Введение

При изучении дисциплины «Автоматизированное проектирование распределенных СРВ» студентам приходится изучать системы реального времени (СРВ) как объекты проектирования. Наиболее сложным для изучения оказывается усвоение механизмов функционирования СРВ в условиях многопроцессорной архитектуры вычислительной системы и совместной работы совокупности задач с различными условиями запуска и взаимодействия.

Выполняя практические задания по проектированию СРВ, студенты должны получить ответы на многие вопросы, среди которых выделяются следующие:

- определение числа станций вычислительной системы;
- получение топологии размещения станций на объекте и закрепление за ними терминальных точек;
- планирование использования ресурсов вычислительной системы для выполнения прикладных функций;
- организация совместной параллельной работы совокупности вычислительных процессов;
- построение моделей и анализ функционирования варианта проектируемой СРВ в модельном представлении;
- оптимизация проектных решений и получение приемлемого варианта проектируемой СРВ.

Принятие решений по данным вопросам предполагает глубокое понимание алгоритмов функционирования компонентов и СРВ в целом, установление зависимостей между параметрами компонентов и характеристиками работы СРВ. Поэтому основное внимание при подготовке учебного пособия уделялось изложению методов разработки моделей функционирования СРВ, которые, по возможности, более полно раскрывают все многообразие отношений между параметрами и характеристиками. Учебное пособие построено так, что вначале рассматриваются более простые модели с укрупненным представлением объектов системы и отношений между ними. А далее, на основе полученных знаний, к изучению предлагаются более сложные модели и методы их построения. Так, для изучения взаимодействия параллельных процессов предварительно рассматриваются сети Керка. Предполагается также, что студенты, изучающие данную дисциплину, знакомы с сетями Петри.

Учебное пособие написано по результатам научной работы, выполняемой на кафедре информатики и проектирования систем Томского политехнического университета. Особо важное место среди этих ре-

зультатов занимает метод моделирования на основе активных моделей. Метод отражает один из принципов технологии модульного проектирования СРВ, которая также представлена в учебном пособии. Технология включает три уровня моделирования: уровень аналитических расчетов, функциональный уровень и уровень комплексного моделирования СРВ в динамике. Наиболее подробно излагаются второй и третий уровни моделирования.

Материал учебного пособия разбит на семь глав. Каждая глава включает несколько параграфов, материал которых структурирован на достаточно мелкие тематически поименованные порции. Такая детальная структуризация упрощает усвоение материала. Кроме того, каждая глава сопровождается перечнем вопросов, которые во многом соответствуют тематическим порциям.

В первой главе рассмотрены общие проблемы проектирования СРВ. Вторая глава посвящена аналитическим методам проектирования СРВ. Третья глава знакомит студентов с технологией модульного проектирования. Методы разработки модели распределенной системы изложены в четвертой главе. Общая модель проектируемой системы представлена совокупностью частных моделей: модель топологии системы; модель программной нагрузки СРВ; модель архитектуры МВС; модель алгоритма функционирования системы. Последующие три главы раскрывают методы анализа систем на основе трех уровней моделирования: аналитических расчетов (глава 5), функционального моделирования (глава 6), комплексного моделирования в динамике (глава 7).

В целом материал учебного пособия носит теоретический характер и полностью соответствует лекционному курсу. В ходе изложения материала в ряде случаев рассматриваются примеры и в качестве иллюстраций используются окна, формируемые программными средствами поддержки технологии проектирования СРВ. Описание данных программных средств вынесено в отдельную методическую литературу, которая наряду с данным учебным пособием используется при выполнении лабораторных и курсовых работ по проектированию СРВ.

1. СРВ И ПРОБЛЕМЫ ИХ АВТОМАТИЗИРОВАННОГО ПРОЕКТИРОВАНИЯ

1.1. Определение СРВ

Предварительно определим термин «реальное время». В самом широком смысле термин «реальное время» можно использовать применительно к деятельности человека или функционированию системы по обработке информации, когда требуется отвечать на поступающие извне входные сигналы, причем задержка ответа должна быть конечной и не превышающей заданного значения.

Среди таких систем можно выделить два крупных класса, критичных к реальному времени. Первый из них характеризуется тем, что реакция системы на команду пользователя или входной сигнал должна составлять определенное время, например, несколько секунд. Тем не менее, если ответ не поступил даже через 2 минуты, то систему нельзя считать непригодной. Пользователь вправе быть недовольным, но если система спроектирована правильно, то ожидаемый результат, в конечном счете, будет получен.

Однако такие условия работы системы не всегда допустимы. Для второго класса систем получение результата за время, большее требуемого, вполне может быть приравнено к неверному результату. В последующем основное внимание будет уделено именно таким системам. Для них характерным является то, что вычислительная система непосредственно связана с некоторым физическим объектом и должна обеспечивать управление и контроль над функционированием этого объекта.

Вычислительной системой реального времени (СРВ) будем называть систему, связанную с некоторым объектом и обрабатывающую поступающую в нее информацию о его состоянии настолько быстро, чтобы результат обработки мог использоваться для своевременного воздействия на протекание процессов в объекте.

Типичными примерами применения таких систем являются:

- автоматизированные системы управления технологическими процессами (АСУ ТП) производственных объектов;
- системы управления летательными аппаратами;
- навигационные системы;
- системы управления роботами и гибкими автоматизированными производствами;
- системы телекоммуникационного обслуживания.

Встроенные и окружающие системы

С появлением высокопроизводительной и надежной микропроцессорной техники широкое распространение получили способы создания СРВ в форме ***встроенной вычислительной системы***. Техническую систему, в которую встраивается вычислительная система, стали называть ***окружающей системой***. При этом встроенная система является неотъемлемой частью окружающей системы. Обе системы, и встроенная и окружающая, образуют единую техническую систему. Это означает, что окружающая система не может функционировать автономно. Такая тесная взаимосвязь между окружающей и встроенной системами достигается тем, что встроенная система уже на стадии выполнения проектных работ по созданию технической системы рассматривается как одна из ее компонентов. В любом случае разработка окружающей системы должна вестись с учетом функциональных возможностей встроенной системы, которая наблюдает за работой окружающей системы и воздействует на нее так, чтобы обеспечить эффективное функционирование технической системы в целом.

В процессе наблюдения и воздействия на окружающую систему фактор времени для встроенной системы играет первостепенную роль. Встроенная система должна своевременно реагировать на изменения, происходящие в окружающей системе. Если эти изменения протекают достаточно быстро, то допустимое время, отводимое встроенной системе для определения управляющих воздействий, может оказаться несопоставимо с производительностью используемых вычислительных средств. В то же время вполне естественным является стремление сократить время запаздывания при вычислении управляющих воздействий. Поэтому при проектировании встроенных систем ограничения на время определения управляющих воздействий, диктуемые окружающей системой, оказываются достаточно жесткими [1, 2].

Важной особенностью встроенной системы является то, что ее функционирование осуществляется не по заранее заданному сценарию, а подчинено событиям, происходящим в окружающей системе. Динамика событий в окружающей системе полностью переносится на встроенную систему и в конечном итоге определяет правила ее функционирования.

Схему взаимодействия встроенной и окружающей систем представим в виде рис. 1.1.

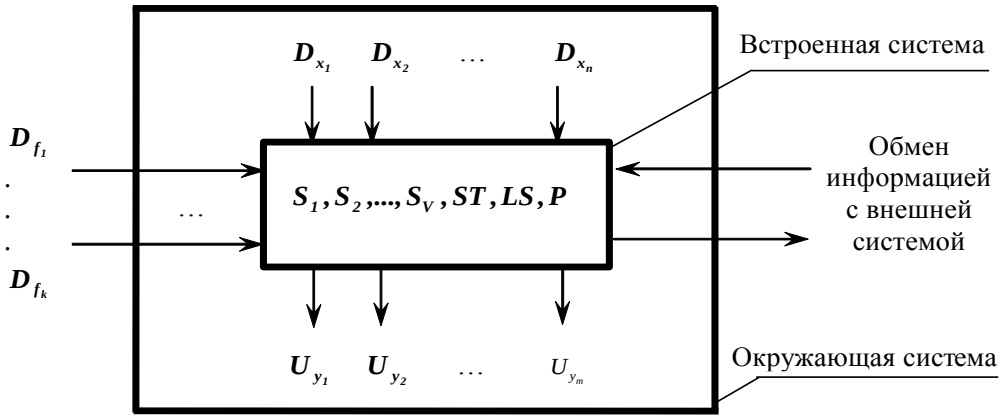


Рис. 1.1. Схема взаимодействия встроенной и окружающей систем

На рис. 1.1 встроенная система представлена совокупностью микропроцессорных станций S_1, S_2, …, S_V, системным таймером ST, локальной сетью LS, программным обеспечением P. Окружающая система представлена совокупностью датчиков и исполнительных механизмов. Взаимодействие окружающей и встроенной систем осуществляется через совокупность информационных входов с датчиков и от оператора D_{x_1}, D_{x_2}, …, D_{x_n} и совокупность информационных выходов для воздействий на исполнительные механизмы и сообщений оператору U_{y_1}, U_{y_2}, …,U_{y_k}.В отдельную группу выделены входные данные D_{f_1}, D_{f_2}, **...**, D_{f_k}, которые обозначают нерегулируемые и неизменяемые параметры внешней среды, сырья, оборудования. На рисунке также показаны входные и выходные данные для обмена с внешними системами.

1.2. Взаимосвязь между окружающей и встроенной системами

Рассмотрим основные принципы организации взаимосвязи между окружающей системой и встроенной. Состояние окружающей системы в момент времени t представляет вектор входных данных $X_t = \{x_1, x_2, ..., x_n\}_t$. В результате работы встроенной системы с вектором X_t получается вектор управляющих воздействий $Y_t = \{y_1, y_2, ..., y_m\}_t$. Таким образом, взаимосвязь встроенной и окружающей систем осуществляется через оператор F, реализующий основную функцию встроенной системы $Y_t = F(X_t)$. Различные способы ор-

ганизации выполнения оператора $\boldsymbol{F}$ порождают соответствующие схемы реализации взаимосвязи встроенной и окружающей систем.

Наиболее простой является схема, в соответствии с которой время разбивается на равные интервалы $\Delta\boldsymbol{t}$ тактирующими сигналами таймера. Выполнение оператора $\boldsymbol{F}$ повторяется в каждом интервале и по времени не должно превышать величину $\Delta\boldsymbol{t}$. Такая циклическая организация связи получила название ***синхронной***.

