Итенберг И.И.). 2008 г.

Основные положения, выносимые на защиту.

1. Алгоритмы автоматической синхронизации взаимодействия конвейеров в многоконвейерной вычислительной системе для случая многомерного гнезда циклов.
2. Алгоритмы и программная реализация автоматической синхронизации взаимодействия конвейеров для случая тесного двумерного гнезда циклов.
3. Модификация и обоснование формулы расчета интервала инициализации итераций.

4. Алгоритмы и программная реализация автоматического расчета интервала инициализации итераций, буферов задержек, выявления обратных дуг, составления расписания в рамках рассматриваемой модели вычислений.
5. Алгоритмы и программная реализация линеаризации выражений для вычисления информационных зависимостей и оптимизации программ, в том числе для конвейеризации циклов.

Структура диссертации.

Диссертация состоит из введения, четырёх глав и заключения.

В первой главе диссертации приводятся общие сведения из теории графов, необходимые для формулирования теорем и результатов работы. Также сообщаются сведения из теории преобразования программ. Рассматриваются понятия информационной зависимости, графа информационных связей, решетчатого графа и его подвидов (элементарного решетчатого графа, минимального решетчатого графа и т.д.). Описываются способы представления решетчатых графов в памяти компьютера.

В последнем параграфе первой главы приводится понятие графа вычислений, который в некоторых источниках называется редуцированным графом алгоритма. Этот граф является промежуточным представлением программы для многоконвейерной модели вычислений.

Отображение графа вычислений на конвейерную архитектуру может быть построено следующим образом: каждой вершине графа вычислений ставится в соответствие процессорный элемент (ПЭ), на котором будет выполняться эта операция, а каждой дуге – канал связи, по которому будут передаваться данные от одного ПЭ к другому.

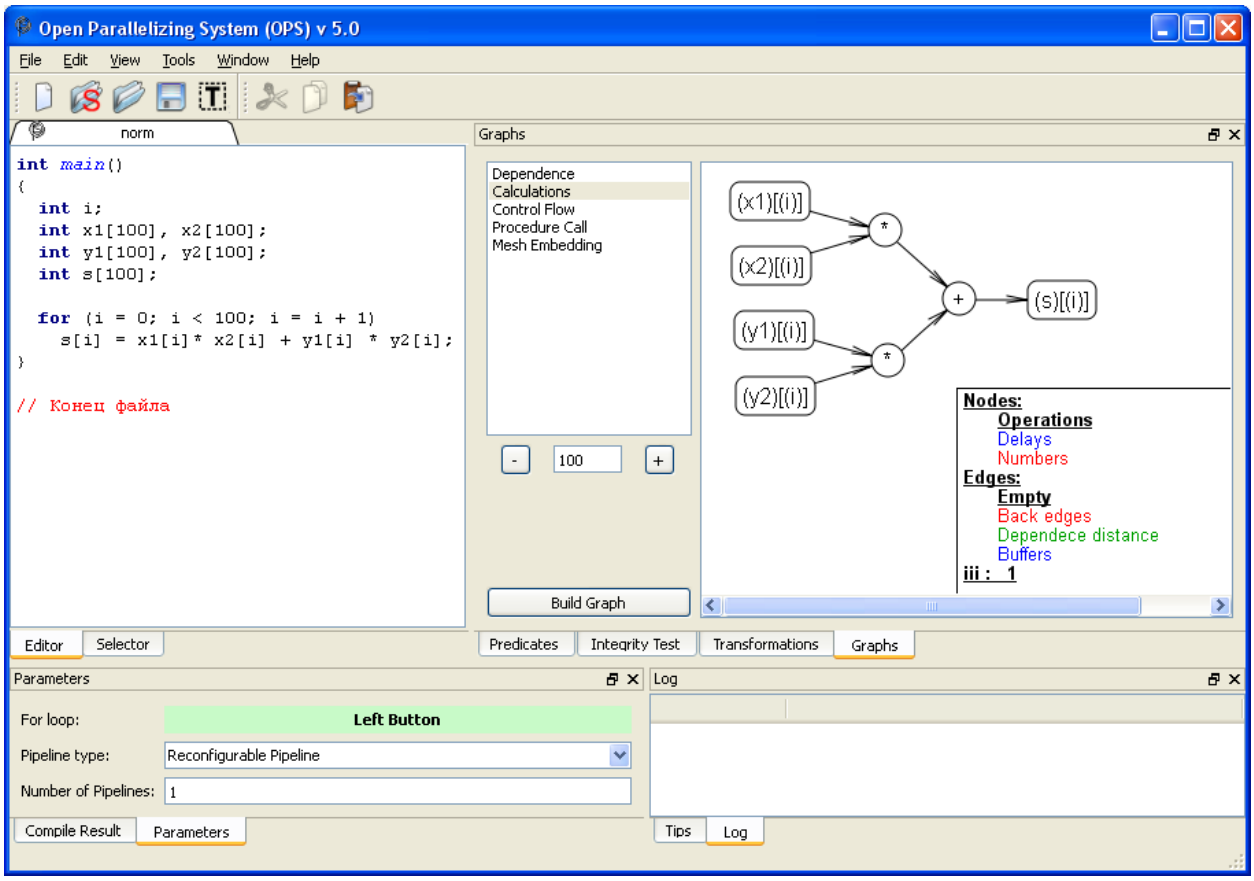

Рис. 1. Граф вычислений в ОРС 5 (ДВОР)

Во второй главе рассматриваются конвейерные вычисления и отображения программ на конвейерные вычислительные устройства. Приводятся алгоритмы отображения одномерных циклов на конвейер, алгоритмы расчета параметров конвейера (интервала инициализации итераций, буферов задержек) и вспомогательные алгоритмы.

Основным результатом этой главы является алгоритм автоматической синхронизации конвейера, включающий расчет интервала инициализации итераций и расчет буферов задержек. Задача вычисления минимального интервала инициализации итераций в общем случае является NP-полной (см. [47]), и точный алгоритм её решения состоит в переборе всех циклов на графе вычислений. Известная формула расчета интервала инициализации итераций (см. [47]) выглядит следующим образом:

$$iii = \max_{c} \lceil d(c) / p(c) \rceil, \qquad (1)$$

где максимум берется по всем c — циклам графа вычислений, $d(c)$ — сумма задержек операций по вершинам цикла c, $p(c)$ — сумма расстояний зависимости по дугам зависимости входящим в цикл c, а символом $\lceil a \rceil$ обозначено округление с избытком числа a.

В диссертации доказывается следующая теорема.

Теорема: Всегда существует элементарный цикл, на котором достигается максимум выражения (1).

Таким образом, предложенный в диссертации алгоритм уменьшает объем вычислений для определения минимального *iii*, т.к. для вычисления значения выражения (1) требуется перебрать не все циклы на графе вычислений, а только элементарные. Также приводятся обоснования корректности этих алгоритмов.

В третьей главе рассматривается модель многоконвейерных вычислений. Вводится понятие задержки между стартами конвейеров. Обсуждается влияние информационных зависимостей на синхронизацию и составление расписания. Для анализа зависимостей используются решетчатые графы.

Вводятся два понятия: цепочка вхождений и функция зависимости цепочки. С помощью этих понятий формулируется и доказывается условие отображения гнезда циклов на многоконвейерную модель вычислений.

Далее в третьей главе рассматривается практически важный случай двумерного гнезда циклов и выводятся упрощенные формулы для вычисления задержки между стартами конвейеров. Обсуждаются вопросы составления расписания для организации вычислений гнезда циклов.

В четвертой главе формулируются вспомогательные и дополнительные результаты, которые были получены в процессе работы над диссертацией: анализ и линеаризация выражений, система тестирования, конвертера с языка C в VHDL.

Линеаризацией выражения относительно набора переменных a_i называется преобразование произвольного арифметического выражения к виду: $e_1*a_1+e_2*a_2+\ldots+e_n*a_n+e_n+1$, где e_i – выражения, a_i – переменные, входящие в исходное выражение, и при этом ни одна из них не входит ни в одно из выражений e_i. Таким образом, программная реализация должна определить по входному выражению и набору переменных a_i, коэффициенты его линейного представления e_i. Также в первом параграфе рассматривается применение линеаризации к различным подпроектам ДВОР. Описывается процесс развития библиотеки, реализующей линеаризацию в ДВОР, с целью соответствия как можно большему покрытию языка C.

Вторым вспомогательным результатом является система тестирования. В втором параграфе, посвященном системе тестирования, разъясняются принципы тестирования, определяются компоненты системы, в частности интерфейс, а также специально разработанный формат описания файлов с входными и выходными данными.

Разработанная автором библиотека позволяет по описаниям файлов входных и выходных данных генерировать случайный файл с данными, проверять, подходит ли конкретный файл с данными под указанное описание, и сравнивать два файла на эквивалентность данных, в нем содержащихся. Используя эти возможности, тестирующая система может осуществлять следующие методики тестирования:

- тестирование на случайном наборе входных данных,
- тестирование на предопределенном наборе входных данных с эталонными результатами,
- тестирование параллельной версии программы на эквивалентность последовательному эталону,
- тестирование масштабируемости параллельной программы,
- тестирование преобразований программ, в частности, входящих в проект ДВОР.

В последнем параграфе четвертой главы приводится информация о проекте экспериментального конвертера с языка C в VHDL. Описывается поэтапное преобразование исходной программы на языке C в описание схемы на VHDL. Основным преимуществом перед всеми существующими на сегодняшний день аналогичными программными средствами является то, что по программе на языке C будет сгенерировано целое семейство описаний схемы на VHDL. Причем эти описания будут получены с помощью оптимизирующих высокоуровневых преобразований исходной программы, ориентированных на распараллеливание. А далее будут выбираться различные, подходящие для конкретной ПЛИС, реализации стандартных функций. Таким образом, инженер-схемотехник получит возможность выбора между различными вариантами автоматически сгенерированных описаний. У этого конвертера промежуточным представлением программ является граф вычислений, который рассматривается во второй главе диссертации.

1. Теория графов и графовые представления программ

1.1. Необходимые понятия из теории графов

В настоящей диссертации активно используются графовые представления программ. Поэтому приведем необходимые понятия из теории графов для полноты изложения.

В рамках этой диссертации под *графом* будем понимать ориентированный граф, допускающий кратные дуги и петли.

Напомним, определение операции *объединение графов* (см. [48]).

Определение 1. Объединением графов $G_1(X_1, U_1)$ и $G_2(X_2, U_2)$ называется граф $G_3(X_3, U_3)$ такой, что $X_3 = X_1 \cup X_2$, $U_3 = U_1 \cup U_2$.

При построении *остовного дерева* (см. [5, 48]) ориентированного графа методом *обхода в глубину* (см. [5]), множество дуг исходного графа можно разбить на четыре непересекающихся подмножества.

Определение 2. Дуги графа, принадлежащие остовному дереву, будем называть дугами дерева.

Определение 3. Дуги графа, для которых существует путь, ведущий от начала дуги, оканчивающийся в конце дуги и содержащий только дуги дерева, будем называть прямыми дугами.

Определение 4: Дуги графа, для которых существует путь, ведущий от конца дуги, оканчивающийся в начале дуги и содержащий только дуги дерева, будем называть обратными дугами.

Определение 5. Дуги графа, начала и концы которых принадлежат разным ветвям остовного дерева, будем называть поперечными дугами.

На следующем рисунке изображены дуги всех четырех типов для некоторого графа.

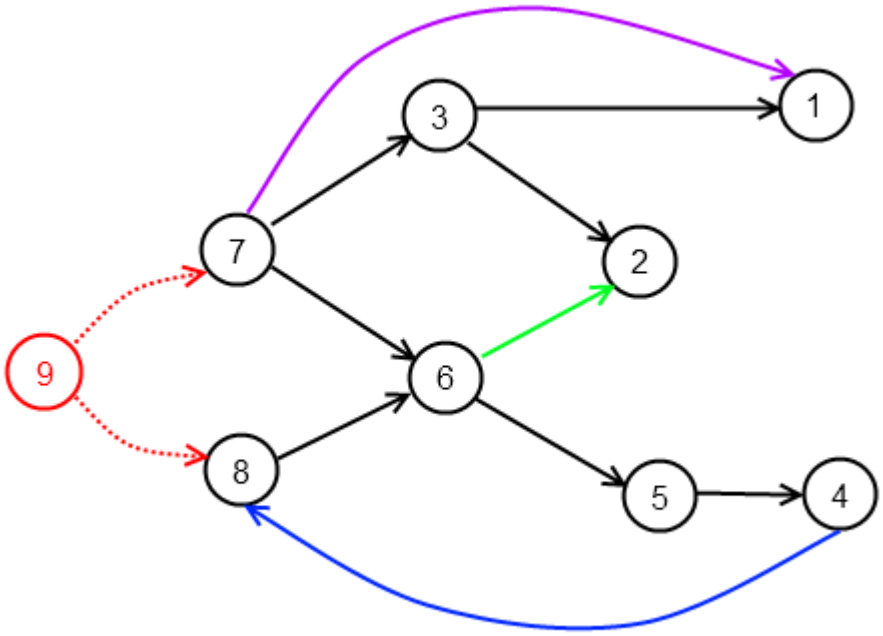

Рис. 2. Типы дуг графа: древесные – черный, прямые – фиолетовый, обратные – синий, поперечные – зеленый. Красная вершина – вспомогательный источник

Красным цветом изображена дополнительная вершина и дополнительные дуги. Если предположить, что поиск в глубину обходил вершины в порядке определенном нумерацией вершин на рисунке, то разбиение дуг на типы соответствует цветам: древесные – черный, прямые – фиолетовый, обратные – синий, поперечные – зеленый.

Введем в рассмотрение новую нумерацию вершин графа. Обходом в глубину построим остовное дерево и одновременно занумеруем вершины графа следующим образом: если из текущей вершины нельзя попасть ни в одну непомеченную вершину и необходимо сделать шаг возврата, то текущая вершина получает очередной номер. Будем называть эту нумерацию *dfs-нумерацией вершин графа*. Пример такой нумерации можно увидеть на рис. 2.

Сформулируем и докажем лемму, которая будет необходима для обоснования корректности алгоритмов, описанных далее.

Лемма 1. Если все дуги классифицировать в соответствии с построением остовного дерева методом обхода в глубину, то любой цикл в ориентированном связном графе содержит хотя бы одну обратную дугу.

Доказательство. Пусть дан граф $G = (X, U)$. Добавим вспомогательную вершину с дугами, ведущими ко всем источникам графа G. Обозначим $\omega(x)$ номер вершины x, в соответствии с указанной нумерацией.

Таким образом, вершина будет иметь тем больший номер, чем позже она будет просмотрена вместе со всеми вершинами, принадлежащими поддереву остовного дерева с корнем в данной вершине. Пусть $u = (x, y)$ дуга в графе G. Очевидно, что если u древесная, прямая или поперечная дуга, то $\omega(x) > \omega(y)$, т.е. номер начала больше номера конца дуги. Если u обратная дуга, то $\omega(x) < \omega(y)$, номер начала меньше номера конца.

Рассмотрим произвольный цикл $(x_1, u_1, x_2, u_2, \ldots, u_n, x_1)$, где x_i – вершины, а u_i – дуги соединяющие x_i и x_{i-1}. Предположим, что этот цикл не содержит обратных дуг. Тогда все дуги древесные, прямые и поперечные. В соответствии со сказанным в предыдущем абзаце, $\omega(x_{i-1}) > \omega(x_i)$, для любого $i \in [2, n]$ и $\omega(x_n) > \omega(x_1)$ больше номера вершины x_1. Но тогда, очевидно, $\omega(x_1) > \omega(x_2) > \ldots > \omega(x_n) > \omega(x_1)$. Полученное противоречие доказывает, что предположение об отсутствии обратных дуг в цикле неверно.

Определение 6. Правильной нумерацией ориентированного ациклического графа называется такая нумерация вершин этого графа, при которой все дуги ведут из вершин с меньшими номерами в вершины с большими номерами.

Определение 7. Будем называть цикл простым, если все дуги этого цикла, встречаются в нем ровно один раз.

Определение 8. Будем называть цикл элементарным, если все вершины этого цикла встречаются в нем ровно один раз.

Очевидно, любой элементарный цикл является простым, но не наоборот. Также очевидно, что для любого графа, после удаления из него прямых и поперечных дуг, каждый элементарный цикл содержит ровно одну обратную дугу и каждая обратная дуга входит ровно в один элементарный цикл.

1.2. Информационные зависимости в программах

В этом параграфе будут изложены основы теории информационных зависимостей [14, 76]. Будут введены такие понятия, как вхождение переменной и информационная зависимость. Также будут рассмотрены типы информационных зависимостей.

Определение 9. Вхождением переменной будем называть пару: переменная и место в тексте программы, где происходит обращение к этой переменной.

Определение 10. Вхождение переменной называется генератором, если в этом месте программы в ячейку памяти, соответствующую этой переменной, происходит запись. В противном случае (чтение данного) вхождение называется использованием.

Определение 11. Два вхождения называются зависимыми, если они обращаются к одной и той же ячейке памяти. При этом говорят, что вхождение *v* зависит от вхождения *u*, если вхождение *v* обращается к некоторой ячейке памяти позже, чем вхождение *u*.

Пример 1

```
a = 10;
b = 100;
c = a + b;
```

В этом фрагменте программы присутствует два вхождения переменной *a*, два вхождения переменной *b* и одно переменной *c*. Второе (по тексту программы) вхождение переменной *a*, зависит от первого (по тексту программы) вхождения переменной *a*. Для вхождений переменной *b* зависимость аналогична. Вхождение переменной *c* не связано зависимостями ни с одним другим вхождением в этом фрагменте программы.

Конец примера.

Любую зависимость между двумя вхождениями можно отнести к одному из четырех типов.

Определение 12. Пусть вхождение *u* зависит от вхождения *v*. Зависимость между *u* и *v* называется истинной (потоковой), если *u* – использование, а *v* – генератор. В английском языке: true (flow) dependence.

Определение 13. Пусть вхождение *u* зависит от вхождения *v*. Зависимость между *u* и *v* называется антизависимостью, если *u* – генератор, а *v* – использование. В английском языке: antidependence.

Пример 2

```
a = 10;
b = a + c;
c = 20;
```

В этом фрагменте программы присутствуют два вхождения переменной *a*, два вхождения переменной *b* и одно переменной *c*. Вхождения переменной *a* связаны истинной зависимостью. Вхождения переменной *b* связаны антизависимостью. На следующем рисунке этот пример проиллюстрирован с помощью системы OPSDemo 5 [89].

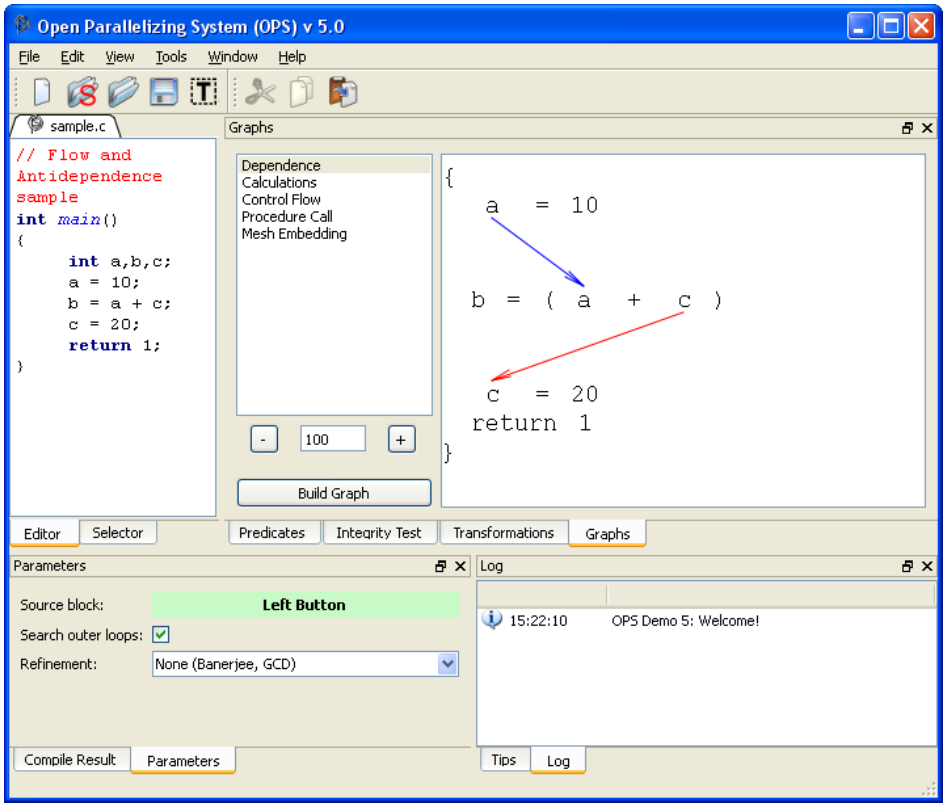

Рис. 3. Скриншот системы OPSDemo 5, демонстрирующий зависимости: синие дуги – истинная зависимость, красные – антизависимость

Конец примера.

Определение 14. Пусть вхождение *u* зависит от вхождения *v*. Зависимость между *u* и *v* называется выходной, если *u* – генератор и *v* – генератор. В английском языке: output dependence.

Определение 15. Пусть вхождение *u* зависит от вхождения *v*. Зависимость между *u* и *v* называется входной, если *u* – использование и *v* – использование. В английском языке: input dependence.

Пример 3

```
a = b + 10;
a = b + 20;
```

В этом фрагменте программы присутствуют два вхождения переменной *a* и два вхождения переменной *b*. Вхождения переменной *a* связаны выходной зависимостью. Вхождения переменной *b* связанны входной зависимостью. На следующем рисунке этот пример проиллюстрирован с помощью системы OPSDemo 5 [89].

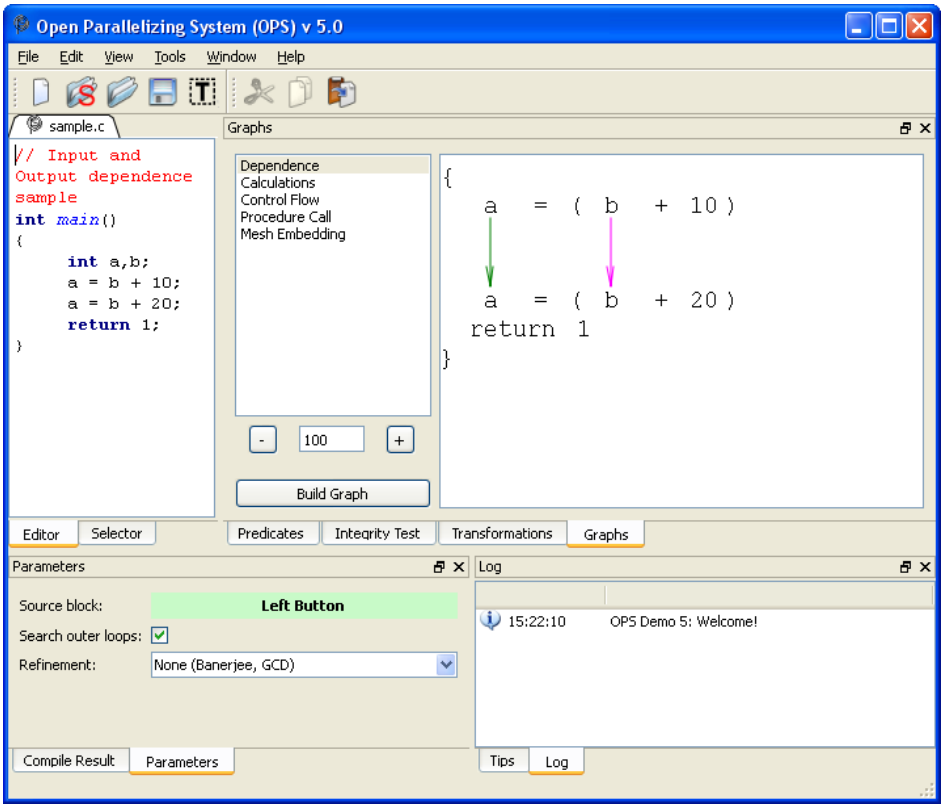

Рис. 4. Скриншот системы OPSDemo 5, демонстрирующий граф зависимости: зеленые дуги – выходная зависимость, фиолетовые – входная

Конец примера.

Есть еще одно понятие, которое важно для понимания движения потоков данных в программе.

Определение 16. Пусть вхождение u зависит от вхождения v_1 и от вхождения v_2, причем обе эти зависимости истинные. Зависимость между u и v_2 называется ложной, если любое данное, записанное генератором v_2, будет перезаписано в ту же ячейку памяти генератором v_1, до того как оно будет использовано вхождением u. В английском языке: false dependence.

Пример 4

```
for (i=0; i<N; ++i)
{
     x[i] = a[i]+b[i];
     y[i] = x[i]*c[i];
     x[i] = a[i]*b[i];
     z[i] = x[i]-c[i];
}
```

Обозначим через v_1 вхождение переменной `x[·]` в первом операторе присваивания, через u_1 вхождение переменной `x[·]` во втором операторе присваивания, через v_2 вхождение переменной `x[·]` в третьем операторе присваивания, через u_2 вхождение переменной `x[·]` в четвертом операторе присваивания.

Вхождения u_1 и u_2 – использования, а вхождения v_1 и v_2 – генераторы. Вхождение u_1 зависит от v_1 и v_2, но истинной зависимостью является только зависимость от v_1. В то же время u_2 зависит от v_1 и v_2, причем в этом случае обе зависимости истинные, но зависимость u_2 от v_1 является ложной.

Конец примера.

Введем в рассмотрение еще одну характеристику информационных зависимостей в программе.

Определение 17. Пусть в программе есть два вхождения u и v массива `x[·]`, причем u зависит от v. Вектором расстояния зависимости будем называть вектор, полученный вычитанием векторов индексных выражений массива вхождения u и v.

Пример 5

Пусть в программе есть вхождение `x[i][j]` и вхождение `x[i+1][j+1]`, причем второе вхождение зависит от первого. Тогда вектор расстояния зависимости будет равен (1, 1).

Конец примера.

1.3. Граф информационных связей

Рассматриваемый в этом параграфе граф используется при контроле эквивалентности преобразований программ, при корректности распараллеливания программ, в оптимизирующих компиляторах и распараллеливающих системах.

Определение 18. Графом информационных связей фрагмента программы (см. [14, 79]) называется граф, в котором множество вершин — это множество вхождений переменных данного фрагмента программы, причем две вершины соединены дугой, направленной от *u* к *v*, если вхождение *v* зависит от вхождения *u*.

Пример 6

```
for (i=0; i<1000; ++i)
     for (0=1; i<1000; ++i)
          for (k=0; i<1000; ++i)
               c[i][j] = c[i][j] + a[i][k]* b[k][j];
```

Граф информационных связей этого фрагмента программы изображен на следующем рисунке.

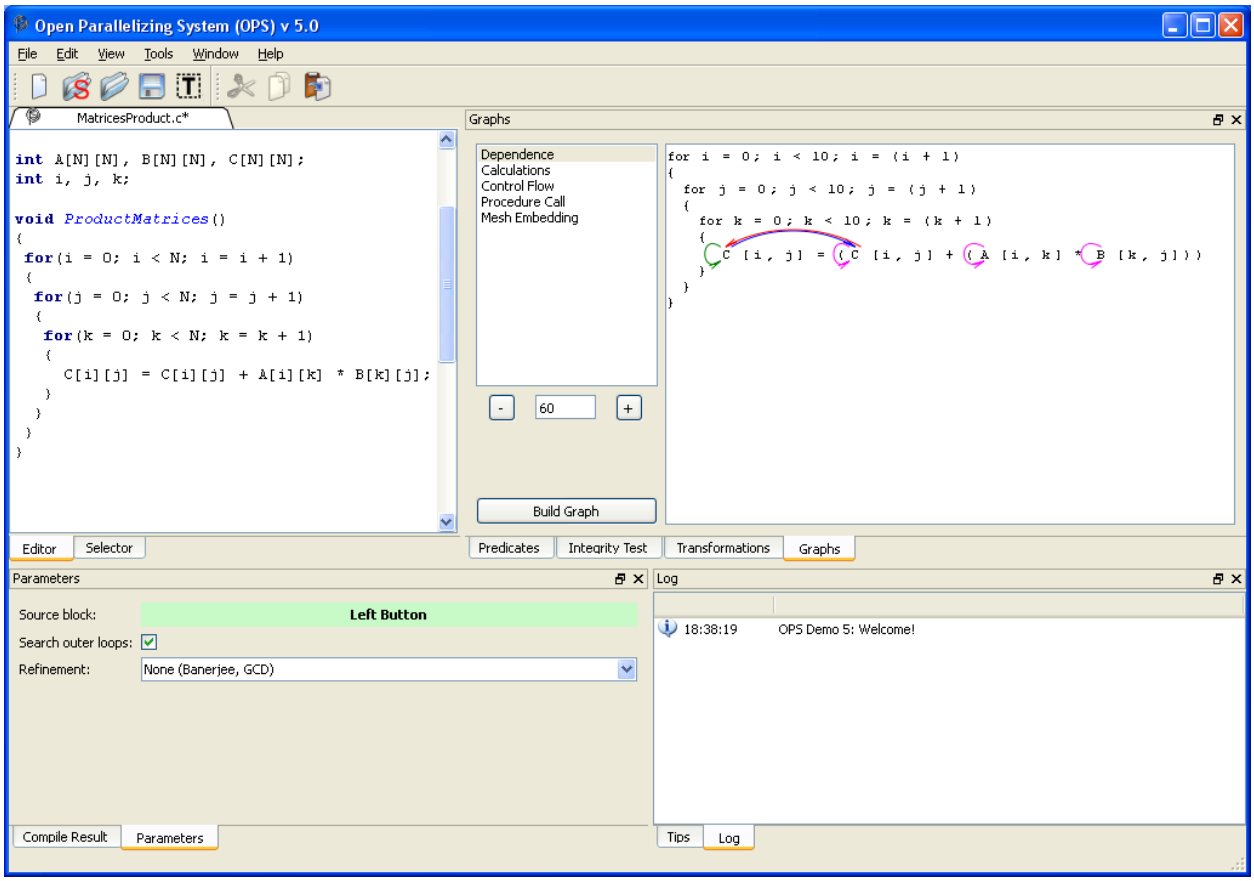

Рис. 5. Скриншот системы OPSDemo 5, демонстрирующий граф информационных связей фрагмента программы

Конец примера.

1.4. Основные понятия теории решетчатых графов

Как было сказано выше, граф информационных связей описывает зависимости между вхождениями переменных. Это позволяет определять возможность параллельного выполнения программных циклов. Но бывают такие гнезда циклов, в которых ни один цикл не распараллеливается, и при этом часть кода может быть выполнена параллельно. Примеры таких гнезд циклов приводятся в работах Л. Лампорта [81, 82], В.А. Вальковского [12, 13], Н.А. Лиходеда [39]. В этих работах неявно используются решетчатые графы. Термин «решетчатый граф» появился в работах В.В. Воеводина [14, 15, 17, 19]. Описание структур данных для эффективного хранения решетчатых графов (в виде функций) появилось независимо в работах В.В. Воеводина [14, 15] и

П. Фотерье [79]. В рамках проекта ОРС [59, 60, 89] была получена программная реализация алгоритмов построения и использования решетчатых графов [71, 73, 74].

В данной работе решетчатые графы используются для расчета задержек между стартами конвейеров при анализе выполнимости гнезда циклов на многоконвейерной вычислительной системе.

1.4.1. Элементарные решетчатые графы

Рассмотрим гнездо из n вложенных друг в друга циклов. Занумеруем операторы циклов в этом гнезде в соответствии с порядком вложенности, начиная с самого внешнего цикла. Обозначим счетчик i-го цикла – I_i, левую границу i-го цикла – L_i, правую границу i-го цикла – R_i.

```
for(I1=L1; I1<=R1; I1++)
{
      Statements(1)
      for(I2=L2; I2<=R2; I2++)
      {
            Statements(2)
            for(I3=L3; I3<=R3; I3++)                    (1)
            …
            for(In=Ln; In<=Rn; In++)
            {
                  Statements(n)
            }
            …
      }
}
```

В общем случае, L_i и R_i линейные функции от счетчиков предыдущих циклов, т.е. от $I_1, I_2, \dots, I_{i-1}$.

Ввиду того, что данная диссертация посвящена не теории решетчатых графов, определения пространства итераций, элементарного решетчатого графа, максимального решетчатого графа и минимального снизу решетчатого графа будут приведены только для частного случая – тесного гнезда циклов, с рассматриваемыми в нем только потоковыми зависимостями. Это сделано с целью, чтобы дать читателю некоторое представление об

этих понятиях, но в то же время не загромождать текст цепочкой определений, которые при чтении данной работы читателю не потребуются, и тем самым не усложнять понимание текста. Более подробно о теории решетчатых графов можно прочесть в [14, 15, 16].

Итак, пусть рассматриваемое гнездо циклов тесное.

Определение 19. Пространством итераций цикла называется множество всех целочисленных векторов $I = (I_1, I_2, \dots, I_n)$, которые удовлетворяют системе неравенств $L_i \le I_i \le R_i$, где $i = 1 \dots n$.

Пример 7

```
for (i=0; i<N; ++i)
     for (j=0; j<M; ++j)
          c[i][j] = a[i][j]+b[i][j];
```

Для цикла реализующего сложение матриц пространством итераций является целочисленный прямоугольник [0, N-1]×[0, M-1].

Конец примера.

Определение 20. Зафиксируем пару вхождений: генератор и использование. Элементарным решетчатым графом называется граф, у которого множество вершин является пространством итераций, а две вершины соединены дугой, если на итерации, соответствующей первой из них генератором записывается данное, которое будет считано использованием на итерации, соответствующей второй из вершин.

Пример 8

```
for (i=1; i<N1; ++i)
     for (j=1; j<N2; ++j)
          x[i+1][j+1] = a[i]+b[j]*x[i][j];
```

Для случая N1 = N2 = 8 элементарный решетчатый граф этого фрагмента программы изображен на рис. 6.

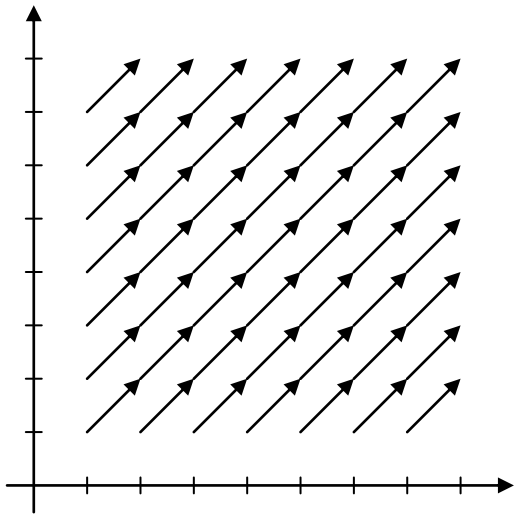

Рис. 6. Элементарный решетчатый граф для вхождений переменной x[·][·]

Конец примера.

Вспомним, что каждая дуга графа информационных связей определяет пару зависимых вхождений. Следовательно, для пары вхождений, которые на графе информационных связей не соединены дугой, решетчатый граф не будет содержать дуг. Таким образом, каждой дуге графа информационных связей соответствует пара вхождений, а каждой паре вхождений — решетчатый граф, т.е. можно говорить о построении решетчатых графов по дугам графа информационных связей.

Доказано (см. [14]), что элементарный решетчатый граф можно представить в виде семейства аффинных (точнее, квазиаффинных [73]) функций, определенных на выпуклых многогранниках. Это семейство называется семейством покрывающих функций. Для этого необходимо множество точек пространства итераций, в которые входят концы дуг решетчатого графа, разбить на выпуклые подмножества $\{D_\alpha\}$, $1 \le \alpha \le p$, на каждом из которых будет определена аффинная функция F_α. Эти аффинные функции ставят в соответствие концам дуг решетчатого графа начала дуг решетчатого графа.

В программных системах OPSDemo3 и ТПП, разработанных группой ОРС, названные функции строятся автоматически.

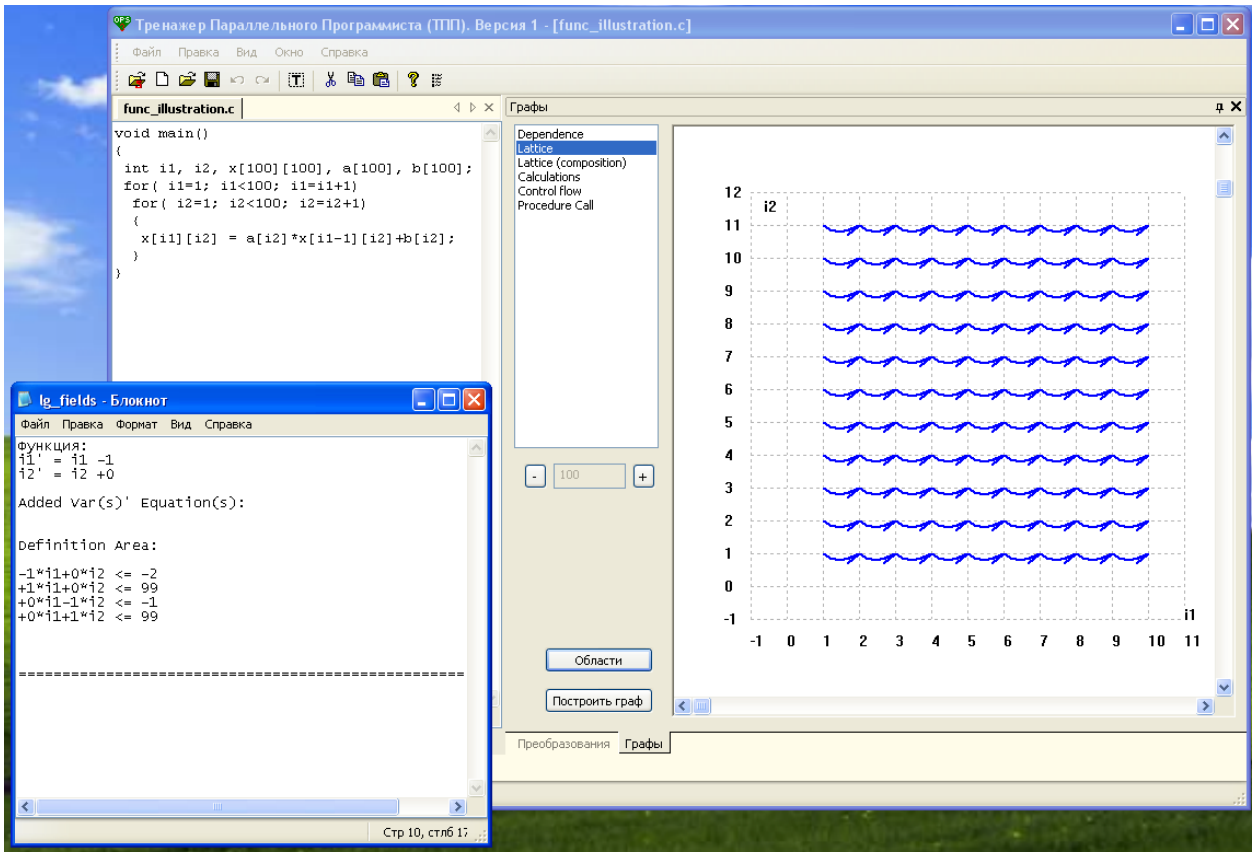

Рис. 7. Автоматическое разбиение пространства итераций на выпуклые подобласти и построение функций аффинных функций, описывающих решетчатый граф

1.4.2. Максимальный и минимальный решетчатые графы

Определение 21. Максимальным решетчатым графом называется граф, у которого множество вершин является пространством итераций, а множество дуг является объединением множеств дуг всех элементарных решетчатых графов, построенных по всем истинным зависимостям этого цикла.