Достоинством синхронного принципа связи является его универсальность, позволяющая реализовать любые сценарии наступления событий в окружающей системе. Основной недостаток синхронной связи заключается в необходимости выполнять в каждом очередном цикле $\boldsymbol{t}+1$ оператор $\boldsymbol{F}$ и в том случае, если состояние встроенной системы не изменилось, то есть вектор $\boldsymbol{X_t} = \boldsymbol{X_{t+1}}$. Это приводит к неоправданно большим объемам вычислений. Поэтому в реальных системах компоненты вектора входных данных $\boldsymbol{X_t}$ разбиваются на группы с одинаковыми скоростями изменений значений и соответственно равными требованиями к цикличности обработки. Для каждой такой группы устанавливается своя величина интервала $\Delta\boldsymbol{t}$. В этом случае принцип синхронной связи сохраняется, но декомпозирован на отдельные группы входных данных.

Во многих случаях требуется более тесное взаимодействие окружающей и встроенной систем, при котором встроенная система должна практически мгновенно реагировать на сигнал прерывания. Компоненты вектора $\boldsymbol{X_t}$, соответствующие таким сигналам, описывают сообщения окружающей системы об аварийных ситуациях, особых состояниях технологического регламента, фиксации моментов времени на выполнение переключений, наступления событий, связанных с выполнением определенных условий. Помимо перечисленных существует и ряд других причин, по которым приходится использовать нециклическую схему связи, получившую название ***асинхронной***. При асинхронной схеме связи сигналам прерывания, как правило, назначаются приоритеты.

В реальных технических системах практически всегда возникает потребность в одновременном использовании всех видов схем связи, что существенно усложняет алгоритм реализации оператора $\boldsymbol{F}$, но вместе с тем создает более благоприятные возможности для удовлетворения самых жестких требований по условиям реального времени.

1.3. Свойства распределенных систем реального времени как объектов проектирования

При построении высокопроизводительных СРВ на базе микропроцессорных станций приходится объединять в систему большое число станций. Эти станции в ***многопроцессорной вычислительной системе*** (МВС) работают асинхронно и могут по собственной инициативе использовать ресурсы системы, что предъявляет высокие требования к организации их взаимодействия.

Наличие большого числа активных средств обработки данных, способных вмешиваться в работу системы и ожидающих быстрой реакции системы, требует очень тщательной проработки организации управления взаимодействием компонентов системы. Требуется разработка новых методов управления, рассчитанных на его ***децентрализацию***. Управление большим числом процессов и объектами в реальном времени, при котором требуется постоянное взаимодействие МВС и внешних устройств, представляющих окружающую систему, также эффективно реализуется в ***распределенных СРВ***.

Таким образом, на основе микропроцессорных станций могут создаваться весьма сложные и малоисследованные системы. При этом наиболее сложной является задача организации взаимодействия компонентов системы при выполнении функций оператора F в реальном времени.

Функциональная и топологическая децентрализация

Принципы построения распределенных СРВ основываются на ***функциональной*** и ***топологической*** децентрализации этих систем. Функциональная децентрализация выполняется путем разбиения оператора F на ряд более мелких операторов, которые соответствуют отдельным прикладным функциям или технологическим процессам. Критерием разбиения является минимум связей между отдельными частями, что в конечном итоге соответствует минимуму объема данных, передаваемых между вычислительными процессами, реализующими эти части.

Топологическая децентрализация предполагает пространственное распределение компонентов окружающей и встроенной систем. При этом обычно станции МВС размещаются по возможности ближе к датчикам и исполнительным устройствам. Критерием размещения является минимум суммарной стоимости кабельной продукции, необходимой для реализации схемы связей.

Как правило, оптимальные варианты функциональной и топологической децентрализации не совпадают. Поэтому приходится искать некоторый компромиссный вариант.

С экономической точки зрения вариант оптимальной топологической децентрализации часто оказывается предпочтительнее, особенно для территориально распределенных объектов окружающей системы.

Параллельность

Одним из наиболее важных свойств, которым должна обладать СРВ, является возможность ***параллельного*** выполнения функций оператора $\boldsymbol{F}$. Это обусловлено тем, что процессы, протекающие в окружающей системе, во многом развиваются параллельно. Кроме того, жесткие ограничения на время реакции встроенной системы на события в окружающей системе предопределяют стремление к параллельному выполнению алгоритмов прикладных функций оператора $\boldsymbol{F}$. ***Распараллеливание*** является одним из основных способов сокращения времени выполнения алгоритмов и широко используется при проектировании программного обеспечения СРВ.

Таким образом, функционирование СРВ можно представить в виде совокупности развивающихся параллельных вычислительных процессов. Поэтому при проектировании СРВ организация совместной работы параллельных процессов становится одной из наиболее важных проблем. Следует также отметить, что и методы построения адекватных моделей взаимодействия параллельных процессов, и механизмы их реализации развиты слабо. Это создает дополнительные трудности при проектировании программного обеспечения СРВ.

Динамизм

При определении СРВ и изображенной на рис. 1.1 схемы взаимодействия окружающей и встроенной систем неявно предполагалось, что конфигурация окружающей системы стабильна во времени. Но для многих систем в большей или меньшей мере присуща нестабильность как по составу компонентов структуры, так и значений их параметров. Поэтому динамизм окружающей системы проявляется не только через изменения значений компонентов вектора $\boldsymbol{X}_t$, но и через изменения состава компонентов данного вектора. В первом случае будем считать, что имеет место ***динамизм первого рода***, а во втором – ***динамизм второго рода.***

Наличие в окружающей системе динамизма второго рода предполагает использование моделей конфигурации системы, представляемых в явном виде. Состав компонентов окружающей системы в момент времени t представим вектором $R_t = \{r_i\}$. Компонента r_i описывает состояние соответствующего i-го устройства и равна 1, если i-е устройство включено в систему, и равна 0 в противном случае.

Таким образом, информацию об изменениях конфигурации окружающей системы можно описать совокупностью переменных $\{r_i\}$. Каждой переменной r_i соответствует подмножество X_t^i входных данных вектора X_t, $\bigcup_i X_t^i = X_t'$, X_t' – клеточный вектор входных данных. Клетки X_t^i включаются или исключаются из вектора X_t' в соответствии с изменениями в конфигурации окружающей системы.

Согласно принятым обозначениям ***трансформация модели*** окружающей системы при динамизме второго рода сводится к отображению значений переменных R_t и X_t' на модель в текущий момент времени t. После трансформации модель используется при выполнении прикладных функций оператора F по определению вектора $Y_t = F(R_t \times X_t')$.

Свойства СРВ

Для современных встроенных систем и систем других форм реализации, относящихся к классу систем реального времени, характерными являются следующие свойства:

- ***распределенность***, которая предполагает топологическую или (и) функциональную децентрализацию при создании СРВ;
- ***параллельность***, которая предполагает функционирование окружающей системы и многопроцессорной вычислительной системы СРВ как совокупности развивающихся параллельных взаимодействующих процессов;
- ***асинхронность***, отражающая независимость запуска процессов и возможные варианты селекции во времени состояний процессов при их взаимодействии;
- ***высокий динамизм***, который приводит к жестким ограничениям на время реакции встроенной системы на события в окру-

жающей системе и, соответственно, к высоким требованиям по производительности МВС.

Проектирование СРВ, функционирующих в указанных условиях, связано с большими трудностями. Это объясняется необходимостью установления соответствия между вычислительными ресурсами МВС и приемлемыми временами реакции программного обеспечения встроенной системы на события в окружающей системе. Для достижения такого соответствия предлагается последовательно (***эволюционным путем***) приближать исходный вариант описания прикладных функций оператора $\boldsymbol{F}$ и описания начального варианта архитектуры МВС к некоторому приемлемому компромиссному варианту СРВ.

1.4. Проблемы автоматизированного проектирования СРВ

Проблемы эволюции моделей

При проектировании современных СРВ все большую актуальность приобретают методы анализа качества их функционирования в модельном представлении. Такой анализ приходится выполнять на моделях, которые зачастую неудовлетворительно описывают СРВ, как объект проектирования. Это обусловлено тем, что на начальной стадии проектирования даже в предположении о наличии хорошего и полного описания прикладных функций, которые должна выполнять СРВ, и алгоритмов их реализации, существует большая степень неопределенности относительно архитектуры МВС, алгоритмов ее функционирования и алгоритмов управления совместной работой большого числа вычислительных процессов, реализующих прикладные функции оператора $\boldsymbol{F}$.

В этих условиях для обоснования принимаемых проектных решений неизбежным и наиболее эффективным инструментом анализа является моделирование. Затраты на разработку моделирующих программных средств оказываются соизмеримыми с затратами на проектирование программного обеспечения СРВ. Это побуждает разработчиков отказываться от этапа моделирования и вести проектирование, полагаясь на накопленный опыт и интуицию.

Это положение усугубляется тем, что при значительной неопределенности относительно приемлемого варианта СРВ, требуется строить не одну, а совокупность последовательно улучшаемых моделей. То есть процесс проектирования необходимо вести путем последовательной трансформации (эволюции) исходного варианта модели СРВ до получения модели приемлемого варианта. Из этого следует, что при ис-

пользовании традиционных подходов к моделированию трансформации приходится подвергать сложные программные комплексы, реализующие алгоритмы, моделирующие функционирование СРВ. Такой подход вступает в противоречие с возможностями существующих систем программирования по трансформации программ. Поэтому в основу построения моделей СРВ должны быть положены другие принципы, обеспечивающие возможность применения эволюционного подхода к проектированию.

Процесс последовательной трансформации моделей СРВ при эволюционном подходе к проектированию можно представить схемой на рис. 1.2.

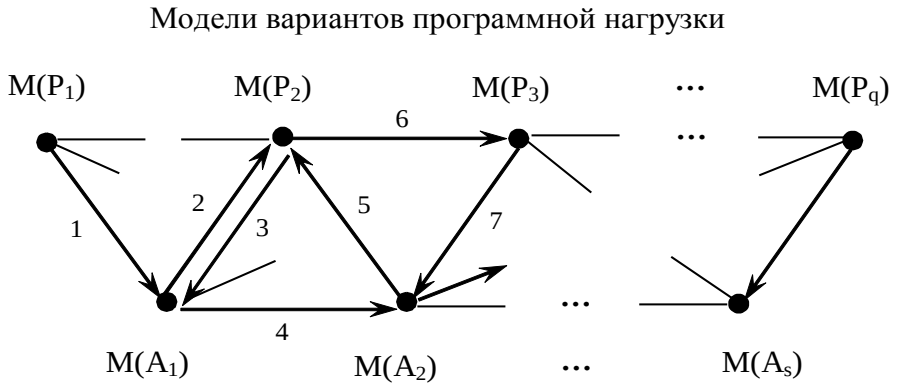


Рис. 1.2. Пример маршрута эволюции моделей

Выделенный на рисунке маршрут {$M(P_1)$, $M(A_1)$, $M(P_2)$, $M(A_1)$, $M(A_2)$, $M(P_2)$, $M(P_3)$, $M(A_2)$…….} отражает один из возможных процессов эволюции модели программной нагрузки и архитектуры МВС. Вид маршрута определяется решениями по трансформации модели, которые принимаются пользователем на основе результатов моделирования.