Пример 9

```
for(i=1; i<=N1; ++i)
      for (j=1; j<=N2; ++j)
      {
            y[i][j-1] = a[i]+y[i][j];
            x[i-1][j] = b[i]*x[i][j];
      }
```

В данном фрагменте программы присутствуют две пары вхождений, порождающих истинные зависимости. Имея для каждой из пар элементарный решетчатый граф, можно построить максимальный решетчатый граф (см. рис. 8).

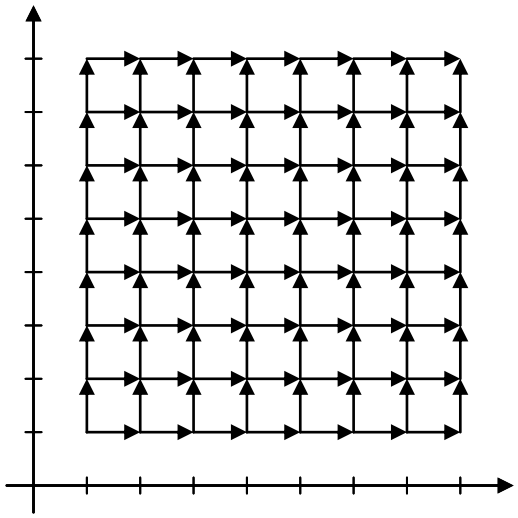

Рис. 8. Максимальный решетчатый граф для фрагмента программы из примера 9

Конец примера.

Построим семейство множеств $\{U_i\}$. Каждое множество в этом семействе, представляет собой множество дуг, входящих в фиксированную вершину максимального решетчатого графа, обладающих тем свойством, что все они порождены парой вхождений одной и той же переменной, при этом каждая дуга максимального решетчатого графа должна находиться хотя бы в одном из множеств семейства $\{U_i\}$. Понятно, что если дуга максимального решетчатого графа порождается p переменными, то она будет находиться в p различных множествах семейства $\{U_i\}$.

Для каждого множества U_i построим множество W_i. Множество W_i состоит из одной дуги, у которой начало лексикографически больше, чем у всех остальных из U_i. Обозначим через W объединение всех множеств семейства $\{W_i\}$.

Определение 22. Минимальным снизу решетчатым графом называется граф, у которого множество вершин является пространством итераций, а множество дуг совпадает с описанным выше множеством W.

Далее под решетчатым графом будем понимать минимальный снизу решетчатый граф.

Пример 10

```
for(i=1; i<=N1; i++)
      for(j=1; j<=N2; j++)
      {
            x[i-1,j-1] = a[j]*x[i-1,j];
            x[i-1,j] = b[i]+x[i,j];
      }
```

Обозначим через v_1 генератор переменной `x[·]` в первом операторе присваивания, через u_1 использование переменной `x[·]` в первом операторе присваивания, через v_2 генератор переменной `x[·]` во втором операторе присваивания, через u_2 использование переменной `x[·]` во втором операторе присваивания.

В данном фрагменте программы присутствуют три пары вхождений порождающих истинные зависимости: v_1 и u_1, v_1 и u_2, v_2 и u_2. Построим элементарные решетчатые графы для этих пар.

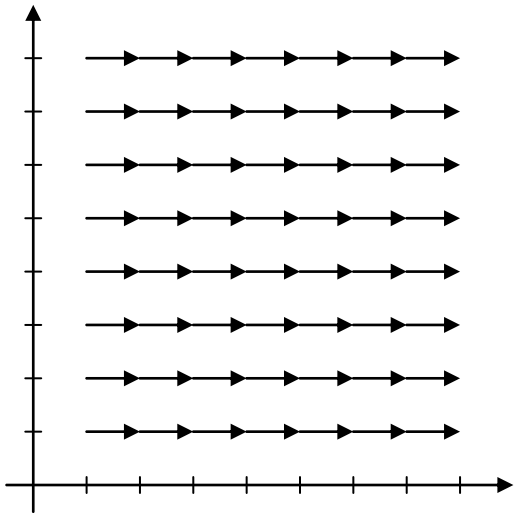

Рис. 9. Элементарный решетчатый граф для пары v_1 и u_1

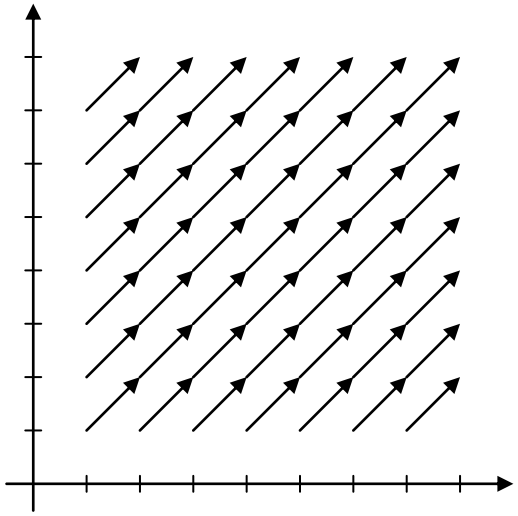

Рис. 10. Элементарный решетчатый граф для пары v_1 и u_2

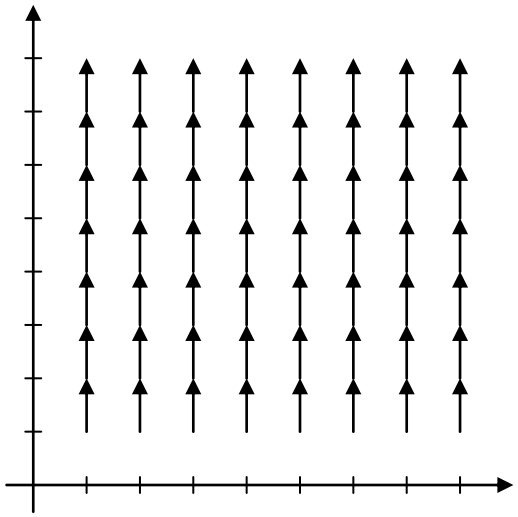

Рис. 11. Элементарный решетчатый граф для пары v_2 и u_2

Вершины максимального решетчатого графа, который получается объединением всех трех графов, можно разбить на четыре подмножества, по количеству входящих в них дуг: $M_1 = \{i{=}1 \ \& \ j{=}1\}$, $M_2 = \{i{=}1 \ \& \ j{>}1\}$, $M_3 = \{i{>}1 \ \& \ j{=}1\}$, $M_4 = \{i{>}1 \ \& \ j{>}1\}$.

Пусть $U_{N2*(i-1)+j}$ – множество дуг, входящих в вершину (i, j). Таким образом, множество U_1 – пустое, множества $U_{N2*(i-1)+j}$, $(i, j) \in M_2$ — состоят из одной дуги, множества $U_{N2*(i-1)+j}$, $(i, j) \in M_3$ — состоят из двух дуг, множества $U_{N2*(i-1)+j}$, $(i, j) \in M_4$ — состоят из трех дуг.

Проанализировав полученное семейство множеств $\{U_i\}$, можно сделать вывод, что семейство множеств $\{W_i\}$, разбивается на три группы: множество W_1 – пустое, множества $W_{N2*(i-1)+j}$, $(i, j) \in M_2$ – содержат дуги вида $(i-1, j) \to (i, j)$, множества $W_{N2*(i-1)+j}$, $(i, j) \in M_3 \cup M_4$ – содержат дуги вида $(i, j-1) \to (i, j)$. Полученный минимальный снизу решетчатый граф изображен на рис. 12.

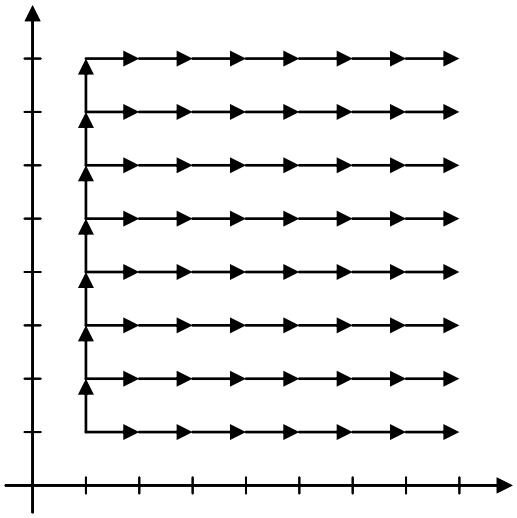

Рис. 12. Минимальный снизу решетчатый граф для фрагмента программы

Конец примера.

1.5. Граф вычислений.

В данном параграфе будет введено понятие графа вычислений, которое является основой для организации отображения на конвейерные архитектуры в рамках данной диссертации.

Обозначим через *OP* – множество операций (чтения данных, записи данных, арифметические операции) некоторого фрагмента программы. Обозначим через *OP(S)* – множество операций (чтения данных, записи данных, арифметические операции) некоторого оператора *S*.

Пусть $o \in OP$ – операция. Тогда для *o* можно определить, какому оператору присваивания она принадлежит. Можно определить *опорное пространство* (см. [14], [18]), порожденное этим оператором. В случае тесного гнезда циклов опорное пространство оператора совпадает с *пространством итераций* гнезда циклов, в котором он находится. Напомним, что в целях улучшения ясности изложения в этой главе рассматриваются только тесные гнезда циклов, хотя полученные результаты легко обобщаются на случай нетесных гнезд циклов.

Обозначим через *PI*(*o*) опорное пространство операции *o*.

Определение 23. Графом оператора S будем называть ориентированный граф, множество вершин которого представляют собой множество $V = \{ (o, I), o \in OP(S), I \in PI(o) \}$, а дуги соединяют такие вершины (x, I) и (y, I), что результат операции x, выполненной на итерации I, является аргументом операции y, выполняемой на итерации I.

Пример 11

```
for(k=1; k<P+1; ++k)
      C[1,2] = C[1,2]+A[1,k]*B[k,2]
```

Граф единственного оператора в этом цикле изображен на рисунке 13.

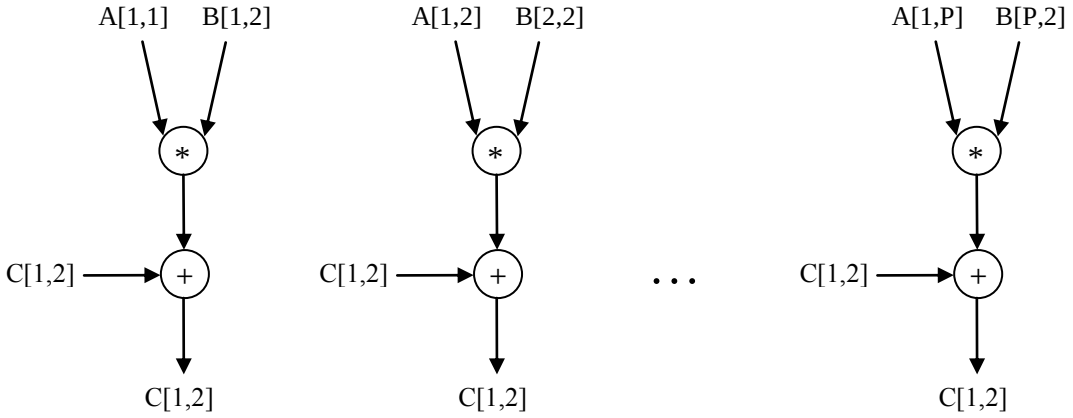

Рис. 13. Граф оператора для алгоритма вычисления одного элемента C[1,2] произведения матриц

Конец примера.

Рассмотрим пару операторов S_1 и S_2 (не обязательно различных) и пару вхождений $u \in S_1$ и $v \in S_2$, из которых v – генератор, u – использование, и они связаны истинной зависимостью. Построим граф, являющийся объединением графов операторов S_1 и S_2. Обозначим его G.

Далее опишем процесс замены одних дуг другими в соответствии с информационными зависимостями, обнаруженными во фрагменте программы. Этот процесс называется *out-in склейкой* (см. [52]).

В вершины, соответствующие генератору v, входит только одна дуга. Обозначим через x операцию, результат которой записывается в генератор v, а значит, дуги, которые входят в вершины, соответствующие v, имеют вид: $(x, I) \rightarrow (v, I)$.

Из вершин, соответствующих использованию u, выходит только одна дуга. Обозначим через y операцию, которая получает в качестве аргумента результат чтения данного использованием u. Дуги, которые выходят из вершин, соответствующих u, имеют вид: $(u, I) \rightarrow (y, I)$.

Рассмотрим элементарный решетчатый граф для вхождений u и v. Для всех итераций I и J, для которых существует дуга $I \rightarrow J$ в этом графе, построим дугу в графе G: $(x, I) \rightarrow (y, J)$. Множество всех таких дуг для пары вхождений u и v, обозначим $A^+(u, v)$. А их объединение по всем парам вхождений u и v, связанных истинной зависимостью в этом фрагменте программы, обозначим $A^+ = \cup A^+(u, v)$.

Для всех итераций J, для которых существует дуга $I \rightarrow J$ в элементарном решетчатом графе, построенном по паре вхождений u и v, множество дуг в графе G, имеющих вид $(x, J) \rightarrow (v, J)$, обозначим $A^-(u, v)$, а множество вершин (x, J), являющихся началами этих дуг обозначим $V^-(u, v)$. Отметим, что в вершины (y, J) будет входить вторая дуга, порожденная зависимостью $I \rightarrow J$ для вхождений u и v.

Пусть $A^- = \cup A^-(u, v)$, $V^- = \cup V^-(u, v)$, где объединение берется по всем вхождениям u – генератор и v – использование.

Определение 24. Графом операций фрагмента программы будем называть ориентированный граф, являющийся объединением всех графов операторов входящих в этот фрагмент программы, с добавлением множества дуг A^+, удалением дуг A^- и их начальных вершин V^-.

Пример 12

```
for (i=0; i<N+1; ++i)
  for (j=0; j<M+1; ++j)
    for (k=0; k<P+1; ++k)
      C[i][j] = C[i][j]+A[i][k]*B[k][j]
```

Граф операций для цикла, реализующего умножение матриц, изображен на рис. 14. Отметим, что множество дуг A^+ для этого примера – это множество дуг, изображенных слева направо и соединяющих операции сложения внутри графа каждого оператора.

Конец примера.

Заметим, что в случае тесного гнезда циклов, пространства итераций всех операторов совпадают.

Определение 25. Сужением графа $G = (V, U)$ на множестве X будем называть такой подграф графа G, у которого множество вершин равно X, а множество дуг состоит из всех тех дуг графа G, которые соединяют вершины из множества X. Будем обозначать сужение графа G на множестве X, следующим образом $G|_X$.

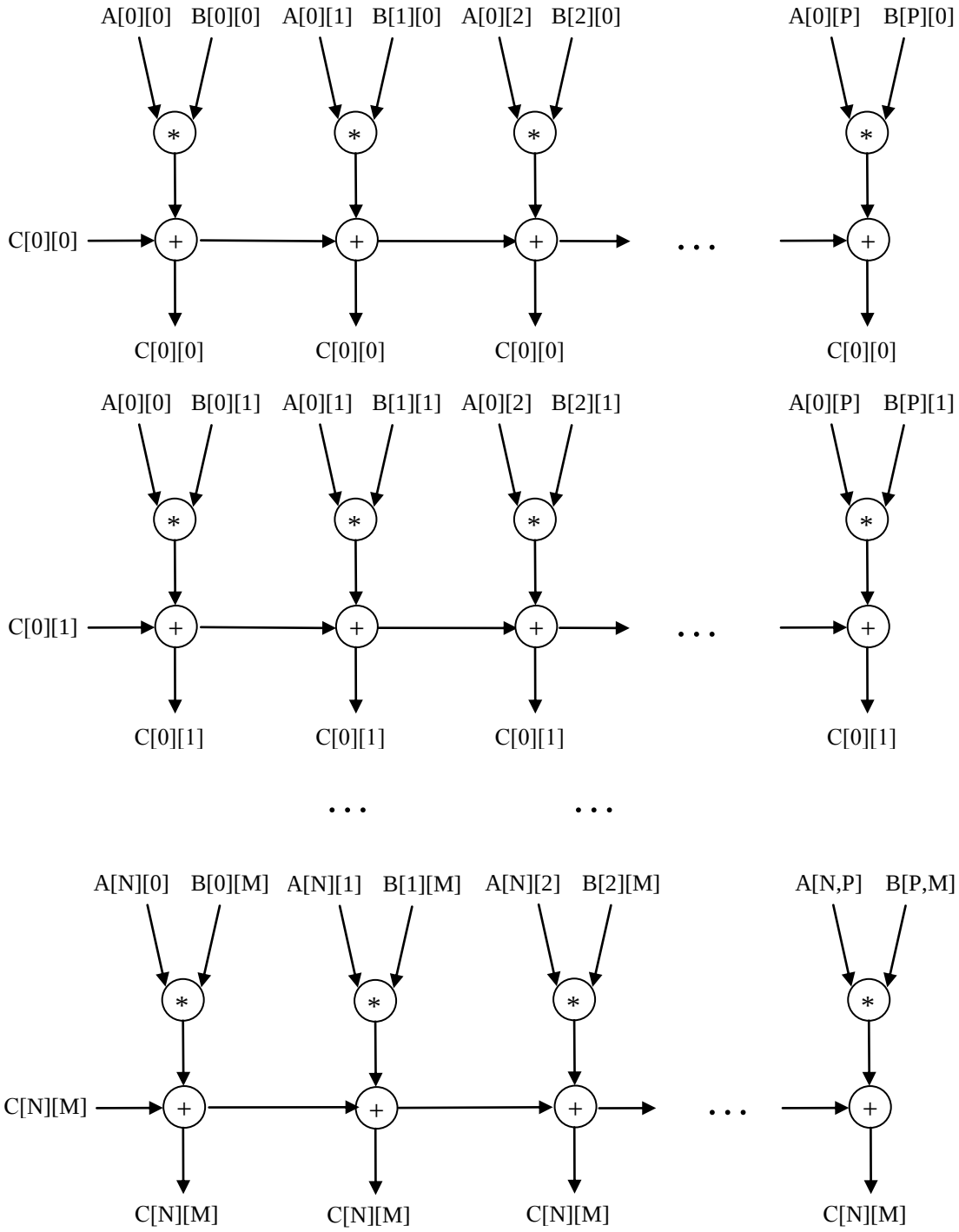

Рис. 14. Граф операций для цикла, реализующего умножение матриц

Введем понятие периодического графа, которое потребуется для определения графа вычислений [45].

Определение 26. Периодическим графом будем называть граф $G = (V, U)$ такой, что существует разбиение $\{V_i\}$ множества вершин V и все сужения $G|_{V_i}$, $i>2$ изоморфны друг другу, а множество V_1 сравнительно мало по мощности по сравнению с объединением $\{V_i\}$, $i>2$. Подграф $G|_{V_i}$ будем называть периодом.

Граф, изображенный на рис 14, – периодический, причем множество V_1 состоит из вершин чтения массива C[·], а его период изображен на рис. 15.

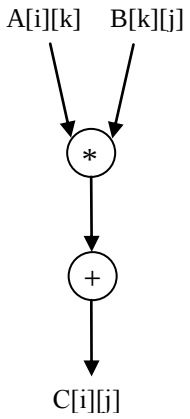

Рис. 15. Период графа операций для цикла, реализующего умножение матриц.

Определение 27. Добавим к периоду дуги, характеризующие межитерационные зависимости по данным, и поставим им в соответствие вес – вектор расстояния зависимости между соответствующими вхождениями массивов, и 1 для скалярных переменных. Полученный таким образом взвешенный граф называют редуцированным графом алгоритма – РГА (см. [45]) или графом вычислений (см. [52]).

Пример 13

```
for (i=0; i<N+1; ++i)
  for (j=0; j<M+1; ++j)
    for (k=0; k<P+1; ++k)
      C[i][j] = C[i][j]+A[i][k]*B[k][j]
```

Граф вычислений для цикла, реализующего умножение матриц, изображен на рис. 16.

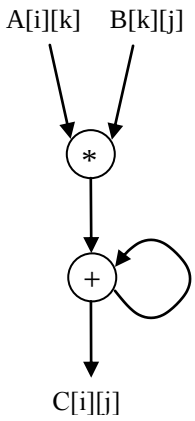

Рис. 16. Граф вычислений для цикла, реализующего умножение матриц

Обратим внимание, что операция сложения выполняется с аргументами «результат умножения» и значением C[i][j], полученным в результате предыдущего сложения. Но самое первое сложение не может получить второй аргумент, так как предыдущего сложения еще не было. В связи с этим, на графе вычислений изображают дуги и вершины инициализации, которые работают только на этапе инициализации конвейера. В данном случае вершина инициализации должна передать по дуге инициализации значение ячейки C[i][j], которое было в ней до начала цикла. На рисунках будем обозначать такие дуги и вершины пунктиром.

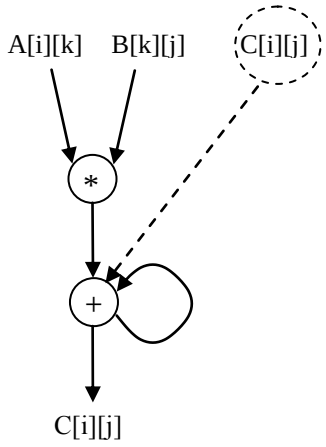

Рис. 17. Граф вычислений для цикла, реализующего умножение матриц с дугой и вершиной инициализации

Конец примера.

Периодический граф операций, как правило, порождается циклом или гнездом циклов. Он очень удобен с точки зрения конвейерных вычислений. Но хранить в памяти и обрабатывать столь большую структуру нецелесообразно, поэтому, как правило,

используют граф вычислений. В случае компьютера с архитектурой допускающей перестраиваемый конвейер, по графу вычислений можно построить задание на коммутацию. В случае архитектуры VLIW, по графу вычислений можно построить расписание для программного конвейера. Когда нет возможностей настроить операции и связи конвейера, можно осуществить попытку определения есть ли у графа вычислений подграф, который можно было бы выполнить на конвейере. С помощью графа вычислений можно определить настройки программирующего сдвигового регистра ПЛИС, что, например, применяется при разработке специальных вычислителей (см. [45]).

1.6. Выводы к первой главе

В данной главе введены основные понятия, которые будут использованы в следующих главах. Рассмотрена классификация зависимостей между вхождениями переменных в программе. Рассмотрены также такие графовые модели, как граф информационных связей, решетчатый граф, граф вычислений.

2. Конвейерные вычисления

2.1. Конвейерные задержки и интервал инициализации итераций

В этой главе будет рассмотрено отображение программного цикла на конвейерный вычислитель. В данном параграфе будут даны определения необходимые для формулирования результатов главы: расчет интервала инициализации итераций, расчет буферов, расчет расписания конвейерного вычислителя.

Определение 28. Задержкой операции будем называть время между моментом поступления операндов на выполнение в ПЭ и моментом выхода результата из ПЭ.

Определение 29. Расстоянием зависимости будем называть количество итераций цикла, которые выполняются между командами при возникновении циклически порожденной зависимости. Для циклически независимой зависимости эту величину принято считать равной 0 (см. [47]).

Понятно, что расстояние зависимости равно разности между индексными выражениями вхождений переменной, соответствующих зависимости.

Определение 30. Интервалом инициализации итераций (интервалом подачи данных, темпом подачи данных) будем называть постоянный интервал времени, по прошествии которого инициируются соседние итерации. Будем обозначать его *iii* (см. [47]).

Формула вычисления интервала инициализации итераций рассмотрена в [47, 52].

$$iii = \max_{c}\left[d(c) / p(c)\right] \qquad (2)$$

где максимум берется по всем c — циклам графа, $d(c)$ — сумма задержек операций по вершинам цикла c, $p(c)$ — сумма расстояний зависимости по дугам зависимости, входящим в цикл c, а символом $\lceil a \rceil$ обозначается округление с избытком числа a.

Для программных конвейеров вычисление минимального интервала инициализации итераций может быть выполнено, например, с помощью итеративного алгоритма [86], с использованием ЦЛП подхода [20, 21, 22]. В работах [47, 52] вычисление *iii* по формуле (2) предполагает просмотр всех циклов ориентированного графа. В данной работе предлагается более эффективный алгоритм, основанный на том, что достаточно просматривать только элементарные циклы на графе.

Необходимо отметить, что для интервала инициализации итераций программного конвейера в случае компьютера с архитектурой VLIW можно получить лишь нижнюю оценку, ввиду ограничений по ресурсам. В случае суперкомпьютера со структурно-процедурной организацией вычислений та же самая оценка является точной, так как в случае нехватки ресурсов конвейеризация откладывается до переразбиения на кадры (см. [28, 29]). Ввиду этого преимущества архитектуры со структурно-процедурной организацией вычислений задача составления расписания перестает быть NP-полной, что имеет место в общем случае для архитектуры VLIW (см. [47]). В общем случае, для архитектур с перестраиваемым конвейером указанный анализ дает информацию, как настроить конвейер, если ресурсов достаточно, в противном случае отображение на конвейер, как правило, считается невозможным.

Пример 14

```
for (i=0; i<N; ++i)
{
  X[i]=A[i]+B[i]*C[i];
  Y[i]=X[i-1]*C[i];
}
```

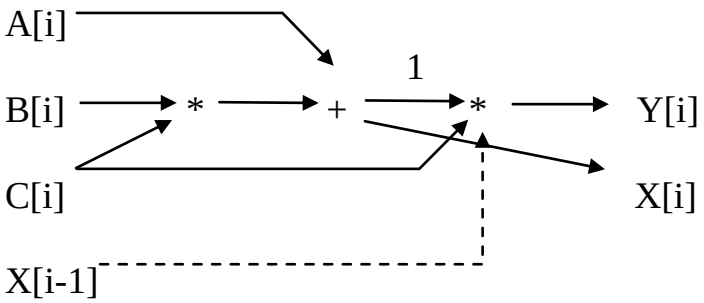

Рис. 18. Граф вычислений

Этот граф не содержит циклов, следовательно, интервал инициализации итераций равен 1, т.е. минимально возможному значению.

Пример 15

```
for (i=0; i<N; ++i)
{
  X[i] = A[i]+B[i]*Y[i-3];
  Y[i] = X[i-1]*C[i];
}
```

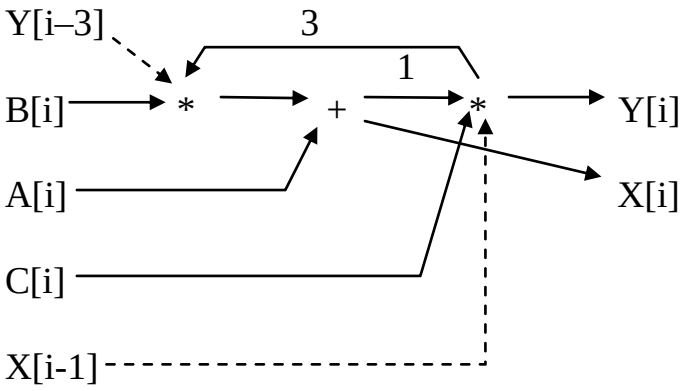

Рис. 19. Граф вычислений

Предположим, что умножение вычисляется 5 тактов, а сложение 3 такта. Этот граф содержит один цикл, следовательно, интервал инициализации итераций оценивается снизу числом $\lceil (5+3+5)/(3+1) \rceil = 4$.

2.2. Вспомогательные утверждения

В этом параграфе излагаются вспомогательные леммы и теоремы, необходимые для доказательства корректности алгоритмов описываемых ниже.

Лемма 2. Любой простой цикл можно разбить на множество элементарных циклов.

Доказательство. Доказательство этого факта аналогично классическому доказательству того факта, что любой цикл можно разбить на последовательность простых циклов.

Конец доказательства.

Лемма 3. Для любых натуральных чисел a_1, a_2, b_1, b_2, существует *i* равное 1 или 2, такое что $\frac{a_1 + a_2}{b_1 + b_2} \le \frac{a_i}{b_i}$.

Доказательство. Предположим, что верно обратное, то есть $\frac{a_1 + a_2}{b_1 + b_2} > \frac{a_1}{b_1}$ и $\frac{a_1 + a_2}{b_1 + b_2} > \frac{a_2}{b_2}$. Тогда из первого неравенства, очевидно, следует $a_2 b_1 > a_1 b_2$, а из второго $a_1 b_2 > a_2 b_1$. Полученное противоречие доказывает утверждение леммы.

Конец доказательства.

Лемма 4. Для любых двух последовательностей из *n* натуральных чисел $\{a_j\}, \{b_j\}$, существует *i* ($1 \le i \le n$), такое что $\frac{\sum_j a_j}{\sum_j b_j} \le \frac{a_i}{b_i}$.

Доказательство. Проведем доказательство конструктивно, т.е. укажем алгоритм, который позволит найти нужное значение *i* для любых последовательностей $\{a_j\}, \{b_j\}$. Обозначим через $u_k = \sum_{j>k} a_j$, $v_k = \sum_{j>k} b_j$, $k = 0..n$, в частности $u_0 = \sum_j a_j$, $v_0 = \sum_j b_j$. Очевидно, что $u_0 = u_1 + a_1$, $v_0 = v_1 + b_1$. Тогда по лемме 3 либо $\frac{u_0}{v_0} \le \frac{a_1}{b_1}$, либо $\frac{u_0}{v_0} \le \frac{u_1}{v_1}$. Если верно первое, то *i*=1, иначе применим ту же лемму к дроби $\frac{u_1}{v_1} = \frac{a_2 + u_2}{b_2 + v_2}$ и так далее пока не получим такое *i*, что $\frac{a_{i-1} + u_{i-1}}{b_{i-1} + v_{i-1}} \le \frac{a_i}{b_i}$. Алгоритм, очевидно, конечен. С другой стороны понятно, что $\frac{u_0}{v_0} \le \ldots \le \frac{u_{i-1}}{v_{i-1}} \le \frac{u_i}{v_i} \le \frac{a_i}{b_i}$. Итак, для любых двух последовательностей из *n* натуральных чисел $\{a_j\}, \{b_j\}$ с помощью вышеописанного алгоритма можно найти требуемое в условии леммы, число *i*, что и требовалось доказать.

Конец доказательства.

Теорема 1. Всегда существует элементарный цикл, на котором достигается максимум выражения (2).

Доказательство. Пусть на графе есть не элементарный цикл, на котором достигается максимум выражения (2). Покажем, что на этом графе существует элементарный цикл, на котором достигается максимум выражения (2). Пусть описанный выше не элементарный цикл разбит на множество простых циклов. Разобьем каждый из них на множество элементарных циклов по лемме 2. Таким образом, мы получаем,

разбиение исходного цикла на множество элементарных. Пусть d_j – сумма задержек операций, а p_j – сумма расстояний зависимости на j элементарном цикле. По лемме 4 найдется такое число i, что $\frac{\sum_j d_j}{\sum_j p_j} \le \frac{d_i}{p_i}$, а значит, один из элементарных циклов имеет не меньшее отношение суммы задержек операций к сумме расстояний зависимости, чем исходный цикл. Т.е. на этом элементарном цикле достигается максимум выражения (2).

Конец доказательства.

После предыдущей теоремы алгоритм нахождения интервала инициализации итераций сводится к перебору всех элементарных циклов на графе, каждый из которых содержит хотя бы одну из обратных дуг. Значит, можно перебирать обратные дуги графа. Остается только составить алгоритм обхода всех элементарных циклов, в которые входит конкретная обратная дуга.

Введем в рассмотрение частичный порядок на множестве пар натуральных чисел следующим образом: $(a_1, b_1) << (a_2, b_2)$, если $(a_1 \le a_2$ и $b_1 > b_2)$ или $(a_1 < a_2$ и $b_1 \ge b_2)$. Очевидно, что для таких пар чисел верно, что $\frac{a_1}{b_1} < \frac{a_2}{b_2}$.

Лемма 5. Пусть даны натуральные числа a_1, a_2, b_1, b_2 такие что $(a_1, b_1) << (a_2, b_2)$. Тогда для любых неотрицательных целых чисел c и d, верно что $(a_1+c, b_1+d) << (a_2+c, b_2+d)$ и, следовательно, $\frac{a_1+c}{b_1+d} < \frac{a_2+c}{b_2+d}$.

Доказательство. Из условия, очевидно, следует, что либо $a_1 + c \le a_2 + c$ и $b_1 + d > b_2 + d$, либо $a_1 + c < a_2 + c$ и $b_1 + d \ge b_2 + d$. Значит, $(a_1+c, b_1+d) << (a_2+c, b_2+d)$ и, следовательно, $\frac{a_1+c}{b_1+d} < \frac{a_2+c}{b_2+d}$.

Конец доказательства.

Лемма 6. Пусть даны конечные множества натуральных чисел A, B. Рассмотрим декартово произведение $A \times B$. Тогда любое множество не сравнимых (в смысле частичного порядка <<) в $A \times B$ пар будет содержать не больше $\min\{|A|, |B|\}$ пар.

Доказательство. Не нарушая общности, предположим, что $|A| < |B|$. Пусть $n = |A|$. Рассмотрим множество пар $C = \{(a_1, b_1), (a_2, b_2), \dots, (a_n, b_n)\}$ такое, что $a_1 < a_2 < \dots < a_n$, $b_1 < b_2 < \dots < b_n$, причем $\{a_1, a_2, \dots, a_n\} = A$, $\{b_1, b_2, \dots, b_n\} \subset B$. Тогда $C \subset A \times B$ и при этом

все пары из множества C не сравнимы в смысле частичного порядка << и $|C| = n = \min\{|A|, |B|\}$. Таким образом, приведен пример множества, у которого количество пар равно $\min\{|A|, |B|\}$.

Покажем, что множества с большим количеством пар не существует. Предположим противное: существует такое множество пар $X = \{(a_1, b_1), (a_2, b_2), \dots, (a_m, b_m)\}$ такое, что $X \subset A \times B$ и все входящие в него пары не сравнимы в смысле частичного порядка <<, и при этом $|X| = m > n$. Рассмотрим набор чисел $a_1, a_2, \dots, a_m$. Все эти числа принадлежат A, так как $X \subset A \times B$, но с другой стороны количество чисел в этом наборе равно $m > n = |A|$. Значит, в этом наборе есть равные числа. Пусть их номера k и l. Тогда для пар с номерами k и l верно либо ($a_k = a_l$ и $b_k > b_l$), либо ($a_k = a_l$ и $b_k < b_l$). В первом случае $(a_k, b_k) << (a_l, b_l)$, а во втором $(a_l, b_l) << (a_k, b_k)$. Таким образом, пары с номерами k и l из множества X сравнимы. Полученное противоречие показывает, что описанное множество X не существует, и, следовательно, любое множество не сравнимых (в смысле частичного порядка <<) в $A \times B$ пар будет содержать не больше $\min\{|A|, |B|\}$ пар.

Конец доказательства.

2.3. Алгоритм поиска множества всех элементарных циклов, содержащих данную обратную дугу

Описываемый в этом параграфе, алгоритм является вспомогательным, и будет использоваться для поиска всех циклов на графе и расчета интервала инициализации итераций.

На вход алгоритма подается произвольный граф $G = G(X, U)$, без дуг инициализации, при этом для каждой дуги графа G уже определен ее тип (древесная, прямая, поперечная, обратная) в соответствии с методом обхода в глубину. А также входным параметром является обратная дуга v, для которой будет осуществляться поиск множества элементарных циклов, ее содержащих.

На выходе алгоритма множество $ElemCycles(v)$ – множество элементарных циклов, содержащих дугу v.

Этот алгоритм состоит в переборе всех путей на графе, начинающихся в одной вершине (конец обратной дуги) и заканчивающихся в другой (начало обратной дуги), и у которых каждая дуга встречается в пути только 1 раз. При этом используется метод обхода в глубину для построения всех таких путей. Обозначим начало обратной дуги — x,

а конец — *y*. Саму обратную дугу будем обозначать *v*. Обозначим граф вычислений $G = G(X, U)$. Также потребуется массив меток *label*(*u*), ставящий в соответствие каждой дуге $u \in U$ либо 0, либо 1. Обозначим через *current* — текущую вершину, с которой работает алгоритм. Кроме того, потребуется стек *path* со стандартными процедурами *push* и *pop*, который содержит текущий путь как последовательность дуг. Также потребуется множество *vertexes* — множество вершин входящих в текущий путь *path*. Для множеств определены стандартные процедуры: *add*(*a*) — добавить вершину *a* к множеству, *exclude*(*a*) — исключить вершину *a* из множества, *in*(*a*) — принадлежит ли вершина *a* множеству (да — 1, нет — 0).

Алгоритм

1. Начало алгоритма.
2. Для всех дуг $u \in U$ установить начальное значение *label*(*u*):=0. Также *current*:=*y*; *path.push*(*v*); *vertexes.add*(*x*).
3. Начало цикла.
4. Если *current*=*y* и все дуги *u*, исходящие из вершины *current* имеют метку *label*(*u*)=1, тогда к шагу 13 (выход из цикла).
5. Если все дуги *u*, исходящие из вершины *current* имеют метку *label*(*u*)=1, тогда к шагу 10 (шаг возврата), иначе к шагу 6 (шаг в глубину).
6. Выбираем любую дугу *q* из множества дуг, исходящих из вершины *current*, такую что *label*(*q*)=0. Обозначим конец дуги *q* через *next*.
7. Если *vertexes.in*(*next*)=0, тогда к шагу 8, иначе *label*(*q*):=1 и к шагу 9.
8. Если *next*=*x*, тогда *path.push*(*q*); *path.push*(*v*); *ElemCycles*(*v*).*add*(*path*); *path.pop*(); *path.pop*(), т.е. текущий путь преобразуем в цикл и, добавив полученный цикл к искомому множеству путей, возвращаем текущий путь в исходное состояние; *label*(*q*):=1 и к шагу 3.
9. *current*:=*next*; *path.push*(*q*) и *vertexes.add*(*current*), т.е. делаем шаг в глубину. Переход к шагу 3.
10. Для всех дуг *u*, исходящих из вершины *current*, установить метки *label*(*u*):=0, чтобы можно было использовать эти дуги в другом пути.
11. *vertexes.exclude*(*current*); *t*:=*path.pop*().Обозначим начало дуги *t* через *prev*.
12. *current*:=*prev*; *label*(*t*):=1, т.е. делаем шаг возврата. Переход к шагу 3.
13. Конец цикла.
14. Конец алгоритма.