Проблемы построения конструктивных моделей

В общем случае предполагается, что исходные варианты описаний программ, реализующих прикладные функции СРВ и архитектуры МВС разработаны независимо друг от друга. При этом особо важным является то, что программы описаны без учета конкретных условий реального времени и без ориентации на конкретную архитектуру МВС.

Возможные варианты архитектур МВС могут быть представлены библиотекой базовых архитектур и включать приемлемые типы процессоров, возможные схемы связи их между собой и другие сведения о технических средствах МВС и способах их взаимодействия. Эволюция

архитектуры МВС проектируемой СРВ может быть сведена к последовательному выбору из библиотеки более предпочтительных базовых архитектур и уточнению для них значений характеристик ресурсов и правил взаимодействия отдельных компонентов. Оценка предпочтительности осуществляется на основе анализа характеристик алгоритмов, реализующих прикладные функции СРВ на выбранном варианте МВС.

При таком подходе к проектированию СРВ важное место занимает проблема анализа характеристик данных алгоритмов или, как принято называть, ***программной нагрузки*** на МВС. Задача рассматривается для общего случая, когда программная нагрузка представляет собой автономную программу или информационно связанный комплекс программ. Программы могут быть представлены на различных уровнях детализации. Для программной нагрузки заданы также ограничения реального времени, циклы съема входных данных и обновления выходных, правила инициирования программ и другие ограничения. Требуется для очередной (предпочтительной) архитектуры МВС и заданной программной нагрузки получить программный продукт, удовлетворяющий условиям реального времени.

Среди известных подходов к моделированию в области проектирования управляющих систем для оценки характеристик производительности наиболее предпочтительными являются сетевые модели, например, сети массового обслуживания или сети очередей [3, 5]. Для моделирования параллельных взаимодействующих процессов широко используются сети Петри [6, 7] и многочисленные их модификации [8–10], в частности, применительно к управляющим системам в гибких автоматизированных производствах [11, 12]. Подходам, ориентированным на использование аналитических методов при проектировании вычислительных систем для АСУ ТП, посвящена работа [13].

Общим недостатком существующих подходов, помимо низкой адекватности, является наличие непреодолимого разрыва между этапом моделирования и этапом разработки программного обеспечения СРВ. Для процесса эволюции моделей это разрыв между этапом анализа и этапом синтеза. Проблема заключается в том, что в этих подходах модельное представление программной нагрузки и архитектуры МВС не является конструктивным для решения задач синтеза. Другими словами, модельное представление программной нагрузки не может быть трансформировано в программное обеспечение проектируемой СРВ. На таких моделях трудно определять и выполнять трансформации, необходимые для улучшения характеристик исследуемых вариантов систем. Здесь важно отметить, что варианты таких моделей, полученных на заключительной стадии эволюции и имеющих приемлемые характеристи-

ки, не могут быть непосредственно использованы на последующих этапах проектирования.

1.5. Построение СРВ на базе типовых станций

В настоящее время производится ряд наборов устройств, из которых можно строить СРВ распределенной структуры. Основная конструктивная единица такой аппаратуры – микропроцессорная станция (контроллер), которую можно установить в том или ином месте окружающей системы или на операторском пункте и подключить к локальной сети. Станция обычно содержит один или несколько микропроцессоров и имеет определенное функциональное назначение. По этому последнему признаку станции делятся на следующие классы [13]:

- ***локальные технологические***, непосредственно связанные с датчиками и исполнительными механизмами окружающей системы;
- ***операторские***, оснащенные средствами вывода информации оператору и средствами восприятия от него управляющих воздействий;
- ***координирующие***, не имеющие непосредственной связи с окружающей системой и оператором и служащие для определения управляющих воздействий;
- ***вспомогательные***, для управления передачей данных, обменом с другой системой, выполнения функций диагностики устройств МВС и т.п.

Прикладные функции, которые должна выполнять СРВ, определяются по результатам обследования объекта управления и указываются в техническом задании на проектирование. Каждая прикладная функция связана с получением информационных выходов: управляющих воздействий, сообщений оператору, информации для внешней системы. Прикладная функция включает операции ввода, обработки и вывода данных. Примерами таких операций являются следующие [13]:

- измерение технологической переменной;
- вычисление среднего значения и интеграла технологической переменной;
- вычисление комплексных показателей технологического процесса;
- вычисление управляющего воздействия;

- ввод дискретных событий и состояний объекта управления;
- определение событий в технологическом процессе;
- вывод информации на экран;
- вывод информации на печать;
- вывод информации на индикацию;
- вывод информации на сигнализацию;
- аналоговый выход на исполнительный механизм;
- дискретный выход на исполнительный механизм;
- цифровой выход в другую систему.

На основе анализа совокупности прикладных функций принимается решение об архитектуре МВС. При этом необходимо ответить на следующие вопросы:

- каково число станций и как их разместить на объекте управления;
- как распределить между станциями массивы данных, операции обработки, программы, реализующие эти операции;
- каковы маршруты прокладки каналов связи между станциями.

Должны быть решены также ряд частных проблем по определению характеристик:

- тип сети передачи данных (радиальная или парная связь между станциями, магистраль, несколько магистралей нижнего уровня, связанных с магистралью верхнего уровня);
- пропускные способности каналов связи;
- метод доступа к сети;
- быстродействие процессора;
- емкость запоминающих устройств станций;
- план размещения на объекте точек ввода и вывода, устройств, операторских пультов.

Приведенные проблемы порождают соответствующие проектные процедуры. В процессе проектирования последовательность выполнения процедур жестко не фиксируется.

При поиске решений используется три уровня показателей функционирования СРВ. К первому уровню относятся общесистемные эко-

номические показатели. Второй уровень соответствует общесистемным частным показателям эффективности СРВ: показатели времени, показатели надежности выполнения прикладных функций, МВС и СРВ в целом. На третьем уровне выделяются показатели функционирования МВС: характеристики загрузки сети, процессоров, запоминающих устройств, суммарная стоимость аппаратуры МВС и т.п.

Совокупность прикладных функций, то есть функциональную структуру программной нагрузки СРВ, можно представить в виде ориентированного графа, в котором вершины соответствуют операциям, программам их выполнения и массивам данных, а дуги – информационным связям между ними.

Массивам или операциям, связанным с внешними источниками или потребителями информации, соответствуют терминальные вершины функционального графа. Каждой прикладной функции соответствует на графе подграф, содержащий в общем случае несколько входных и одну выходную терминальную вершины. Если при выполнении операции вычисляется несколько выходных переменных, имеющих самостоятельный прикладной смысл, то с каждой такой переменной связывается отдельная функция и выполнение одной операции обеспечивает реализацию нескольких прикладных функцией СРВ.

Множество прикладных функций можно разделить на два в общем случае пересекающихся подмножества. К первому относятся периодические функции, выполняемые с известной частотой в рамках регламентированного цикла управления. Второе подмножество составляют непериодические функции, вызываемые по случайным запросам в критических или аварийных ситуациях. Функции, принадлежащие к пересечению указанных подмножеств, в зависимости от ситуации могут выполняться как периодически, так и по случайным запросам.

Пример построения СРВ производства карбамида

Производство карбамида [13] состоит из 2-х линий, в каждой из которых выполняются следующие технологические процессы:

- подготовки технического азота;
- получения пара;
- химической подготовки воды;
- компрессии;
- синтеза;
- концентрирования;

- гранулирования;
- очистки стоков.

Общий алгоритм функционирования СРВ включает 5 алгоритмов (рис. 1.3):

АИ – ввода и обработки аналоговой информации;

ДИ – ввода дискретной информации;

УВ – вычисления и вывода управляющих воздействий;

ОО – обращений оператора;

ДС – диспетчер системы.

Первые три алгоритма выполняются циклически, но с разным периодом: АИ и ДИ – 10с; УВ – 4 мин. Алгоритмы АИ, ДИ, УВ могут прерываться алгоритмом ОО для запроса некоторой видеограммы или изменения уставок, параметров регулирования.

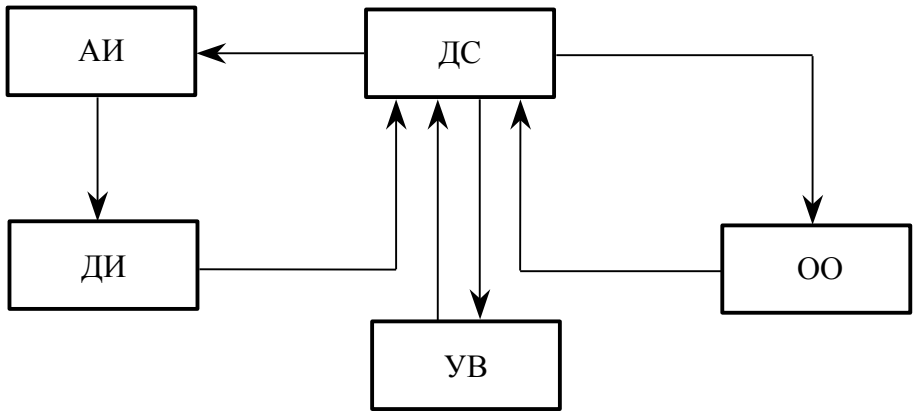


Рис.1.3. Блочная схема алгоритма функционирования СРВ производства карбамида

Прерывания выполняются алгоритмом ДС. Алгоритм АИ вводит результаты аналого-цифрового преобразования, масштабирует, линеаризует, фильтрует и проверяет достоверность. Результатом является запись в массив аналоговой информации ***А*** обработанных значений технологических переменных.

Алгоритм ДИ переносит дискретные данные из устройства ввода в массив ***Д*** базы данных.

Алгоритм УВ содержит семь алгоритмов, вычисляющих уставки, и алгоритм ПИД – регулирования. Все исходные данные для алгоритма содержатся в массиве ***А***. Уставки и управляющие воздействия записы-

ваются в массивы ***У*** и ***В***. Управляющие воздействия выводятся также на исполнительные механизмы.

Алгоритм ОО инициируется оператором с монитора. Исходные данные при вызове видеограммы содержатся в массиве формы кадров ***ФК*** и массивах ***А***, ***Д***, ***У***, ***В***. При ручном изменении уставки или настройки регулирования исходные данные содержатся также в регистре клавиатуры ***К***, откуда они выводятся не только на дисплей, но также вводятся в массивы уставок ***У*** и параметров ***Н***.