Пример 16

```
for (i=3; i<N; ++i)
{
  X[i] = A[i]+B[i]*Y[i-3];
  Y[i] = X[i-1]*C[i]+X[i-2];
}
```

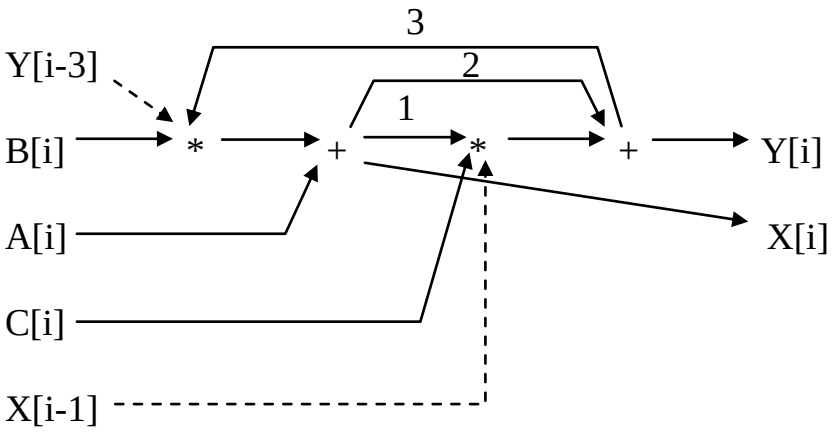

Рис. 20. Пример графа вычислений

Для этого графа будет найдено два цикла по единственной обратной дуге, ведущей от последней операции сложения к первой операции умножения.

2.4. Алгоритм вычисления интервала инициализации итераций

Алгоритм, описываемый в данном параграфе, является одним из основных результатов в данной главе. Этот алгоритм будет использоваться далее в параграфах, в которых будет составляться расписание.

На входе алгоритма граф вычислений $G = G(X, U)$, без дуг инициализации, функция $d(x)$, ставящая в соответствие каждой вершине x графа G задержку операции в этой вершине, и функция $p(u)$, ставящая в соответствие каждой дуге u, порожденной истинной информационной зависимостью, расстояние зависимости для этой дуги.

На выходе алгоритма число *iii*.

С учетом теоремы 1 для вычисления *iii* не обязательно просматривать все циклы на графе вычислений, а достаточно лишь элементарные. Предположим теперь, что построено остовное дерево с помощью обхода в глубину и для всех дуг определен их тип в

соответствии с этим построением. Тогда каждый цикл будет содержать хотя бы одну обратную дугу (см. лемму 1). Следовательно, и любой элементарный цикл также будет содержать обратную дугу. Отсюда следует, что с помощью перебора обратных дуг можно перебрать все элементарные циклы в графе.

Алгоритм

1. К графу вычислений G добавить вспомогательную вершину и дуги, ведущие из этой вершины ко всем источникам. Полученный граф обозначим G'.
2. Построить для графа G' остовное дерево методом поиска в глубину, корнем которого будет являться единственный источник графа G'.
3. Найти множество всех обратных дуг U_{back}.
4. Для каждой дуги $u \in U_{\text{back}}$ найдем множество всех элементарных циклов, содержащих эту дугу $ElemCycles(u)$.
5. Перебирая все дуги $u \in U_{\text{back}}$, переберем все циклы $c \in ElemCycles(u)$ и из них найдем цикл с наибольшим отношением суммы задержек операций к сумме расстояний зависимости и запишем это отношение в iii.
6. Конец алгоритма.

Замечание 1: Как было сказано выше, любой элементарный цикл будет содержать обратную дугу. Поэтому способ перебора всех элементарных циклов на графе использованный в алгоритме корректен. Но не оптимален, так как не любой элементарный цикл содержит ровно одну обратную дугу, следовательно, некоторые дуги будут просмотрены более одного раза. В качестве иллюстрации приведем пример цикла и графа вычислений, на котором есть цикл, содержащий две обратные дуги.

Пример 17

```
for (i=2; i<N; ++i)
{
  X[i] = A[i]+B[i]*Y[i-2];
  Y[i+1] = (X[i-2]+Z[i-2])*C[i];
  Z[i+1] = X[i-1]*Y[i];
}
```

Занумеруем операции описанного выше программного цикла сверху вниз справа налево (аддитивные S1 и S2, а мультипликативные M1, M2 и M3). На графе вычислений, изображенном на рис. 21, есть две обратные дуги: M2→M1, M3→S2. Воспользовавшись описанным выше алгоритмом, можно найти три элементарных цикла: M1-S1-S2-M2-M1,

S2-M2-M3-S2, M1-S1-M3-S2-M2-M1. Необходимо отметить, что третий из найденных элементарных циклов содержит две обратные дуги.

Этот пример показывает: множество элементарных циклов в общем случае не находится во взаимно однозначном соответствии с множеством обратных дуг. Таким образом, каждый элементарный цикл содержит не менее одной обратной дуги.

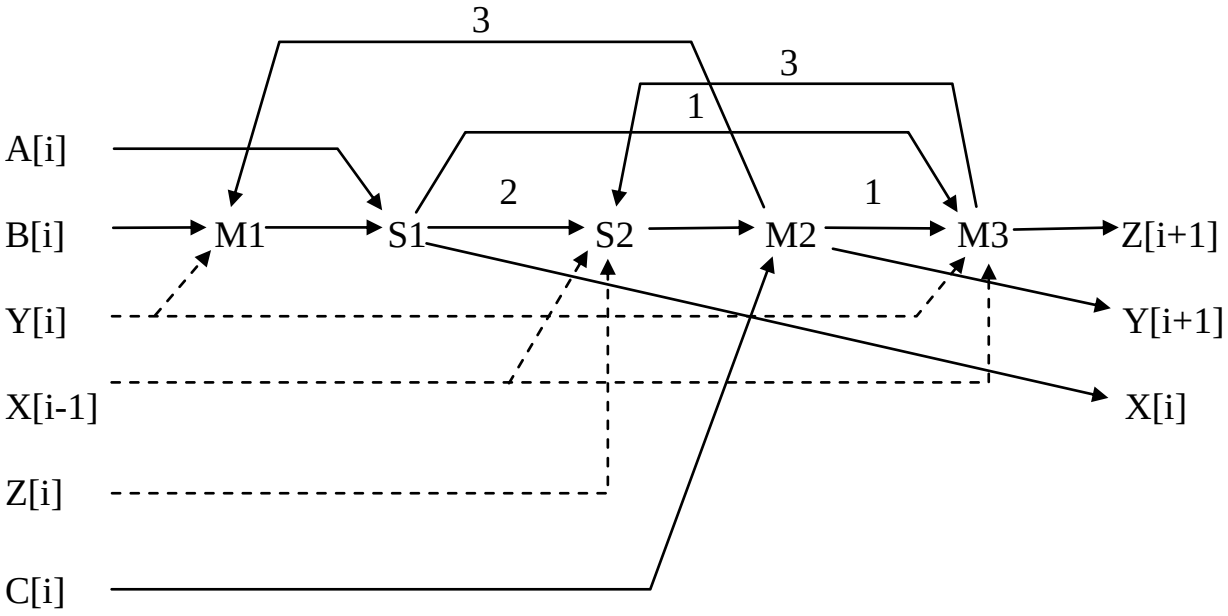

Рис. 21. Пример графа вычислений

Замечание 2: Если в архитектуре вычислительной системы встречаются не конвейерные ВЭ, то интервал инициализации итераций не может быть меньше максимума среди тех неконвейерных операций, которые есть на графе вычислений. Тогда в этом случае в алгоритм расчета *iii* необходимо добавить еще один шаг, который можно выразить следующей формулой:

$$iii_{total} = \max\left\{iii,\ \max_{x \in X'}\{d(x)\}\right\},$$

где X' – множество вершин графа вычислений, соответствующих неконвейерным операциям, $d(x)$ – время выполнения не конвейерной операции, соответствующей вершине x, *iii* – это интервал инициализации итераций, вычисленный в результате алгоритма описанного выше, а iii_{total} – это интервал инициализации итераций, учитывающий неконвейерность операций из X'.

Сформулируем и докажем корректность алгоритма расчета *iii*.

Теорема 2. Пусть граф вычислений $G = (X, U)$ содержит только 1 обратную дугу и ни одной поперечной или прямой дуги. Пусть C – единственный элементарный цикл в

этом графе, а x – произвольная вершина в C. Пусть u_1 – дуга, входящая в x и принадлежащая C, u_2 – дуга, входящая в x и не принадлежащая C. Если iii рассчитано в соответствии с алгоритмом, описанным в начале параграфа, то аргумент, поступающий по дуге u_1 (данные по этой дуге будут поступать после этапа инициализации конвейера), поступит в вершину x не раньше аргумента, поступающего по дуге u_2.

Доказательство. Пусть цикл C имеет вид (x_1, c_1, x_2, c_2, … , x_n, c_n, x_1), где c_i – дуги, а x_i – вершины которые они соединяют, при этом дуга c_i соединяет x_i и x_{i+1} для $1 \le i \le n-1$, а c_n соединяет x_n и x_1. Обозначим через o_1, o_2, … , o_n – операции соответствующие вершинам x_1, x_2, … , x_n.

Обозначим через $t_k = t + \sum_{s=1}^{k} d(x_s)$, $i_k = i + \sum_{s=1}^{k} p(x_s)$, где $1 \le k \le n$.

Отметим, что в случае графа с единственным циклом данные по дугам, не принадлежащим C, будут поступать по простым путям в графе. Следовательно, своевременность их поступления определяется только тем, что для первой итерации все аргументы должны поступить своевременно, а затем они будут поступать с интервалом iii для каждой следующей итерации программного цикла. Таким образом, достаточно проверить корректность поступления данных в одну выбранную вершину цикла C. Поэтому, не нарушая общности, будем считать, что $x = x_1$. Тогда $u_1 = c_n$.

Предположим на вход вершины x_1 поступили оба аргумента в момент времени t, необходимые для вычисления операции o_1 на итерации i. Тогда через $d(x_1)$ тактов результат этого вычисления попадет на вход операции o_2. Можно сделать вывод, что через $\sum_{i=1}^{j} d(x_i)$ тактов данные, попавшие на вход операции o_1, пройдут по зависящим от операции o_1 операциям, соответствующим вершинам цикла C, до операции o_{j+1}.

Заметим, что данные, вычисленные операцией o_1 для итерации i на такте $t+d(x_1)$, будут востребованы операцией o_2 для вычисления значения на итерации $i+p(c_1)$. Аналогично, можно заключить, что данные, вычисленные операцией o_k для итерации i_{k-1} на такте t_k, будут востребованы операцией o_{k+1} для вычисления значения на итерации i_k.

Для итерации i_{n-1} операция o_n выдаст результат вычисления на такте t_n и полученное данное необходимо будет для вычисления результата операции o_1 на итерации i_n, так как c_n соединяет x_n и x_1. Другой аргумент для той же итерации операции o_1 придет на вход на такте $t + iii * (i_n - i)$ по дуге u_2, так как данные поступают по дуге u_2 с частотой iii тактов. Следовательно, условие теоремы можно сформулировать в виде неравенства:

$$t + iii * (i_n - i) \ge t_n. \qquad (3)$$

Для доказательства этого неравенства воспользуемся формулой (2), с учетом теоремы 1 и того, что в графе только один элементарный цикл:

$$iii = \lceil d(C) / p(C) \rceil .$$

Следовательно,

$$iii \geq \sum_{k=1}^{n} d(x_k) / \sum_{k=1}^{n} p(u_k) .$$

Тогда

$$iii * \sum_{k=1}^{n} p(u_k) \geq \sum_{k=1}^{n} d(x_k) .$$

Из последнего неравенства, очевидно, следует, что неравенство (3) верно, что и требовалось доказать.

Конец доказательства.

В заключение этого параграфа стоит отметить, что полный перебор всех путей от одной вершины до другой может осуществляться за экспоненциальное время. В качестве примера можно привести граф, для которого количество путей от вершины *s* до вершины *f* равно 2^n, где n – это количество вершин на графе.

Рис. 22. Пример графа с экспоненциальным количеством путей

Таким образом, описанный в этом параграфе алгоритм вычисления *iii*, вообще говоря, имеет экспоненциальную сложность.

2.5. Полиномиальный алгоритм вычисления интервала инициализации итераций

Алгоритм, описываемый в данном параграфе, будет использоваться далее в параграфах, в которых будет составляться расписание.

На входе алгоритма граф вычислений $G = G(X, U)$, без дуг инициализации, функция $d(x)$, ставящая в соответствие каждой вершине x графа G задержку операции в этой вершине, и функция $p(u)$, ставящая в соответствие каждой дуге u, порожденной истинной информационной зависимостью, расстояние зависимости для этой дуги. Пусть также $d(x)$ принимает ограниченное количество различных значений, не зависящее от $|X|$.

На выходе алгоритма число *iii*.

В процессе работы алгоритма, строится множество характеристик $C(x)$, которое представляет собой множество пар неотрицательных целых чисел.

С учетом теоремы 1 для вычисления *iii* не обязательно просматривать все циклы на графе вычислений, а достаточно лишь элементарные. Предположим теперь, что построено остовное дерево с помощью обхода в глубину и для всех дуг определен их тип в соответствии с этим построением. Тогда каждый цикл будет содержать хотя бы одну обратную дугу (см. лемму 1). Следовательно, и любой элементарный цикл также будет содержать обратную дугу. Отсюда следует, что с помощью перебора обратных дуг можно построить нижнюю оценку *iii* для всех элементарных циклов содержащих данную обратную дугу.

Алгоритм

1. К графу вычислений G добавить вспомогательную вершину и дуги, ведущие из этой вершины ко всем источникам. Полученный граф обозначим G'.
2. Построить для графа G' остовное дерево методом поиска в глубину, корнем которого будет являться единственный источник графа G'. Обозначим это дерево T. В процессе построения остовного дерева, найти множество всех обратных дуг U_{back}.
3. Для каждой дуги $u \in U_{\text{back}}$ найдем множество $D(u)$ всех вершин достижимых из конца дуги u в построенном выше остовном дереве T. Обозначим через s конец текущей дуги u, а начало через f.
4. Построим граф $T_u = T|_{D(u)}$. Этот граф является деревом. Корнем дерева T_u, очевидно, будет s, а начало дуги u также будет принадлежать этому дереву.
5. Построим правильную нумерацию для вершин дерева T_u.
6. Для корня дерева s определим множество характеристик $C(s) = \{(d(s), 0)\}$.
7. Будем обходить вершины T_u в порядке правильной нумерации, пропустив корень s. Для каждой вершины x дерева T_u будем строить множество характеристик $C(x)$. Каждая характеристика представлена парой неотрицательных чисел.

8. Положим $C(x) = \varnothing$.
9. Для текущей вершины x будем перебирать все вершины y такие, что существует дуга $v = (y, x)$.
10. Для каждой характеристики $(a, b) \in C(y)$ проверим, есть ли пара $(a', b') \in C(x)$, такая что пара (a', b') сравнима с парой $(a + d(x), b + p(v))$. Если пара (a', b') не сравнима с парой $(a + d(x), b + p(v))$, то добавим пару $(a + d(x), b + p(v))$ в множество характеристик $C(x)$. В противном случае проверим, если $(a', b') << (a + d(x), b + p(v))$, то заменим пару (a', b') на пару $(a + d(x), b + p(v))$.
11. Конец цикла по вершинам y.
12. Если $x = f$, то переходим к шагу 14.
13. Конец цикла по вершинам x.
14. Вычислим $iii(u) = \max\limits_{(a,b)\in C(f)} \dfrac{a}{b + p(u)}$.
15. Очистим все множества характеристик $C(x)$.
16. Конец цикла по дугам u.
17. Вычислим $iii = \max\limits_{u \in U_{back}} iii(u)$.
18. Конец алгоритма.

Замечание 3: В алгоритме, описанном в параграфе 2.4, используется полный перебор всех путей между двумя заданными вершинами, что делает этот алгоритм экспоненциальным. В данном параграфе перебор всех путей между двумя заданными вершинами отсутствует, но при вычислении характеристик в каждой вершине учитываются все пути, которые могут привести в данную вершину. Идея алгоритма аналогична тому, как находятся кратчайшие пути в волновых алгоритмах или в алгоритме Дейкстры [48].

Шаг 11 алгоритма требует пояснения. Предположим, в некоторую вершину x ведут два различных простых пути из вершины s. Предположим также, что сумма задержек операций по первому из путей равна a_1, а по второму a_2, сумма расстояний зависимостей пути равна b_1 и b_2, соответственно. Тогда в $C(x)$ должны быть обе пары (a_1, b_1) и (a_2, b_2). Предположим также, что есть элементарный цикл c_1, частью которого является первый из путей. Тогда очевидно, что существует элементарный цикл c_2, частью которого является второй из путей.

Если для пар (a_1, b_1) и (a_2, b_2) справедливо, что $(a_1, b_1) << (a_2, b_2)$, то, очевидно, что $\frac{a_1}{b_1} < \frac{a_2}{b_2}$. Следовательно, по лемме 5 оценка *iii* для цикла c_1 будет меньше, чем для цикла c_2. Следовательно, в процессе анализа путей из s в f на шаге 11 алгоритма можно выбросить из рассмотрения первый путь и его характеристику. Таким образом, множество характеристик $C(x)$ в любой момент будет содержать только пары не сравнимые в смысле частичного порядка <<.

Замечание 4: Описанный алгоритм является полиномиальным. Обратим внимание, что первый элемент каждой характеристики $C(x)$ является суммой задержек операций по некоторому пути, ведущему из s в x, а второй элемент – это сумма расстояний зависимости по тому же пути. При этом различных значений задержек операций конечное число, зависящее от набора различных значений $d(x)$. Таким образом, количество различных сумм задержек операций можно оценить полиномом от количества вершин. Тогда по лемме 6 количество различных пар характеристик в множестве $C(x)$, оценивается полиномом от количества вершин, а значит, и вычисление оценки $iii(u)$ выполняется за полиномиальное время. Таким образом, алгоритм, описанный в этом параграфе, является полиномиальным.

На практике при построении графа вычислений функция $d(x)$ будет иметь конечное число различных значений, так как значения этой функции – это задержки операций, встроенных в вычислительное устройство, на которое будет отображаться программа.

2.6. Составление расписания для организации конвейерных вычислений одномерного цикла

2.6.1. Примеры составления расписания для конвейерных вычислений

В этой главе будем составлять расписание цикла в форме таблицы, на примерах, чтобы облегчить дальнейшее восприятие алгоритма построения такой таблицы. Под расписанием будем понимать двумерную таблицу, у которой столбцы соответствуют операциям в программе (вершинам на графе вычислений), строки соответствуют номерам тактов, а в ячейках таблицы стоит прочерк (-), если соответствующая операция на этом такте бездействует, в противном случае перечень номеров итераций цикла, которые

выполняются этой операцией на этом такте. Поскольку вершины инициализации задают действия, выполняемые до старта конвейера, столбцы соответствующие этим вершинам включать в таблицу расписания не будем.

Кроме того, в данном параграфе на примерах будут проиллюстрированы такие понятия, как вершины и дуги инициализации графа вычислений.

Рассмотрим пример построения расписания.

Пример 18

```
for(i=2; i<N; ++i)
{
      X[i] = A[i]+B[i]*Y[i+3];
      Y[i] = C[i]-X[i-2];
}
```

Предположим, что операции чтения и записи выполняются 1 такт, операция умножения – 5 тактов, аддитивные операции – 3 такта. Предположим также, что все умножения и сложения – конвейерные операции и позволяют каждый такт получать новые данные. Обозначим вершину, соответствующую сложению – S1, умножению – M1, вычитанию – S2. Граф вычислений этого цикла изображен на рис. 23.

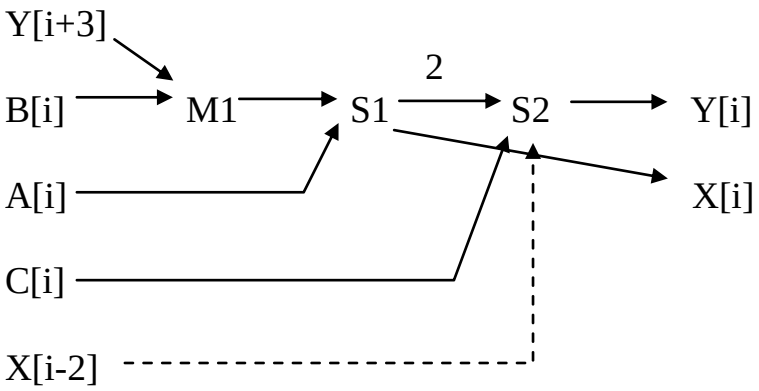

Рис. 23. Пример графа вычислений

Граф вычислений на рис. 23 не содержит циклов, следовательно, *iii* = 1. Приведем расписание для этого программного цикла.

Таблица 1

№ такта	A	B	C	Y(r)	M1	S1	S2	X(w)	Y(w)
1	-	1	-	1	-	-	-	-	-
2	-	2	-	2	1	-	-	-	-
3	-	3	-	3	1,2	-	-	-	-
4	-	4	-	4	1,2,3	-	-	-	-
5	-	5	-	5	1,2,3,4	-	-	-	-
6	1	6	-	6	1,2,3,4,5	-	-	-	-
7	2	7	1	7	2,3,4,5,6	1	-	-	-
8	3	8	2	8	3,4,5,6,7	1,2	1	-	-
9	4	9	3	9	4,5,6,7,8	1,2,3	1,2	-	-
10	5	10	4	10	5,6,7,8,9	2,3,4	1,2,3	1	-
11	6	11	5	11	6,7,8,9,10	3,4,5	2,3,4	2	1
12	7	12	6	12	7,8,9,10,11	4,5,6	3,4,5	3	2
13	8	13	7	13	8,9,10,11,12	5,6,7	4,5,6	4	3
14	9	14	8	14	9,10,11,12,13	6,7,8	5,6,7	5	4
15	10	15	9	15	10,11,12,13,14	7,8,9	6,7,8	6	5

Таблица, приведенная выше, является только частью расписания, длина которой зависит от количества итераций цикла. Здесь и далее, будем изображать только часть таблицы, из которой ясно, как выглядит она целиком.

Замечание 5. Необходимо отметить, что, зная номер такта s, на котором начинает выполнение первая итерация некоторой операции, несложно рассчитать момент старта любой итерации с номером k для этой операции, по формуле $s + iii*(k - 1)$. Кроме того, для всех вершин, исключая вершины инициализации, необходимо рассчитать момент остановки по формуле $s + iii*(N - 1) + d - 1$, где d – задержка операции, соответствующей этой вершине, N – количество итераций цикла.

Пример 19

```
for(i=1; i<N; ++i)
{
      X[i] = A[i]+Y[i];
      Z[i] = X[i-1]*B[i]*C[i];
      Y[i+2] = X[i]*C[i]+Z[i-1];
}
```

Предположим, что операции чтения и записи выполняются по 1 такту, операция умножения – 5 тактов, аддитивные операции по 3 такта (тем самым задержки операций

определены). Предположим также, что все умножения и сложения – конвейерные операции и позволяют каждый такт получать новые данные. Обозначим вершины, соответствующие аддитивным операциям – S1 и S2, в порядке их появления в теле цикла сверху вниз слева направо. Обозначим вершины, соответствующие мультипликативным операциям – M1, M2 и M3, в порядке их появления в теле цикла сверху вниз слева направо. Чтобы не загромождать рисунок, граф вычислений этого цикла изображен на рис. 24 без дуг и вершин инициализации.

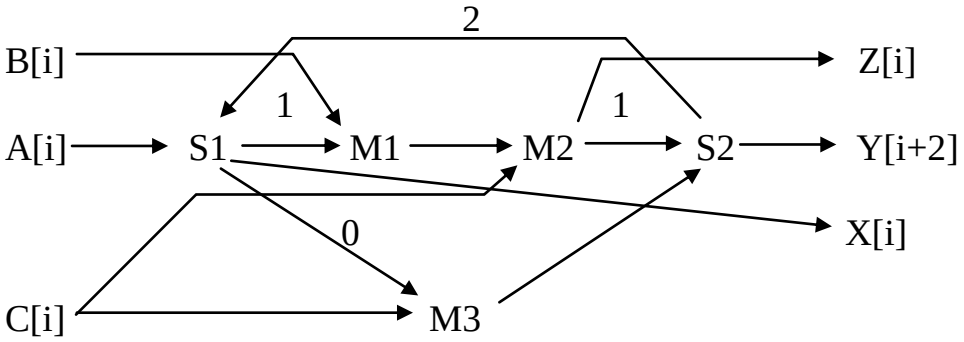

Рис. 24

Граф вычислений на рис. 24 содержит два цикла, проходящих через одну и ту же обратную дугу с началом в S2 и концом в S1. Таким образом, $iii = \max\{\lceil (3+5+5+3)/(1+1+2);\ (3+5+3)/(2+0) \rceil\} = 6$. Приведем расписание для этого программного цикла.

Таблица 2

№ такта	A	B	C	S1	S2	M1	M2	M3	X(w)	Y(w)	Z(w)
1	-	1	-	-	-	-	-	-	-	-	-
2	-	-	-	-	-	1	-	-	-	-	-
3	-	-	-	-	-	1	-	-	-	-	-
4	1	-	-	-	-	1	-	-	-	-	-
5	-	-	-	1	-	1	-	-	-	-	-
6	-	-	1	1	-	1	-	-	-	-	-
7	-	2	-	1	-	-	1	-	-	-	-
8	-	-	-	-	-	2	1	1	1	-	-
9	-	-	-	-	-	2	1	1	-	-	-
10	2	-	-	-	-	2	1	1	-	-	-
11	-	-	-	2	-	2	1	1	-	-	-
12	-	-	2	2	-	2	-	1	-	-	1
13	-	3	-	2	1	-	2	-	-	-	-
14	-	-	-	-	1	3	2	2	2	-	-
15	-	-	-	-	1	3	2	2	-	-	-
16	3	-	-	-	-	3	2	2	-	1	-
17	-	-	-	3	-	3	2	2	-	-	-
18	-	-	3	3	-	3	-	2	-	-	2
19	-	4	-	3	2	-	3	-	-	-	-
20	-	-	-	-	2	4	3	3	3	-	-

По таблице 1 (из предыдущего примера) можно определить, что буферы задержек для операций в примере 18 не нужны. В то же время, по табл. 2 наверняка можно указать, где и какие буферы задержек необходимы. Итак, на дуге ведущей от S2 к S1, необходим буфер размером в 1 такт; на дуге, ведущей от M2 к S2, необходим буфер размером в 1 такт; на дуге, ведущей от C к M3, необходим буфер размером в 1 такт.

2.6.2. Вспомогательный алгоритм составления расписания и вычисления буферных задержек на подграфе графа вычислений, который содержит один источник и не содержит обратных дуг

Алгоритм, описываемый в этом параграфе, носит вспомогательный характер. Он является составной частью алгоритма составления расписания.

Для работы алгоритма потребуется функция $s(x)$, вычисляемая в процессе работы алгоритма, ставящая в соответствие каждой вершине x момент старта вычислений этой операции в расписание. Также потребуется вспомогательная функция $h(u)$, вычисляемая в процессе работы алгоритма и ставящая в соответствие каждой дуге u свернутого графа цикла размер буфера для нее.

На входе алгоритма: подграф $F = F(Y; V)$ графа вычислений цикла $G = G(X; U)$, содержащий ровно один источник и не содержащий обратных дуг, функция $p(u)$, ставящая в соответствие каждой дуге u, порожденной истинной информационной зависимостью, расстояние зависимости для этой дуги, а для остальных дуг равна нулю, и функция $d(x)$, ставящая в соответствие каждой вершине x графа G задержку операции в этой вершине.

На выходе алгоритма: $s(x)$ для всех $x \in Y$ (т.е. рассчитываем расписание для множества столбцов, соответствующих вершинам из множества Y) и $h(u)$ для всех $u \in V$ (т.е. рассчитываем буферы для всех дуг из множества V).

1. Найдем единственный источник на графе F и обозначим его *source*.
2. $s(source) := 1$.
3. Начало цикла. Перебираем вершины из множества V в порядке правильной нумерации. Текущую вершину обозначаем *current*.
4. Если *current* имеет две входящих дуги $u1$ и $u2$ с началами в $x1$ и $x2$, тогда $s(current) := \max\{s(x1) + d(x1) - p(u1)*iii,\ s(x2) + d(x2) - p(u2)*iii\} + 1$, иначе (случай, когда *current* имеет одну входящую дугу u с началом в вершине x) $s(current) := s(x) + d(x) - p(u)*iii$.
5. Если *current* имеет две входящих дуги $u1$ и $u2$ с началами в $x1$ и $x2$, тогда обозначим через u, с началом x, ту из двух дуг, которая дает меньшее значение выражению $s(x) + d(x) - p(u)*iii$, тогда $h(u) := s(current) + p(u)*iii - (s(x) + d(x))$.
6. Конец цикла.

Замечание 6. Отметим тот факт, что граф F всегда будет связный и ациклический. Связный, потому что у него только один источник, а ациклический, потому что отсутствуют обратные дуги. Ацикличность обосновывает правильную нумерацию на шаге 3. Корректность шага 4 обосновывается связностью графа, а также тем, что операции могут быть либо бинарные (случай двух входящих дуг), либо унарные (случай одной входящей дуги). Также надо отметить, что алгоритм не зависит от выбора правильной нумерации.

Пример 20

Возьмем программный цикл из примера 19. Из графа вычислений, изображенного на рис. 24, выделим подграф, который не содержит вершин чтения и записи, а также обратных дуг. Полученный подграф обозначим *F* (см. рис. 25).

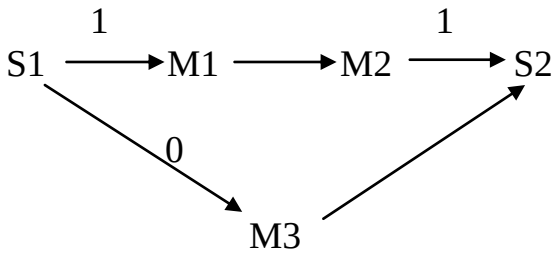

Рис. 25

Напомним, что *iii* = 6, задержки мультипликативных операций равны 5, а аддитивных операций – 3, и все эти операции конвейерные.

Находим источник – S1. Устанавливаем *s*(S1) := 1. Далее двигаемся в порядке правильной нумерации по вершинам графа. Правильная нумерация не единственна, но для работы этого алгоритма подходит любая. Например: S1, M1, M3, M2, S2. Таким образом, дальше выполняем 4 раза шаг 4 последнего алгоритма. Итак,

s(M1) = *s*(S1) + *d*(S1) – *p*(S1;M1)**iii* = –2,

s(M3) = *s*(S1) + *d*(S1) – *p*(S1;M2)**iii* = 4,

s(M2) = *s*(M1) + *d*(M1) – *p*(M1;M2)**iii* = 3,

s(S2) = max{*s*(M2) + *d*(M2) – *p*(M2;S2)**iii*, *s*(M3) + *d*(M3) – *p*(M3;S2)**iii*} =

= max{2,9} = 9.

И один раз нужно выполнить шаг 5 последнего алгоритма:

h(M2;S2) = *s*(S2) + *p*(M2;S2)**iii* – (*s*(M2) + *d*(M2)) = 7.

Таким образом, получаем рассчитанный буфер для дуги (M2;S2) и расписание в следующей таблице:

Таблица 3.

№ такта	S1	S2	M1	M2	M3
-2	-	-	1	-	-
-1	-	-	1	-	-
0	-	-	1	-	-
1	1	-	1	-	-
2	1	-	1	-	-
3	1	-	-	1	-
4	-	-	2	1	1
5	-	-	2	1	1
6	-	-	2	1	1
7	2	-	2	1	1
8	2	-	2	-	1
9	2	-	-	2	-
10	-	1	3	2	2
11	-	1	3	2	2
12	-	1	3	2	2
13	3	-	3	2	2
14	3	-	3	-	2
15	3	-	-	3	-

2.6.3. Склеивание двух подграфов с рассчитанными расписаниями (вспомогательный алгоритм)

Алгоритм, описываемый в этом параграфе, носит вспомогательный характер. Он является составной частью алгоритма составления расписания.

Для работы алгоритма потребуется функция $s(x)$, вычисляемая в процессе работы алгоритма, ставящая в соответствие каждой вершине x момент старта вычислений этой операции в таблице расписания. Также потребуется вспомогательная функция $h(u)$, вычисляемая в процессе работы алгоритма и ставящая в соответствие каждой дуге u свернутого графа цикла размер буфера для нее.

На входе алгоритма: подграфы $F_1 = F_1(Y_1; V_1)$ и $F_2 = F_2(Y_2; V_2)$ графа вычислений цикла $G = G(X; U)$, без обратных дуг, множество V_3 – множество дуг графа G, соединяющих вершины подграфа F_1 с вершинами подграфа F_2; функция $s(x)$, рассчитанная для каждого из подграфов отдельно; функция $p(u)$, ставящая в соответствие каждой дуге u, порожденной истинной информационной зависимостью, расстояние

зависимости для этой дуги, а для остальных дуг равна нулю; функция $d(x)$, ставящая в соответствие каждой вершине x графа G задержку операции в этой вершине.

На выходе алгоритма: подграф $F = F(Y; V)$, где $Y = Y_1 \cup Y_2$, $V = V_1 \cup V_2 \cup V_3$, $s(x)$ для всех $x \in Y$ (т.е. рассчитываем расписание для множества столбцов, соответствующих вершинам из множества Y) и $h(u)$ для всех $u \in V$ (т.е. рассчитываем буферы для всех дуг из множества V).

Замечание 7. Отметим тот факт, что граф F всегда будет ациклический, так как он не будет содержать обратных дуг, и дуги из множества V_3 соединяют вершины подграфа F_1 с вершинами подграфа F_2. И это является главным условием при невыполнении, которого алгоритм работать не будет.

Для работы алгоритма потребуется вспомогательная функция $f(u)$, ставящая в соответствие каждой дуге $u \in V_3$ сдвиг таблицы расписания F_1 относительно этой дуги.

1. Начало цикла. Перебираем все дуги из множества V_3, при этом текущую дугу обозначаем *u_current*, ее начало – $x1$, а конец – $x2$.
2. $f(u\_current) := s(x2) + p(u\_current)*iii - (s(x1) + d(x1))$.
3. В переменную *minf* запоминаем текущее минимальное значение функции $f(u)$ для уже просмотренных дуг.
4. Конец цикла.
5. Начало цикла. Перебираем вершины графа F_1. Текущую вершину обозначаем *x_current*.
6. $s(x\_current) := s(x\_current) - minf$.
7. Конец цикла.
8. Начало цикла. Перебираем все дуги из множества V_3, при этом текущую дугу обозначаем *u_current*.
9. $h(u\_current) := f(u\_current) - minf$.
10. Конец цикла.

Пример 21

```
for(i=2; i<N; ++i)
{
      X[i] = (X[i-2]+C[i]*B[i])*Y[i-1]+Z[i-1];
      Z[i] = B[i-2]-C[i];
      Y[i] = A[i-2]*Z[i];
}
```

Предположим, что операции чтения и записи выполняются 1 такт, операция умножения – 5 тактов, аддитивные операции – 3 такта (тем самым задержки операций определены). Предположим также, что все умножения и сложения реализованы как конвейерные операции и позволяют каждый такт получать новые данные. Обозначим вершины, соответствующие аддитивным операциям – S1, S2 и S3, в порядке их появления в теле цикла сверху вниз слева направо. Обозначим вершины, соответствующие мультипликативным операциям – M1, M2 и M3, в порядке их появления в теле цикла сверху вниз слева направо. Чтобы не загромождать рисунок, граф вычислений этого цикла изображен, без дуг и вершин инициализации (рис. 26).

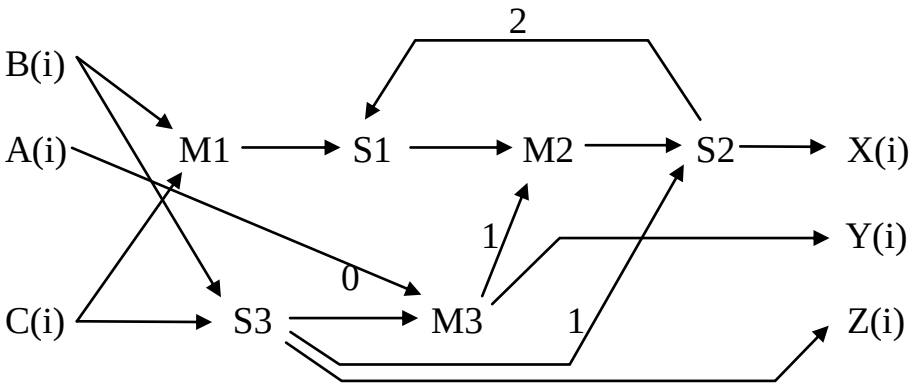

Рис. 26. Граф вычислений для примера 21

Предположим, что *iii* = 6 уже рассчитано. Предположим также, что этот свернутый граф цикла уже разбит на два подграфа F_1 (см. рис. 27) и F_2 (см. рис. 28). Для алгоритма неважно содержат ли подграфы вершины чтения и записи, но этот вспомогательный алгоритм в дальнейшем потребуется только для работы с подграфами без вершин чтения и записи.

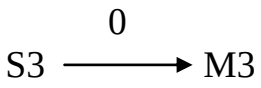

Рис. 27. Фрагмент графа вычислений из примера 21

Графы получаются очень простые, но нас интересует склеивание этих графов. По условию алгоритма для этих подграфов расписание должно быть уже рассчитано. Итак, s(M1) = 1; s(S1) = 6; s(M2) = 9; s(S2) = 14 – для подграфа F_1, s(S3) = 1; s(M3) = 4 – для подграфа F_2.

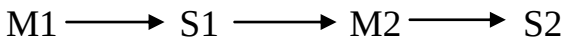

Рис. 28. Фрагмент графа вычислений из примера 21

Теперь воспользуемся последним алгоритмом. Множество V_3 состоит из двух дуг $u1$ = (M3;M2) и $u2$ = (S3;S2), следовательно, шаг 2 последнего алгоритма будет выполнен дважды:

$f(u1) := s(M2) + r(u1)*iii - (s(M3) + z(M3)) = 6,$

$f(u2) := s(S2) + r(u2)*iii - (s(S3) + z(S3)) = 16,$

$minf := \min\{f(u1), f(u2)\}=6.$

После этого пересчитываем расписание для вершин из F_1:

$s(S3) := s(S3) - minf = -5,$

$s(M3) := s(M3) - minf = -2.$

И рассчитываем буфер для дуги $u2$:

$h(u2) := f(u2) - minf = 10.$

Таким образом, получаем расписание следующего вида:

Таб. 4

№ такта	S1	S2	S3	M1	M2	M3
-5	-	-	1	-	-	-
-4	-	-	1	-	-	-
-3	-	-	1	-	-	-
-2	-	-	-	-	-	1
-1	-	-	-	-	-	1
0	-	-	-	-	-	1
1	-	-	2	1	-	1
2	-	-	2	1	-	1
3	-	-	2	1	-	-
4	-	-	-	1	-	2
5	-	-	-	1	-	2
6	1	-	-	-	-	2
7	1	-	3	2	-	2
8	1	-	3	2	-	2
9	-	-	3	2	1	-
10	-	-	-	2	1	3
11	-	-	-	2	1	3
12	2	-	-	-	1	3
13	-	-	3	3	1	3
14	-	1	-	3	-	3
15	-	1	-	3	2	-
16	2	1	-	3	2	4

2.6.4. Алгоритм составления расписания и вычисления буферных задержек на дугах графа вычислений

Алгоритм, описываемый в этом параграфе, является одним из основных результатов диссертации.

Перед изложением алгоритма, для улучшения восприятия, приведем короткое его описание:

Этап 1. Из графа вычислений для цикла, без дуг инициализации, удаляются вершины чтения и записи (вместе с их дугами), а также обратные дуги. Полученный граф обозначаем G''.

Этап 2. Для G'' производится разбиение на связные подграфы, содержащие по одному источнику из G''. При этом необходимо, чтобы для любых двух дуг, не вошедших ни в один из подграфов разбиения, и любых двух подграфов этого разбиения было бы

неверно утверждение, что начало первой из дуг принадлежит первому подграфу, конец – второму, а для второй дуги наоборот. Подграфы разбиения обозначаем G_1, G_2, …,G_m.

Этап 3. Каждый G_k рассчитываем вспомогательным алгоритмом.

Этап 4. Склеиваем G_k вспомогательным алгоритмом и получаем G'' с рассчитанным расписанием.

Этап 5. Рассчитываем буферные задержки на обратных дугах.

Этап 6. Добавляем в расписание вершины чтения и, если необходимо, на их дугах определяем буферные задержки.

Этап 7. Добавляем в расписание вершины записи.

Этап 8. Нормализуем расписание, т.е. делаем так, чтобы первая строка таблицы расписания содержала такт с номером 1.

Далее при изложении алгоритма будем использовать определение 25 сужения графа на множестве вершин.

Обозначим через $G = G(X, U)$ свернутый граф цикла, без дуг инициализации. Для работы алгоритма потребуется функция $s(x)$, вычисляемая в процессе работы алгоритма, ставящая в соответствие каждой вершине x момент старта вычислений этой операции в расписании. Также потребуется вспомогательная функция $h(u)$, вычисляемая в процессе работы алгоритма и ставящая в соответствие каждой дуге u свернутого графа цикла размер буферной задержки для нее.

На входе алгоритма: граф G, без дуг инициализации; функция $p(u)$, ставящая в соответствие каждой дуге u, порожденной истинной информационной зависимостью, расстояние зависимости для этой дуги, а для остальных дуг равна нулю; функция $d(x)$, ставящая в соответствие каждой вершине x графа G задержку операции в этой вершине.

На выходе алгоритма: функции $s(x)$, $h(u)$.

1. К графу G добавить вершину и дуги, ведущие из этой вершины ко всем источникам. Полученный граф обозначим G'.
2. Для графа G' построить остовное дерево методом обхода в глубину, корнем которого будет являться единственный источник графа G'.
3. Найти множество всех обратных дуг U_{back}.
4. Из графа G удалим все дуги, принадлежащие множеству U_{back}, а также все вершины чтения и записи вместе с принадлежащими им дугами. Полученный граф обозначим G''.
5. $k := 0$; $G_0 := G''$; X_0 – множество вершин графа G_0.
6. Начало цикла (для графа G'' построим набор подграфов G_1 ,G_2 , …, G_m).

7. Если G_0 – пустой граф, тогда к шагу 10 (выход), иначе $k := k + 1$.
8. Из графа G_0 выберем любой источник. Обозначим через X_k – множество вершин графа G_0 достижимых по направлению дуг из этого источника. Найдем X_k.
9. Определим $G_k := G_0\big|_{X_k}$, переопределим $X_0 := X_0 \setminus X_k$, и после этого переопределим $G_0 := G_0\big|_{X_0}$.
10. Конец цикла.
11. $m := k$ – количество подграфов в разбиении графа G.
12. Начало цикла. Для k от 1 до m выполнять следующее:
13. К графу G_k применить алгоритм составления расписания для графа вычислений с одним источником.
14. Конец цикла.
15. Начало цикла.
16. Склеиваем графы G_k, так чтобы получился граф G'', применяя вспомогательный алгоритм склейки рассчитанных подграфов.
17. Конец цикла.
18. Начало цикла.
19. Для каждой дуги $u \in U_{\text{back}}$ рассчитываем буфер $h(u)$ следующим образом: пусть x – начало дуги u, y – конец дуги u; $h(u) := s(y) + p(u)*iii - (s(x) + d(x))$.
20. Конец цикла. В результате работы этого цикла будет полностью рассчитано расписание для графа G''.
21. Начало цикла.
22. Для каждой вершины чтения x рассчитать $s(x)$ следующим образом: пусть $E(x)$ – множество вершин смежных с x; $s(x) := \min\{s(y) + p(u)*iii\} - d(x)$, где $y \in E(x)$, $u = (x,y)$.
23. Для каждой дуги u исходящей из вершины x: $h(u) := s(y) - s(x) - d(\text{x})$, где y – конец дуги u.
24. Конец цикла.
25. Начало цикла.
26. Для каждой вершины записи x рассчитать $s(x)$ следующим образом: пусть y – вершина смежная с x, тогда $s(x) := s(y) + T_{\text{send}}$.
27. Конец цикла.
28. Найти $s := \min\{s(x)\}$, где $x \in X$.
29. Начало цикла.

30. Для каждой вершины x графа G: $s(x) := s(x) - s + 1$.

31. Конец цикла.

Замечание 8. От порядка перебора источников на шагах 6–10 зависит то, какое расписание построит алгоритм и соответственно размеры и расположения буферов на графе. Но в любом случае расписание будет корректно с точки зрения последовательного выполнения исходного цикла.

Замечание 9. Способ разбиения на подграфы на шаге 8 диктуется требованием вспомогательного алгоритма, склеивающего два подграфа. Напомним, что множество дуг между подграфами (в указанном алгоритме обозначено V_3) должно удовлетворять условию направленности дуг от одного подграфа к другому.

Пример 22

Возьмем программный цикл из примера 21. Тогда последним алгоритмом будет получен граф G'', изображенный на рис. 29.

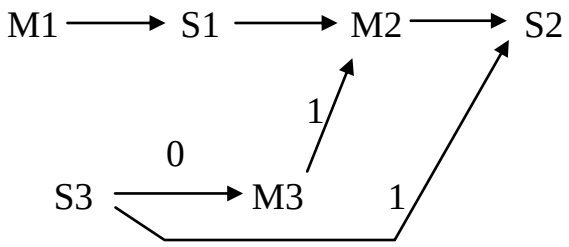

Рис. 29. G'' – фрагмент графа вычислений

У графа G'' два источника: M1 и S3. Предположим, источники перебирались в следующем порядке: M1, S3. Тогда подграф G_1 будет выглядеть так, как это изображено на рис. 28, а подграф G_2 так, как это изображено на рис. 27. Расчет расписания далее пойдет, как и в примере 21. А затем после шагов 18–31, будет вычислен буфер для единственной обратной дуги (он равен 1), а также буферы для операций чтения. В итоге расписание приобретет вид, показанный в табл. 5.

Таблица 5.

№ такта	A	B	C	S1	S2	S3	M1	M2	M3	X(w)	Y(w)	Z(w)
1	-	1	1	-	-	-	-	-	-	-	-	-
2	-	-	-	-	-	1	-	-	-	-	-	-
3	-	-	-	-	-	1	-	-	-	-	-	-
4	1	-	-	-	-	1	-	-	-	-	-	-
5	-	-	-	-	-	-	-	-	1	-	-	1
6	-	-	-	-	-	-	-	-	1	-	-	-
7	-	2	2	-	-	-	-	-	1	-	-	-
8	-	-	-	-	-	2	1	-	1	-	-	-
9	-	-	-	-	-	2	1	-	1	-	-	-
10	2	-	-	-	-	2	1	-	-	-	1	-
11	-	-	-	-	-	-	1	-	2	-	-	2
12	-	-	-	-	-	-	1	-	2	-	-	-
13	-	3	3	1	-	-	-	-	2	-	-	-
14	-	-	-	1	-	3	2	-	2	-	-	-
15	-	-	-	1	-	3	2	-	2	-	-	-
16	3	-	-	-	-	3	2	1	-	-	2	-
17	-	-	-	-	-	-	2	1	3	-	-	3
18	-	-	-	-	-	-	2	1	3	-	-	-
19	-	4	4	2	-	-	-	1	3	-	-	-
20	-	-	-	2	-	4	3	1	3	-	-	-
21	-	-	-	2	1	4	3	-	3	-	-	-
22	4	-	-	-	1	4	3	2	-	-	3	-
23	-	-	-	-	1	-	3	2	4	-	-	4
24	-	-	-	-	-	-	3	2	4	1	-	-
25	-	5	5	3	-	-	-	2	4	-	-	-

2.7. Выводы ко второй главе

В данной главе приведены модификация формулы расчета интервала инициализации итераций и ее обоснование. Предлагается алгоритм поиска всех элементарных циклов на графе, который является основой нового алгоритм расчета минимального *iii*. Последний алгоритм уменьшает объем вычислений. Приводятся обоснования корректности этих алгоритмов.

Также приведены алгоритмы автоматического расчета характеристик конвейера, таких как буферные задержки, определение обратных дуг, и алгоритм составления расписания работы каждого конвейера в рамках многоконвейерной модели вычислений.

3. Многоконвейерные вычисления

3.1. Многоконвейерная модель вычислений

Предположим, что в нашем распоряжении есть суперкомпьютер с достаточно большим числом ресурсов: перестраиваемых конвейеров. Предположим также, что нам необходимо произвести вычисления по следующей формуле:

$$y = A*x+b,$$

где A – матрица размера $N \times N$, а x, y, b – вектора размерности N. Примерами задач, требующих выполнения таких вычислений, могут быть итерационные матричные методы. Вполне естественно записать эти вычисления в виде программы следующим образом:

```
for (i=0; i<N; ++i)
{
     y[i] = b[i];
     for (j=0; j<N; ++j)
          y[i] = y[i]+A[i][j]*x[j];
}
```

Для этой программы представляется вполне логичным организовать вычисления на N конвейерах и запустить их одновременно. При этом на каждом конвейере будет выполняться внутренний цикл по j, а переменная i будет фиксированным числом для каждого конвейера. В этом примере конвейеры могут работать независимо друг от друга, так как на решетчатом графе этого гнезда циклов не будет ни одной горизонтальной дуги (см. рисунки ниже).

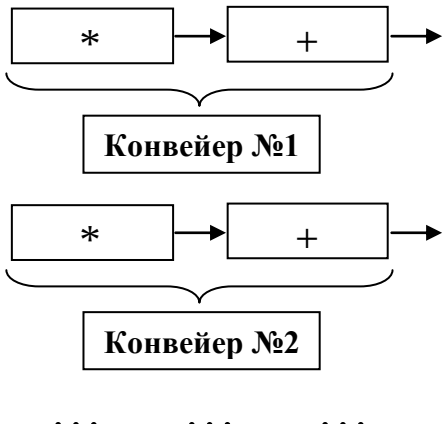

Рис. 30. Организация вычислений без передачи информации между конвейерами

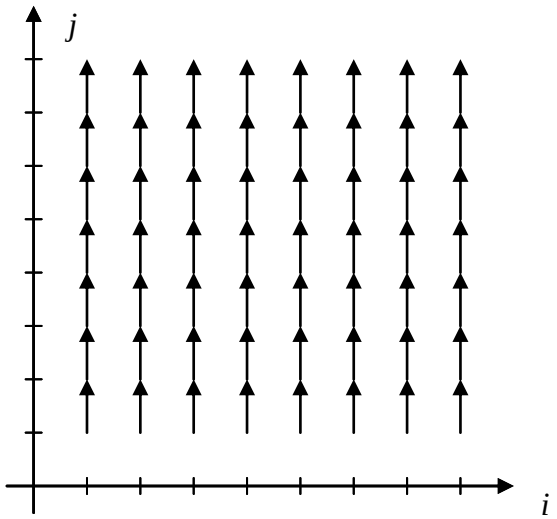

Рис. 31. Решетчатый граф

Рассмотрим теперь процесс вычислений, заданный следующей формулой:

$$x_{ij} = a_{ij} * x_{i-1\,j} + b_{ij} * x_{i\,j-1} + c_{ij},$$

где $A = (a_{ij})$, $B = (b_{ij})$, $C = (c_{ij})$, $X = (x_{ij})$ – матрицы размера $N \times M$. Вычисления по такой формуле могут быть необходимы, например, в сеточных итерационных методах. Вполне естественно записать эти вычисления в виде программы следующим образом:

```
for (i=1; i<N; ++i)
      for (j=1; j<M; ++j)
            x[i][j] = a[i][j]*x[i-1][j]+b[i][j]*x[i][j-1]+c[i][j];
```

Для этой программы представляется вполне логичным организовать вычисления на N конвейерах, аналогично предыдущему примеру на каждом конвейере будет выполняться внутренний цикл по j, а переменная i будет фиксированным числом для

каждого конвейера. Но запустить их одновременно уже не удастся. Это связано с тем, что конвейерам придется обмениваться информацией, т.е. значение $x[k][l]$, вычисленное на k-ом конвейере, потребуется следующему конвейеру для вычисления своего результата $x[k+1][l]$. На решетчатом графе этого цикла будут присутствовать горизонтальные дуги, указывающие на эту проблему. Таким образом, так организованные конвейерные вычисления потребуют синхронизации конвейеров.

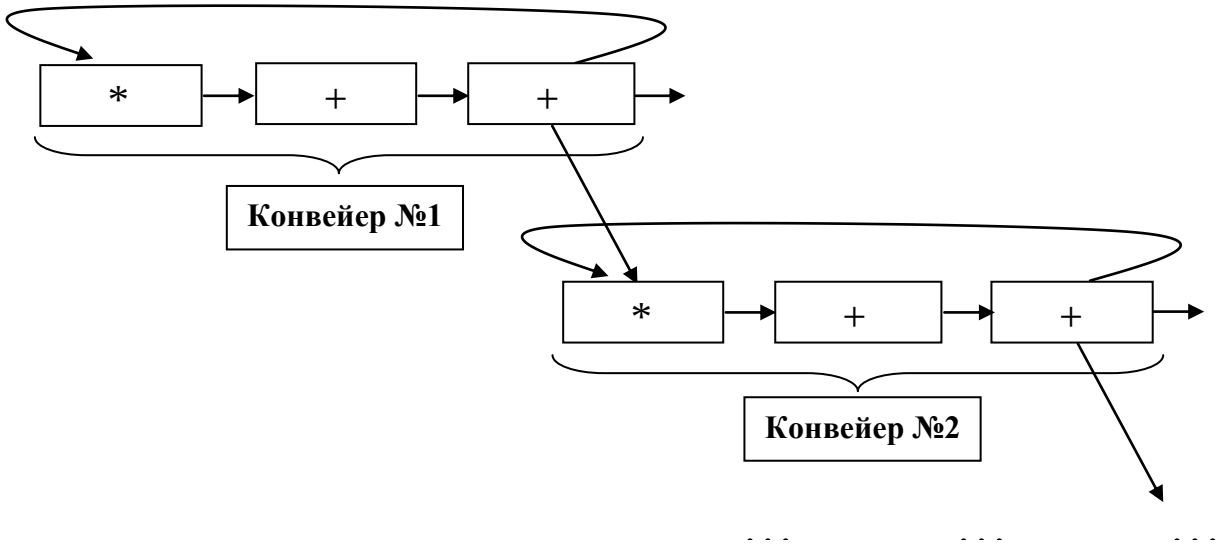

Рис. 32. Организация вычислений с передачей информации между конвейерами.

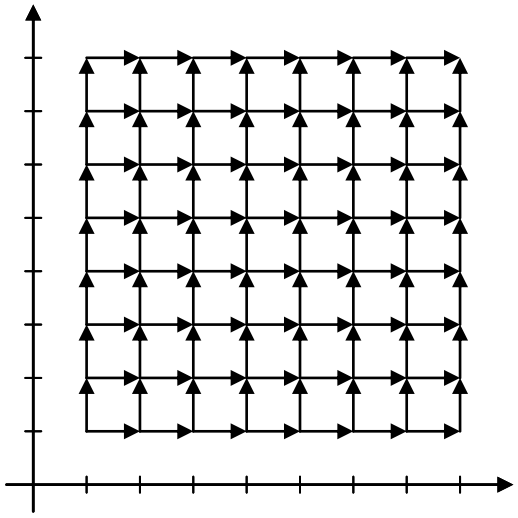

Рис. 33. Решетчатый граф программы, показывающий невозможность ее выполнения на независимых конвейерах

В общем случае можно говорить о семействах конвейеров, между которыми нет передачи данных? и семействами, между конвейерами которых есть передача данных. И это можно проиллюстрировать следующими схемами:

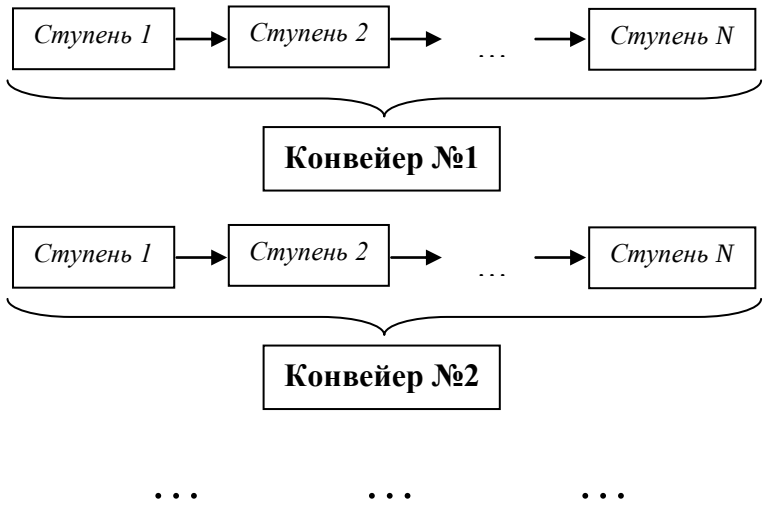

Рис. 34. Организация вычислений без передачи информации между конвейерами

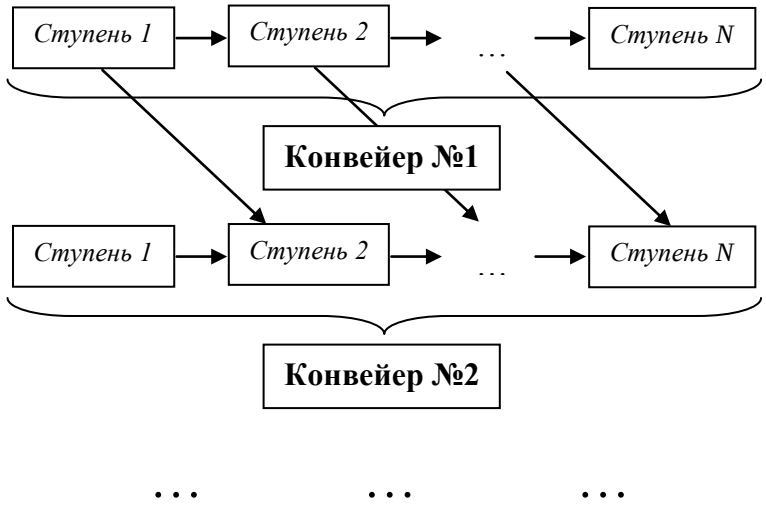

Рис. 35. Организация вычислений с передачей информации между конвейерами

В данной работе будет рассматриваться многоконвейерная вычислительная система с передачей информации между конвейерами и будет вычисляться необходимая задержка между стартами конвейеров, которая определяется этой передачей информации.

3.2. Отображение программ на многоконвейерную модель вычислений

Здесь и далее в этой главе будут рассматриваться многомерные гнезда циклов вида:

```
for(k1=0; k1<N[1]; k1++)
{
      Statements(1)
      for(k2=0; k2<N[2]; k2++)
      {
            Statements(2)
            for(k3=0; k3<N[3]; k3++)                                    (2)
            …
            for(ks=0; ks<N[s]; ks++)
            {
                  Statements(s)
            }
            …
      }
}
```

где Statements(α) — это формальное обозначение сложного оператора, состоящего только из операторов присваивания.

Модель многоконвейерных вычислений подразумевает, что конвейерно будет выполняться внутренний цикл гнезда (2), а счетчики вышестоящих циклов определяют запуски конвейеров. Таких запусков будет N[1]×N[2]×…×N[s-1] штук, причем некоторые из них могут быть параллельными.

Уточним условия, при которых рассматривается отображение гнезда циклов (2) на многоконвейерную модель вычислений.

1. Будем считать, что количество вычислительных ресурсов не ограничено, т.е. в один момент времени мы можем запустить любое количество конвейеров.
2. Будем предполагать, что время дискретно.
3. Будем считать, что время между стартами соседних конвейеров одинаково. Будем называть её задержкой между стартами конвейеров или просто задержкой, и обозначать *z*.

4. Значащие элементы любого n-мерного массива X[k1][k2][…][kn], вхождения которого принадлежат описанному выше фрагменту программы, находятся не только в целочисленном прямоугольнике [0,N(1)]×[0,N(2)]×…×[0,N(n)], но и за его пределами, т.е. для любых неотрицательных (ограничение языка C) k1,k2,…,kn ячейка памяти X[k1][k2][…][kn] содержит значение важное для работы программы. Разумно предположить, что в правильно работающей программе индексные выражения находятся в пределах границ массива.

В дальнейшем эти условия, формирующие математическую модель многоконвейерных вычислений, будем называть *соглашениями*. Отметим, что переход от полученного в итоге решения к решению задачи с ограниченным количеством ресурсов и/или переменным количеством конвейеров, стартующих одновременно, представляется несложным.

Время пересылки данного между вычислительными элементами системы обозначим T_{send}. Обозначим через $N_{\text{pipe}}(I)$ номер конвейера, на котором будет вычисляться итерация I. Тогда $N_{\text{pipe}}(I)$ определяется формулой

$$N_{\text{pipe}}(I) = \left[\sum_{\alpha=1}^{s-2} \left(i_\alpha \prod_{\beta=\alpha}^{s-2} N[\beta+1] \right) + i_{s-1} + 1 \right], \qquad (3)$$

Время выполнения гнезда циклов (2) на вычислительной системе с архитектурой, позволяющей запустить набор конвейеров, вычисляется по формуле:

$$T = L + iii*(N[s] - 1) + z*\left(\prod_{k=1}^{s-1} N[k] - 1 \right), \qquad (4)$$

где T – общее время выполнения гнезда циклов (2), L – время выполнения данных на критическом пути конвейера (все конвейеры по структуре будут одинаковы), iii – интервал инициализации итераций, $N[s]$ – количество итераций внутреннего цикла, z – задержка между стартами конвейеров, $\prod_{k=1}^{s-1} N[k]$ – количество конвейеров.

Для тесного гнезда циклов расписание выполнения фрагмента программы будет строиться как последовательность конвейеров, сформированных по самому вложенному циклу. И учитываться будут только зависимости, которые находятся в блоке операторов Statements(s).

В общем случае предполагается составление расписания сначала для всего семейства конвейеров, а операторы, составляющие Statements(α), α < s будут добавляться в расписание, после того как расписание для всех конвейеров уже составлено. При таком

способе составления расписания, во-первых, желательно знать заранее задержку в стартах конвейеров. Во-вторых, при добавлении в расписание операторов, не вошедших в серию конвейеров, должны быть учтены информационные зависимости между вхождениями, попавшими на конвейер, и вхождениями, не попавшими на конвейер.

Для того чтобы составить расписание работы одного конвейера необходимо знать момент старта самого конвейера, относительные моменты старта вычислений каждой из операций (арифметической, чтения/записи в память и т.п.), занятой в работе конвейера, а также интервал инициализации итераций (*iii*). Под относительными моментами понимается интервал времени, который пройдет с момента старта самого конвейера.

Расписание работы семейства конвейеров задается следующим набором параметров: старт первого в семействе, задержка в стартах конвейеров, моменты старта вычислений каждой из операций (арифметической, чтения/записи в память и т.п.) занятой в работе каждого конвейера относительно момента старта самого конвейера, а также интервал инициализации итераций (*iii*) каждого конвейера. В этом случае момент старта каждого отдельного конвейера легко определить, исходя из его номера и задержки между стартами конвейеров.

Подход к составлению расписания, описанный выше, – не самый простой при настоящих соглашениях, но учитывает тот факт, что в реальных системах количество ресурсов все-таки ограничено и при реализации алгоритмов, изложенных в данной работе, на реальной вычислительной системе в эти алгоритмы вносить существенные изменения не придется. Необходимо будет лишь добавить дополнительные ограничения на возможность вычисления гнезда циклов при данном количестве ресурсов.

Предположим, что в гнезде циклов (2) есть антизависимости. Тогда предлагается это гнездо циклов преобразовать так, чтобы от них избавиться. В этом может помочь преобразование растягивания скаляров (array expansion) или комбинация преобразований: введение временных переменных (temp arrays) и разрезания циклов (loop distribution [76]). С выходными зависимостями внутри цикла нужно бороться аналогичным образом.

Обозначим через Δ пространство итераций цикла (2), т.е. Δ определяется равенством $\Delta = \{I \in Z^s \mid 1 \le i_\alpha \le N(\alpha),\ 1 \le \alpha \le s\}$.

Пусть $I=(i_1, i_2, \ldots, i_s)$, $J=(j_1, j_2, \ldots, j_s)$ некоторые итерации цикла (2), т.е. $I, J \in \Delta$. Введём в рассмотрение бинарное отношение линейного порядка $\prec$ на множестве Z^s следующим образом: $I \prec J$ означает, что существует такое σ $(1 \le \sigma \le s)$, что $i_\alpha = j_\alpha$, для $1 \le \alpha < \sigma$ и $i_\sigma < j_\sigma$. Будем называть это бинарное отношение лексикографическим сравнением.

Пусть $M \subset Z^s$ – конечное множество. Определим lexmax(M) следующим образом: lexmax(M) = I, где $I \in M$ и для любого $J \in M$ такого, что $I \neq J$ выполняется $J \prec I$. Будем называть эту функцию лексикографическим максимумом.

Пример 23

```
for(k1=0; k1<N1; k1++)
for(k2=0; k2<N2; k2++)
{
      x[k1] = y[k2]
      for(k3=0; k3<N3; k3++)
      {
            z[k3] = x[k2] + a[k3]
            for(k1=0; k1<N1; k1++)
                  x[k4] = y[k3+50] + c[k4]
      }
}
```

В соответствии с соглашениями итерация $I = (0,0,0,0)$ будет выполняться на конвейере номер 1, итерация $I = (0,0,1,0)$ будет выполняться на конвейере номер 2, итерация $I = (0,0,2,0)$ будет выполняться на конвейере номер 3 и т.д. Итерация $I = (0,1,0,0)$ будет выполняться на конвейере номер N3 + 1, итерация $I = (0,1,1,0)$ будет выполняться на конвейере номер N3 + 2 и т.д. Итерация $I = (1,0,0,0)$ будет выполняться на конвейере номер N2×N3 + 1, итерация $I = (1,0,1,0)$ будет выполняться на конвейере номер N2×N3 + 2 и т.д.

Конец примера.

Обозначим через $T_{\text{begin}}(I)$ время старта конвейера, на котором будет выполняться итерация $I \in \Delta$. Тогда

$$T_{\text{begin}}(I) = z \times N_{\text{pipe}}(I) \qquad (5)$$

Обозначим через u — генератор, а через v — использование, принадлежащие гнезду циклу (2). Предположим, что на итерации $I \in \Delta$ вычисляется значение для генератора u. Обозначим через $T_{\text{calc}}(u, I)$ время, которое пройдет с момента старта итерации I на конвейере $N_{\text{pipe}}(I)$ до момента, когда будет получен результат вычисления

значения генератора u, принадлежащего самому вложенному циклу. Обозначим через $T_{\text{calc}}(u)$ время, которое потребуется для вычисления значения генератора u, не принадлежащего самому вложенному циклу.

Предположим $I \prec J$. Пусть $T_{\text{gen}}(u, I)$ — это момент времени, когда будет окончено вычисление значения генератора u на итерации I. Этот момент времени определяется моментом старта конвейера и моментом старта итерации I относительно этого конвейера.

$$T_{\text{gen}}(u, I) = T_{\text{begin}}(I) + iii \times i_s + T_{\text{calc}}(u, I) \quad (6)$$

Пусть $T_{\text{use}}(v, J)$ — это момент времени, когда потребуется значение использования v на итерации J.

$$T_{\text{use}}(v, J) = T_{\text{begin}}(J) + iii \times j_s \quad (7)$$

Тогда разность $T_{\text{use}}(v, J)$ и $T_{\text{gen}}(u, I)$ определяет промежуток времени между этими событиями, т.е. время, допустимое для передачи данного. Обозначим через $T_{\text{pipe}}(u, v, I, J)$ допустимое время передачи данного.

$$T_{\text{pipe}}(u, v, I, J) = T_{\text{begin}}(J) + iii \times j_s - (T_{\text{begin}}(I) + iii \times i_s + T_{\text{calc}}(u, I)) \quad (8)$$

Надо отметить, что все описанные обозначения введены без учета факта наличия/отсутствия информационной зависимости между генератором u на итерации I и использованием v на итерации J.

3.3. Влияние зависимостей в самом вложенном цикле на задержку в стартах конвейеров

Предположим, что u генератор, а v использование, принадлежащие самому вложенному циклу в гнезде циклов (2). Предположим также, что использование v на итерации J зависит от генератора u на итерации I. Тогда время $T_{\text{pipe}}(u, v, I, J) - T_{\text{send}}$ должно быть не отрицательно. Следовательно,

$$z \times \left[\sum_{\alpha=1}^{s-2} \left((j_\alpha - i_\alpha) \prod_{\beta=\alpha}^{s-2} N(\beta+1) \right) + j_{s-1} - i_{s-1} \right] + iii \times (j_s - i_s) - T_{\text{calc}}(u, I) - T_{\text{send}} \geq 0$$

Обозначим через d дугу графа информационных связей, которая соединяет вхождения u и v. Обозначим через

$$T_1(d, I, J) = \frac{T_{\text{calc}}(u, I) + T_{\text{send}} - iii \times (j_s - i_s)}{\sum_{\alpha=1}^{s-2}\left((j_\alpha - i_\alpha)\prod_{\beta=\alpha}^{s-2} N(\beta+1)\right) + j_{s-1} - i_{s-1}}$$

Тогда

$z \geq T_1(d, I, J)$

Введем в рассмотрение G — граф информационных связей для тела самого вложенного цикла в гнезде циклов (2). Для каждого $d \in G$ система функций $\{F_{d,\alpha}\}$, $1 \leq \alpha \leq p_d$ описывает элементарный решетчатый граф, построенный для зависимости d, а набор выпуклых множеств $\{D_{d,\alpha}\}$, $1 \leq \alpha \leq p_d$ является набором областей определения для соответствующих функций $F_{d,\alpha}$. Тогда

$$z \geq \max_{d \in G} \max_{1 \leq \alpha \leq p_d} \max_{J \in D_\alpha} T_1(d, F_{d,\alpha}(J), J) \qquad (9)$$

Таким образом, доказана следующая теорема.

Теорема 3. Для организации вычисления гнезда циклов (2) на многоконвейерной вычислительной системе эквивалентно последовательному выполнению достаточно, чтобы для задержки z между стартами конвейеров выполнялось неравенство (9).

3.4. Влияние зависимостей между вхождениями самого вложенного цикла и вхождениями в остальных циклах гнезда на задержку в стартах конвейеров

В этом разделе мы покажем влияние цепочек вхождений на задержку между стартами конвейеров. Для этого нам потребуются еще несколько обозначений.

Пусть вхождения u и v принадлежат самому вложенному циклу в гнезде циклов (2), причем u — генератор, а v — использование. Пусть $\{u_\alpha\}$ и $\{v_\alpha\}$, $1 \leq \alpha \leq q$ последовательности вхождений, обладающие следующими свойствами:

1. Последовательность $\{u_\alpha\}$ — последовательность генераторов, а $\{v_\alpha\}$ — последовательность использований.
2. Для любого $\alpha \in [1; q]$ вхождения u_α и v_α принадлежат одному оператору.

3. На графе вычислений существует путь, начинающийся в вершине u и заканчивающийся в вершине v, проходящий через все элементы последовательностей $\{u_\alpha\}$ и $\{v_\alpha\}$,.

Определение 31. В приведенных обозначениях, цепочкой вхождений будем называть последовательность вхождений $\{w_\alpha\}$, $1 \le \alpha \le 2q + 2$, в которой $w_1 = u$, $w_{2q+2} = v$, а для $\alpha \in [1; q]$: $w_{2\alpha} = v_\alpha$, $w_{2\alpha+1} = u_\alpha$.

Пример 24

```
for(i=0; i<=10; i++)
{
     z[i][i] = x[i][i] + a[i];
     y[i][i+6] = z[i-6][i-6] + b[i];

     for(j=0; j<=20; j++)
          x[i+1][j] = y[i][j+10] + c[j];
}
```

Обозначим операторы по порядку сверху вниз: Statement1, Statement2, Statement3.

Обозначим через u вхождение переменной x(·) в левую часть оператора Statement3. Обозначим через u_1 вхождение переменной z(·) в левую часть оператора Statement1. Обозначим через u_2 вхождение переменной y(·) в левую часть оператора Statement2. Обозначим через v вхождение переменной y(·) в правую часть оператора Statement3. Обозначим через v_1 вхождение переменной x(·) в правую часть оператора Statement1. Обозначим через v_2 вхождение переменной z(·) в правую часть оператора Statement2. Тогда последовательность u, v_1, u_1, v_2, u_2, v является цепочкой вхождений. Граф информационных связей для этого примера изображен на следующем рисунке.

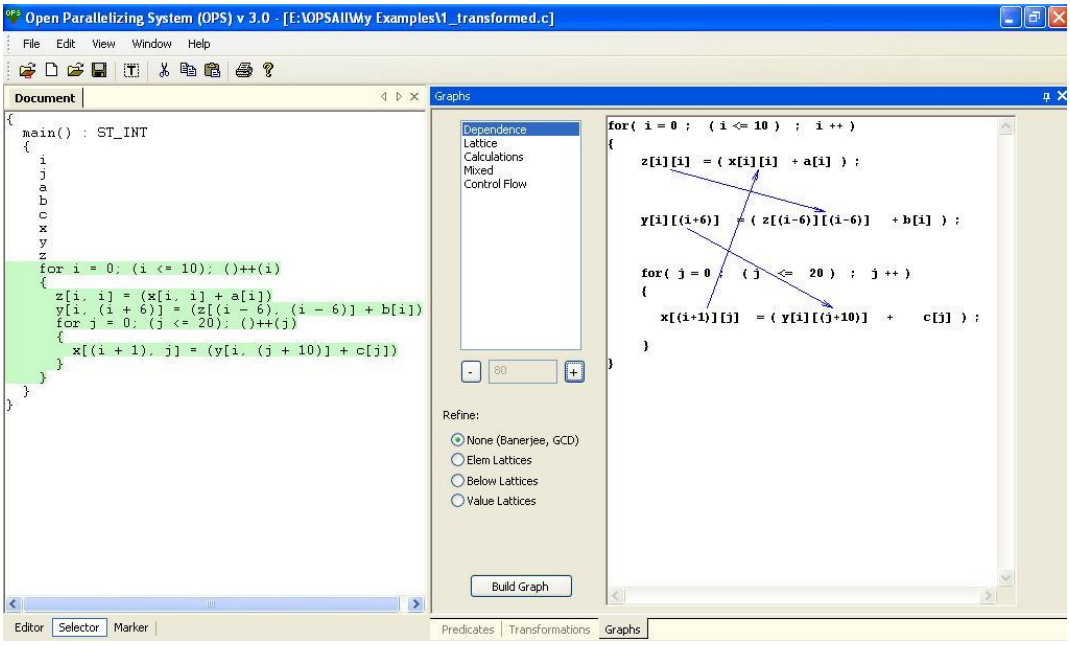

Рис. 36. Экранная форма OPSDemo3. Синие стрелки изображают истинную (потоковую) зависимость

Заметим, что данное, записанное на итерации (1; 2) генератором u в ячейку памяти `x[2][2]`, затем будет считано вхождением v_1 на итерации (2). Данное, считанное вхождением v_1 на итерации (2), будет являться аргументом для вычисления результата сложения в операторе Statement1, который будет записан генератором u_1 на той же итерации в ячейку памяти `z[2][2]`. Данное, записанное на итерации (2) генератором u_1, затем будет считано вхождением v_2 на итерации (8). Данное, считанное вхождением v_2 на итерации (8), будет являться аргументом для вычисления результата сложения в операторе Statement2, который будет записан генератором u_2 на той же итерации в ячейку памяти `y[8][14]`. Данное, записанное на итерации (8) генератором u_2, затем будет считано вхождением v на итерации (8; 4) и будет использоваться для вычисления `x[9][4]` в генераторе u. Таким образом, значение `x[2][2]` проходит цепочку вхождений, а `x[9][4]` опосредованно через всю цепочку вхождений зависит от `x[2][2]`.

Надо принять во внимание тот факт, что данное `x[2][2]` вычисляется на конвейере с номером 2, а `x[9][4]` на конвейере с номером 9. Для этого определим моменты времени, когда в расписании конвейеров будут выполняться итерации (1; 2) и (8; 4), на которых вычисляются значения `x[2][2]` и `x[9][4]` соответственно и разность между этими моментами времени. Воспользуемся формулой (8).

$$T_{pipe}(u, v, (1; 2), (8; 4)) = T_{begin}(8; 4) + iii \times j_s - (T_{begin}(1; 2) + iii \times i_s + T_{calc}(u, (1; 2)))$$

Так как внутри конвейеров зависимости отсутствуют, то $iii = 1$. Воспользуемся формулой (5).

$$T_{pipe}(u, v, (1; 2), (8; 4)) = z \times 9 + 4 - (z + 2 + T_{calc}(u, (1; 2))) = z \times 8 + 2 - T_{calc}(u, (1; 2)))$$

Теперь вычислим минимальное время, необходимое для прохождения цепочки вхождений, значением, вычисленным генератором u на итерации $(1; 2)$, чтобы в конце цепочки быть использованным на итерации $(8; 4)$. Обозначим это время $T_{chain}(u, v, (1; 2), (8; 4))$. Тогда $T_{chain}(u, v, (1; 2), (8; 4))$ вычисляется как сумма времени пересылки данного из u в v_1, времени вычисления значения для генератора u_1, времени пересылки данного из u_1 в v_2, времени вычисления значения для генератора u_2 и времени пересылки данного из u_2 в v.

$$T_{chain}(u, v, (1; 2), (8; 4)) = T_{send} + T_{calc}(u_1) + T_{send} + T_{calc}(u_2) + T_{send}$$

Тогда величина z должна быть выбрана таким образом, чтобы $T_{chain}(u, v, (1; 2), (8; 4)) < T_{pipe}(u, v, (1; 2), (8; 4))$. В противном случае, оператор Statement3 на итерации $(8; 4)$ начнет вычисляться раньше, чем будет готово данное для использования v, которое, следуя последовательной логике программы, должно быть готово к началу вычисления оператора Statement3. Таким образом, величина задержки в стартах конвейеров должна удовлетворять неравенству

$$z > \{T_{calc}(u, (1; 2))) + 2 + T_{chain}(u, v, (1; 2), (8; 4))\} / 8$$

Учитывая тот факт, что данные аналогично проходят по цепочке от вхождения u на итерациях $(0; 1)$, $(2; 3)$, $(3; 4)$ к вхождению v на итерациях $(7; 3)$, $(9; 5)$, $(10; 6)$ соответственно, следовательно, должны быть составлены еще 3 неравенства для ограничения величины z. Обоснование того факта, что этих неравенств в этом примере всего 4 и что они порождаются именно описанным здесь набором итераций гнезда циклов, будет приведено далее.