В качестве основы для выделения операции при построении функционального графа служит алгоритмический модуль. В общем случае одному алгоритмическому модулю может соответствовать несколько операций, и наоборот. Рассмотрим этот вопрос более подробно.

Представлять один модуль несколькими операциями можно, если он используется многократно для работы с различными исходными данными, например, для обработки ряда входных сигналов от разных технологических переменных. Но вводить при этом несколько операций нужно не всегда. Если исходные данные для модуля берутся из одного массива, а результаты его работы также помещаются в один массив, то нет необходимости вводить несколько операций. В этом случае модуль может быть представлен одной операцией, объем вычислений по которой составляется из общего объема повторных вычислений по данному модулю.

Иначе обстоит дело, если предположить, что вычислительные операции по данному модулю будут выполняться в разных станциях МВС. Тогда выделение операций обработки данных для включения в качестве вершин в функциональный граф зависит от того, как предварительно распределяются терминальные точки между станциями локальной сети.

В рассматриваемом примере принято решение распределить терминальные точки по 5 станциям: 4 локальных технологических, по 2 на каждую линию производства и одна операторская станция, общая на обе линии. На каждой линии одна станция связывается с 41 аналоговыми и 51 дискретными входами и с 5 аналоговыми выходами, другая станция – с 52 аналоговыми и 51 дискретными входами и 4 аналоговыми выходами.

Алгоритмический модуль ОО выполняется в операторской станции. Диспетчер ДС должен функционировать в каждой локальной технологической станции. Данный модуль служит только для системной организации вычислений и не решает прикладных задач. Поэтому он в функциональном графе не отражается.

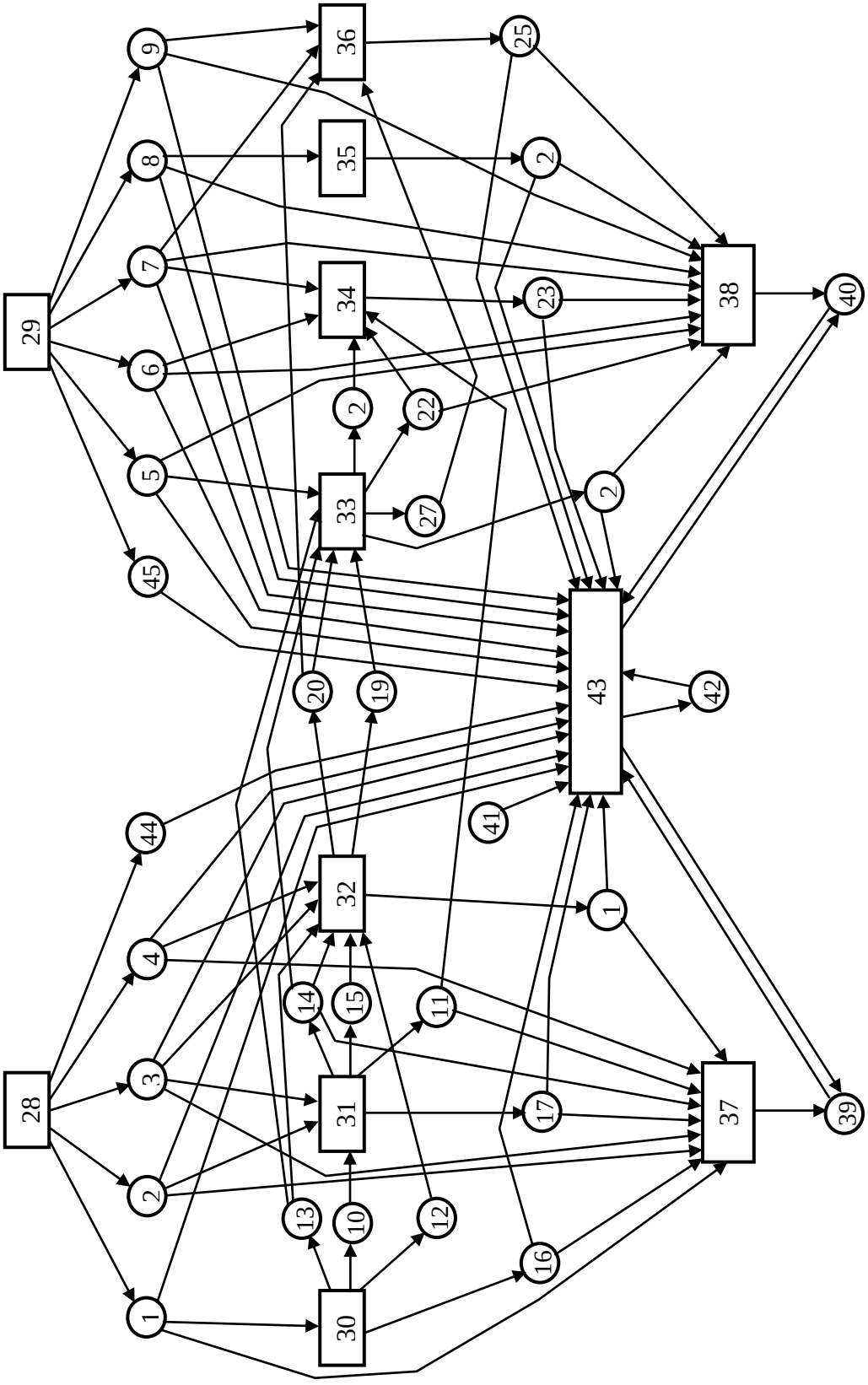


Рис. 1.4. Функциональный граф

В рассматриваемой СРВ модули АИ и ДИ, выполняемые всегда последовательно, могут быть объединены в одну операцию – ввод информации (ВИ). В функциональном графе следует предусмотреть по 4 операции ввода ВИ1-ВИ4 и ПИД-регулирования ПИ1-ПИ4 – по одной операции на каждую станцию.

При выделении массивов нужно руководствоваться тем, что если несколько массивов формируются одной операцией и используются затем одними и теми же операциями, то их следует объединить в один массив. Управляющие воздействия можно объединить в один массив ***В***, так как все они связаны только с операциями ПИ и ОО. По этому же принципу объединяются в массивы обработанные операциями ВИ аналоговые и дискретные данные. Как правило, данные с пренебрежимо малой частотой обращений к ним в функциональный граф не включаются. По этой причине в граф не включены исходные для операции ОО массивы ***К*** и ***Н***.

Массив ***ФК***, содержащий формы кадров, хранит практически постоянную информацию и может рассматриваться как источник информации. Массив ***ВГ*** хранит полную видеограмму, вызванную оператором и построенную операцией ОО.

Вершинам функционального графа ставятся в соответствие характеристики операций, массивов и программ, которые могут понадобиться при решении задач проектирования. К этим характеристикам относятся:

- длина массива;
- объем необходимых для выполнения операции вычислений, выраженный, например, средним числом элементарных машинных команд;
- частота выполнения операции в нормальном режиме функционирования;
- объем рабочей памяти, необходимый для выполнения операции;
- длина программы.

В функциональном графе, изображенном на рис. 1.4, для обозначения операций и массивов использована сплошная нумерация. В табл. 1.1 приведены содержание и доступные к началу построения графа характеристики операций, а в табл. 1.2 – содержание и характеристики массивов данных.

Таблица 1.1

Номер операции	Содержание операции	Частота исполнения, $мин^{-1}$	Требуемая емкость ОЗУ $x10^2$ байт
28	ВИ1 – ввод и обработка аналоговой и дискретной информации	6	5
29	ВИ2 – ввод и обработка аналоговой информации	6	5
30	УВ1 – управление концентрацией кислорода в потоке двуокиси углерода	0.25	8
31	УВ2 – управление степенью конверсии углерода	0.25	10
32	УВ3 – управление степенью разложения карбамата	0.25	8
33	УВ4 – управление степенью разложения карбамата аммония в колонне дистилляции среднего давления	0.25	7
34	УВ5 – управление составом раствора угле-аммонийных солей на выходе из промывной колонны	0.25	12
35	УВ6 – управление отношением расходов аммиака в аммиачный теплообменник и двуокиси углерода	0.25	8
36	УВ7 – управление составом паровой фазы на выходе из десорбера	0.25	8
37	ПИ1 – ПИД - регулирование	0.25	10
38	ПИ2 – ПИД - регулирование	0.25	10
43	ОО – обращение оператора	0.005	20

Таблица 1.2

Номер массива	Длина массива	Содержание данных
1 2 3 4 44	9 24 3 39 201	Обработанные значения аналоговых и дискретных технологических переменных – результаты операции ВИ1
5 6 7 8 9 45	30 21 9 15 30 204	Обработанные значения аналоговых и дискретных технологических переменных – результаты операции ВИ2

Продолжение табл. 1.2

Номер массива	Длина массива	Содержание данных
10	21	Промежуточные данные – результаты операций УВ1-УВ7
11	36	
12	15	
13	15	
14	15	
15	15	
16	3	
17	9	
18	3	
19	51	
20	39	
21	60	
22	21	
23	3	
24	3	
25	3	
26	3	
27	51	
39	15	Управляющие воздействия – результат операции ПИ1
40	12	Управляющие воздействия – результат операции ПИ2
41	14000	Формы кадров
42	2000	Видеограмма

При оценке характеристик объема вычислений, как правило, указывают две величины: одну оценку – среднюю для нормального режима, а вторую – максимальную для аварийного режима работы объекта. Аналогично задают частоту выполнения операций. Для нормального режима дают оценку средней частоты, для аварийного же указывают значительно большее значение.

Длину программ и объем рабочей памяти в условиях, когда программы еще не написаны, можно получить только экспертным путем, с использованием опыта, полученного ранее при программировании аналогичных задач.

Структура МВС для рассматриваемого примера представлена на рис. 1.5.

Заметим также, что при описании функционального графа (рис. 1.4) и структуры МВС (рис. 1.5) показана работа операторской станции (ОС) и станций С1, С2 первой линии, работа второй линии на станциях С3, С4 аналогична.

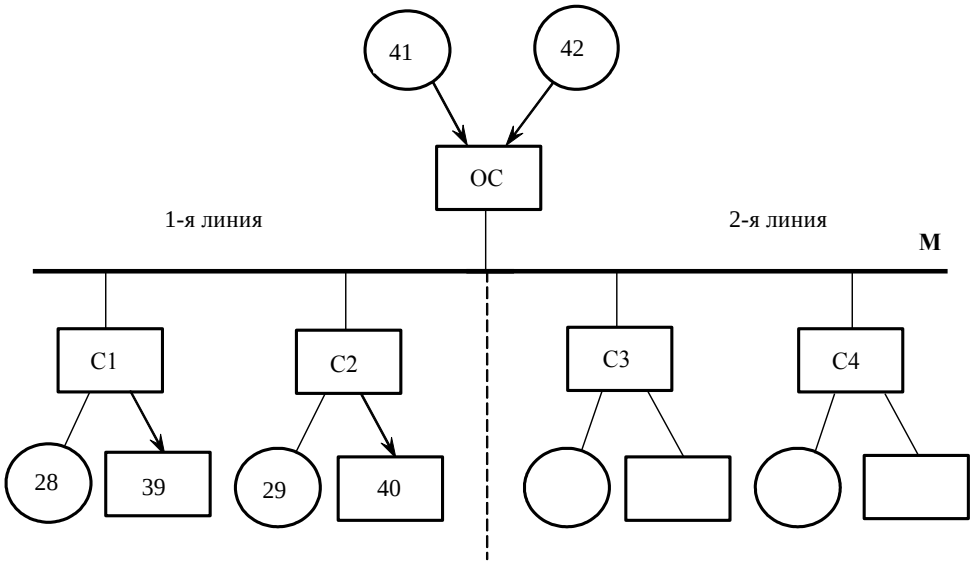


Рис. 1.5. Структура МВС

Вопросы для контроля усвоения знаний

1. Дать определение встроенных и окружающих систем, их роли в общей технической системе и условий функционирования.
2. Пояснить основные принципы организации взаимосвязи встроенной и окружающей систем.
3. Сопоставить функциональную и топологическую децентрализацию при построении СРВ.
4. Раскрыть свойство параллельности в функционировании СРВ и его использование при проектировании.
5. Пояснить разницу между динамизмом первого рода и динамизмом второго рода.
6. Перечислить основные свойства СРВ как объекта проектирования и дать их характеристику.
7. Обосновать необходимость эволюционного подхода при проектировании СРВ. Пояснить понятие маршрута эволюции модели.
8. В чем проявляется конструктивность модели? Являются ли конструктивными модели в форме сетей массового обслуживания, сетей Петри, Марковских цепей, сетей очередей?

9. Дать определение типовых станций и прикладных функций, которые должна выполнять СРВ. Привести примеры прикладных функций. Периодические и непериодические прикладные функции.
10. Какие задачи следует решить при проектировании СРВ на базе типовых станций?
11. Пояснить три уровня показателей функционирования СРВ, используемых при поиске решений задач проектирования.
12. Дать определение функционального графа, используемого для описания прикладных функций.
13. Изложить процесс построения СРВ на базе типовых станций на примере СРВ производства карбамида.

2. АНАЛИТИЧЕСКИЕ МЕТОДЫ И МОДЕЛИ ПРИ ПРОЕКТИРОВАНИИ СРВ

2.1. Общая характеристика задач проектирования СРВ

При проектировании СРВ рассмотрению подлежат три основных компонента системы:

- ***архитектура МВС***, определяемая составом станций и структурой связей между ними;
- ***программная нагрузка на МВС***, определяемая фиксированным набором прикладных функций и правил их инициирования;
- ***алгоритм функционирования СРВ***, включающий алгоритм управления совместной работой совокупности вычислительных процессов, реализующих программную нагрузку.

Исследования могут проводиться на системном уровне, когда изучаются общесистемные свойства и характеристики, или касаться отдельных компонентов системы: функционирование процессора, внешнего запоминающего устройства и канала ввода-вывода, обмен данными между станциями МВС, взаимодействие процессов. При этом свойства компонентов системы изучаются относительно принятых критериев качества с учетом взаимодействия компонентов с другими частями системы.

Задачи проектирования СРВ делят на ***задачи анализа*** и ***задачи синтеза***. Под анализом понимается определение функционирования по заданному описанию системы. Синтез – это построение описания системы по заданному функционированию [2].

Задачи анализа

Анализ может производиться путем измерения характеристик как непосредственно на объекте, так и на его модельном представлении.

Модель – это физическая или абстрактная система, адекватно представляющая объект исследования. В теории СРВ используются преимущественно абстрактные модели – описания объекта исследования на некотором языке. Абстрактность модели проявляется в том, что компонентами модели являются не физические элементы, а понятия, в качестве которых наиболее широко используются математические. ***Абстрактная модель, представленная на языке математических отношений, называется математической моделью.***

Математическая модель имеет форму функциональной зависимости:

$$\boldsymbol{Y = F(X),}$$

где $\boldsymbol{Y = \{y_1, y_2, ..., y_m\}}$ и $\boldsymbol{X = \{x_1, x_2, ..., x_n\}}$ – соответственно характеристики и параметры моделируемой системы, а $\boldsymbol{F}$ – функция, воспроизводимая моделью. Построение модели сводится к выявлению функции $\boldsymbol{F}$ и представлению ее в форме, пригодной для вычисления значений $\boldsymbol{Y = F(X)}$. Модель позволяет оценивать характеристики $\boldsymbol{Y}$ для заданных параметров $\boldsymbol{X}$ и выбирать значения параметров, обеспечивающие требуемые характеристики с использованием процедур оптимизации.

Модель может воспроизводить полную совокупность свойств системы либо их подмножество. Состав свойств устанавливается в зависимости от цели исследований и возможностей применяемого типа моделей воспроизводить требуемые свойства. При выборе математического аппарата моделирования и построении модели учитываются следующие факторы:

- состав воспроизводимых характеристик $\boldsymbol{Y = \{y_i\}, \quad i = 1, 2, ..., m}$;
- состав параметров $\boldsymbol{X = \{x_j\}, \quad j = 1, 2, ..., n}$, изменение которых влияет на характеристики $\boldsymbol{Y}$;
- область изменения параметров $\boldsymbol{x_j \in x_j^*, \quad j = 1, 2, ..., n, \quad x^* = \{x_j^*\}}$ – область определения модели;
- точность – предельно допустимая погрешность оценки характеристик $\boldsymbol{Y}$ на основе модели.

Состав характеристик $\boldsymbol{Y}$ определяется в зависимости от исследуемых свойств системы и должен гарантировать полноту отображения этих свойств. Состав параметров $\boldsymbol{X}$ должен охватывать все существенные аспекты организации системы, изучение влияния которых на качество функционирования составляет цель исследования производимого с помощью модели.

Область определения модели характеризует диапазон исследуемых вариантов системы. Чем шире состав характеристик и параметров, а также область определения модели, тем универсальнее модель в отношении задач, которые можно решать с ее использованием.

Допустимые погрешности оценки характеристик и точность задания параметров определяют требования к точности модели.

Модель, удовлетворяющая вышеперечисленным требованиям по составу характеристик и параметров и точности воспроизведения характеристик по всей области определения, называется ***адекватной***.

Задача идентификации

В общем случае задача анализа формулируется следующим образом: исходя из целей исследования устанавливается набор характеристик $\boldsymbol{Y}=\{y_1, y_2, ..., y_m\}$ исследуемого объекта и точность $\Delta=\{\delta_1,\ \delta_2,\ ...,\ \delta_m\}$, с которой они должны быть определены. Требуется определить характеристики $\boldsymbol{Y}$ объекта с заданной точностью Δ.

При анализе существующих систем в процессе их эксплуатации оценка характеристик $\boldsymbol{Y}$ производится, как правило, измерением ряда параметров функционирования с последующей их обработкой. Для сокращения затрат на анализ измеряют по возможности меньшее число параметров $\boldsymbol{X}=\{x_1, x_2, ..., x_n\}$, а недостающий набор характеристик доопределяют с помощью зависимостей $y_i = f_i(\boldsymbol{X}),\ i=1,\ 2,\ ...,\ m$.

В ходе эксплуатации системы часто возникает необходимость в расширении и изменении выполняемых функций и повышении эффективности работы МВС. В этих случаях следует оценить возможный эффект, для чего необходима модель системы. ***Построение модели системы на основе априорных сведений об ее структуре, алгоритму функционирования и данных измерений называют идентификацией системы***. Порядок идентификации системы представлен на рис. 2.1.

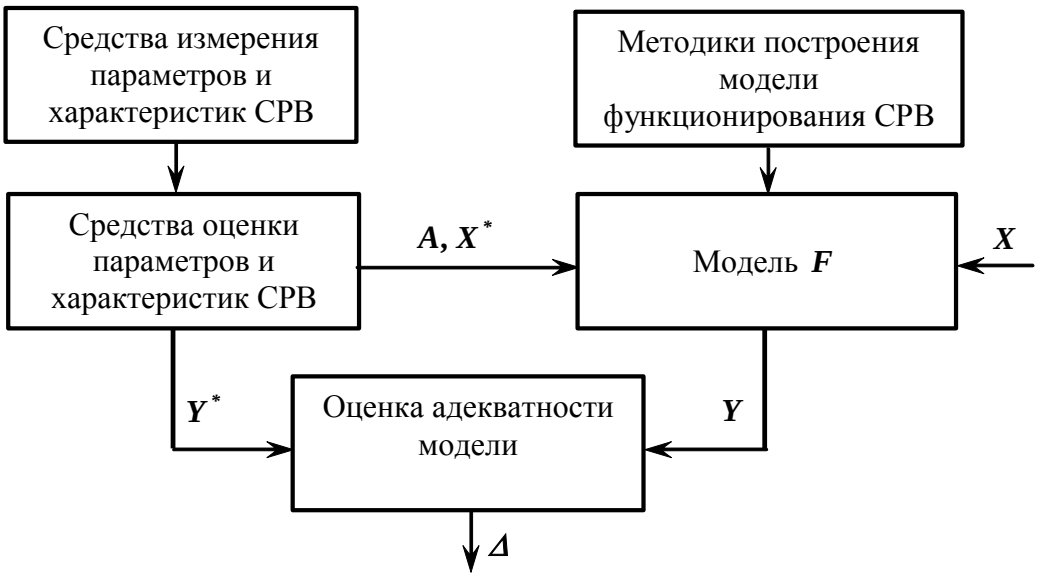


Рис. 2.1. Функциональная схема идентификации

На основе измеренных данных вычисляются параметры $\boldsymbol{X}^*$ и характеристики $\boldsymbol{Y}^*$ системы, а также параметры $\boldsymbol{A}$, которые полностью доопределяют модель $\boldsymbol{F}$. Значения $\boldsymbol{X}^*$ и $\boldsymbol{Y}^*$ используются для проверки адекватности модели, то есть оценки погрешности Δ воспроизведения моделью характеристик системы. Оценка производится путем сравнения значений характеристик $\boldsymbol{Y} = \boldsymbol{F}\left(\boldsymbol{X}^*\right)$, порождаемых моделью, с зарегистрированными характеристиками $\boldsymbol{Y}^*$, полученными по результатам измерений. Если модель адекватна системе, то она может использоваться для прогнозирования свойств системы, что сводится к вычислению на основе модели характеристик $\boldsymbol{Y} = \boldsymbol{F}\left(\boldsymbol{X}\right)$, соответствующих новым значениям $\boldsymbol{X}$ параметров системы.