В заключении к этому примеру, хотелось бы отметить, что циклы такой длины не очень интересно конвейеризовать. Но границы циклов в этом примере выбраны малыми величинами, лишь для наглядности и вполне очевидно, что при циклах большей длины может сложиться аналогичная ситуация с зависимостями внутри циклов.

Конец примера.

Пример 25

```
for(i=0; i<=10; ++i)
{
      z[i][i] = x[i][i] + a[i];
      y[i][i+6] = z[i-6][i-6] + b[i];

      for(j=0; j<=20; ++j)
            x[i+1][j] = y[i][j-10] + c[j];
}
```

Введем обозначения вхождений и операторов аналогично примеру 24. Этот пример отличается от предыдущего только индексным выражением для вхождения *v*. Но в результате не может быть составлено ни одного неравенства, ограничивающего величину *z*.

Действительно, генератор *u* на итерации (0;1) записывает данное в ячейку памяти `x[1][1]`, а затем это данное будет считано вхождением v_1 на итерации (1). Данное, считанное использованием v_1 на итерации (1), будет являться аргументом для вычисления результата сложения в операторе Statement1, который будет записан генератором u_1 на той же итерации в ячейку памяти `z[1][1]`. Данное, записанное на итерации (1) генератором u_1, затем будет считано использованием v_2 на итерации (7). Данное, считанное использованием v_2 на итерации (7), будет являться аргументом для вычисления результата сложения в операторе Statement2, который будет записан генератором u_2 на той же итерации в ячейку памяти `y[7][13]`. Данное, записанное на итерации (7) генератором u_2, не будет считано использованием *v* ни на одной итерации. Таким образом, значение `x[1][1]` не пройдет цепочку вхождений.

Аналогично и на других итерациях этого гнезда циклов данные будут проходить лишь часть цепочки вхождений.

Конец примера.

Обозначим через $\{d_\alpha\}$, $1 \le \alpha \le q + 1$ последовательность дуг на графе вычислений (и на графе информационных связей) таких, что $d_\alpha = (w_{2\alpha-1}, w_{2\alpha})$ для $\alpha \in [1; q + 1]$.

Напомним, что время работы каждого конвейера определяется интервалом инициализации итераций и длиной критического пути, а друг относительно друга соседние конвейеры работают со сдвигом в *z* итераций. Таким образом, можно определить время, которое будет работать последовательность конвейеров, начиная с того момента,

когда вхождение *u* сгенерирует данное, и до того момента, когда использование *v* должно получить данное. Отметим, что за это время должны произойти все пересылки данных и выполнение всех операторов, которые находятся между вхождениями, принадлежащими цепочке.

Для каждой дуги d_α графа информационных связей построим элементарный решетчатый граф в виде семейства аффинных функций $F_\alpha = \{F_{\alpha,\beta}\}$, $1 \le \beta \le m(\alpha)$, где $m(\alpha) = p(d_\alpha)$ в соответствии с работой [14]. Набор областей определения соответствующих функциям из последовательности $\{F_{\alpha,\beta}\}$ обозначим через $\{D_{\alpha,\beta}\}$, $1 \le \beta \le m(\alpha)$. Обозначим также через $D_\alpha = D_{\alpha,1} \cup D_{\alpha,2} \cup \ldots \cup D_{\alpha,m(\alpha)}$ всю область определения для функции F_α.

Определение 32. Будем называть функцией зависимости для цепочки вхождений $\{w_\alpha\}$ композицию функций следующего вида: $F = F_1 \circ F_2 \circ \ldots \circ F_{q+1}$ с естественной областью определения *D*, имеющей следующий вид:

$$D = D_{q+1} \cap F_{q+1}^{-1}\left(\ldots D_3 \cap F_3^{-1}\left(D_2 \cap F_2^{-1}(D_1)\right)\ldots\right)$$

Замечание 10. Определение функции зависимости имеет смысл только при $D \neq \varnothing$.

Определение 33. Путь на графе вычислений, проходящий через все элементы цепочки $\{w_\alpha\}$, будем называть *влияющим на задержку путем* в том случае, если для этой цепочки существует функция зависимости, т.е. $D \neq \varnothing$.

Поясним последние определения на примере.

Пример 26

```
for(i=0; i<=10; i++)
{
     z[i][i] = x[i][i] + a[i];
     y[i][i+6] = z[i-6][i-6] + b[i];

     for(j=0; j<=20; j++)
          x[i+1][j] = y[i][j+10] + c[j];
}
```

Вернемся к рассмотрению фрагмента программы из примера 24. Обозначим все вхождения и операторы так, как это уже было сделано в примере 24.

Многоконвейерная модель подразумевает 11 запусков конвейерных вычислений. Для каждого запуска внутренний цикл, соответствующий значению счетчика `i`, отображается на конвейерный вычислитель, а также отображается еще два оператора (Statement1 и Statement2), выполняемые последовательно, которые необходимо включить в расписание работы вычислительной системы с учетом зависимостей.

Введем последовательность дуг $\{d_\alpha\}$ как описано выше, т.е. $d_1 = (u, v_1)$, $d_2 = (u_1, v_2)$, $d_3 = (u_2, v)$.

Рассмотрим, на каких итерациях, по каким из дуг $\{d_\alpha\}$ передаются данные. Для этого необходимо рассмотреть элементарные решетчатые графы для каждой из дуг последовательности $\{d_\alpha\}$. Решетчатый граф F_1 для пары вхождений d_1 соединяет точки вида $(k, k + 1)$ с точками $(k + 1)$, для $k \in [0,9]$.

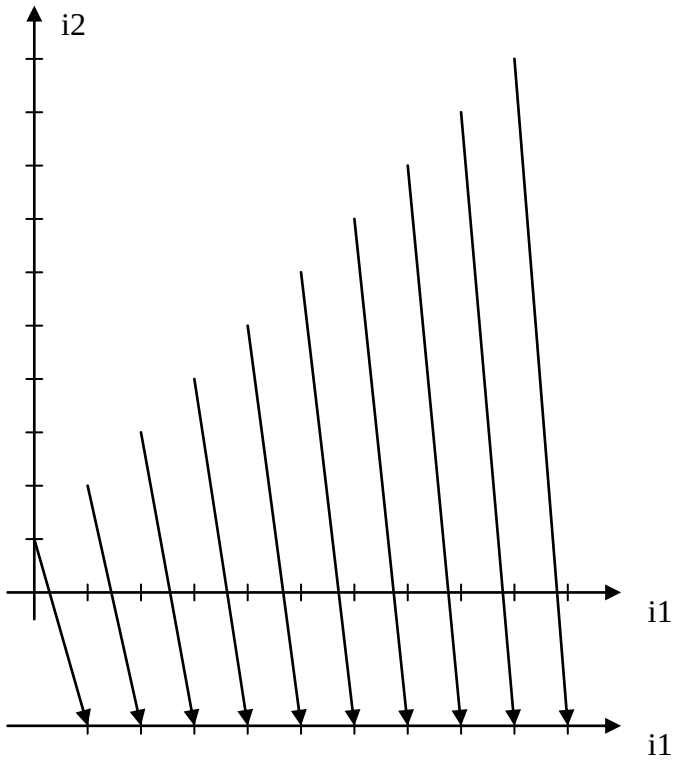

Рис. 37. Решетчатый граф для пары вхождений $d_1 = (u, v_1)$

Решетчатый граф F_2 для пары вхождений d_2 соединяет точки вида (k) с точками вида $(k + 6)$, для $k \in [0, 4]$.

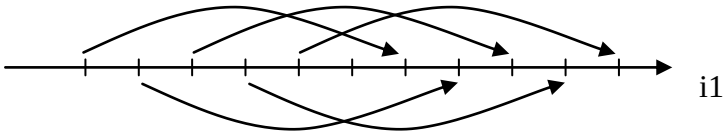

Рис. 38. Фрагмент решетчатого графа для пары вхождений $d_2 = (u_1, v_2)$

Решетчатый граф F_3 для пары вхождений d_3 соединяет точки вида (k) с точками $(k, k-4)$, для $k \in [4, 10]$.

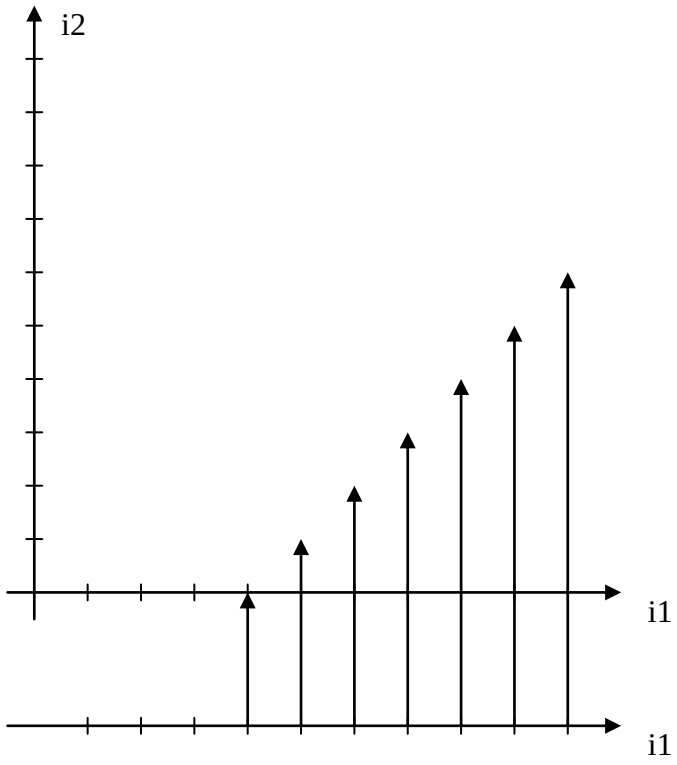

Рис. 39. Фрагмент решетчатого графа для пары вхождений $d_3 = (u_2, v)$

В данном примере каждый из трех решетчатых графов представляется в виде одной аффинной функции, что, конечно, упрощает рассмотрение примера. Напомним также, что семейство аффинных функций, в виде которого удобно хранить решетчатые графы, обладает тем свойством, что область определения этих функций– это множества концов дуг, а область действия — это множество начал дуг решетчатого графа. Итак, построим множество D:

$$D = D_3 \cap F_3^{-1}\left(D_2 \cap F_2^{-1}\left(D_1\right)\right)$$

Области определения задаются следующими соотношениями:

$D_1 = \{i_1 \in Z | \ 1 \le i_1 \le 9\}$, $D_2 = \{i_1 \in Z | \ 6 \le i_1 \le 10\}$,

$D_3 = \{I \in Z^2 | \ i_1 - i_2 = 4, \ i_1 \in [0, 10], \ i_2 \in [0, 20]\} = \{(4, 0), (5, 1), (6, 2), (7, 3), (8, 4), (9, 5), (10, 6)\}$

Теперь несложно вычислить, что

$D = \{I \in Z^2 | \ I \in D_3, \ i_1 \ge 7\} = \{(7, 3), (8, 4), (9, 5), (10, 6)\}$.

Таким образом, путь на графе вычислений, проходящий через цепочку вхождений $(u, v_1, u_1, v_2, u_2, v)$, является путем, влияющим на задержку в стартах конвейеров. А, следовательно, величина задержки z должна выбираться с учетом неравенств, о которых шла речь в примере 24.

Конец примера.

С помощью механизма, использованного в примере, можно определить, что путь, образованный цепочкой вхождений на графе вычислений, не является влияющим на задержку путем.

Пример 27

```
for(i=0; i<=10; i++)
{
     z[i][i] = x[i][i] + a[i];
     y[i][i+6] = z[i-6][i-6] + b[i];

     for(j=0; j<=20; j++)
          x[i+1][j] = y[i][j-10] + c[j];
}
```

Введем обозначения вхождений и операторов аналогично примеру 25. Но теперь в отличие от примера 25, вывод о том, что в цикле нет зависимостей, влияющих на задержку в стартах конвейеров, будет математически обоснован. И сделано это будет с помощью механизма построения функции зависимости для цепочки вхождений $(u, v_1, u_1, v_2, u_2, v)$.

В данном примере имеют место тоже три решетчатых графа, и все они состоят из одной выпуклой области с заданной на ней одной аффинной функцией. Итак,

$D_1 = \{i_1 \in Z \mid 1 \le i_1 \le 9\}$, $D_2 = \{i_1 \in Z \mid 6 \le i_1 \le 10\}$,

$D_3 = \{I \in Z^2 \mid i_1 - i_2 = -16, i_1 \in [0, 10], i_2 \in [0, 20]\} = \{(0, 16), (1, 17), (2, 18), (3, 19), (4, 20)\}$

Тогда

$D = \{I \in Z^2 \mid I \in D_3, i_1 \ge 7\} = \varnothing$.

Из чего сразу можно сделать вывод, что ни одно данное вычисляемое в цикле не проходит по всей цепочке вхождений (u, v_1, u_1, v_2, u_2, v).

Конец примера.

Теорема 4. Для каждой цепочки вхождений существует не более одного влияющего на задержку пути.

Эта теорема несложно доказывается, исходя из определения 31.

Итак, предположим, что на графе вычислений есть цепочка вхождений $\{w_\alpha\}$, начинающаяся с генератора u и заканчивающаяся использованием v. Предположим также, что для этой цепочки существует функция зависимости F с областью определения D и соответственно влияющий на задержку путь. Вычислим время, которое пройдет с момента как некоторое значение будет вычислено для генератора u на итерации I до того момента, когда потребуется аргумент для использования v на итерации J, прошедший по всему влияющему на задержку пути.

$$T_{\text{pipe}}(u, v, I, J) = z \times \left[\sum_{\alpha=1}^{s-2} \left((j_\alpha - i_\alpha) \prod_{\beta=\alpha}^{s-2} N(\beta+1) \right) + j_{s-1} - i_{s-1} \right] + iii \times (j_s - i_s) - T_{\text{calc}}(u, I)$$

Теперь вычислим время, которое потребуется тому же данному на прохождение пути, влияющего на задержку. Обозначим это время через $T_{\text{chain}}(u, v, I, J)$.

$$T_{\text{chain}}(u, v, I, J) = q \times T_{\text{send}} + \sum_{\alpha=1}^{q} T_{calc}(w_{2\alpha+1})$$

Введем вспомогательную функцию $T_1(u, v, I, J)$.

$$T_1(u, v, I, J) = \frac{T_{\text{calc}}(u, I) + q \times T_{\text{send}} + \sum_{\alpha=1}^{q} T_{calc}(w_{2\alpha+1}) - iii \times (j_s - i_s)}{\sum_{\alpha=1}^{s-2} \left((j_\alpha - i_\alpha) \prod_{\beta=\alpha}^{s-2} N(\beta+1) \right) + j_{s-1} - i_{s-1}}$$

Очевидно, что $T_{\text{pipe}}(u, v, I, J) - T_{\text{chain}}(u, v, I, J) \geq 0$. Тогда задержка в стартах конвейеров должна быть выбрана с учетом этого неравенства.

$$z \geq T_1(u, v, I, J)$$

Теперь надо учесть тот факт, что $I = F(J)$, а $J \in D$. Тогда

$$z \geq \max_{J \in D} T_1(u, v, F(J), J)$$

Предположим теперь, что на графе вычислений есть семейство цепочек вхождений, которые порождают влияющие на задержку пути. Обозначим это семейство $\{W_\alpha\}$, $1 \leq \alpha \leq r$, причем $W_\alpha = \{w_{\alpha, \beta}\}$, $1 \leq \beta \leq 2q(\alpha)+2$. Для каждой цепочки W_α построим функцию зависимости F_α. Получим семейство функций зависимости $\{F_\alpha\}$, $1 \leq \alpha \leq r$ с областями определения D_α. Тогда задержка в стартах конвейеров должна удовлетворять следующему неравенству.

$$z \geq \max_{1 \leq \alpha \leq r} \max_{J \in D_\alpha} T_1(w_{\alpha, 1}, w_{\alpha, 2q(\alpha)+2}, F_\alpha(J), J) \qquad (10)$$

Основной результат этого параграфа может быть сформулирован в виде следующей теоремы.

Теорема 5. Для организации вычисления гнезда циклов (2) на многоконвейерной вычислительной системе эквивалентно последовательному выполнению достаточно, чтобы для задержки z между стартами конвейеров выполнялось неравенство (10).

3.5. Составление расписания для вычисления гнезда циклов

Теорема 6. При отображении гнезда циклов (1) на многоконвейерную модель вычислений достаточно, чтобы задержка между стартами конвейеров z удовлетворяла неравенствам (7) и (10).

Достаточность следует из теорем 3, 5 и доказуема только в рамках модели, требования к которой описаны в главе 3.

Предположим, что в гнезде циклов (2) есть истинные информационные зависимости только между вхождениями, принадлежащими составному оператору Statements(s). Тогда предлагается цикл разрезать (loop distribution) таким образом, чтобы получить тесное гнездо циклов, что упростит задачу составления расписания.

Предположим, что в гнезде циклов (2) есть дуга информационной зависимости, ведущая от генератора *u*, принадлежащего составному оператору Statements(s), к использованию *v*, принадлежащему составному оператору Statements(α), $\alpha < s$. Если других информационных зависимостей во фрагменте программы не существует, то на задержку в стартах конвейеров зависимость между вхождениями *u* и *v* не влияет. Это обосновывается тем, что конвейеры могут работать независимо от Statements(α), а операторы в блоке Statements(α) можно поставить в расписание сразу после получения необходимого результата в генераторе *u*.

Предположим, что в гнезде циклов (2) есть дуга информационной зависимости, ведущая от генератора *u*, принадлежащего составному оператору Statements(α), $\alpha < s$, к использованию *v*, принадлежащему составному оператору Statements(s). Если других информационных зависимостей во фрагменте программы не существует, то на задержку в стартах конвейеров зависимость между вхождениями *u* и *v* не влияет. Это обосновывается тем, что конвейеры могут работать независимо от Statements(α), а операторы в блоке Statements(α) можно поставить в расписание сразу после получения необходимого результата в генераторе *u*. Более того, зависимость между вхождениями *u* и *v* влияет на задержку в стартах конвейеров только в том случае, если существует влияющий на задержку путь, проходящий через дугу (*u*, *v*). И, следовательно, если влияющих на задержку путей на графе вычислений нет, то на задержку в стартах конвейеров влияют только те зависимости, вхождения которых находятся в самом вложенном цикле.

Приведем алгоритм, который использует результаты, описанные в предыдущих параграфах.

Алгоритм вычисления параметров, определяющих расписание выполнения гнезда циклов на многоконвейерной архитектуре

1. Устраняем все анти- и выходные зависимости из гнезда циклов.
2. Строим граф информационных связей.
3. Находим все зависимости между вхождениями самого внутреннего цикла.
4. Строим граф вычислений.
5. Находим все пути на графе вычислений, которые проходят через цепочки вхождений.
6. Среди полученного множества путей отбираем влияющие на задержку пути.
7. Для самого вложенного цикла строим усовершенствованный граф вычислений.

8. По усовершенствованному графу вычислений находим интервалы инициализации итераций и определяем моменты старта операций задействованных в работе конвейеров.
9. Вычисляем все оценки для задержки между стартами конвейеров. Выбираем из них максимум, который и будет величиной задержки между стартами конвейеров.

3.6. Формула вычисления задержки в стартах конвейеров для тесного двумерного гнезда циклов

3.6.1. Постановка задачи

Качество распараллеливаемости цикла на компьютере с несколькими процессорами с разделённой или разделяемой памятью можно рассматривать по нескольким параметрам: средняя загруженность каждого процессора, объём вычислений выполняемых последовательно, количество пересылок данных и т.д. Но сама возможность распараллеливания или преобразования цикла к распараллеливаемому виду, так же как и все эти параметры качества распараллеливаемости, зависит от зависимостей между данными. Так, например, есть циклы, которые ни распараллеливать, ни преобразовывать нельзя именно ввиду этих зависимостей.

В данном параграфе будет рассмотрен двумерный цикл с одной информационной зависимостью:

```
for(i1=1; i1<=N1; ++i1)
      for(i2=1; i2<=N2; ++i2)
      {
            X[a1*i1+c1*i2+d11][a2*i1+c2*i2+d12] = …
            … =… X[b1*i1+c1*i2+d21][b2*i1+c2*i2+d22];
      }
```

(11)

где параметры N1,N2,a1,a2,b1,b2,c1,c2,d11,d12,d21,d22 принимают целочисленные значения. Влияние нескольких информационных зависимостей на задержку в стартах конвейеров можно оценить как максимум задержек, вычисленных для каждой отдельной зависимости.

Предположим, этот цикл должен быть реализован на многоконвейерной архитектуре, т.е. конвейерно будет выполняться внутренний цикл, а запусков конвейеров будет N1 штук.

Условия, при которых возможно отображение на многоконвейерную модель, смотри в §3.2.

В дальнейшем эти условия будем называть условиями на математическую модель или соглашениями. Отметим, что переход от полученного в итоге решения к решению задачи с ограниченным количеством ресурсов и/или переменным количеством конвейеров стартующих одновременно, представляется несложным.

Целью данной работы является выявление циклически порожденных зависимостей в цикле (11) и вычисление задержки между стартами конвейеров. Это особенно актуально для суперкомпьютеров с перестраиваемой архитектурой, потому что для этого класса компьютеров есть возможность работать с любым циклом такого вида, но нужно соблюсти все зависимости между данными.

Понятно, что при описанных допущениях на математическую модель задержка будет определяться информационными зависимостями между итерациями цикла (11). Для нахождения её потребуется построить граф, который будет описывать информационные зависимости между итерациями цикла.

В начале рассмотрим произвольную не вертикальную дугу решетчатого графа и посмотрим, как она влияет на задержку между стартами конвейеров. Предположим, что эта дуга единственная и пусть её начало (i_1; i_2), а конец (j_1; j_2). Так как дуга не вертикальная, то $i_1 < j_1$. Предположим, что $i_2 = j_2$. Тогда задержка между стартами конвейеров с номерами i_1 и j_1, в соответствии с соглашениями, будет равно $T_{it} + T_{send}$, где T_{it} – время выполнения одной итерации внутреннего цикла, а T_{send} – время пересылки данного. Предположим, что $i_2 > j_2$. Тогда задержка между стартами i_1 и j_1 конвейеров будет равна $T_{it} + T_{send} + i_2 - j_2$. Предположим, что $i_2 < j_2$. Тогда задержка между стартами конвейеров с номерами i_1 и j_1 будет равна 0, при $j_2 - i_2 \geq T_{it} + T_{send}$, и равна $T_{it} + T_{send} + i_2 - j_2$, при $j_2 - i_2 < T_{it} + T_{send}$.

Теперь введём следующее обозначение: z[i_1; j_1] – это минимально-допустимая задержка между i_1 и j_1 конвейерами. Тогда для случая единственной не вертикальной дуги в решетчатом графе справедливо следующее равенство:

$$z[i_1; j_1] = \max_{\substack{i_2, j_2: \\ (i_1;i_2;j_1;j_2)\in \mathbf{G}}} \begin{cases} T_{it} + T_{send} + i_2 - j_2, & T_{it} + T_{send} + i_2 - j_2 > 0; \\ 0, & T_{it} + T_{send} + i_2 - j_2 \leq 0. \end{cases}$$

Введём в рассмотрение следующую функцию:

$$w(x) = \begin{cases} x, & x > 0; \\ 0, & x \leq 0. \end{cases}$$

Тогда в новых обозначениях:

$$z[i_1; j_1] = \max_{\substack{i_2, j_2: \\ (i_1;i_2;j_1;j_2)\in \mathbf{G}}} \mathrm{w}(T_{it} + T_{send} + i_2 - j_2).$$

Отметим, что с учётом соглашения 3 (см. §3.1) задержка z между стартами соседних конвейеров выражается формулой:

$$z = \max_{1 \le i_1 < j_1 \le N_1} \frac{z[i_1; j_1]}{j_1 - i_1}, \qquad (12)$$

причём она ограничена снизу 0 ввиду соглашения 1.

В частном случае, если граф состоит из одной дуги с началом $(i_1; i_2)$ и концом $(j_1; j_2)$:

$$z = \frac{\mathrm{w}(T_{it} + T_{send} + i_2 - j_2)}{j_1 - i_1}.$$

Соглашение 4, которое ещё не было упомянуто, позволяет нам не акцентировать внимание на индексных выражениях в массиве. Если же мы ограничимся прямоугольными пространствами данных, придётся в алгоритме поставить дополнительные ограничения на индексные выражения. Более того, для некоторых пар пространство данных – пространство итераций возникнет два вопроса: почему в цикле происходит обращение к ячейке памяти вне прямоугольного пространства данных и какое решение принимать в такой ситуации?

Пример 28

Пусть дан цикл следующего вида:

```
for(i1=1; i1<=N1; ++i1)
      for(i2=1; i2<=N2; ++i2)
      {
            X[i1-i2][2*i1+i2] = …
            … =… X[i1-i2+1][i1+i2+1];
      }
```

Решетчатый граф для этого цикла (N1=N2=8) изображен на рис. 40. Этот граф показывает, что на 5-ой итерации внешнего цикла существует зависимость от данных полученных на 4-ой итерации внешнего цикла, т.е. работа 5-го конвейера должна быть синхронизирована с работой 4-го конвейера так, чтобы не нарушить логику последовательных вычислений.

Итак, $z[4;5] = w(T_{it} + T_{send} - 2)$, $z[6;8] = w(T_{it} + T_{send} - 3)$, для других пар значений i_1, j_1: $z[i_1; j_1] = 0$. Тогда $z = w(T_{it} + T_{send} - 2)$.

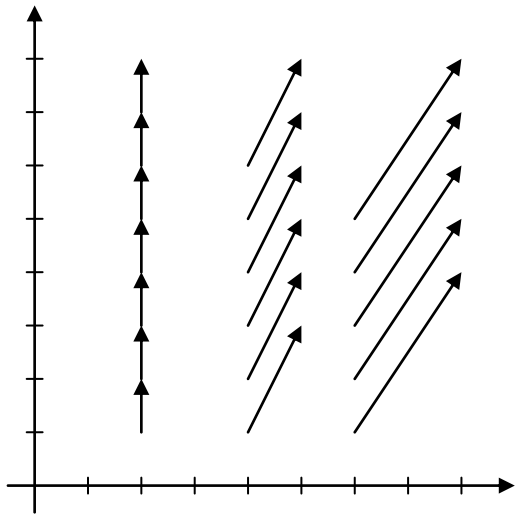

Рис. 40. Решетчатый граф для примера 28

Конец примера.

Вернёмся к общей постановке задачи.

Введём в рассмотрение множество $\Delta = \{(x; y) \in Z^2 \mid 0 \le x \le \mathrm{N1} \ \& \ 0 \le y \le \mathrm{N2}\}$.

Введём в рассмотрение бинарное отношение линейного порядка $\prec$ на множестве Z^2 следующим образом: $(i_1; i_2) \prec (j_1; j_2)$ означает либо $i_1 < j_1$, либо $i_1 = j_1$ и $i_2 < j_2$. Будем называть это бинарное отношение лексикографическим сравнением. Пусть $M \subset Z^2$ – конечное множество. Определим lexmax(M) следующим образом: lexmax(M) = I, где $I \in M$ и для любого $I' \in M$ такого, что $I \ne I'$ выполняется $I' \prec I$. Будем называть эту функцию лексикографическим максимумом.

Пусть генератор на итерации $I = (i_1; i_2)$ обращается к той же ячейке памяти, что и использование на итерации $J = (j_1; j_2)$. Тогда пара $(I; J)$ удовлетворяет системе диофантовых уравнений:

$$\begin{cases} a_1 i_1 + c_1 i_2 = b_1 j_1 + c_1 j_2 + d_1, \\ a_2 i_1 + c_2 i_2 = b_2 j_1 + c_2 j_2 + d_2, \end{cases} \qquad (13)$$

где $d_1 = d21 - d11$, $d_2 = d22 - d12$, а все остальные параметры системы равны одноимённым параметрам цикла (1). В матричном виде эта система имеет вид:

$$AI = BJ + D, \qquad (14)$$

где

$$A = \begin{pmatrix} a_1 & c_1 \\ a_2 & c_2 \end{pmatrix}, \quad B = \begin{pmatrix} b_1 & c_1 \\ b_2 & c_2 \end{pmatrix}, \quad D = \begin{pmatrix} d_1 \\ d_2 \end{pmatrix}.$$

С учётом предшествующих обозначений и того, что $I = (i_1; i_2)$ – итерация генератора и $J = (j_1; j_2)$ – итерация использования одной и той же ячейки памяти, в нашей задаче естественными являются следующие ограничения:

$$I, J \in \Delta, I \prec J, I = \text{lexmax } K, \qquad (15)$$

где K – множество всех таких $I' \in Z^2$, что $(I'; J)$ – решение системы (14).

Теперь нужно отметить, что *все решения системы* (14)–(15) *задают дуги в решетчатом графе цикла* (1), причём среди них могут быть и вертикальные тоже. Для поиска решения этой системы, мы рассмотрим пять случаев:

1. $|A| \neq 0, |B| \neq 0$.
2. $|A| = 0, |B| \neq 0$.
3. $|A| \neq 0, |B| = 0$.
4. $|A| = 0, |B| = 0, \ c_1^2 + c_2^2 \neq 0$.
5. $|A| = 0, |B| = 0, \ c_1^2 + c_2^2 = 0$.

Для каждого из этих пяти случаев мы укажем

1) пример, удовлетворяющий условиям данного случая;
2) исследование задачи при ограничениях, наложенных условиями данного случая;
3) условия существования зависимостей, а следовательно, и дуг графа зависимостей;
4) систему равенств, которая задаёт решения системы (14)–(15), а значит, задаёт дуги графа зависимостей;
5) задержку между стартами конвейеров.

В связи с тем, что в начале каждого исследованного случая демонстрируется пример, мы рекомендуем читателю, в «сомнительных» на его взгляд ситуациях обращаться к этим примерам.

3.6.2. Случай $|A| \neq 0, |B| \neq 0$.

Пример 29

```
for(i1=1; i1<=N1; ++i1)
      for(i2=1; i2<=N2; ++i2)
      {
            X[2*i1-i2][i1+i2] = …
            … =… X[i1-i2][i1+i2];
      }
```

Выпишем систему (14)–(15) для данного примера:

$$\begin{cases} 2i_1 - i_2 = j_1 - j_2, \\ i_1 + i_2 = j_1 + j_2, \\ I, J \in \Delta, \quad I \prec J, \quad I = \text{lex max } K. \end{cases}$$

Отметим, что в этом случае, уравнение (14) не может иметь двух разных решений с одинаковыми значениями $J = (j_1; j_2)$, поэтому условие лексикографического максимума в наборе ограничений (15) является избыточным. Итак, приведём последнюю систему к виду:

$$\begin{cases} 2i_1 - i_2 = j_1 - j_2, \\ 3i_1 = 2j_1, \\ I, J \in \Delta, \quad I \prec J. \end{cases}$$

Решим диофантово уравнение из второй строчки:

$$\begin{cases} 2i_1 - i_2 = j_1 - j_2, \\ i_1 = 2t, \\ j_1 = 3t, \quad t \in \mathbf{Z}, \\ I, J \in \Delta, \quad I \prec J. \end{cases}$$

Тогда

$$\begin{cases} i_2 - j_2 = t, \\ i_1 = 2t, \\ j_1 = 3t, \quad t \in \mathbf{Z}, \\ I, J \in \Delta, \quad I \prec J. \end{cases}$$

Чтобы проиллюстрировать пример рисунком выберем $N_1 = N_2 = 8$. Из всего множества целых чисел необходимо выбрать допустимые значения параметра t, т.е. множество целых чисел, удовлетворяющее ограничениям (15). Обозначим через $D = \{t \in Z | 1 \le 2t \le 3t \le 8 \ \& \ 1 \le |t| \le 8-1\}$ множество допустимых значений параметра t. Тогда $D = \{1; 2\}$. Теперь несложно построить граф информационных зависимостей (см. рис. 41).

Для того чтобы вычислить задержку необходимо отобрать те значения параметра t, которые, с одной стороны, относятся только к не вертикальным дугам (т.е. $i_1 < j_1$), а с другой стороны, для которых значения функции $z[i_1; j_1]$ были больше 0 (т.е. $w(T_{it} + T_{send} + i_2 - j_2) > 0$). Обозначим через $H = D \cap \{t \in Z | 2t < 3t \ \& \ T_{it} + T_{send} + t > 0\}$. Тогда $H = \{1; 2\}$, так как $T_{it} > 1$. Тогда $z = \max \{w(T_{it} + T_{send} + 1); w(T_{it} + T_{send} + 2) \times 0{,}5\} = w(T_{it} + T_{send} + 1)$.

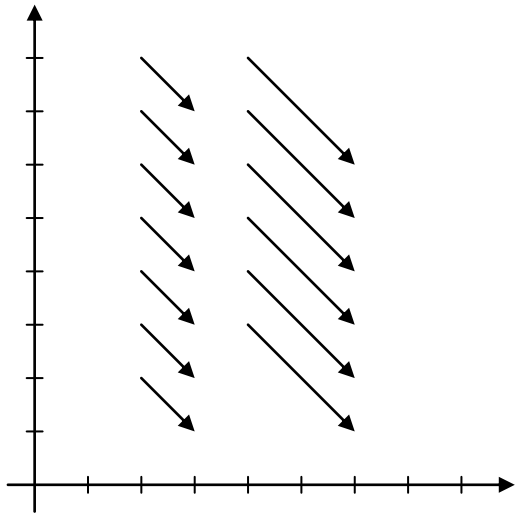

Рис. 41. Решетчатый граф для примера 29

Исследование

Отметим, что в этом случае, уравнение (14) не может иметь двух разных решений с одинаковыми значениями $J = (j_1; j_2)$, поэтому условие лексикографического максимума в наборе ограничений (15) является избыточным.

Очевидным является то, что $c_1^{\,2} + c_2^{\,2} \neq 0$, иначе $|A| = |B| = 0$. Не умаляя общности, предположим, что $c_1 \neq 0$. Тогда мы можем утверждать, что в этом случае систему (14)–(15) с помощью вычитания уравнений можно привести к виду:

$$\begin{cases} a_1 i_1 + c_1 i_2 = b_1 j_1 + c_1 j_2 + d_1, \\ a_2' i_1 = b_2' j_1 + d_2', \\ I, J \in \Delta, \quad I \prec J, \end{cases} \tag{16}$$

где $a_2' = c_1 a_2 - c_2 a_1$, $b_2' = c_1 b_2 - c_2 b_1$, $d_2' = c_1 d_2 - c_2 d_1$.

Пусть $\gamma = НОД(a_2'; -b_2')$. Предположим $\dfrac{d_2'}{\gamma} \in Z$, иначе второе уравнение не имеет решений, а значит и вся система (16) решений иметь не будет. Тогда существуют $i_0, j_0 \in Z$ такие, что $a_2' i_0 = b_2' j_0 + d_2'$. Решаем диофантово уравнение во второй строке системы (16):

$$\begin{cases} a_1 i_1 + c_1 i_2 = b_1 j_1 + c_1 j_2 + d_1, \\ i_1 = i_0 + b_2' t, \\ j_1 = j_0 + a_2' t, \quad t \in Z, \\ I, J \in \Delta, \quad I \prec J. \end{cases} \tag{17}$$

После подстановки выражений для i_1 и j_1 в первое уравнение, получается:

$$i_2 - j_2 = (a_2 b_1 - a_1 b_2)t + (b_1 j_0 - a_1 i_0 + d_1)/c_1.$$

Предположим $b_1j_0 - a_1i_0 + d_1$ делиться нацело на c_1, иначе последнее уравнение не имеет решений, а значит и вся система (17) решений не имеет.

$$\begin{cases} i_1 = i_0 + b_2' t, \\ j_1 = j_0 + a_2' t, \quad t \in Z, \\ i_2 = j_2 + \beta(t), \\ I, J \in \Delta, \quad I \prec J, \end{cases}$$

где $a_2' = c_1a_2 - c_2a_1$, $b_2' = c_1b_2 - c_2b_1$, $\beta(t) = (a_2b_1 - a_1b_2)t + (b_1j_0 - a_1i_0 + d_1)/c_1$.

Условия существования дуг в решетчатом графе

Обозначения:

$$a_2' = c_1a_2 - c_2a_1,\ b_2' = c_1b_2 - c_2b_1,\ d_2' = c_1d_2 - c_2d_1,$$
$$\exists i_0, j_0 \in Z : a_2'i_0 = b_2' j_0 + d_2',$$
$$\beta(t) = \left(a_2b_1 - a_1b_2\right)t + \left(b_1j_0 - a_1i_0 + d_1\right)/c_1$$
$$D = \left\{ t \in Z \,\middle|\, 1 \le i_0 + b_2't \le j_0 + a_2't \le N_1 \ \&\ 1 \le \left|\beta(t)\right| \le N_2 - 1 \right\}$$

Условия:

$(b_1j_0 - a_1i_0 + d_1)/c_1 \in Z$ & $D \neq \varnothing$.

Формула вычисления дуг графа

$$\begin{cases} i_1 = i_0 + b_2' t, \\ j_1 = j_0 + a_2' t, \\ i_2 = j_2 + \beta(t), \\ t \in D. \end{cases}$$

Задержка

Рассмотрим выражение:

$$\frac{z[i_1; j_1]}{j_1 - i_1} = \max_{\substack{i_2, j_2: \\ (i_1; j_1; i_2; j_2) \in \mathbf{G}}} \frac{w(i_2 - j_2 + T_{it} + T_{send})}{j_1 - i_1} = \frac{w(\beta(k) + T_{it} + T_{send})}{j_0 - i_0 + (a_2' - b_2')k}$$

где $i_1 < j_1$.

Так как нас интересует максимум этого выражения, то нас интересуют только $k \in H \subset D$, где $H = D \cap \{k \in Z |\ i_0 + b_2'k < j_0 + a_2'k$ & $T_{it} + T_{send} + \beta(k) > 0\}$. Отметим также, что в числителе и знаменателе линейные по k функции. Так как множество D – конечно, то и множество H тоже конечно. Тогда существуют минимальный и максимальный элементы

во множестве H, если оно не пусто. Пусть $k_{\min}$, $k_{\max}$ такие числа, что $k_{\min} \le k \le k_{\max}$, для любого $k \in H$.