Задача синтеза

Исходными данными для задачи являются:

- совокупность прикладных функций;
- ограничения на характеристики системы;
- критерий эффективности, устанавливающий способ оценки качества системы.

Необходимо выбрать оптимальную конфигурацию системы и алгоритм функционирования, удовлетворяющие заданным ограничениям.

Введем обозначения:

$\boldsymbol{Z}$ – вектор параметров, характеризующий фиксированный набор задач (прикладных функций) и правил их инициирования;

$\boldsymbol{G_i}$ – вектор параметров, определяющий $\boldsymbol{i}$-ю конфигурацию архитектуры системы;

$\boldsymbol{R_j}$ – вектор параметров, характеризующий $\boldsymbol{j}$-й вариант алгоритма функционирования системы.

В принятых обозначениях модель системы может быть записана в виде

$$\boldsymbol{Y} = \boldsymbol{F}(\boldsymbol{Z},\boldsymbol{G},\boldsymbol{R}), \quad \boldsymbol{G} = \left\{\boldsymbol{G_i}\right\}, \quad \boldsymbol{R} = \left\{\boldsymbol{R_j}\right\}.$$

Ограничения на характеристики и параметры $\boldsymbol{p_k}, \ldots, \boldsymbol{p_s} \in \left(\boldsymbol{Y} \cup \boldsymbol{X}\right)$ представим в виде

$$\boldsymbol{p_k} \in \boldsymbol{p_k^*}, \ \ldots, \boldsymbol{p_s} \in \boldsymbol{p_s^*},$$

где $p_k^*, ..., p_s^*$ – области допустимых значений соответствующих характеристик и параметров.

Критерий эффективности $E = \varphi(Y)$ зависит от полученных значений характеристик. Тогда задача синтеза заключается в определении конфигурации G_i и алгоритма R_j, максимизирующих эффективность системы

$$\max E = \max_{\substack{G_i \in G, \\ R_i \in R}} \varphi(Y)$$

при условиях $p_k \in p_k^*, ..., p_s \in p_s^*$.

Сложность решения задачи синтеза обусловлена трудностями в получении модели $Y = F(Z, G, R)$, большим числом варьируемых параметров и областью варьирования. При общей постановке задачи синтеза, когда множества G и R включают в себя все мыслимые варианты построения системы и различные способы управления совместной работой процессов, сложность задачи синтеза превосходит возможности методов моделирования и оптимизации.

В целях упрощения задача синтеза сводится к многократному решению задачи анализа в эволюционном процессе проектирования. Такая схема решения представлена на рис. 2.2.

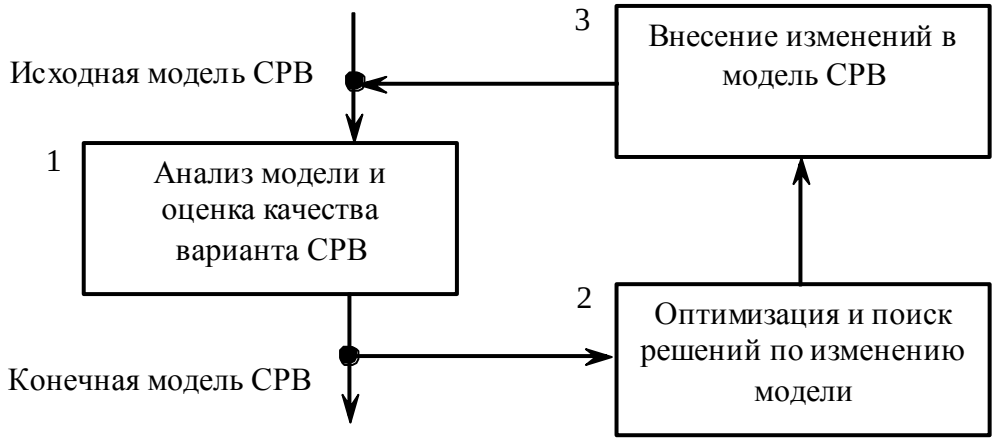


Рис. 2.2. Схема решения задачи синтеза через задачу анализа

Блоки 1, 2, 3 образуют итерационный процесс последовательных преобразований модели СРВ, начиная с исходного варианта модели до

конечного, удовлетворяющего условиям реального времени и приемлемым оценкам критериев качества.

Для решения возникающих при этом задач используются как формальные, так и эвристические методы, причем на долю последних приходится большинство задач.

Ранее отмечалось, что модель может воспроизводить лишь некоторое подмножество свойств проектируемой СРВ. Как правило, это происходит не потому, что часть характеристик вектора $\boldsymbol{Y}$ не интересует исследователя. Чаще всего применяемые модели способны воспроизводить только отдельные свойства проектируемой системы. Ниже перечислим некоторые свойства и типы моделей, используемых для оценок соответствующих характеристик.

Дискретные и ***непрерывные марковские цепи*** используются для оценки надежности системы.

Детерминированные и ***стохастические сети очередей***, ***сети массового обслуживания*** позволяют оценить ряд характеристик, определяющих производительность системы в целом и ее отдельных компонентов.

Статистические модели, в частности ***регрессионный анализ***, широко используется для установления статистических зависимостей между характеристиками и параметрами. Применение статистических моделей обусловлено как необходимостью воспроизведения статистической природы процессов, протекающих в системе, так и отсутствием теории, достаточной для построения математических моделей, описывающих физические свойства элементов системы и отношений между ними.

Математические модели, в которых отношения между характеристиками и параметрами удается представить в виде математических выражений, называют ***аналитическими***. Для анализа характеристик такие модели имеют большую ценность, но построить их удается лишь в тех редких случаях, когда имеется хорошо разработанная теория исследуемых объектов.

Динамику функционирования СРВ можно представить некоторой совокупностью процессов, которые протекают параллельно и взаимодействуют друг с другом. Такие процессы можно сопоставить с прикладными функциями, реализующими задачи СРВ. В этой связи при проектировании СРВ важное место занимают задачи организации взаимодействия совокупности параллельных процессов. Для анализа корректности организации совместной работы совокупности параллельных процессов разработаны ***сети Петри***. Аппарат сетей Петри и их много-

численных модификаций широко используется в задачах синхронизации процессов, выявления тупиковых состояний в динамике функционирования проектируемых систем.

Ниже рассмотрим другой тип менее известных, чем сети Петри, моделей, используемых при анализе взаимодействия процессов. В этих моделях используется явное задание моментов времени запуска процессов, продолжительности выполнения процессов и механизмов селекции данных при передаче их от одного процесса к другому. По имени одного из авторов, такие модели названы ***моделями Керка***. Данные модели рассматриваются здесь в значительной степени для познавательных целей в вопросах организации взаимодействия параллельных процессов.

2.2. Анализ взаимодействия процессов на сетях Керка

В сетях Керка описание прикладных функций, составляющих программную нагрузку СРВ, осуществляется в виде множества взаимосвязанных параллельно выполняемых процессов [14–17].

Программная система СРВ рассматривается как состоящая из взаимосвязанных подсистем. Каждая подсистема рассматривается в свою очередь как система. Процесс разбиения на подсистемы, подподсистемы и т.д. продолжается, пока не будет достигнута приемлемая детальность описания. Система на нижнем уровне описания называется процессом. Обычно процесс выполняет элементарную функционально законченную задачу и вычисляет значения переменных, которые характеризуют решение этой задачи. Значения переменных образуют ***состояние процесса***.

Состояние системы в каждый момент времени определяется совокупностью состояний всех процессов. Заметим также, что в общем случае этап декомпозиции программной системы на процессы формализовать не удается.

Модель должна фиксировать взаимодействие процессов и тем самым вводить на множестве процессов частичную упорядоченность, определяющую последовательность выполнения процессов. Это означает, что отдельные процессы связываются с определенными ситуациями в окружающей системе или с определенными моментами времени. Моменты времени могут быть как относительными (по отношению к событию), так и абсолютными, фиксируемыми в астрономическом времени. Активизация части процессов может зависеть от результата работы программной системы.

Кроме фиксации последовательности выполнения процессов необходимо определять объем информации и моменты ее передачи

между процессами. Модель ориентирована на метод обмена сообщениями, который имеет ряд преимуществ по сравнению с методом разделяемых переменных, например, методом семафоров.

Пусть $\boldsymbol{P}=\{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n\}$ – множество процессов программной нагрузки. Обозначим через $\boldsymbol{S_i}(\boldsymbol{t})$ состояние процесса $\boldsymbol{p_i}$ в момент времени $\boldsymbol{t}$. Тогда состояние системы в момент времени $\boldsymbol{t}$ определяется совокупностью $\boldsymbol{S}(\boldsymbol{t})=\{\boldsymbol{S}_1(\boldsymbol{t}), \ldots, \boldsymbol{S_n}(\boldsymbol{t})\}$. Здесь $\boldsymbol{t}$ принимает только дискретные значения.

В СРВ процессы нередко имеют так называемую внутреннюю память. Например, состояние процесса $\boldsymbol{p_i}$ в момент $\boldsymbol{t_m}$ зависит от состояния процесса $\boldsymbol{p_j}$ в моменты времени $\boldsymbol{t_m}, \boldsymbol{t_{m-1}}, \ldots, \boldsymbol{t_{m-k}}$. Поэтому состояние системы $\boldsymbol{S}$ может включать состояния некоторого процесса, вычисленные в разные моменты времени.

В моделях Керка запуски процесса производятся в определенные моменты времени. С каждым процессом $\boldsymbol{p_i}\in\boldsymbol{P}$ связывается множество ***пусковых моментов*** $\boldsymbol{T}(\boldsymbol{p_i})$ этого процесса. $\boldsymbol{T}(\boldsymbol{p})$ строго упорядоченное, для любого $\boldsymbol{p}\in\boldsymbol{P}\rightarrow\min(\boldsymbol{T}(\boldsymbol{p}))=0$, то есть у всех процессов существует общий начальный пусковой момент. Мощность множества $[0, \boldsymbol{t}]\cap\boldsymbol{T}(\boldsymbol{p})$ конечна, то есть в любом конечном промежутке времени процесс $\boldsymbol{p}\in\boldsymbol{P}$ может быть активизирован конечное число раз.

Для любых процессов $\boldsymbol{p}\in\boldsymbol{P}$ и последовательных элементов $(\boldsymbol{t}, \boldsymbol{t}')\in\boldsymbol{T}(\boldsymbol{p})\rightarrow\boldsymbol{t}_{\min}(\boldsymbol{p})\le(\boldsymbol{t}'-\boldsymbol{t})\le\boldsymbol{t}_{\max}(\boldsymbol{p})$, где $\boldsymbol{t}_{\min}(\boldsymbol{p})$ и $\boldsymbol{t}_{\max}(\boldsymbol{p})$ – заданные из практических соображений функции с соблюдением условия $\boldsymbol{t}_{\min}(\boldsymbol{p})\le\boldsymbol{t}_{\max}(\boldsymbol{p})$.

Чтобы гарантировать циклическое выполнение любого процесса $\boldsymbol{p}\in\boldsymbol{P}$, активизированного в момент $\boldsymbol{t}\in\boldsymbol{T}(\boldsymbol{p})$, нужно, чтобы время выполнения процесса $\boldsymbol{d}(\boldsymbol{p},\boldsymbol{t})$ было строго ограничено сверху, либо выполнялось условие

$$\boldsymbol{d}(\boldsymbol{p},\boldsymbol{t})\in[\alpha(\boldsymbol{p}), \beta(\boldsymbol{p})], \quad \alpha(\boldsymbol{p})<\beta(\boldsymbol{p}).$$

Процесс может принимать данные не только в начале выполнения, а в любой момент. Поэтому необходимо оценить задержку между запуском процесса потребителя и моментом требования входных дан-

ных. Эту задержку обозначим $\boldsymbol{d_1(\sigma_{ij}, t)}$, $\boldsymbol{t \in T(p)}$, где σ_{ij} – средство, обеспечивающее связь между процессами $\boldsymbol{p_i}$ (производители) и $\boldsymbol{p_j}$ (потребители).

Величина $\boldsymbol{d_1(\sigma_{ij}, t)}$ в общем случае является случайной величиной на интервале $\left[\alpha_1(\sigma_{ij}), \beta_1(\sigma_{ij})\right]$, где $\alpha_1(\sigma_{ij}), \beta_1(\sigma_{ij})$ – функции, заданные из практических соображений, при этом $\alpha_1(\sigma_{ij}) \le \beta_1(\sigma_{ij})$.

Предполагается также, что каждый процесс выполняется на собственном процессоре. При этом процесс не имеет информации о потребителях результатов своей работы, он знает только необходимые входные данные.

Каналы взаимодействия процессов

Данными в моделях Керка могут быть только состояния процессов. Поэтому необходимо иметь логические устройства ввода, которые снабжают процессы правильными данными в нужный момент времени. ***Логическое устройство ввода будем называть каналом.***

Канал между процессом производителем $\boldsymbol{p_i}$ и процессом потребителем $\boldsymbol{p_j}$ обозначим $\sigma_{ij} = (\boldsymbol{p_i}, \boldsymbol{p_j})$, $(\boldsymbol{p_i}, \boldsymbol{p_j}) \in \sum$, где $(\boldsymbol{p_i}, \boldsymbol{p_j})$ упорядоченная пара, а $\sum = \boldsymbol{P} \times \boldsymbol{P}$ – множество каналов, необходимых для осуществления взаимодействий в системе.

Для разных типов взаимодействий требуются разные типы каналов.

Нулевой канал $(\sigma_{ij} \in \sum_n)$ не передает информацию, но гарантирует работу процессов $\boldsymbol{p_i}$ и $\boldsymbol{p_j}$ на одном множестве пусковых моментов, то есть $\boldsymbol{T(p_i) = T(p_j)}$.

Синхронный канал $(\sigma_{ij} \in \sum_s)$ передает заранее определенное число последовательных во времени состояний процесса производителя и в то же время гарантирует работу процессов $\boldsymbol{p_i}$ и $\boldsymbol{p_j}$ на одном и том же множестве пусковых моментов.

Полусинхронный канал, или канал Петри $(\sigma_{ij} \in \sum_p)$, передает информацию как и синхронный канал. Связываемые процессы выполняются с одинаковой частотой, то есть процесс-потребитель запускается

после завершения работы процесса-производителя. Тем самым процесс-производитель генерирует множество пусковых моментов процесса-потребителя.

Асинхронный канал $\left(\sigma_{ij} \in \sum_a\right)$ передает информацию как и синхронный канал, но связывает процессы, выполняющиеся на разных множествах пусковых моментов.

В общем случае допускается ввод других типов каналов, поэтому для перечисленной совокупности типов каналов должно выполняться условие $\sum_n \cup \sum_s \cup \sum_p \cup \sum_a = \sum$.

Канал σ_{ij}, связывающий процессы $\boldsymbol{p_i}$ и $\boldsymbol{p_j}$, можно определить как отображение

$$\sigma_{ij} : \boldsymbol{T}\left(\boldsymbol{p_i}\right) * \boldsymbol{T}\left(\boldsymbol{p_j}\right) * \boldsymbol{val}\ \boldsymbol{p_i} \rightarrow \boldsymbol{dom}\ \boldsymbol{p_j},$$

где $\boldsymbol{val}\ \boldsymbol{p_i}$ –(value) область изменения значений состояний процесса $\boldsymbol{p_i}$;

$\boldsymbol{dom}\ \boldsymbol{p_j}$ – (domain) область определения входных данных процесса $\boldsymbol{p_j}$.

Это отображение создает постоянную связь между процессами, то есть состояния процесса $\boldsymbol{p_i}$, определенные в моменты времени $\boldsymbol{t} \in \boldsymbol{T}\left(\boldsymbol{p_i}\right)$, доступны процессу $\boldsymbol{p_j}$ в моменты $\boldsymbol{t'} \in \boldsymbol{T}\left(\boldsymbol{p_j}\right)$ при условии, что $\boldsymbol{t'} \geq \boldsymbol{t}$.

Функция канала

Для обеспечения избирательности взаимодействия процессов во времени введена ***функция канала*** $\boldsymbol{K}\left(\sigma_{ij}, \boldsymbol{t}\right)$. Функция канала определяет подмножество множества $\boldsymbol{T}\left(\boldsymbol{p_i}\right)$, $\boldsymbol{K}\left(\sigma_{ij}, \boldsymbol{t}\right) \subset \boldsymbol{T}\left(\boldsymbol{p_i}\right)$, $\sigma_{ij} \in \sum$, $\boldsymbol{t} \in \boldsymbol{T}\left(\boldsymbol{p_i}\right)$. Функция $\boldsymbol{K}\left(\sigma_{ij}, \boldsymbol{t}\right)$ указывает на совокупность состояний производителя $\boldsymbol{p_i}$, полученных от запусков в моменты времени $\boldsymbol{t'} \in \boldsymbol{K}\left(\sigma_{ij}, \boldsymbol{t}\right)$, $\boldsymbol{t'} \leq \boldsymbol{t}$ и доступных потребителю $\boldsymbol{p_j}$.

В ряде случаев для синхронного канала и канала Петри функцию канала можно задавать на строго упорядоченном множестве $\boldsymbol{T}\left(\boldsymbol{p_i}\right)$ в виде интервала $\left[\mu, \nu\right]$, то есть $\boldsymbol{K}\left(\sigma_{ij}, \boldsymbol{t}\right) = \left[\mu, \nu\right]$. Интервал пусковых

моментов задается по отношению к текущему моменту. Принимается также, что $\nu = 0$ соответствует текущему выполнению процесса $\boldsymbol{p_i}$, а $\nu = 1$ соответствует непосредственно предыдущему выполнению и т.д. Величина μ обозначает число состояний производителя одновременно доступных потребителю. Например, функция $\boldsymbol{K}\left(\sigma_{ij}, \boldsymbol{t}\right) = [2, 1]$ определяет, что через этот канал потребителю одновременно доступны два состояния производителя, непосредственно предшествующие текущему выполнению.

Отметим также, что буфер для сбора состояний организован в канале, то есть процесс производитель не должен заботиться о буферизации своих состояний.

Для асинхронных каналов функция $\boldsymbol{K}\left(\sigma_{ij}, \boldsymbol{t}\right)$ также может быть определена в виде интервала $[\mu, \nu]$. Поскольку взаимосвязанные процессы работают на разных множествах пусковых моментов, необходимо определить момент времени, начиная с которого состояния процесса производителя $\boldsymbol{p_i}$ в принципе доступны потребителю. Все состояния производителя $\boldsymbol{p_i}$, вычисление которых началось до момента $\boldsymbol{t'} \in \boldsymbol{T}\left(\boldsymbol{p_i}\right)$ включительно, где $\boldsymbol{t'}$ удовлетворяет условию

$$\boldsymbol{t' < t + d_1(\sigma_{ij}, t) - d(p_i, t'),}$$

доступны процессу потребителю $\boldsymbol{p_j}$ в момент времени $\boldsymbol{t} \in \boldsymbol{T}\left(\boldsymbol{p_j}\right)$. Из доступных состояний производителя потребитель выбирает нужные ему состояния при помощи функции канала.

Пример анализа взаимодействия процессов на моделях Керка

Рассмотрим пример использования модели Керка для анализа работы программной нагрузки регулятора непосредственного цифрового управления [17] на разных конфигурациях вычислительной системы. Регулятор получает данные от системы управления, в составе которой он работает. На основе этих данных регулятор вычисляет управляющие воздействия по заданному алгоритму. На этапе анализа точный алгоритм не задается, указываются лишь его временные характеристики. Если окажется, что временные характеристики удовлетворяют заданным требованиям, то в регулятор можно встроить пропорционально-

интегральный (ПИ), пропорционально-интегрально-дифференциальный (ПИД) или любой из алгоритмов многомерного регулирования.

Вычисленные значения управляющего воздействия регулятор выводит на исполнительное устройство. Для повышения надежности работы системы введена обратная связь от исполнительного устройства. В зависимости от условий применения нужно принять решение относительно конфигурации вычислительной системы, реализующей регулятор на вычислительной сети, на мультипроцессорной ЭВМ или же на одном процессоре.

Модель регулятора представлена на рис. 2.3 в виде графа, в котором узлами являются процессы, а дугами каналы. Программная нагрузка регулятора включает следующие процессы:

АЛГ – алгоритмы управления;

БУФ – имитирование буфера или линии передачи данных;

ОУТ – подготовка сигналов для исполнительного устройства;

ИСП – имитирование исполнительного устройства;

СИН – синхронизация исполнительной части регулятора.