Рассмотрим дробно-линейную функцию $h(k) = (q_1k + p_1)/(q_2k + p_2)$ с вещественным аргументом $k \in [k_{\min}; k_{\max}]$, обладающую следующим свойством: $-p_2/q_2 \notin [k_{\min}; k_{\max}]$. Отметим, что

i) функция $h(k)$, на отрезке $k \in [k_{\min}; k_{\max}]$ монотонна (либо возрастает, либо убывает в зависимости от знака числа $q_1 p_2 - q_2 p_1$).

ii) функция $h(k)$, на отрезке $k \in [k_{\min}; k_{\max}]$ непрерывна.

Рассмотрим дробно-линейную функцию $g(k) = (\beta(k) + K_{ik} + K_{send})/(j_0 - i_0 + k(a'_2 - b'_2))$ с вещественным аргументом $k \in [k_{\min}; k_{\max}]$, так как $j_0 - i_0 + k(a'_2 - b'_2) > 0$, $k \in H$. Из свойств i, ii следует, что на отрезке $k \in [k_{\min}; k_{\max}]$: $\max g(k) = \max \{g(k_{\min}); g(k_{\max})\}$. Тогда на множестве $k \in H$: $\max g(k) = \max \{g(k_{\min}); g(k_{\max})\}$.

Итак,

$$\max_{1 \le i_1 < j_1 \le N_1} \frac{z[i_1; j_1]}{j_1 - i_1} = \max\{g(k_{\min}); g(k_{\max})\}$$

следовательно,

$$z = \begin{cases} \max\left\{ \dfrac{w(\beta(k_{\min}) + T_{it} + T_{send})}{j_0 - i_0 + (a'_2 - b'_2)k_{\min}}; \dfrac{w(\beta(k_{\max}) + T_{it} + T_{send})}{j_0 - i_0 + (a'_2 - b'_2)k_{\max}} \right\}, & H \ne \phi \\ 0, & H = \phi \end{cases}$$

3.6.3. Случай |A| = 0, |B| ≠ 0.

Пример 30

```
for(i1=1; i1<=N1; ++i1)
      for(i2=1; i2<=N2; ++i2)
      {
            X[i1-i2][1] = …
            … =… X[2*i1-i2-8][i1-3];
      }
```

Выпишем систему (14)–(15) для данного примера:

$$\begin{cases} i_1 - i_2 = 2j_1 - j_2 - 8, \\ 0 = j_1 - 4, \\ I, J \in \Delta, \quad I \prec J, \quad I = \text{lex max } K, \end{cases}$$

где K – множество всех таких $I' \in Z^2$,что $(I'; J)$ – решение системы (14).

Пусть $i_1 = 4 - k$, $0 \le k \le N_1$. Тогда

$$\begin{cases} i_1 - i_2 = 2j_1 - j_2 - 8, \\ j_1 = 4, \\ i_1 = 4 - k, \\ 0 \le k \le N_1, \quad I, J \in \Delta, \quad I \prec J, \quad I = \text{lex max } K, \end{cases}$$

Подставим второе и третье уравнение в первое:

$$\begin{cases} i_2 - j_2 = 4 - k, \\ j_1 = 4, \\ i_1 = 4 - k, \\ 0 \le k \le N_1, \quad I, J \in \Delta, \quad I \prec J, \quad I = \text{lex max } K, \end{cases}$$

Обозначим через $\beta(k) = i_2 - j_2 = 4 - k$. Введём в рассмотрение следующие множества:

$D_0 = \{j \in Z \cap [1; N_2] | \beta(0) < 0 \ \& \ j + \beta(0) \in j \in Z \cap [1; N_2]\}$,

$$D_k = \{j \in Z \cap [1; N_2] | \alpha - k \in Z \cap [1; N_1] \ \& \ j + \beta(k) \in Z \cap [1; N_2]\} \setminus \left(\bigcup_{s=0}^{k-1} D_s \right), \ 1 \le k \le N_1$$

Найдём D_0:

$D_0 = \varnothing$

Найдём D_1:

$D_1 = \{j \in Z \cap [1; 8] | 4 - 1 \in Z \cap [1; 8] \ \& \ j + 4 - 1 \in Z \cap [1; 8]\} \setminus D_0 = \{1; 2; 3; 4; 5\}$

Найдём D_2:

$D_2 = \{j \in Z \cap [1; 8] | 4 - 2 \in Z \cap [1; 8] \ \& \ j + 4 - 2 \in Z \cap [1; 8]\} \setminus (D_0 \cup D_1) = \{6\}$

Найдём D_3:

$D_3 = \{j \in Z \cap [1; 8] | 4 - 3 \in Z \cap [1; 8] \ \& \ j + 4 - 3 \in Z \cap [1; 8]\} \setminus (D_0 \cup D_1 \cup D_2) = \{7\}$

Отметим, что для любого $k \ge 4$ выражение $4 - k \le 0$, следовательно, $D_k = \varnothing$, при $k \ge 4$. Теперь мы можем описать все решения нашей задачи следующим образом:

$$\bigcup_{k \in H} \begin{cases} i_1 = 4 - k, \\ j_1 = 4, \\ i_2 = j_2 + \beta(k), \\ j_2 \in D_k, \end{cases}$$

где $H = \{k \in \mathrm{N} \cup \{0\} \mid D_k \neq \varnothing\} = \{1; 2; 3\}$.

Тогда

$$\begin{cases} i_1 = 3, \\ j_1 = 4, \\ i_2 = j_2 + 3, \\ j_2 \in \{1,2,3,4,5\}, \end{cases} \cup \begin{cases} i_1 = 2, \\ j_1 = 4, \\ i_2 = j_2 + 2, \\ j_2 \in \{6\}, \end{cases} \cup \begin{cases} i_1 = 1, \\ j_1 = 4, \\ i_2 = j_2 + 1, \\ j_2 \in \{7\}. \end{cases}$$

Теперь несложно построить граф информационных зависимостей (см. рис. 42). Далее для вычисления задержки необходимо воспользоваться формулой (12):

$z = \max \{w(T_{it} + T_{send} + 3); w(T_{it} + T_{send} + 2)/2; w(T_{it} + T_{send} + 1)/3\} = w(T_{it} + T_{send} + 3)$.

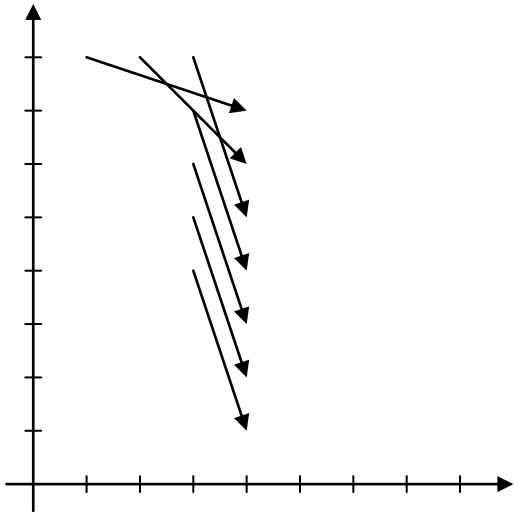

Рис. 42. Решетчатый граф для примера 30

Исследование

Очевидным является то, что $c_1^2 + c_2^2 \neq 0$, иначе $|B| = 0$. Не умаляя общности, предположим, что $c \neq 0$. Тогда мы можем утверждать, что в этом случае систему (14)–(15) с помощью вычитания уравнений можно привести к виду с 0-строкой в матрице A:

$$\begin{cases} a_1 i_1 + c_1 i_2 = b_1 j_1 + c_1 j_2 + d_1 \\ 0 = b_2 j_1 + d_2 \\ I, J \in \Delta, \quad I \prec J, \quad I = \operatorname{lex\,max} K, \end{cases}$$

где K – множество всех таких $I' \in Z^2$,что $(I'; J)$ – решение системы (14). Коэффициент при переменной j_2 во втором уравнении равен 0 вследствие того, что он должен быть равен коэффициенту при i_2, так как был ему равен до преобразования системы (14).

Пусть $\alpha = -d_2/b_2$, следовательно, $j_1 = \alpha$. Тогда система не имеет решений, если $\alpha \notin Z \cap [1; N_1]$.

Введём следующее обозначение: пусть k такое число, что $i_1 = \alpha - k$, $0 \le k < N_1$. Тогда первое уравнение можно записать в виде:

$a_1(\alpha - k) + c_1 i_2 = b_1\alpha + c_1 j_2 + d_1 \Leftrightarrow c_1(i_2 - j_2) = (b_1 - a_1)\alpha + d_1 + a_1 k$.

Пусть $\beta(k) = ((b_1 - a_1)\alpha + d_1 + a_1 k)/c_1$. Тогда система не имеет решений, если для любого $0 < k < N_1$ выполняется $\beta(k) \notin Z \cap [-N_2; N_2]$ и $\beta(0) \notin Z \cap [-N_2 + 1; -1]$.

С учетом новых обозначений получаем:

$$\begin{cases} i_1 = \alpha - k \\ j_1 = \alpha \\ i_2 = j_2 + \beta(k) \\ 0 \le k < N_1, \\ I, J \in \Delta, \quad I \prec J, \quad I = \text{lex max } K, \end{cases} \tag{18}$$

где K – множество всех таких $I' \in Z^2$,что $(I'; J)$ – решение системы (14). Чтобы систему (18) избавить от использования лексикографического максимума необходимо ввести систему множеств $\{D_k\}$, $0 \le k \le N_1$:

$D_0 = \{j \in Z \cap [1; N_2] | \beta(0) < 0 \ \& \ j + \beta(0) \in j \in Z \cap [1; N_2]\}$,

$$D_k = \{j \in Z \cap [1; N_2] | \alpha - k \in Z \cap [1; N_1] \ \& \ j + \beta(k) \in Z \cap [1; N_2]\} \setminus \left(\bigcup_{s=0}^{k-1} D_s \right), \ 1 \le k \le N_1$$

Отметим, что если при $k = 0$ есть решения системы, то граф информационных зависимостей имеет вертикальные дуги, при $\beta(k) < 0$.

Условия существования дуг графа

Обозначения:

$\alpha = -d_2/b_2$, $\beta(k) = ((b_1 - a_1)\alpha + d_1 + a_1 k)/c_1$

$D_0 = \{j \in Z \cap [1; N_2] | \beta(0) < 0 \ \& \ j + \beta(0) \in Z \cap [1; N_2]\}$,

$$D_k = \{j \in Z \cap [1; N_2] | \alpha - k \in Z \cap [1; N_1] \ \& \ j + \beta(k) \in Z \cap [1; N_2]\} \setminus \left(\bigcup_{s=0}^{k-1} D_s \right), \ 1 \le k \le N_1$$

$D = \cup D_k$, $0 \le k < N_1$.

На практике рекомендуется перебрать $j \in Z \cap [1; N_2]$ и в первую очередь построить разбиение на D_k в соответствии с определением этой системы множеств.

Условия:

$\alpha \in Z \cap [1; N_1]$, $D \ne \varnothing$.

Формула вычисления дуг графа

$$\bigcup_{k\in H}\begin{cases} i_1 = \alpha - k, \\ j_1 = \alpha, \\ i_2 = j_2 + \beta(k), \\ j_2 \in D_k. \end{cases}$$

где $H = \{k \in \mathrm{N} \cup \{0\} \mid D_k \neq \varnothing\}$.

Ввиду того, что сказано в пункте «Условия существования дуг графа», множество H – конечно.

Задержка

Рассмотрим выражение:

$$\frac{z[i_1; j_1]}{j_1 - i_1} = \max_{\substack{i_2, j_2: \\ (i_1; j_1; i_2; j_2)\in \mathbf{G}}} \frac{w(i_2 - j_2 + T_{it} + T_{send})}{j_1 - i_1} = \frac{w(\beta(k) + T_{it} + T_{send})}{k}.$$

Пусть $k_{\min}$, $k_{\max}$ такие числа, что $k_{\min} \le k \le k_{\max}$, для любого $k \in H \setminus \{0\}$. Так как нас интересует максимум выражения $z[i_1; j_1]/(j_1 - i_1)$, то мы можем воспользоваться вспомогательными результатами, рассмотренными в пункте «Задержка» для случая $|A| \neq 0$, $|B| \neq 0$.

Итак,

$$\max_{1\le i_1 < j_1 \le N_1} \frac{z[i_1; j_1]}{j_1 - i_1} = \max\left\{\frac{w(\beta(k_{\min}) + T_{it} + T_{send})}{k_{\min}}; \frac{w(\beta(k_{\max}) + T_{it} + T_{send})}{k_{\max}}\right\},$$

следовательно,

$$z = \begin{cases} \max\left\{\dfrac{w(\beta(k_{\min}) + T_{it} + T_{send})}{k_{\min}}; \dfrac{w(\beta(k_{\max}) + T_{it} + T_{send})}{k_{\max}}\right\}, & H \setminus \{0\} \neq \phi \\ 0, & H \setminus \{0\} = \phi \end{cases}$$

3.6.4. Случай $|A| \neq 0$, $|B| = 0$

Пример 31

```
for(i1=1; i1<=N1; ++i1)
      for(i2=1; i2<=N2; ++i2)
      {
            X[i1+2*i2][i1-3] = …
            … = … X[3*i1+2*i2-5][1];
      }
```

Выпишем систему (14)–(15) для данного примера:

$$\begin{cases} i_1 + 2i_2 = 3j_1 + 2j_2 - 5, \\ i_1 = 4, \\ I, J \in \Delta, \quad I \prec J, \quad I = \text{lex max } K, \end{cases}$$

где K – множество всех таких $I' \in Z^2$,что $(I'; J)$ – решение системы (14).

Отметим, что в этом случае, уравнение (14) не может иметь двух разных решений с одинаковыми значениями $J = (j_1; j_2)$, поэтому условие лексикографического максимума в наборе ограничений (15) является избыточным.

Пусть $j_1 = 4 + k$, $0 \le k \le N_1$. Введём в рассмотрение множество $D = \{k > 0 | \ 4 + k \in Z \cup [1; 8]\}$. Тогда

$$\begin{cases} i_1 + 2i_2 = 3j_1 + 2j_2 - 5, \\ i_1 = 4, \\ j_1 = 4 + k, \\ 0 \le k < 8, \quad I, J \in \Delta, \quad I \prec J. \end{cases}$$

Подставим второе и третье уравнение в первое:

$$\begin{cases} 2(i_2 - j_2) = 3 + 3k, \\ i_1 = 4, \\ j_1 = 4 + k, \\ 0 \le k < 8, \quad I, J \in \Delta, \quad I \prec J. \end{cases}$$

Обозначим через $\beta(k) = 1{,}5(1 + k)$. Введём в рассмотрение следующее множество:

$H = \{k \in \mathrm{N} | \ 4 + k \in Z \cup [1; 8] \ \& \ |\beta(k)| \in Z \cup [1; 8 - 1]\} \cap$

$\cap \{k \in \{0\} | \ \beta(k) \in Z \cup [-7; -1]\} = \{1; 3\}$

Тогда

$$\begin{cases} i_1 = 4, \\ j_1 = 4 + k, \\ i_2 - j_2 = 1{,}5(1 + k), \\ k \in \{1{,}3\}, \quad I, J \in \Delta. \end{cases}$$

Теперь несложно построить граф информационных зависимостей. Для случая N1 = 8, N2 = 8 этот граф изображен на рис. 43.

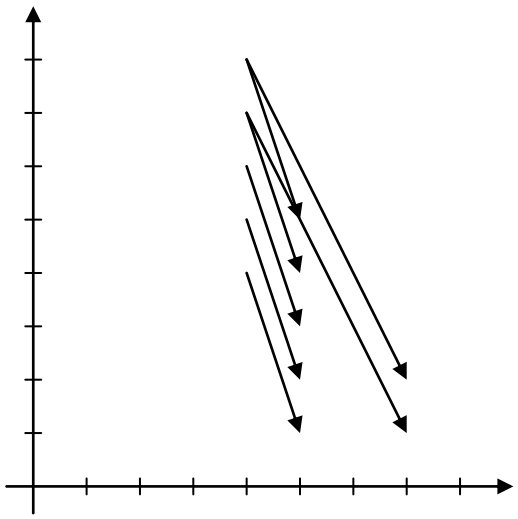

Рис 43. Решетчатый граф для примера 31

Далее для вычисления задержки необходимо воспользоваться формулой (12):
$z = \max\{w(3 + T_{it} + T_{send}); w(6 + T_{it} + T_{send})/3\} = w(3 + T_{it} + T_{send})$.

Исследование

Отметим, что в этом случае, уравнение (14) не может иметь двух разных решений с одинаковыми значениями $J = (j_1; j_2)$, поэтому условие лексикографического максимума в наборе ограничений (15) является избыточным.

Очевидным является то, что $c_1^2 + c_2^2 \neq 0$, иначе $|A| = 0$. Не умаляя общности, предположим, что $c_1 \neq 0$. Тогда мы можем утверждать, что в этом случае систему (14)–(15) с помощью вычитания уравнений можно привести к виду с 0-строкой в матрице B:

$$\begin{cases} a_1 i_1 + c_1 i_2 = b_1 j_1 + c_1 j_2 + d_1, \\ a_2 i_1 = d_2, \\ I, J \in \Delta, \quad I \prec J, \end{cases} \tag{19}$$

где K – множество всех таких $I' \in Z^2$,что (I'; J) – решение системы (14). Коэффициент при переменной i_2 во втором уравнении равен 0 по тем же причинам, что и в случае $|A| = 0$, $|B| \neq 0$.

Пусть $\alpha = d_2/a_2$, следовательно, $i_1 = \alpha$. Тогда система не имеет решений, если $\alpha \notin Z \cup [1; N_1]$.

Пусть $D = \{k > 0 | \alpha + k \in Z \cup [1; N_1]\}$. Если $D = \varnothing$, то решений системы (19) нет. Предположим, что $D \neq \varnothing$. Пусть $j_1 = \alpha + k$, $k \in D$. Тогда первое уравнение можно записать в виде:

$$a_1\alpha + c_1 i_2 = b_1(\alpha + k) + c_1 j_2 + d_1 \quad <=>$$

$$c_1(i_2 - j_2) = (b_1 - a_1)\alpha + d_1 + b_1 k$$

Пусть $\beta(k) = ((b_1 - a_1)\alpha + d_1 + b_1 k)/c_1$. Тогда система не имеет решений, если для любого $k > 0$: $|\beta(k)| \notin Z \cup [1; N_2]$.

Теперь систему (19) можно переписать в виде:

$$\begin{cases} i_1 = \alpha, \\ j_1 = \alpha + k, \\ i_2 = j_2 + \beta(k), \\ k > 0, \\ I, J \in \Delta, \quad I \prec J, \end{cases}$$

где K – множество всех таких $I' \in Z^2$,что (I'; J) – решение системы (14).

Отметим, что различным $k > 0$ могут соответствовать различные решения этой системы, что легко увидеть на приведённом выше примере.

Условия существования дуг графа

Обозначения:

$$\begin{cases} \alpha = \dfrac{d_2}{a_2}, \\ \beta(k) = \dfrac{(b_1 - a_1)\alpha + d_1 + b_1 k}{c_1} \\ H = D \cap \left(\left\{k \in \mathrm{N} \,\middle|\, |\beta(k)| \in Z \cap [1; N_2 - 1]\right\} \cup \left\{k \in \{0\} \,\middle|\, \beta(k) \in Z \cap [-N_2 + 1; -1]\right\}\right). \end{cases}$$

Условия:

$$\begin{cases} \alpha \in Z \cap [1; N_1], \\ H \neq \phi. \end{cases}$$

Отметим, что условие $D \neq \varnothing$ будет выполняться автоматически, при $H \neq \varnothing$.

Формула вычисления дуг графа

$$\begin{cases} i_1 = \alpha, \\ j_1 = \alpha + k, \\ i_2 = j_2 + \beta(k), \\ k \in H, \\ i_2, j_2 \in Z \bigcap [1; N_2]. \end{cases}$$

Задержка

Рассмотрим выражение:

$$\frac{z[i_1; j_1]}{j_1 - i_1} = \max_{\substack{i_2, j_2: \\ (i_1; j_1; i_2; j_2) \in \mathbf{G}}} \frac{w(i_2 - j_2 + T_{it} + T_{send})}{j_1 - i_1} = \frac{w(\beta(k) + T_{it} + T_{send})}{k},$$

где $k \in H \backslash \{0\}$.

Пусть $k_{\min}$, $k_{\max}$ такие числа, что $k_{\min} \le k \le k_{\max}$, для любого $k \in H \backslash \{0\}$. Так как нас интересует максимум выражения $z[i_1; j_1]/(j_1 - i_1)$, то мы можем воспользоваться свойствами функций, описанными в пункте «Задержка» для случая $|A| \neq 0$, $|B| \neq 0$.

Итак,

$$\max_{1 \le i_1 < j_1 \le N_1} \frac{z[i_1; j_1]}{j_1 - i_1} = \max\left\{ \frac{w(\beta(k_{\min}) + T_{it} + T_{send})}{k_{\min}}; \frac{w(\beta(k_{\max}) + T_{it} + T_{send})}{k_{\max}} \right\},$$

следовательно,

$$z = \begin{cases} \max\left\{ \frac{w(\beta(k_{\min}) + T_{it} + T_{send})}{k_{\min}}; \frac{w(\beta(k_{\max}) + T_{it} + T_{send})}{k_{\max}} \right\}, & H \setminus \{0\} \neq \phi \\ 0, & H \setminus \{0\} = \phi \end{cases}$$

3.6.5. Случай |*A*| = 0, |*B*| = 0, $c_1^2 + c_2^2 \neq 0$.

Пример 32

```
for(i1=1; i1<=N1; ++i1)
      for(i2=1; i2<=N2; ++i2)
      {
            X[2*i1+i2][1] = …
            … = … X[i1+i2+4][1];
      }
```

Выпишем систему (14)–(15) для данного примера:

$$\begin{cases} 2i_1 + i_2 = j_1 + j_2 + 4, \\ I, J \in \Delta, \quad I \prec J, \quad I = \text{lex max } K, \end{cases} \tag{20}$$

где K – множество всех таких $I' \in Z^2$, что $(I'; J)$ – решение системы (14).

Обозначим через $k = j_1 - i_1$. Обозначим через $\beta(k, j_1) = 2k - 5 - j_1$. Тогда систему (20) можно привести к виду:

$$\begin{cases} j_1 - i_1 = k, \\ i_2 - j_2 = 2k + 4 - j_1, \\ 0 \le k < 8, \\ I, J \in \Delta, \quad I \prec J, \quad I = \text{lex max } K. \end{cases}$$

Построим следующую систему множеств:

$$D_0 = \left\{ (j_1; j_2) \in \Delta \mid j_2 + \beta(0, j_1) \in \mathbf{Z} \cap [1; j_2 - 1] \right\},$$

$$D_k = \left\{ (j_1; j_2) \in \Delta \mid j_1 - k \in \mathbf{Z} \cap [1; N_1] \,\&\, j_2 + \beta(k, j_1) \in \mathbf{Z} \cap [1; N_2] \right\} \setminus \left(\bigcup_{s=0}^{k-1} D_s \right),$$

где $0 \le k < N_1$.

Найдём D_0:

$$D_0 = \left\{ (j_1; j_2) \in \Delta \mid j_2 - j_1 + 4 \in \mathbf{Z} \cap [1; j_2 - 1] \right\} = \{(5;2);(5;3);(5;4);(5;5);(5;6);(5;7);(5;8)\} \cup$$
$$\cup \{(6;3);(6;4);(6;5);(6;6);(6;7);(6;8)\} \cup \{(7;4);(7;5);(7;6);(7;7);(7;8)\} \cup \{(8;5);(8;6);(8;7);(8;8)\}.$$

Найдём D_1:

$$D_1 = \left\{ (j_1; j_2) \in \Delta \mid j_1 - 1 \in \mathbf{Z} \cap [1; N_1] \,\&\, j_2 - j_1 + 6 \in \mathbf{Z} \cap [1; N_2] \right\} \setminus D_0 =$$
$$= \{(2;1);(2;2);(2;3);(2;4)\} \cup \{(3;1);(3;2);(3;3);(3;4);(3;5)\} \cup \{(4;1);(4;2);(4;3);(4;4);(4;5);(4;6)\} \cup$$
$$\cup \{(5;1)\} \cup \{(6;1);(6;2)\} \cup \{(7;2);(7;3)\} \cup \{(8;3);(8;4)\}.$$

Найдём D_2:

$$D_2 = \{(j_1; j_2) \in \Delta \mid j_1 - 2 \in \mathbf{Z} \cap [1; N_1] \,\&\, j_2 - j_1 + 8 \in \mathbf{Z} \cap [1; N_2]\} \setminus (D_0 \cup D_1) =$$
$$= \{(7;1)\} \cup \{(8;1);(8;2)\}.$$

Введём также следующую последовательность множеств:

$$\overline{D}_k = \{ j_1 \in Z \cap [1; N_1] \mid \exists\, j_2 \in Z \cap [1; N_2] : (j_1; j_2) \in D_k \},$$

где $0 \le k < N_1$.

Найдём $\overline{D}_0, \overline{D}_1, \overline{D}_2$:

$$\overline{D}_0 = \{5,6,7,8\},\ \overline{D}_1 = \{1,2,3,4,5,6,7,8\},\ \overline{D}_2 = \{7,8\}.$$

Тогда решения имеют вид:

$$\bigcup_{0 \le k < N_1} \begin{cases} j_1 \in \overline{D}_k \\ i_1 = j_1 - k, \\ i_2 = j_2 + 2k + 4 - j_1. \end{cases}$$

Теперь несложно построить граф информационных зависимостей. Для случая N1 = 8, N2 = 8 этот граф изображен на рис. 44.

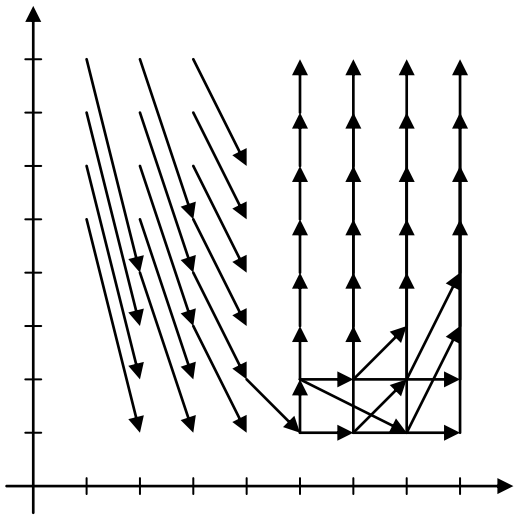

Рис 44. Решетчатый граф для примера 32

Далее для вычисления задержки необходимо воспользоваться формулой (12):

$$z = \max_{k \in G} \max_{j_1 \in \overline{D}_k} \frac{\mathrm{w}(2k + 4 - j_1 + t_e)}{k} = \max\left\{ \max_{j_1 \in \overline{D}_k} \frac{\mathrm{w}(6 - j_1 + t_e)}{1}; \max_{j_1 \in \overline{D}_k} \frac{\mathrm{w}(8 - j_1 + t_e)}{2} \right\} =$$

$$= \max\left\{ \mathrm{w}(6 - 1 + t_e); \frac{\mathrm{w}(8 - 7 + t_e)}{2} \right\} = \mathrm{w}(5 + t_e)$$

.

Исследование

Предположим $c_1 \neq 0$. Тогда систему (14)–(15) с помощью вычитания уравнений с коэффициентом c_2/c_1 можно привести к виду:

$$\begin{cases} a_1 i_1 + c_1 i_2 = b_1 j_1 + c_1 j_2 + d_1, \\ 0 = d_2 - \dfrac{c_2}{c_1} d_1, \\ I, J \in \Delta, \quad I \prec J, \quad I = \text{lex max}\, K. \end{cases} \tag{21}$$

где K – множество всех таких $I' \in Z^2$,что $(I'; J)$ – решение системы (14). Коэффициенты при i_1, j_1 во втором уравнении необходимо будут равны нулю ввиду того, что $|A| = 0$, $|B| = 0$.

Предположим, что $d_2 - d_1 c_2 / c_1 = 0$. Очевидно, что для любого решения системы (21) выполняется: $i_1 = j_1 - k$, $0 \leq k < N_1$. Далее в этом пункте под k будем понимать, введённое таким образом число.

Обозначим через $\beta(k, j_1) = (a_1 k\, d_1 + (b_1 - a_1) j_1)/c_1$. Тогда систему (21) можно привести к виду:

$$\begin{cases} j_1 - i_1 = k, \\ i_2 - j_2 = \beta(k, j_1) \\ 0 \leq k < N_1 \\ I, J \in \Delta, \quad I \prec J, \quad I = \text{lex max}\, K, \end{cases} \tag{22}$$

где K – множество всех таких $I' \in Z^2$,что $(I'; J)$ – решение системы (14).

Теперь нам необходимо избавиться от условия лексикографического максимума. Для этого введем, как и в случае $|A| = 0$, $|B| \neq 0$, последовательность множеств, следующим образом:

$$D_0 = \left\{ (j_1; j_2) \in \Delta \mid j_2 + \beta(0, j_1) \in \mathbf{Z} \cap [1; j_2 - 1] \right\},$$

$$D_k = \left\{ (j_1; j_2) \in \Delta \mid j_1 - k \in \mathbf{Z} \cap [1; N_1] \,\&\, j_2 + \beta(k, j_1) \in \mathbf{Z} \cap [1; N_2] \right\} \setminus \left(\bigcup_{s=0}^{k-1} D_s \right),$$

где $0 \leq k < N_1$.

Введём также следующую последовательность множеств:

$$\overline{D}_k = \left\{ j_1 \in Z \cap [1; N_1] \mid \exists\, j_2 \in Z \cap [1; N_2] : (j_1; j_2) \in D_k \right\},$$

где $0 \leq k < N_1$.

Отметим, что при фиксированном значении переменной k и $j_1 \in \overline{D}_k$ условие $I = \text{lex max}\, K$ в системе (22) будет избыточным, так же как и остальные условия из набора (15). Тогда систему (22) можно привести к виду:

$$\bigcup_{0\le k<N_1}\begin{cases} j_1 \in \overline{D}_k, \\ i_1 = j_1 - k, \\ i_2 = j_2 - \beta(k, j_1) \end{cases} \tag{23}$$

Условия существования дуг графа

Обозначения:

$$\begin{cases} \beta(k, j_1) = \dfrac{a_1 k + d_1 + (b_1 - a_1) j_1}{c_1}, \\ D_0 = \{(j_1; j_2) \in \Delta \mid j_2 + \beta(0, j_1) \in Z \cap [1; j_1 - 1]\}, \\ D_k = \{(j_1; j_2) \in \Delta \mid j_1 - k \in Z \cap [1; N_1] \,\&\, j_2 + \beta(k, j_1) \in Z \cap [1; N_2]\} \setminus \left(\bigcup_{s=0}^{k-1} D_s \right), \\ \overline{D}_k = \{ j_1 \in Z \cap [1; N_1] \mid \exists\, j_2 \in Z \cap [1; N_2] : (j_1; j_2) \in D_k \}, \quad 0 \le k < N_1. \end{cases}$$

Условия:

$$\bigcup_{0\le k<N_1} D_k \ne \phi .$$

Формула вычисления дуг графа

Смотри формулу (23).

Задержка

Рассмотрим выражение:

$$\frac{z[i_1; j_1]}{j_1 - i_1} = \max_{\substack{i_2, j_2: \\ (i_1; j_1; i_2; j_2) \in \mathbf{G}}} \frac{w(i_2 - j_2 + T_{it} + T_{send})}{j_1 - i_1} = \frac{w(\beta(k, j_1) + T_{it} + T_{send})}{k},$$

где $k > 0, \quad j_1 \in \overline{D}_k$.

Обозначим через $H = \{k \in Z \cap [1; N_1] \mid \overline{D}_k \ne \varnothing\}$. Теперь отметим, что

$$z = \max_{1\le i_1 < j_1 \le N_1} \frac{z[i_1; j_1]}{j_1 - i_1} = \max_{k \in H} \max_{j_1 \in \overline{D}_k} \frac{w(\beta(k, j_1) + T_{it} + T_{send})}{k}.$$

3.6.6. Случай |*A*| = 0, |*B*| = 0, $c_1^2 + c_2^2 = 0$.

Пример 33

```
for(i1=1; i1<=N1; ++i1)
      for(i2=1; i2<=N2; ++i2)
      {
            X[i1][2*i1] = …
            … = … X[i1-3][2*i1-6];
      }
```

Выпишем систему (14)–(15) для данного примера:

$$\begin{cases} i_1 = j_1 - 3, \\ 2i_1 = 2j_1 - 6, \\ I, J \in \Delta, \quad I \prec J, \quad I = \text{lex max } K, \end{cases}$$

где K – множество всех таких $I' \in Z^2$,что (I'; J) – решение системы (14).

Обратим внимание на то, что неизвестные i_2 , j_2 должны удовлетворять только требованию лексикографического максимума, а равенства ограничений на них не накладывают.

Рассмотрим следующую систему уравнений:

$$\begin{cases} i_1 = j_1 - 3, \\ 2i_1 = 2j_1 - 6. \end{cases}$$

Обозначим множество решений системы этой системы через D. Введём следующие два множества:

$$D_= = \{(i; j) \in D \mid 1 \le i = j \le N_1\},$$
$$D_< = \{(i; j) \in D \mid 1 \le i < j \le N_1\}.$$

В данном примере $D_= = \varnothing$, а $D_<$ для случая N1 = 8, N2 = 8 равно {(4;1), (5;2), (6;3), (7;4), (8;5)}.

Теперь несложно построить граф информационных зависимостей. Для случая N1 = 8, N2 = 8 этот граф изображен на рис. 45.

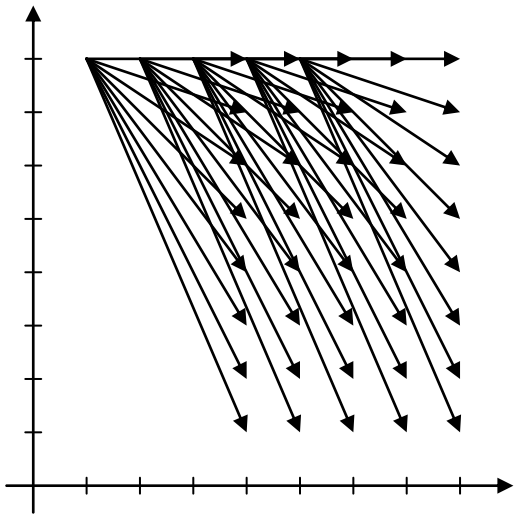

Рис. 45. Решетчатый граф для примера 33

Исследование

В этом случае система (14)–(15) принимает вид:

$$\begin{cases} a_1 i_1 = b_1 j_1 + d_1 \\ a_2 i_1 = b_2 j_1 + d_2 \\ I, J \in \Delta, \quad I \prec J, \quad I = \operatorname{lex\,max} K, \end{cases}$$

где K – множество всех таких $I' \in Z^2$,что (I'; J) – решение системы (14).

Рассмотрим систему уравнений следующего вида:

$$\begin{cases} a_1 i_1 = b_1 j_1 + d_1 \\ a_2 i_1 = b_2 j_1 + d_2 \end{cases} \tag{24}$$

В целых числах система (24) имеет 0, 1 или бесконечное множество решений. Обозначим множество решений системы (24) через D. Как найти это множество, мы считаем очевидным, и задерживать внимание читателя на этом не будем.

Введём следующие два множества:

$$D_= = \{(i; j) \in D \mid 1 \le i = j \le N_1\},$$
$$D_< = \{(i; j) \in D \mid 1 \le i < j \le N_1\}.$$

Таким образом, система (24) эквивалентна следующей системе:

$$\begin{cases} (i_1, j_1) \in D_= \cup D_< \\ I, J \in \Delta, \quad I \prec J, \quad I = \operatorname{lex\,max} K, \end{cases}$$

Теперь избавимся от условия лексикографического максимума и получим объединение следующих систем:

$$\begin{cases} (i_1, j_1) \in D_= \\ i_2 = j_2 - 1 \\ 2 \le j_2 \le N_2 \end{cases} \bigcup \begin{cases} (i_1, j_1) \in D_< \\ i_2 = N_2 \\ 1 \le j_2 \le N_2 \end{cases} \quad (25)$$

Условия существования графа

Обозначения:

D – множество решений системы (24).

$D_= = \{(i; j) \in D \mid 1 \le i = j \le N_1\}$.

$D_< = \{(i; j) \in D \mid 1 \le i < j \le N_1\}$.

Условия:

$D_= \bigcup D_< \ne \phi$.

Формула вычисления дуг графа

Смотри формулу (25).

Задержка

Рассмотрим выражение:

$$\frac{z[i_1; j_1]}{j_1 - i_1} = \max_{\substack{i_2, j_2: \\ (i_1; j_1; i_2; j_2) \in \mathbf{G}}} \begin{cases} \dfrac{i_2 - j_2 + T_{it} + T_{send}}{j_1 - i_1}, & i_2 - j_2 + T_{it} + T_{send} > 0 \\ 0, & i_2 - j_2 + T_{it} + T_{send} \le 0 \end{cases}$$

$$= \begin{cases} \dfrac{N_2 - 1 + T_{it} + T_{send}}{j_1 - i_1}, & N_2 - 1 + T_{it} + T_{send} > 0 \\ 0, & N_2 - 1 + T_{it} + T_{send} \le 0 \end{cases}$$

где $(i_1; j_1) \in D_<$.

Теперь отметим, что

$$z = \max_{1 \le i_1 < j_1 \le N_1} \frac{z[i_1; j_1]}{j_1 - i_1} = \begin{cases} \max\limits_{(i_1, j_1) \in D_<} \dfrac{w(N_2 - 1 + T_{it} + T_{send})}{j_1 - i_1}, & D_< \ne \phi \\ 0, & D_< = \phi \end{cases} =$$

$$\begin{cases} \dfrac{w(N_2 - 1 + T_{it} + T_{send})}{\min\limits_{(i_1, j_1) \in D_<} (j_1 - i_1)}, & D_< \ne \phi \\ 0, & D_< = \phi \end{cases}$$

Отметим, что множество D может содержать 0, 1 или бесконечно много элементов (ввиду определения множеств D), в зависимости от коэффициентов в системе уравнений (24). В случае $|D| = 1$, единственный элемент этого множества вычисляется по формулам из линейной алгебры, остается только проверить его принадлежность множеству $D_<$. В случае $|D| = \infty$ ясно, что все точки множества $D_<$ принадлежат одной прямой $a_1 i_1 = b_1 j_1 + d_1$. Предположим, что множество $D_<$ упорядочено по возрастанию i_1 (это возможно в случае $b_1 \neq 0$). Тогда все точки этого множества принадлежат отрезку, соединяющему первую и последнюю точки этого множества. Для того чтобы найти $\min\{j_1 - i_1\}$, $(i_1; j_1) \in D_<$ нам достаточно определить на упомянутом отрезке самую близкую точку к прямой $j_1 = i_1$. Понятно, что это будет один из концов этого отрезка. На вопрос, какой из двух концов – нам ответит угол наклона прямой $a_1 i_1 = b_1 j_1 + d_1$.

Обозначим через $T = \left\{1 \le i < N_1 \,\middle|\, \exists j : (i; j) \in D_< \right\}$. Обозначим через

$$t = \begin{cases} \min\limits_{i_1 \in T} i_1, & a_1 \ge b_1, \\ \max\limits_{i_1 \in T} i_1, & a_1 < b_1. \end{cases}$$

Тогда самая близкая точка – это $(t; b_1^{-1}(a_1 t - d_1))$. Тогда $\min\limits_{(i_1; j_1) \in D_<} \{j_1 - i_1\} = \dfrac{a_1 t - d_1}{b_1} - t$.

В случае $b_1 = 0$, получаем $a_1 \neq 0$ (т.к. $|D_<| \ge 1$). Тогда самая близкая точка – это $(a_1^{-1} d_1; a_1^{-1} d_1 + 1)$. Тогда $\min\limits_{(i_1; j_1) \in D_<} \{j_1 - i_1\} = \dfrac{d_1}{a_1} + 1 - \dfrac{d_1}{a_1} = 1$.

Обозначим через

$$m = \begin{cases} \dfrac{a_1 t - d_1}{b_1}, & |D| = +\infty, b_1 \neq 0, \\ 1, & |D| = +\infty, b_1 = 0, \\ \dfrac{a_1 d_2 - a_2 d_1 + b_2 d_1 - b_1 d_2}{a_2 b_1 - a_1 b_2}, & |D| = 1. \end{cases}$$

Подведём итог:

$$z = \begin{cases} \dfrac{w\left(N_2 - 1 + T_{it} + T_{send}\right)}{m}, & D_< \neq \phi \\ 0, & D_< = \phi \end{cases}$$

3.7. Выводы к третьей главе

В данной главе впервые рассмотрена задача отображения многомерных гнезд циклов (как тесных, так и не тесных) на многоконвейерную архитектуру. При этом используется информация о зависимостях, описанная с помощью решетчатых графов. Обсуждается влияние информационных зависимостей на синхронизацию и составление расписания.

Разработаны алгоритмы синхронизации конвейеров в рамках многоконвейерной модели вычислений, а также методы и алгоритмы эффективного автоматического отображения программ на многоконвейерную модель вычислений.

В главе также предложен упрощенный алгоритм отображения двумерных гнезд циклов на многоконвейерную архитектуру. Отметим, что двумерные гнезда циклов достаточно часто встречаются в реальных программах. Указанный алгоритм эффективнее решает поставленную задачу, чем общий алгоритм отображения многомерных гнезд циклов, но при этом класс программ, для которого он применим, меньше.

4. Вспомогательные результаты

4.1. Линейная форма представления выражений

В процессе разработки проектов ОРС и ДВОР потребовалось создать дополнительные программные инструменты для представления, хранения и обработки информации о программе. Одним из таких инструментов является представление выражений в линейной форме. Этот вспомогательный инструмент реализован лично автором в проекте ДВОР.

Определим, что такое анализ линейности выражения и сопутствующие ему термины. Предположим, дано выражение, состоящее из арифметических операций, переменных и констант. Линеаризацией выражения относительно набора переменных a_i называется преобразование произвольного выражения к виду: $e_1*a_1+e_2*a_2+\ldots+e_n*a_n+e_n+1$, где e_i – выражения, a_i – переменные, входящие в исходное выражение, и при этом ни одна из них не входит ни в одно из выражений e_i. Анализом линейности выражения называется анализ возможности осуществления линеаризации. Будем называть полученное представление линейной формой выражения (линейным разложением). Будем называть базой линеаризации набор переменных a_i, e_i – коэффициентами, и при этом e_n+1 – свободным членом линейного разложения.

В первых версиях ОРС использовался разбор выражений и представление их в линейной форме по алгоритмам, запрограммированным Д.Н. Черданцевым [51]. Эти алгоритмы предполагали наличие только целочисленных коэффициентов в линейных выражениях, более того все арифметические операции использовали семантику типа int языка С. В текущей (пятой) версии ОРС эту функциональность решено было расширить в соответствии с решением более широкого класса задач.

Разбор линейных выражений, в том числе и индексных выражений массивов, необходим для реализации следующих проектов ОРС:

- Граф информационных связей [73]
- Решетчатый граф
- Распараллеливание рекуррентных циклов
- Генерация MPI-кода

Разработка алгоритмов анализа и линеаризации выражений началась с простого случая – разбор индексных выражений в массивах с константными коэффициентами.

Например, дано выражение a[i+j-2]. Для построения графа зависимостей и решетчатого графа необходимо определить коэффициенты при счетчиках цикла и свободный член. В результате разбора индексного выражения пользователь получает информацию в виде набора чисел {1,1,-2}. А также есть возможность проверить является ли выражение линейным по некоторому набору переменных (база линеаризации).

После создания первой версии библиотеки классов, решающих задачу в указанном случае, появилась возможность запустить проекты «граф информационных связей» и «решетчатый граф», но только для вхождений массивов, содержащих выражения, зависящие только от счетчиков циклов. Реализация первой версии библиотеки позволила определить проблемы в решении задачи в целом и дальнейшие направления развития.

Во-первых, если пользователь пытается построить любой из упомянутых графов с параметрами, то и разбор выражений должен подразумевать возможность параметрических линейных выражений. Например, верхняя граница цикла может быть выражением, зависящим от параметров, и тогда граф информационных связей и решетчатый граф должны суметь получить нужную им информацию об этих выражениях.

Во-вторых, не всегда в качестве базы линеаризации выступают счетчики циклов. Например, для распараллеливания рекуррентных циклов база линеаризации состоит из вхождений массивов. А графу информационных зависимостей нужно получать информацию о коэффициентах при внешних параметрах (не счетчиках цикла), для чего также используется представление выражений в линейной форме.

В-третьих, при приведении подобных над константами выполняются арифметические операции и необходимо учесть семантику языка при вычислении коэффициентов. Например, если дано выражение i+127 – (j-1) надо учитывать типы коэффициентов и в зависимости от типов линейное выражение будет иметь либо коэффициенты {1,-1,-128} в 8-битной арифметике, либо {1,-1,128} в 16-битной (или более) арифметике. Таким образом, потребовалось усовершенствовать библиотеку классов так, чтобы она предусматривала упомянутые проблемы первой версии библиотеки.

Во второй версии библиотеки появилась возможность разбирать параметрические линейные выражения и отличать выражения с параметрами от выражений, зависящих только от переменных, входящих в базу линеаризации. Например, если для выражения *i*+2**j*-n в качестве базы линеаризации выбраны переменные *i, j*, то оно будет считаться зависящим от внешнего параметра n, а если в качестве базы линеаризации выбрать переменные *i, j, n*, то будет считаться не параметрическим.

Опыт разработки этой версии библиотеки позволил выделить еще несколько проблем в решении задачи в целом.

Во-первых, в качестве элементов базы линеаризации по-прежнему нельзя выбирать вхождения массивов, так как $a[i]$ и $a[i\text{-}1]$ не отличаются именем переменной, а отличаются индексными выражениями. Например, если дано рекуррентное выражение вида $a[i] = a[i\text{-}1] + a[i\text{-}2] - 2^* a[i\text{-}1]$, то собрать коэффициенты при разных вхождениях массива a невозможно. Но с использованием функций второй версии библиотеки появилась возможность разработать механизм сравнения на равенство двух вхождений массива a и учитывать при этом наличие параметров.

Во-вторых, при разработке второй версии библиотеки стало понятно, что для выполнения операций над константами разных типов необходимо универсальное представление констант с использованием длинных вычислений. Кроме того, необходимо разработать механизм преобразований констант разных типов с учетом семантики языка C, а в будущем и Fortran.

В-третьих, косвенная адресация не разбирается второй версией библиотеки.

Описанные проблемы решаются в реализации третьей версии библиотеки классов, позволяющих представить выражение в линейном виде.

4.2. Система тестирования

В процессе разработки проектов ОРС и ДВОР потребовалось создать дополнительные программные инструменты для тестирования параллельных программ и тестирования преобразований программ и графа информационных связей в ДВОР.

Кроме того, была проведена работа по тестированию пакетов программ входящих в состав высокопроизводительного программного комплекса HPC-NASIS [10].

В рамках выше упомянутых проектов было создано несколько программных инструментов для тестирования, таких как тестирование интерфейса, тестирование методом «белого ящика», тестирование методом «черного ящика». В этом параграфе будет рассказано о тестировании, с использованием метода «черного ящика», так как автор принимал непосредственное участие в разработке этого программного инструмента.

Тестирование методом «черного ящика» подразумевает полное отсутствие информации о структуре тестируемой программы. Но требуется информация о структуре

входных и выходных данных. Тогда появляется возможность протестировать на разных наборах входных данных, и сравнить результаты.

Тестирование методом «черного ящика» [40] позволяет проверять программы на корректность, масштабируемость, сравнение с эталоном, тестирование на случайном наборе входных данных.

Набор входных тестовых данных может быть заранее сформирован вручную или сгенерирован автоматически с помощью датчика случайных чисел. Вручную подготовленный тестовый набор входных данных позволяет с большей вероятностью обнаружить ошибку в программе. Автоматическая генерация входных наборов с помощью датчика случайных чисел позволяет прогнать через программу значительно большее количество тестов, чем при ручной подготовке.

Оба подхода – ручное и автоматическое тестирование – дополняют друг друга.

Должна быть возможность проверить правильность результатов выполнения программы на тестовых наборах входных данных. Если входные данные числовые, то иногда такая проверка может производиться автоматически.

В некоторых случаях для тестируемой программы имеется дублирующая программа, выполняющая те же функции, что и тестируемая, но, быть может, на другом компьютере, в другой среде программирования или разработанная по другому эквивалентному алгоритму. Особенно это относится к параллельным программам, поскольку перед созданием такой программы у многих разработчиков уже есть последовательный прототип.

Чтобы система автоматизации тестирования была ориентирована не на одну программу, а на некоторый класс программ, следует генератор входных данных и подсистему сравнения результатов сделать ориентированными на некоторый представительный класс программ. Генератор входных данных должен быть оснащен редактором данных. Подсистема сравнения выходных данных должна быть способна сравнивать выходные данные с некоторой заранее задаваемой точностью.

В соответствии с описанными выше требованиями, было разработано инструментальное программное средство BlackBoxTest [61, 88], предназначенное для автоматизации тестирования методом «черного ящика». В настоящее время это программное средство продолжает дорабатываться и расширяться [24].

Работа системы тестирования заключается в последовательном запуске тестируемой программы с различными входными данными и на различных конфигурациях вычислительной системы. По итогам каждого запуска автоматически

производится оценка успешности теста. По результатам выполнения всех тестов формируется файл отчета.

Входные и выходные данные тестируемых программ задаются в специальном формате IODataDescription. На сегодняшний день этот формат позволяет описать данные типа String, Integer, Double и Const String. Последовательности токенов описаний представляют собой описание строки файла с данными, а последовательность описания строк описывает весь файл целиком.

Это XML-описание можно создать как в обычном текстовом редакторе, так и в специальном редакторе IODataDesriptionEditor. Указанный редактор визуально представляет описание данных в файле как последовательность токенов, представленная в виде таблицы, в каждой ячейке которой можно указать описание того или иного данного.

Описание IODataDescription используется в библиотеке IODataTuner. С помощью этой библиотеки, система тестирования получает следующие возможности:

1. Генерация файлов со случайными входными данными по описанию.
2. Проверка корректности (соответствия описанию) входных/выходных файлов с данными.
3. Проверка эквивалентности двух файлов с выходными данными (с точки зрения описания). Если, например, один файл содержит вещественное число 3.1415, а второй 3.1414, и при этом в описании задана точность этого данного 0.001, то такие данные считаются равными, а файлы их содержащие – эквивалентными.

Отметим, что файлов, содержащих входные данные, может быть несколько и между ними могут быть семантические связи. Например, размерность матрицы хранится в одном файле, а сама матрица храниться в другом. В связи с этим описание входных данных представляет собой набор XML-файлов, которые кроме описания данных содержат ссылки на значения из других файлов, если это необходимо. Полный список входных файлов содержится в файле конфигурации входных данных.

Для файлов, содержащих выходные данные, ситуация аналогичная и потому описание выходных данных представляет собой набор XML-файлов и файл конфигурации выходных данных, который содержит полный список требуемых файлов с отдельными описаниями.

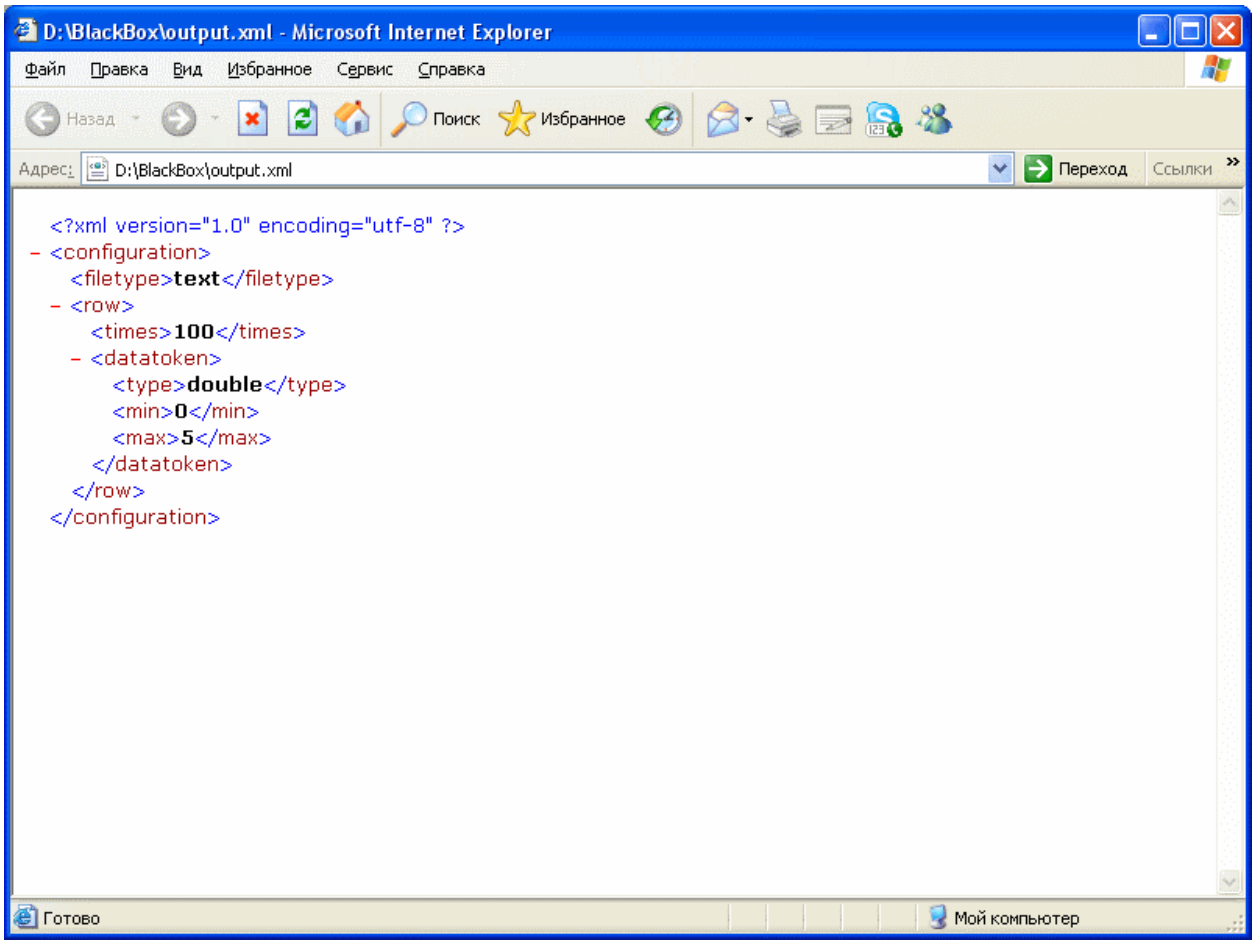

Рис. 46. Пример файла с описанием выходных данных

Входные/выходные данные могут иметь неочевидные множества допустимых значений, либо могут иметь структуру, зависимую от самих данных. Рассмотрим пример описания файла с данными: если файл содержит в первой строке 1, то дальше идет матрица 50x50, а если 0, то дальше будет вектор 1x50. Чтобы решить эту проблему, было решено ввести в описание форматов входных и выходных возможность добавлять свои скрипты на языке Python, которые будут описывать генерацию, проверку корректности и проверку эквивалентности данных. Скрипты являются опциональной возможностью, и если пользователь тестирующей системы хочет воспользоваться стандартными процедурами генерации, проверки корректности и проверки эквивалентности данных, то он не создает эти скрипты, и система по умолчанию будет работать стандартно с токенами, соответствующих входных/выходных файлов.

Скрипты хранятся в отдельных файлах с расширением «.py», а в xml-файлах описания файлов с данными указываются функции, которые требуются для генерации, проверки корректности и проверки эквивалентности данных.

На следующем рисунке демонстрируется процесс редактирования токена данных в редакторе IODatadescriptionEditor. Здесь можно увидеть, что для генерации данных выбрана функция generate_me из файла скриптов.

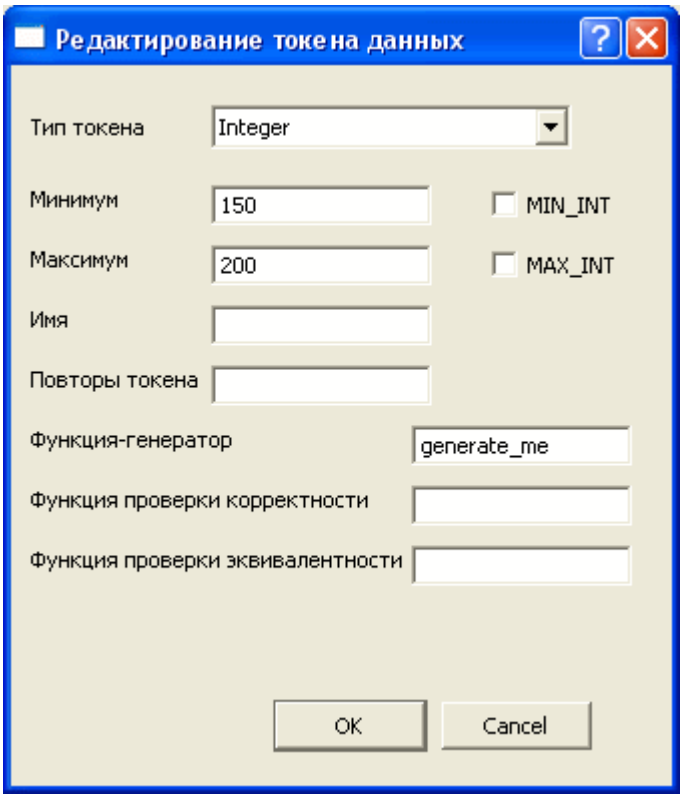

Рис. 47. Редактирование токена в редакторе IODataDescriptionEditor

Система тестирования выполнена в виде консольной программы и имеет следующие параметры командной строки:

- Путь к тестируемой программе.
- Файл конфигурации с описанием списка тестовых случаев.
- Имя списка тестовых случаев, который будет использован для выполнения тестирования.
- Тайм-аут завершения тестируемой программы в секундах.
- Подробную справку о параметрах командной строки можно получить, запустив программу с ключом «-?».

Кроме того, создан графический интерфейс пользователя, который позволяет осуществлять тестирование интерактивно.

Графический интерфейс позволяет:

- выбрать файл для тестирования
- выбрать XML-файл с конфигурацией тестирования

- указать имя тестового набора из файла конфигурации
- проверить параметры командной строки
- выполнить запуск тестирования
- отредактировать файлы с описаниями форматов входных данных

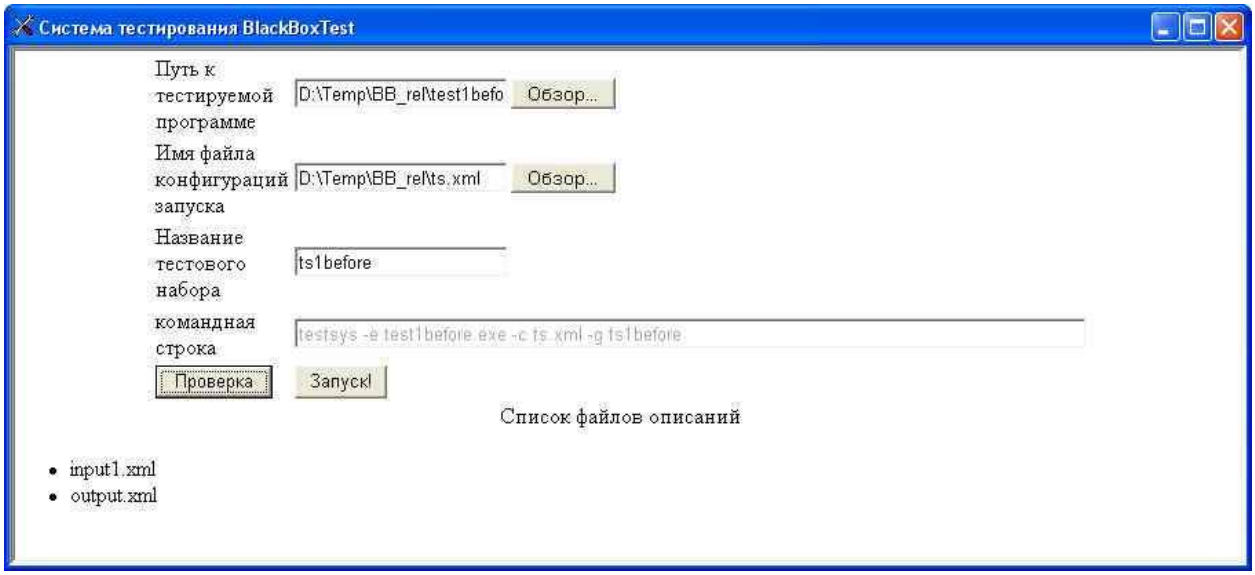

Рис. 48. Графический интерфейс

На последнем рисунке продемонстрирован вид графического интерфейса после выбора тестируемой программы, тестовой конфигурации, тестового набора. После нажатия кнопки «Проверка» появляется информация о командной строке запуска тестирующей системы и заполняется список файлов описаний входных и выходных данных. Нажав дважды левой кнопкой мыши на имени файла, открывается редактор IODataDescriptionEditor.

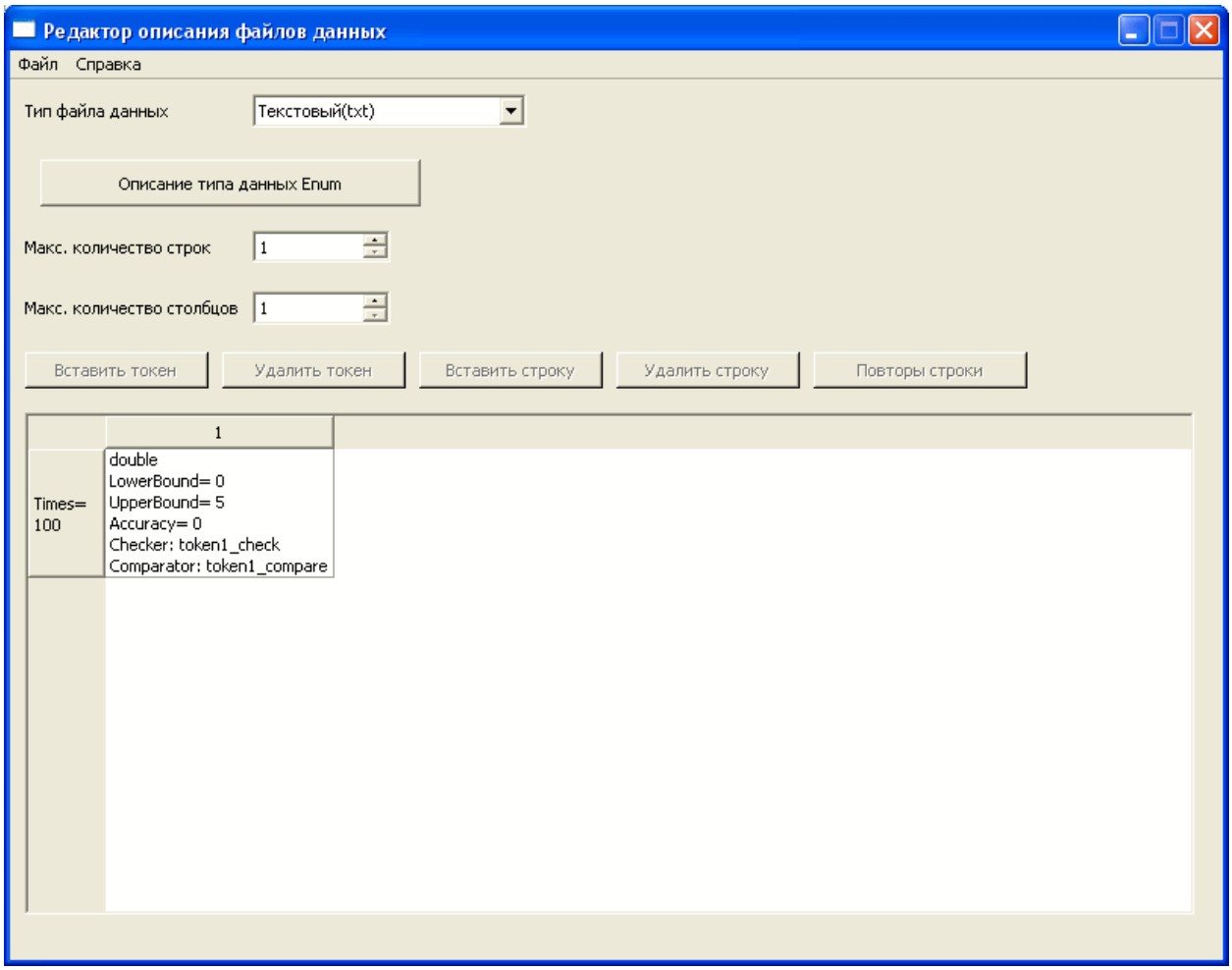

Рис. 49. Запуск редактора IODataDescriptionEditor из интерфейса тестирующей системы

Данная программная реализация позволяет тестировать программы с использованием следующих методик:

- тестирование на случайном наборе входных данных
- тестирование на предопределенном наборе входных данных с эталонными результатами
- тестирование параллельной версии программы на корректность
- тестирование масштабируемости параллельной программы на корректность
- тестирование преобразований программ

4.3. Применение автоматического построения графа вычислений к генерации HDL-описаний. Конвертер с языка C в VHDL

При разработке микросхем на базе ПЛИС используются различные подходы к процессу проектирования. В этом параграфе будем рассматривать один из достаточно распространенных подходов. Приведем примерную разбивку на этапы разработки микросхемы при использовании указанного подхода:

1. Составление ТЗ и согласование с Заказчиком.
2. Программист составляет программу на языке высокого уровня (как правило, C), реализующую алгоритм, который должен быть заложен в основу разрабатываемой микросхемы.
3. Тестирование программы и проверка алгоритма на соответствие ТЗ.
4. С учетом написанной программы, инженер-схемотехник разрабатывает описание на HDL языке.
5. Верификация программы на C и модели микросхемы.
6. Программирование ПЛИС, в соответствии с уже верифицированным описанием на HDL языке.

Пункт 5 этого процесса может занимать около двух месяцев для стандартных проектов. Таким образом, возникает задача оптимизации процесса разработки электронных схем. С этой задачей может справиться конвертер с языка C в один из HDL языков (VHDL или Verilog). Работы в этом направлении ведутся уже несколько лет [3, 4, 7, 25, 26, 50]. В качестве примеров зарубежных программных продуктов можно привести следующие:

- «Catapult C» от компании «Mentor Graphics»
- «C2Verilog» от компании «C Level Design»
- «C-To-FPGA» от компании «Stone Ridge Technology» (анонсирован 01.09.2010)

Это далеко не полный список продуктов, представленных на рынке, но все они характеризуются узким классом входных алгоритмов, которые реализованы аппаратно в виде шаблонов, а также высокой стоимостью самого ПО.

Проект ДВОР [62, 89], разрабатываемый в Южном федеральном университете, представляет собой программный инструментальный комплекс, ориентированный на разработку распараллеливающих и оптимизирующих компиляторов с языков

программирования высокого уровня (C, Fortran), систем полуавтоматического распараллеливания. ДВОР включает в себя набор лексических анализаторов, поддержку внутреннего представления программ (см. [42]), такие графовые представления программ, как граф информационных зависимостей, решётчатый граф, граф вызовов процедур, граф вычислений, а также большую библиотеку оптимизирующих преобразований, использующих графовые представления программ.

На базе проектов ОРС (см. [41, 57-59]) и ДВОР (см. [42, 62, 63]) в 2008 году была начата разработка конвертера C2VHDL [26]. Этот программный продукт должен работать по следующему алгоритму:

1. Разбор программы пользователя на языке С во внутреннее представление ДВОР [42].
2. Применение преобразований программ во внутреннем представлении так, чтобы получить несколько потенциально удачных с точки зрения распараллеливания эквивалентных представлений исходной программы.
3. Для каждого представления исходной программы строится граф вычислений.
4. По графу вычислений строится промежуточное внутреннее представление HDL-описания (возможно не одно).
5. По внутреннему представлению HDL-описания выводится текст на VHDL.

На этапе 4 планируется использовать различные шаблоны реализации стандартных алгоритмов сложения, деления и прочих элементарных операций, а также вычисления более сложных функций, таких как синус, косинус, умножение матриц и т.п.

Основным преимуществом перед всеми существующими на сегодняшний день программными средствами является то, что по программе на языке С будет сгенерировано целое семейство (на этапах 3 и 5) описаний схемы на VHDL. Причем эти описания будут получены с помощью оптимизирующих высокоуровневых преобразований исходной программы, ориентированных на распараллеливание на этапе 3. А на этапе 5 будут выбираться различные, подходящие для конкретной ПЛИС, реализации стандартных функций. Таким образом, инженер-схемотехник получит возможность выбора между различными вариантами автоматически сгенерированных описаний.

Более того, планируется отфильтровать сгенерированное семейство описаний. У описаний на HDL-языках есть конкретные числовые характеристики, как показатели качества описываемой схемы, такие как тактовая частота схемы, количество вентилей и т.д. Эти показатели могут предоставить специальные программные инструменты,

выпускаемые производителями ПЛИС. В связи с этим на множестве автоматически сгенерированных описаний можно установить частичный порядок по указанным критериям. Этот частичный порядок позволит отсеять те варианты сгенерированных описаний, которые получились хуже по всем показателям, чем другие варианты.

4.4. Выводы к четвертой главе

В данной главе изложены вспомогательные результаты, полученные в процессе работы над диссертацией. Тем не менее эти результаты имеют самостоятельную ценность. Так, например, линеаризация выражений используется в системе ДВОР не только при построении графа вычислений, но и для обработки вхождений переменных при построении графа информационных связей, при преобразовании рекуррентных выражений и при упрощении выражений.

Описывается система тестирования, позволяющая тестировать не только программы на масштабируемость, эквивалентность другой программе, но и преобразования программ.

Описывается экспериментальный конвертер C2HDL, позволяющий по программе, написанной на языке C (с некоторыми ограничениями), получить описание схемы, реализующей этот алгоритм на VHDL.

Заключение

В диссертационной работе описана математическая модель многоконвейерных вычислений. Проведены исследования по отображению программ на языке C на указанную модель.

В рамках работы разработан и сформулирован алгоритм синхронизации нескольких конвейеров, полученных в результате отображения многомерного гнезда циклов на многоконвейерную архитектуру. Доказаны теоремы показывающие корректность этого алгоритма.

Также приводятся формулы для вычисления характеристик конвейеров и их синхронизации, в случае двумерного тесного гнезда циклов. Этот частный случай видится практически значимым, так как подобные циклы встречаются довольно часто в реальных программах.

Получена модифицированная формула вычислений интервала инициализации итераций (*iii*) и доказана ее эквивалентность общеизвестной формуле. В работе сформулирован алгоритм вычисления интервала инициализации итераций, использующий модифицированную формулу вычисления *iii*. Предлагается анализировать только элементарные циклы и доказывается, что этого достаточно для получения точной оценки интервала инициализации итераций.

В работе сформулированы и обоснованы алгоритмы вычисления различных характеристик конвейера: алгоритм определения обратных дуг на графе вычислений, алгоритм составления расписания, алгоритм вычисления буферных задержек. Приводятся обоснования корректности этих алгоритмов.

В работе приводится пример конвертера с языка C в VHDL описание электронных схем. В основе указанного конвертера лежат граф вычислений и описанные в работе алгоритмы синхронизации конвейеров.

Некоторые теоретические результаты, полученные в диссертации, реализованы автором программно в рамках проектов ОРС и ДВОР: граф вычислений, алгоритм расчета *iii*, алгоритм расчета буферов и алгоритм построения расписания. Кроме того, автор реализовывал вспомогательные программные инструменты: тестирующую систему и линеаризацию выражений. Экспериментальный конвертор C2HDL, входящий в состав ДВОР, в основе своей реализации содержит разработанную автором программу построения графа вычислений.

Таким образом, можно заключить, что в диссертации разработаны методы автоматизации отображения программ на конвейерные и многоконвейерные вычислительные архитектуры.

Приложение 1

В этом приложении рассматривается проект специализированного устройства для быстрого вычисления локального выравнивания двух последовательностей по алгоритму Смита-Уотермана. В работе [1] рассматривается решение задачи локального выравнивания нуклеотидных последовательностей (биоинформатика) с помощью алгоритма Смита-Уотермана. Указанный алгоритм использует динамическое программирование. Рассмотрим основной цикл обработки данных, который вычисляет матрицу H.

```
// строим матрицу H по следующему принципу:
// находим максимум среди элементов:
//   0,
//   H[i][j-1] + w(S2[j],_ ),
//   H[i - 1][j - 1] + w( S1[i], S2[j] ),   (1)
//   H[i-1][j] + w( S1[i],_ ),
// где w(S1[i],S2[j]) = match, если S1[i] = S2[j],
// и w(S1[i],S2[j]) = mismatch в противном случае,
// w(S1[i],_ ) = w(S2[j],_ )= - 1;
// заносим максимумы матрицы H в стэк MaxLoc, при этом
// в стэке хранится не только максимальное значение, но
// и указатель на этот элемент в другом стэке, в который
// записываем для каждого элемента матрицы H индексы i и j и
// индексы элементов из которых мы получили этот элемент
// (согдасно построению (1))
for (i = 1; i <= M; i++ )
     for (j = 1; j <= N; j++ ){
          max = 0;
          maxI = i - 1;
          maxJ = j - 1;
          previous = 0;
          if( S1[i-1]==S2[j-1])
               wp0 = match;
          else
```

```
            wp0 = mismatch;
        if (S1[i-1] != '_')
            wp1 = mismatch;
        else
            wp1 = match;
        if (S2[j-1] != '_')
            wp2 = mismatch;
        else
            wp2 = match;

        if (H[i-1][j-1] + wp0 > max) {
            max = H[i-1][j-1] + wp0 ;
            maxI = i-1;
            maxJ = j-1;
            previous = 1;
        }
        if (H[i-1][j] + wp1 > max) {
            max = H[i-1][j] + wp1 ;
            maxI = i-1;
            maxJ = j;
            previous = 2;
        }
        if (H[i][j-1] + wp2 > max){
            max = H[i][j-1] + wp2 ;
            maxI = i;
            maxJ = j-1;
            previous = 3;
        }

        H[i][j] = max;
        Prev[i][j] = previous;

        if (H[i][j] >= maximum)
            maximum = H[i][j];
    }
```

Предлагается преобразовать эту программу к виду удобному для организации многоконвейерных вычислений. На сегодняшний день конвертер C2HDL [26] не делает подобные преобразования автоматически, но в будущем это планируется реализовать.

Необходимо избавится от выходных и антизависимостей в исходном цикле. Первым делом, отделим вычисление скалярных переменных wp0, wp1, wp2 в отдельный цикл и применим растягивание скаляров. Кроме того, чтобы не отвлекаться от вычисления массива H, опустим все рассуждения, которые относятся к вспомогательному массиву Prev, уберем его из рассматриваемого фрагмента программы и вернем уже в итоговом цикле. Итак, исходный фрагмент программы примет вид:

```
// цикл инициализации начальных значений
for (i = 1; i <= M; i++ )
     for (j = 1; j <= N; j++ )
     {
          if(S1[i-1] == S2[j-1])
               wp0[i][j] = match;
          else
               wp0[i][j] = mismatch;
          if (S1[i-1] != '_')
               wp1[i][j] = mismatch;
          else
               wp1[i][j] = match;
          if (S2[j-1] != '_')
               wp2[i][j] = mismatch;
          else
               wp2[i][j] = match;
     }
// основной цикл
for (i = 1; i <= M; i++ )
     for (j = 1; j <= N; j++ )
     {
          max = 0;
          if (H[i][j-1] + wp2[i][j] > max )
               max = H[i][ j-1] + wp2[i][j] ;
```

```
            if (H[i-1][j-1] + wp0[i][j] > max )
                 max = H[i-1][j-1] + wp0[i][j] ;

            if (H[i-1][j] + wp1[i][j] > max )
                 max = H[i-1][j] + wp1[i][j] ;

            H[i][j] = max;

            if (H[i][j] >= maximum)
                 maximum = H[i][j];
      }
```

Теперь основной цикл вычисляет максимум из трех выражений и нуля, а также запоминает глобальный максимум по всему массиву H. Обратим внимание на то, что массив H в нулевой строке и нулевом столбце заполнен нулями, а, следовательно, нам достаточно искать максимум только из трех выражений, стоящих в условиях операторов if. Введем в рассмотрение предикаты P0, P1, P2 соответствующие этим условиям. И изменим основной цикл следующим образом:

```
// основной цикл
for (i = 1; i <= M; i++ )
      for (j = 1; j <= N; j++ )
      {
            P0 = (H[i-1][j-1] + wp0[i][j] > H[i][j-1] + wp2[i][j])
&& (H[i-1][j-1] + wp0[i][j] > H[i-1][j] + wp1[i][j]);
            P1 = (H[i-1][j] + wp1[i][j] > H[i-1][j-1] + wp0[i][j])
&& (H[i-1][j] + wp1[i][j] > H[i-1][j-1] + wp2[i][j]);
            P2 = (H[i][j-1] + wp2[i][j] > H[i-1][j-1] + wp0[i][j])
&& (H[i][j-1] + wp2[i][j] > H[i-1][j] + wp1[i][j]);
            H[i][j] = P0*(H[i-1][j-1] + wp0[i][j]) + P1*(H[i-1][j]
+ wp1[i][j]) + P2*(H[i][j-1] + wp2[i][j]);
            if (H[i][j] >= maximum)
                 maximum = H[i][j];
      }
```

Анти- и выходные зависимости для полученного гнезда циклов будут иметь вид:

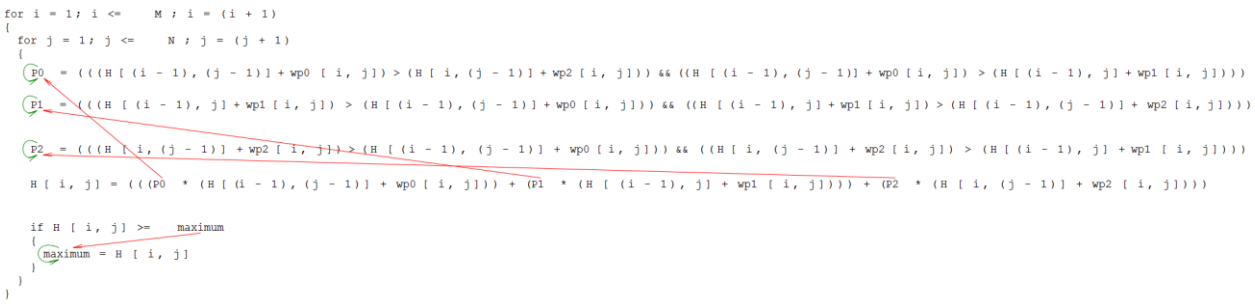

Рис. 50. Граф информационных связей (анти- и выходные зависимости) для основного гнезда циклов алгоритма Смита-Уотермана. Скриншот программы OpsDemo 5

Чтобы избавиться от анти- и выходных зависимостей в этом фрагменте программы достаточно сделать подстановку вперед для скалярных переменных P0, P1, P2. Оставшийся оператор if можно выделить в отдельный цикл с помощью разрезания циклов. Получаем:

```
// цикл инициализации начальных значений
for (i = 1; i <= M; i++ )
      for (j = 1; j <= N; j++ )
      {
            if(S1[i-1] == S2[j-1])
                  wp0[i][j] = match;
            else
                  wp0[i][j] = mismatch;
            if (S1[i-1] != '_')
                  wp1[i][j] = mismatch;
            else
                  wp1[i][j] = match;
            if (S2[j-1] != '_')
                  wp2[i][j] = mismatch;
            else
                  wp2[i][j] = match;
      }
// основной цикл
for (i = 1; i <= M; i++ )
      for (j = 1; j <= N; j++ )
            H[i][j] = ((H[i-1][j-1] + wp0[i][j] > H[i][j-1] +
```

```
wp2[i][j]) && (H[i-1][j-1] + wp0[i][j] > H[i-1][j] +
wp1[i][j]))*(H[i-1][j-1] + wp0[i][j]) + ((H[i-1][j] + wp1[i][j]
> H[i-1][j-1] + wp0[i][j]) && (H[i-1][j] + wp1[i][j] > H[i-1][j-
1] + wp2[i][j]))*(H[i-1][j] + wp1[i][j]) + ((H[i][j-1] +
wp2[i][j] > H[i-1][j-1] + wp0[i][j]) && (H[i][j-1] + wp2[i][j] >
H[i-1][j] + wp1[i][j]))*(H[i][j-1] + wp2[i][j]);

// поиск глобального максимума
for (i = 1; i <= M; i++ )
     for (j = 1; j <= N; j++ )
          if (H[i][j] >= maximum)
               maximum = H[i][j];
```

Для полученного основного цикла граф вычислений будет иметь вид:

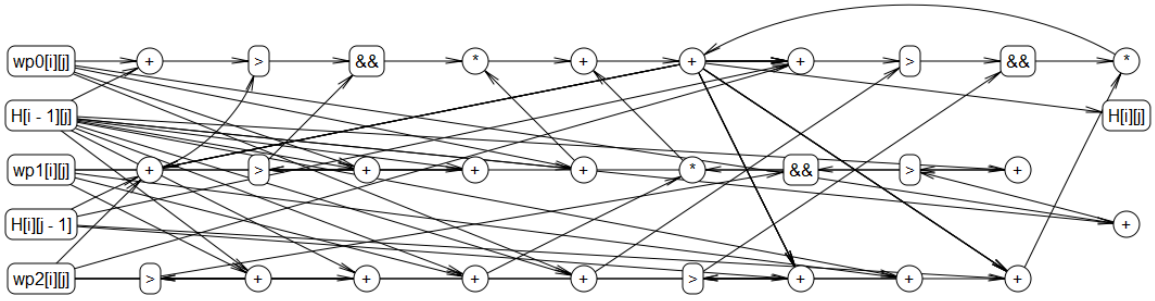

Рис. 51. Граф вычислений для основного цикла алгоритма Смита-Уотермана. Скриншот программы OpsDemo 5

Этот граф показывает настройки отдельного конвейера, семейство которых может быть запущено на многоконвейерной системе. Для удобства восприятия картинки перегруппируем (вручную) вершины.

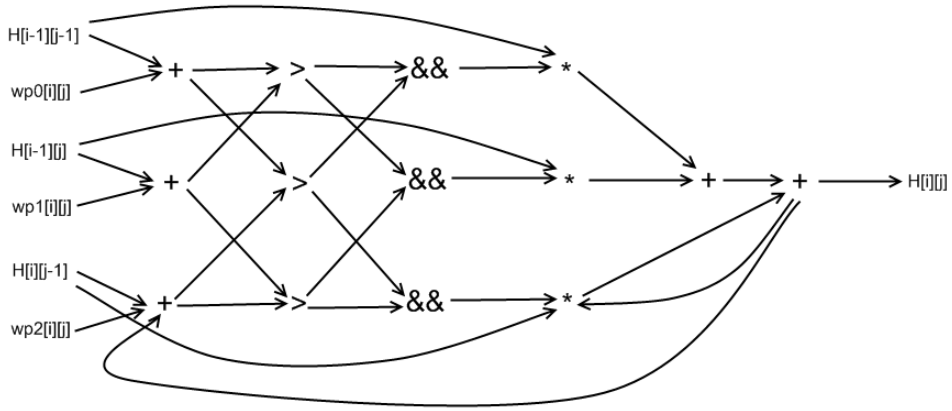

Рис. 52. Схема конвейера, вычисляющего матрицу H в алгоритме Смита-Уотермана

Чтобы определить межконвейерные связи необходимо построить решетчатый граф.

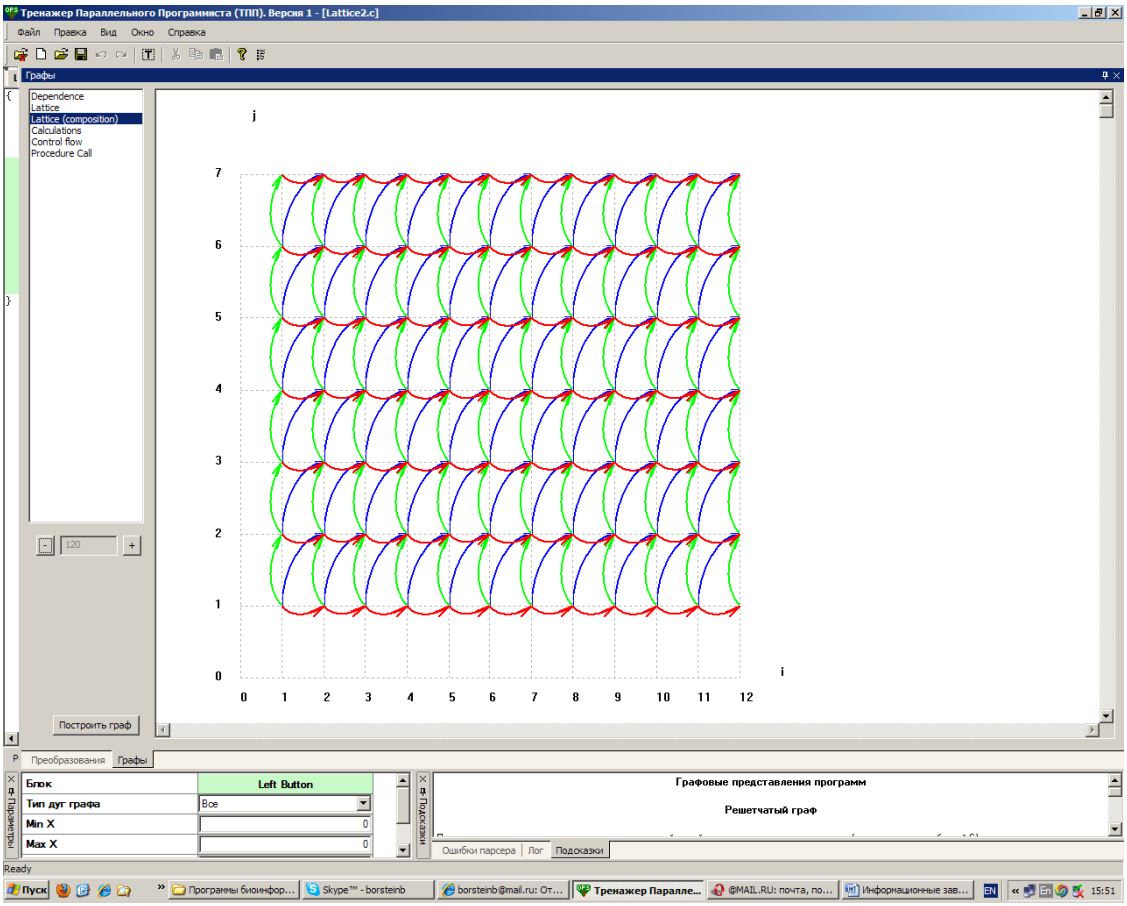

Рис. 53. Решетчатый граф зависимостей для основного цикла, вычисляющего матрицу H в алгоритме Смита-Уотермана. Скриншот программы ТПП

Учитывая диагональные (синие) и горизонтальные (красные) дуги, можно построить многоконвейерную схему вычисления матрицы H следующим образом:

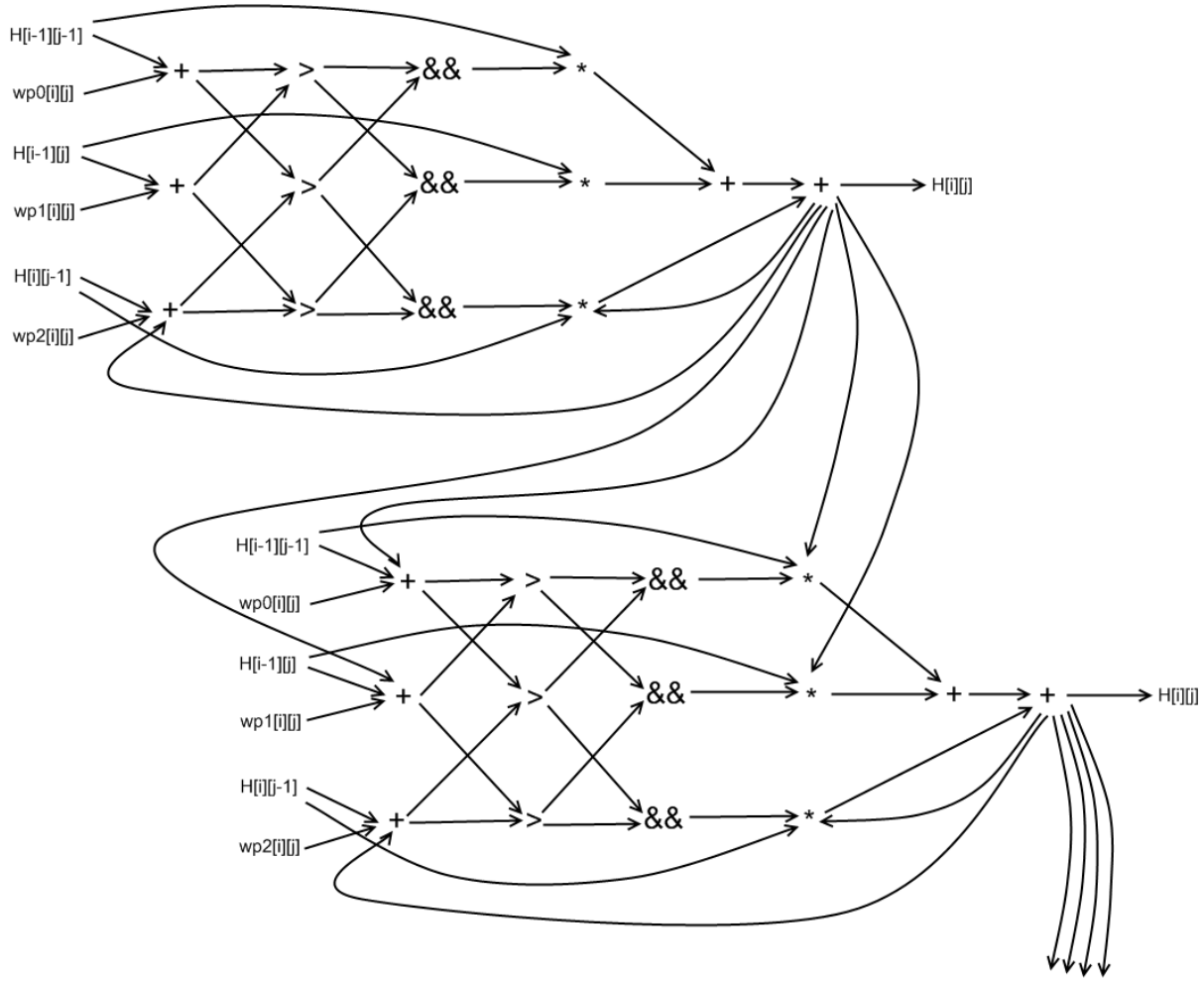

Рис. 54. Схема многоконвейерных вычислений, реализующих алгоритм Смита-Уотермана

Приведем итоговый фрагмент программы с учетом всех преобразований.

```
// цикл инициализации начальных значений
for (i = 1; i <= M; i++ )
     for (j = 1; j <= N; j++ )
     {
          if(S1[i-1] == S2[j-1])
               wp0[i][j] = match;
          else
```

```
                wp0[i][j] = mismatch;
            if (S1[i-1] != '_')
                wp1[i][j] = mismatch;
            else
                wp1[i][j] = match;
            if (S2[j-1] != '_')
                wp2[i][j] = mismatch;
            else
                wp2[i][j] = match;
        }
// основной цикл
for (i = 1; i <= M; i++ )
        for (j = 1; j <= N; j++ )
        {
            H[i][j] = ((H[i-1][j-1] + wp0[i][j] > H[i][j-1] +
wp2[i][j]) && (H[i-1][j-1] + wp0[i][j] > H[i-1][j] +
wp1[i][j]))*(H[i-1][j-1] + wp0[i][j]) + ((H[i-1][j] + wp1[i][j]
> H[i-1][j-1] + wp0[i][j]) && (H[i-1][j] + wp1[i][j] > H[i-1][j-
1] + wp2[i][j]))*(H[i-1][j] + wp1[i][j]) + ((H[i][j-1] +
wp2[i][j] > H[i-1][j-1] + wp0[i][j]) && (H[i][j-1] + wp2[i][j] >
H[i-1][j] + wp1[i][j]))*(H[i][j-1] + wp2[i][j]);
            Prev[i][j] = P0+2*P1+3*P2;
        }
// поиск глобального максимума
for (i = 1; i <= M; i++ )
        for (j = 1; j <= N; j++ )
            if (H[i][j] >= maximum)
                maximum = H[i][j];
```

Далее конвертер C2HDL должен сгенерировать описание схемы на VHDL. На сегодняшний день, конвертер еще не генерирует VHDL для подобных программ, но в ближайшее время это будет реализовано.

Тем не менее, учитывая тот факт, что полученная вычислительная система синхронна, можно произвести теоретическую оценку времени выполнения алгоритма на ПЛИС, в соответствии с рассчитанными на графе вычислений параметрами конвейеров.

Итак, на полученном графе вычислений есть следующие операции: целочисленные сложение и умножение, а также чтение и запись данных в память. Предположим, что сложение и умножение выполняются 5 тактов, а чтение и запись – 1 такт. Тогда расчеты показывают, что *iii* = 23 такта, *z* = 24 такта. Предположим также, что ПЛИС такова, что на ней достаточно «места», чтобы разместить 10 конвейеров, тактовая частота полученной схемы 125 MHz. Все значения описанных параметров ПЛИС, упомянутые здесь, выбраны так, что на большинстве ПЛИС эти значения вполне достижимы при создании реальных устройств.

Воспользуемся формулой (4), с учетом того, что на ПЛИС будет расположено 10 конвейеров. Полученные значения необходимо перевести из тактов в секунды, с учетом указанной выше частоты 125 MHz. В результате получим следующую таблицу.

Таблица 6

Длина строк	Время , с					Теоретическая оценка времени при конвейе
	Алгоритм Смита-Уотермана (последовательный)	Метод гиперплоскостей (последовательно)	Метод гиперплоскостей (параллельно, 4 ядра)	Блочный алгоритм (последовательно)	Блочный алгоритм (параллельно, 4 ядра)	
15796 15608 (размер блока 200)	14.8	31.1	18.6	12.8	7.1	4,5
111640 111630 (размер блока 550)	-	-	-	600	350	229
223280 223260 (размер блока 800)	-	-	-	2522	1463	919
307662 310525 (размер блока 1000)	-	-	-		2442	1741

Данные в первой колонке определяют размерность входных данных. В колонках с номерами 2–6 (считая слева направо) приведены данные полученные экспериментально Ж.М. Абу-Халил в рамках работы [1]. В последней (седьмой) колонке приведены численные оценки времени выполнения этого же алгоритма, но с использованием многоконвейерных вычислений на ПЛИС, с параметрами описанными в предыдущих абзацах.

Таким образом, ожидаемое (теоретическое) время меньше на 30-40%, чем время при решении этой же задачи на 4-х ядерном процессоре с использованием технологии OpenMP и технологии разбиения на блоки (tiling).

Литература

1. Абу-Халил Ж.М. Локальное выравнивание двух последовательностей, содержащих разрывы: материалы IV Международной научно-практическая конференции «Актуальные проблемы биологии, нанотехнологий и медицины». Ростов-на-Дону, 22-25 сентября 2011г. С. 42.
2. Адигеев М.Г. Разбиение программы на кадры для конвейерных вычислительных систем с динамически перестраиваемой архитектурой // Известия высших учебных заведений. Северо-Кавказский регион. 2009. №3. С. 5–7.
3. Андреев С.С., Дбар С.А., Лацис А.О., Плоткина Е.А. Система программирования. Автокод HDL и опыт ее применения для схемной реализации численных методов в FPGA // Научный сервис в сети Интернет: масштабируемость, параллельность, эффективность: труды Всероссийской суперкомпьютерной конференции. Новороссийск, 21-26 сентября 2009 г., М.: Изд-во МГУ, 2009. С. 237.
4. Андреев С.С., Давыдов А.А., Дбар С.А., Лацис А.О., Плоткина Е.А. О моделях и технологиях программирования суперкомпьютеров с нетрадиционной архитектурой // Научный сервис в сети Интернет: суперкомпьютерные центры и задачи: труды Всероссийской научной конференции. Новороссийск, 20-25 сентября 2010г., М.: Изд-во МГУ, 2010. С. 186-187.
5. Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов. М.: Мир, 1979. 536 с.
6. Бобков С.Г. Методика проектирования микросхем для компьютеров серии «Багет» // Информационные технологии. 2008. №3.
7. Бозоян Ш.Е., Егиазарян В.С. Язык Alex описания схем // Информационные технологии. 2007. №4.
8. Булычев Д.Ю. Разработка программно-аппаратных систем на основе описания макроархитектуры // Системное программирование. СПб., 2004. С. 7-22.
9. Булычев Д.Ю., Симановский А.А., Смирнов М.Н., Шапоренков Д.А. PADLA-based Codesign Toolset: технический отчет / Санкт-Петербургский государственный университет, институт информационных технологий. СПб, 2002.
10. Бухановский А.В., Ковальчук С.В., Марьин С.В. Интеллектуальные высокопроизводительные программные комплексы моделирования сложных систем: концепция, архитектура и примеры реализации // Известия вузов. Приборостроение. 2009. Т. 52, № 10.

11. Бухановский А. В., Марьин С. В., Князьков К. В., Сиднев А. А., Жабин С. Н., Баглий А. П., Штейнберг Р. Б., Шамакина А. В., Воеводин В. В., Головченко Е. Н., Фалалеев Р. Т., Духанов А. В., Тарасов А. А., Шамардин Л. В., Моисеенко А. И. Результаты реализации проекта «Мобильность молодых ученых» в 2010 году: развитие функциональных элементов технологии iPSE и расширение состава прикладных сервисов // Известия вузов. Приборостроение. 2011. № 10. С. 80-87.
12. Вальковский В.А. Параллельное выполнение циклов. Метод параллелепипедов // Кибернетика. 1982. № 2. С. 51-62.
13. Вальковский В.А. Параллельное выполнение циклов. Метод пирамид // Кибернетика. 1983. № 5. С. 51-55.
14. Воеводин В. В., Воеводин Вл.В. Параллельные вычисления. СПб: БХВ-Петербург, 2002. 608 с.
15. Воеводин В. В., Пакулев В.В. Определение дуг графа алгоритма. Препринт № 228 М.: Отд. вычисл. математики АН СССР. 1989. 22 с.
16. Воеводин В. В. Математические основы параллельных вычислений. М.: МГУ, 1991. – 345 с.
17. Воеводин В. В. Математические модели и методы в параллельных процессах. М.: Наука, 1986. 296 с.
18. Воеводин Вл. В. Статистические оценки возможности выявления параллельной структуры последовательных программ // Программирование. 1990. №4. С. 44-54.
19. Воеводин В. В. Информационная структура алгоритмов. М.: МГУ, 1997. 139 с.
20. Вьюкова Н.И., Галатенко В.А., Самборский С.В. Программная конвейеризация циклов методом планирования по модулю // Программирование. 2007. №6.
21. Вьюкова Н.И., Галатенко В.А., Самборский С.В. Использование ЦЛП-подхода для совмещения программной конвейеризации внутреннего цикла с разверткой внешних циклов при компиляции гнезда вложенных циклов : труды научной конференции РАСО-2008. М.: ИПУ РАН, 2008. С 1208-1220.
22. Вьюкова Н.И., Галатенко В.А., Самборский С.В. Обобщение задачи программной конвейеризации для гнезд циклов // Информационные технологии. 2009. №3.
23. Гуда С.А. Оценки длины критического пути в решетчатом графе // Труды конференции РАСО-2008. М.: ИПУ РАН, 2008. С 1253-1267.
24. Голозубов А.О. Разработка пользовательского интерфейса для системы автоматизированного тестирования параллельных программ // Научный сервис в сети Интернет: суперкомпьютерные центры и задачи: труды международной

суперкомпьютерной конференции. Новороссийск, 20-25 сентября 2010г., М.: Изд-во МГУ, 2010. С. 659-662.

25. Давыдов С.В., Давыдов Д.А., Фоменко С.А. Автоматизированная система проектирования устройств трехмерной голографической памяти // Информационные технологии. 2007. №12.
26. Дубров Д.В., Штейнберг Р.Б. Экспериментальный конвертер с языка С в HDL на основе диалогового высокоуровневого оптимизирующего распараллеливателя // Труды конференции РАСО-2010. М.: ИПУ РАН, 2010. С. 865-878.
27. Каляев А.В. Многопроцессорные однородные вычислительные структуры // Радиоэлектроника., 1978. №12. С. 5-17.
28. Каляев А.В., Левин И.И. Модульно-наращиваемые многопроцессорные системы со структурно-процедурной организацией вычислений. М.: Янус-К, 2003. 380 с.
29. Каляев И.А., Левин И.И., Семерников Е.А., Шмойлов В.И. Реконфигурируемые мультиконвейерные вычислительные структуры. Изд. 2-е, перераб. и доп. / под общ. ред. И.А. Каляева. Ростов-н/Д: Изд-во ЮНЦ РАН, 2009. 344с.
30. Касьянов В.Н., Евстигнеев В.А. Графы в программировании: обработка, визуализация и применение. СПб.: БХВ-Петербург, 2003. 1104 с.
31. Корнеев В.В. Архитектура вычислительных систем с программируемой структурой // Новосибирск: Наука, 1985.
32. Корнеев В.В. Программная настраиваемость аппаратной структуры // Открытые системы. 2007. №10.
33. Корнеев В.В. Следующее поколение суперкомпьютеров // Открытые системы. 2008. №8.
34. Корнеев В.В. Модель программирования: смена парадигмы. // Открытые системы. 2010. №3.
35. Кочерга М.С., Шмойлов В.И. Построение реконфигурируемых вычислительных систем на однородных вычислительных средах // Вестник ЮНЦ РАН. 2008. Т. 4, №2.
36. Коуги П.М. Архитектура конвейерных ЭВМ. М., Радио и связь, 1985. 358с.
37. Крюков В.А., Клинов М.С. Автоматическое распараллеливание Фортран-программ. Отображение на кластер // Вестник Нижегородского университета им. Н.И. Лобачевского. 2009. № 2. С. 128–134.
38. Курин Е.А. Сейсморазведка и суперкомпьютеры // Научный сервис в сети Интернет: суперкомпьютерные центры и задачи: труды международной суперкомпьютерной

конференции. Новороссийск, 20-25 сентября 2010г., М.: Изд-во МГУ, 2010. С. 366-372.

39. Лиходед Н.А. Распределение операций и массивов данных между процессорами// Программирование. 2003. №3. С. 73-80.
40. Мейерс Г. Искусство тестирования программ. М.: Финансы и статистика, 1982.
41. Петренко В. В. Внутреннее представление ОРС 3// Научный сервис в сети Интернет: технологии распределенных вычислений: труды Всероссийской научной конференции. Новороссийск, 19-24 сентября 2005 г., М.: Изд-во МГУ, 2005. С. 106-108.
42. Петренко В.В. Внутреннее представление Reprise распараллеливающей системы // Параллельные вычисления и задачи управления: труды IV международной конференции PACO-2008. Москва, 27-29 октября 2008г., М.: ИПУ РАН. С. 1241-1245.
43. Пузырев Д.А., Немнюгин С.А., Петров А.Ю. Оптимизация высокопроизводительных вычислений, предоставляемая в качестве веб-сервиса // Научный сервис в сети Интернет: суперкомпьютерные центры и задачи: труды международной суперкомпьютерной конференции. Новороссийск, 20-25 сентября 2010г., М.: Изд-во МГУ, 2010. С. 509-510.
44. Самофалов К.Г., Луцкий Г.М. Основы теории многоуровневых конвейерных вычислительных систем. М.: Радио и связь, 1989. 272с.
45. Сергиенко А.М. VHDL для проектирования вычислительных устройств. – Киев.: ЧП «Корнейчук»; ООО «ТИД «ДС», 2003. 208 с.
46. Суперкомпьютерные технологии в науке, образовании и промышленности / под редакцией акад. РАН В.А. Садовничего, акад. РАН Г.И. Савина, чл.-кор. РАН Вл.В. Воеводина. М.: Изд-во МГУ, 2009. 231с.
47. Французов Ю.А. Обзор методов распараллеливания кода и программной конвейеризации // Программирование, 1992. № 3. С. 16-37.
48. Харари Ф. Теория графов. М.: Мир, 1973. 300 с.
49. Хаханова И.В. Модели проектирования конвейерных устройств цифровой обработки сигналов // Радиоэлектроника и информатика. 2006. №3. С 53-58.
50. Хаханова И.В. Модели и методы системного проектирования вычислительных структур на кристаллах для цифровой обработки сигналов. Диссертация на соискание ученой степени доктора технических наук. Харьков: ХНУРЭ, 2008. С.334.

51. Черданцев Д.Н. Линеаризация выражений в оптимизирующих или распараллеливающих компиляторах // Известия вузов. Северо-Кавказский регион. Естественные науки. 2009. №2.
52. Штейнберг Б. Я. Математические методы распараллеливания рекуррентных циклов для суперкомпьютеров с параллельной памятью // Ростов н/Д: Изд-во РГУ, 2004. 192 с.
53. Штейнберг Б. Я. Подстановка и переименование в многомерных циклах при распараллеливании // СуперЭВМ и многопроцессорные вычислительные системы (МВС-2002): материалы международной научно-технической конференции, Таганрог, Россия, 26-30 июня 2002 г. С. 161-164.
54. Штейнберг Б. Я. Открытая распараллеливающая система // Параллельные вычисления и задачи управления: труды международной конференции PACO-2001. М.: ИПУ РАН, 2001. С. 214-220.
55. Штейнберг Б. Я. Распараллеливание программ для суперкомпьютеров с параллельной памятью и Открытая распараллеливающая система: диссертация на соискание ученой степени доктора технических наук. Ростов н/Д: РГУ, 2004.
56. Штейнберг Б.Я., Арутюнян О.Э., Бутов А.Э., Гуфан К.Ю., Морыле Р.И., Науменко С.А., Петренко В.В., Тузаев А., Черданцев Д.Н., Шилов М.В., Штейнберг Р.Б., Шульженко А.М. Обучающая распараллеливанию программа на основе ОРС // Современные информационные технологии в образовании: Южный федеральный округ: труды научно-методической конференции. Ростов-на-Дону, 12-15 мая 2004 г. С. 248-250.
57. Штейнберг Б.Я., Бутов А.Э., Науменко С.А., Петренко В.В., Черданцев Д.Н., Штейнберг Р.Б., Шульженко А.М. Полуавтоматическое распараллеливание на основе ОРС // Параллельные вычисления в задачах математической физики: труды Всероссийской научно-технической конференции. Ростов-на-Дону, 21-25 июня 2004г. С. 186-194.
58. Штейнберг Б. Я., Макошенко Д. В., Черданцев Д. Н., Шульженко А. М. Внутреннее представление в Открытой распараллеливающей системе // Искусственный интеллект (научно-теоретический журнал) / Институт проблем искусственного интеллекта НАНУ. Украина, Донецк: ДонДИШИ; Наука и Освита. 2003. №4. С. 89-97.
59. Штейнберг Б.Я., Нис З.Я., Петренко В., Черданцев Д., Штейнберг Р.Б., Шульженко А.М. Открытая распараллеливающая система 2006.: труды

международной конференции «Параллельные вычисления и задачи управления» PACO'2006. Москва, 2-4 октября 2006 г.М: ИПУ РАН, 2006. С. 526-541.

60. Штейнберг Б.Я., Нис З.Я., Петренко В., Черданцев Д., Штейнберг Р.Б., Шульженко А.М. Состояние и возможности открытой распараллеливающей системы (лето 2006г.): труды семинара «Наукоемкое программное обеспечение» в рамках VI международной конференции памяти академика А.П. Ершова «Перспективы систем информатики». Новосибирск, Академгородок, 28-29 июня 2006 г. Новосибирск, 2006. С. 122-125.
61. Штейнберг Б.Я., Алымова Е.В., Баглий А.П., Морылев Р.И., Нис З.Я., Петренко В.В., Штейнберг Р.Б. Автоматизация тестирования элементов высокопроизводительного программного комплекса // Научный сервис в сети Интернет: масштабируемость, параллельность, эффективность: труды Всероссийской суперкомпьютерной конференции. Новороссийск, 21-26 сентября 2009 г. М.: Изд-во МГУ, 2009. С. 287-291.
62. Штейнберг Б.Я., Абрамов А.А., Алымова Е.В., Баглий А.П., Гуда С.А., Дубров Д.В., Кравченко Е.Н., Морылев Р.И., Нис З.Я., Петренко В.В., Полуян С.В., Скиба И.С., Шаповалов В.Н., Штейнберг О.Б., Штейнберг Р.Б., Юрушкин М.В. Диалоговый высокоуровневый оптимизирующий распараллеливатель (ДВОР) // Научный сервис в сети Интернет: суперкомпьютерные центры и задачи: труды международной суперкомпьютерной конференции. Новороссийск, 20-25 сентября 2010г. М.: Изд-во МГУ, 2010. С. 71-75.
63. Штейнберг Б.Я., Абрамов А.А., Баглий А.П., Морылев Р.И., Петренко В.В., Полуян С.В., Штейнберг Р.Б. Уточнение зависимостей программы в ДВОР // Параллельные вычисления и задачи управления: труды международной конференции PACO-2010. Москва, 26-28 октября 2010 г. М.: ИПУ РАН. С. 855-864.
64. Штейнберг Б.Я., Кравченко Е.Н., Морылев Р.И., Нис З.Я., Петренко В.В., Скиба И.С., Шаповалов В.Н., Штейнберг О.Б., Штейнберг Р.Б. Особенности реализации распараллеливающих преобразований программ в системе ДВОР // Известия вузов. Приборостроение. 2011. № 10. С. 87-89.
65. Штейнберг Р.Б. Вычисление задержки в стартах конвейеров для суперкомпьютеров со структурно процедурной организацией вычислений // Искусственный интеллект (научно-теоретический журнал) / Институт проблем искусственного интеллекта НАНУ. Украина, Донецк: ДонДИШИ; Наука и Освита. 2003. № 4. С. 105-112.

66. Штейнберг Р.Б. Реализация модели конвейерных вычислений в ОРС // XIII Международная конференция «Математика. Экономика. Образование» : тезисы докладов. Ростов н/Д, 2005. 195 с. ISBN 5-94153-097-8.
67. Штейнберг Р.Б. Некоторые оптимизирующие преобразования программ для компьютеров, допускающих многоконвейерную обработку данных // Научный сервис в сети Интернет: технологии распределенных вычислений: труды Всероссийской научной конференции. Новороссийск, 19-23 сентября 2005г. М.: Изд-во МГУ, 2005. С. 85-87.
68. Штейнберг Р.Б. Вычисление задержки между стартами конвейеров с учётом времени пересылки данных // Труды международной конференции «Параллельные вычисления и задачи управления» PACO-2006. Москва, 2-4 октября 2006 г. М: ИПУ РАН, 2006. С. 542-564.
69. Штейнберг Р.Б. Использование решетчатых графов для исследования многоконвейерной модели вычислений // Известия вузов. Северо-Кавказский регион. Естественные науки. 2009. №2. С. 16-18.
70. Штейнберг Р. Б. Отображение гнезд циклов на многоконвейерную архитектуру // Программирование. 2010. №3. С. 69–80.
Steinberg R. Mapping loop nests to multipipelined architecture // Programming and Computer Software. MAIK Nauka/Interperiodica distributed exclusively by Springer Science+Business Media LLC. May 2010. Vol 36. №3. P. 177-185.
71. Шульженко А. М. Преимущества определения информационных зависимостей в программе с помощью решетчатых графов // XIII Международная конференция «Математика. Экономика. Образование»: тезисы докладов. Ростов н/Д, 2005. 195 с. ISBN 5-94153-097-8. С. 137
72. Шульженко А. М. Расщепление многомерных циклов. // Труды всероссийской научно-технической конференции «Параллельные вычисления в задачах математической физики». Ростов-на-Дону, 21-25 июня 2004 г. С. 186--194.
73. Шульженко А. М. Решетчатый граф и использующие его преобразования программ в Открытой распараллеливающей системе: труды Всероссийской научной конференции «Научный сервис в сети Интернет: технологии распределенных вычислений». Новороссийск, 19-24 сентября 2005 г. С. 82-85.
74. Шульженко А. М. Автоматическое определение циклов ParDo в программе // Известия вузов. Северо-Кавказский регион. Естественные науки. Приложение 11, 2005. С. 77–88.

75. Яджак М.С. Высокопараллельные алгоритмы и методы для решения задач массовых арифметических и логических вычислений: диссертация на соискание ученой степени д.ф.-м.н. / Институт прикладных проблем механики и математики. Львов, 2009. 298с. (на украинском языке).
76. Allen R., Kennedy K. Optimizing compilers for modern architectures. Morgan Kaufmann Publishers, An Imprint of Elsevier. 2002. 790 p.
77. Bondalapati K. Modeling and Mapping for Dynamically Reconfigurable Hybrid Architecture. Ph.D. Thesis, University of Southern California, August, 2001.
78. Dally W. et al. Merrimac: Supercomputing with Streams SC'03, November 15-21, 2003, Phoenix, Arizona, USA.
79. Feautrier P. Data Flow Analysis for Array and Scalar References // International Journal of Parallel Programming / 1991. Vol. 20. №1. Feb. P. 23-53.
80. Feautrier P. Some efficient solutions for the affine scheduling problem. Part 1. One-dimentional Time. // International Journal of Parallel Programming. 1992. Vol. 21. № 5, Oct.
81. Lamport L. The parallel execution of DO loops // Commun. ACM. 1974. Vol.17. №.2. P.83-93.
82. Lamport L. The Coordinate Method for the parallel execution of DO loops // Sagamore Computer Conference on Parallel Processing. 1973. P. 1-12.
83. O'Neal T. New Supercomputing Model Enables Breakthrough Findings in Plate Tectonics // Supercomputing online – online journal. 2010. Aug.
84. Pugh W. The Omega test: a fast and practical integer programming algorithm for dependence analysis // Communications of ACM. 1992. Aug. 8:102-114.
85. Pugh W., Wonnacott D. Going beyond integer programming with Omega test to eliminate false data dependences // Technical Report CS-TR-2993, Dept. of Computer Science, University of Maryland, Collage Park. 1992. Dec. An earlier version of this paper appeared at the SIGPLAN PLDI'92 conference.
86. Ramakrishna Rau B. Iterative Modulo Scheduling: An algorithm for software pipelining loops // MICRO 27 Proceedings of the 27th annual international symposium on Microarchitecture. P. 63-74.
87. Steinberg B., Abramov A., Alymova E., Baglij A., Guda S., Demin S., Dubrov D., Ivchenko A., Kravchenko E., Makoshenko D., Molotnikov Z., Morilev R., Nis Z., Petrenko V., Povazhnij A., Poluyan S., Skiba I., Suhoverkhov S., Shapovalov V., Steinberg O., Steinberg R. Dialogue-based Optimizing Parallelizing Tool and C2HDL Converter.

Proceedings of IEEE East-West Design & Test Symposium (EWDTS'09). Moscow, Russia, 2009. Sept. 18-21. P. 216-218.

88. Steinberg B., Alimova E., Baglij A., Morilev R., Nis Z., Petrenko V., Steinberg R. The System for Automated Program Testing. Proceedings of IEEE East-West Design & Test Symposium (EWDTS'09). Moscow, Russia, 2009. Sept. 18-21. P. 218 – 220.
89. [Электронный ресурс] http://ops.rsu.ru
90. [Электронный ресурс] http://parallel.ru
91. [Электронный ресурс] http://www.ImpulseC.ru
92. Свидетельство о государственной регистрации программы для ЭВМ № 2011617205 «Диалоговый высокоуровневый оптимизирующий распараллеливатель программ» Правообладатель: Федеральное государственное автономное образовательное учреждение высшего профессионального образования «Южный федеральный университет» / Штейнберг Б.Я., Штейнберг Р.Б., Морылев Р.И., Петренко В.Вл., Полуян С.В., Штейнберг О.Б., Баглий А.П., Нис З.Я., Скиба И.С., Юрушкин М.В., Шаповалов В.Н., Алымова Е.Вл., Кравченко Е.Н., Гуда С.А.; Заяв. №2011615302 от 15.07.2011; Опубл. 15.09.2011.
93. Свидетельство о государственной регистрации программы для ЭВМ № 2011617950 «Конвертер C2HDL с языка Си в язык описания электронных схем» Правообладатель: Федеральное государственное автономное образовательное учреждение высшего профессионального образования «Южный федеральный университет» / Дубров Д.Вл., Штейнберг Р.Б.; Заяв. №2011616150 от 12.08.2011; Опубл. 11.10.2011.