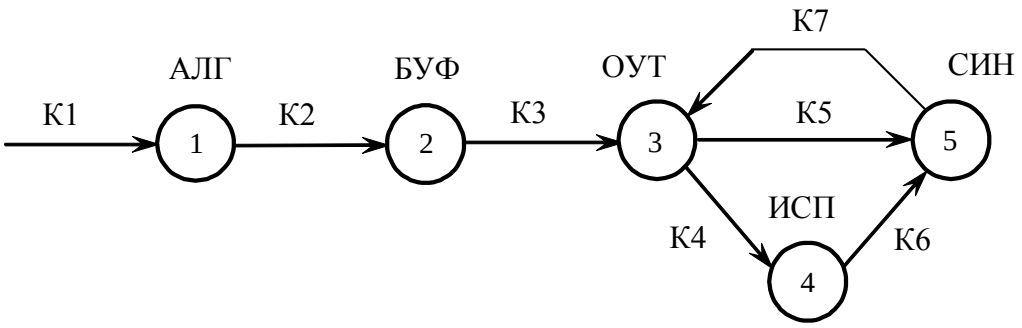


Рис. 2.3. Граф модели регулятора в виде сети Керка

Напомним, что в модели Керка процессы выполняют элементарные функциональные задачи, а каналы обеспечивают взаимодействия процессов.

Анализ графа модели Керка

После фиксации совокупности и каналов у проектировщика имеется несколько возможностей для реализации проекта.

Рассмотрим влияние изменения типа и функции каналов на поведение проектируемой системы. На рис. 2.4 изображено поведение регулятора с тремя разными множествами каналов:

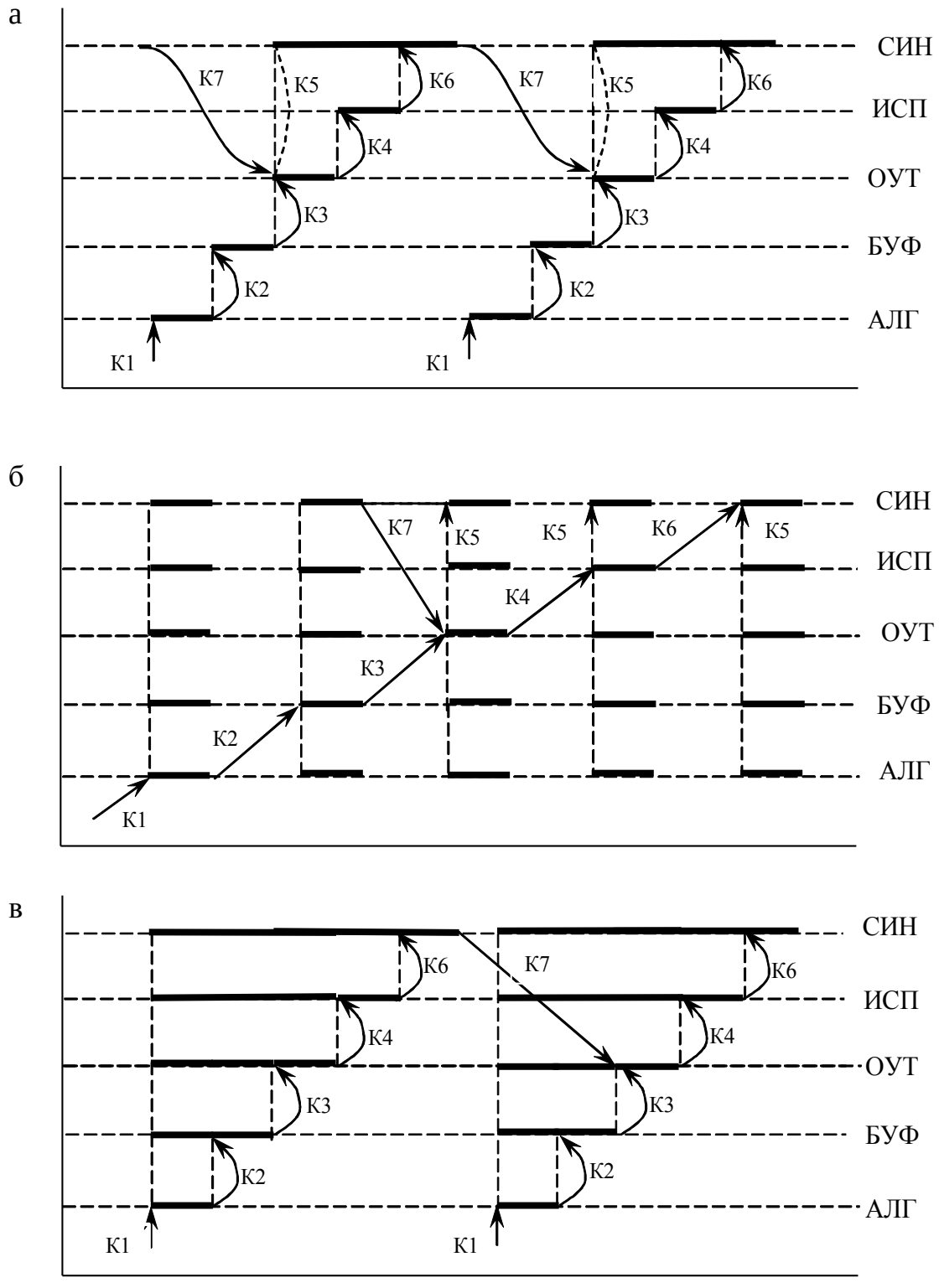


Рис. 2. 4. Варианты временных диаграмм работы регулятора: а – для множества каналов A; б – для множества каналов B; в – для множества каналов C

- множество каналов А; здесь К1, К2, К3, К4 – каналы Петри с функциями [0, 0], К5 – нулевой канал, К6 – асинхронный канал, К7 – синхронный канал с функцией [1, 1];
- множество каналов В; здесь К1, К2, К3, К4, К6, К7 – синхронные каналы с функцией [1, 1], К5 – нулевой канал;
- множество каналов С; здесь К1, К2, К3, К4, К6 – синхронные каналы с функцией [0, 0], К5 – нулевой канал, К7 – синхронный канал с функцией [1, 1].

Временные диаграммы, соответствующие множествам каналов А, В, С, представлены на рис. 2.4 а, 2.4 б, 2.4 в. При этом времена выполнения процессов приняты одинаковыми.

Из сравнения диаграмм очевидны следующие выводы.

Модель Керка с каналами Петри хорошо имитирует поведение сетевых моделей, в которых запуск процесса осуществляется при наличии данных. Соответствующий проект может быть реализован на любой ВС, так как нет особых требований к точности синхронизации. Процесс СИН здесь явно работает как сторожевой таймер.

Модель с множеством каналов В подходит для реализации на мультипроцессорной ВС. Вычислительная система на однопроцессорной ЭВМ будет работать неэффективно, а на сети сложно обеспечить точную синхронизацию отдельных ветвей. Заметим также, что системы с синхронным запуском процессов обеспечивают высокую точность во времени.

Модель с множеством каналов С описывает мультипрограммную работу однопроцессорной ЭВМ, так как процессы в такой схеме взаимодействия могут продвигаться только последовательно.

Анализ временных характеристик работы СРВ на моделях Керка

В качестве объекта исследования берется СРВ, в которой выявлены параллельные взаимодействия отдельных процессов. Все процессы данной СРВ распределены по процессорам. В общем случае число процессоров может быть меньше числа процессов. С каждым процессом связывается множество пусковых моментов и время его выполнения. Алгоритм анализа включает следующие операции.

- Формируется ***матрица пусковых моментов*** $\boldsymbol{TPR(P, N)}$, массив времен выполнения процессов, совокупность каналов взаимосвязи процессов и ряд других сведений.

- Выполняется расчет ***канальных пусковых моментов***. В случае, если канал окажется каналом Петри или синхронным, то это будут моменты времени, когда данные процесса производителя могут быть использованы процессом потребителем. Если канал окажется асинхронным, то это значение укажет в какой момент времени данные процесса производителя в принципе доступны процессу потребителю. Значения канальных пусковых моментов процесса определяются путем прибавления к соответствующим элементам матрицы ***TPR*** времени выполнения процесса.

- Полученные значения канальных пусковых моментов оформляются в виде матрицы $\boldsymbol{T}1(\boldsymbol{P}, \boldsymbol{N})$. Если при анализе учитывается время задержки данных в канале, то матрица $\boldsymbol{T}1$ формируется с учетом этих задержек. Заметим также, что такие задержки имеют место при передаче данных в каналах, связывающих процессы, выполняемые на разных процессорах.

- Построение и вывод ***временной диаграммы*** выполнения процессов. На горизонтальных линиях диаграммы наносятся времена выполнения соответствующих процессов. Для этого используются данные матриц начала и окончания выполнения процессов, матриц ***TPR*** и $\boldsymbol{T}1$. Время работы процесса на диаграмме помечается жирными линиями. При несоответствии пусковых моментов и времени работы процессов конфликтующие участки помечаются жирными линиями удвоенной толщины. Затем, используя сведения о совокупности каналов, на диаграмме наносятся взаимосвязи процессов в виде стрелок, имеющих на конце указание номера процесса производителя.

- Анализ состава каналов для заданных пусковых моментов. Анализ выполняется на основе матриц $\boldsymbol{T}1$ и ***TPR***. Результаты анализа формируются в матрицу состава каналов $\boldsymbol{TKB}(\sum, \boldsymbol{N})$. Элементы матрицы могут принимать следующие значения:

1 – канал Петри;

2 – ***тупиковая ситуация 1-го типа***, которая соответствует нереальному пусковому моменту. В этой ситуации два взаимосвязанных процесса выполняются на одном процессоре и, следовательно, предполагается строгая последовательность выполнения процессов. Тупик возникает в случае, когда процесс-потребитель запросил данные раньше, чем процесс-производитель их выработал. В канале образуется нетранспортная задержка.

3 – асинхронный канал;

4 – ***тупиковая ситуация 2-го типа***, когда несколько процессов одновременно обращаются к одному процессу;

5 – синхронный канал.

- Расчет ***нетранспортной задержки данных***, то есть расчет времени, которое простаивает процесс потребитель в ожидании данных.

- Расчет ***пропускной способности каналов*** и формирование матрицы $\boldsymbol{PSK}$. Пропускная способность канала $\left(\boldsymbol{p_i}, \boldsymbol{p_j}\right)$ рассчитывается по усредненным данным от $\boldsymbol{N}$ пусковых моментов. Вначале определяется среднее время, которое канал может быть загружен передачей данных процесса производителя,

$$\boldsymbol{z} = \frac{1}{\boldsymbol{N}} \sum_{\boldsymbol{m}=1}^{\boldsymbol{N}} (\boldsymbol{TPR}(\boldsymbol{p_j}, \boldsymbol{m}) - \boldsymbol{T}1(\boldsymbol{p_i}, \boldsymbol{m})).$$

При этом суммируются положительные слагаемые, то есть

$$\boldsymbol{TPR}(\boldsymbol{p_j}, \boldsymbol{m}) - \boldsymbol{T}1(\boldsymbol{p_i}, \boldsymbol{m}) > 0.$$

Далее определяется величина $\boldsymbol{r}$ среднего времени, которое «отводит» процесс-потребитель каналу для получения данных,

$$\boldsymbol{r} = \frac{1}{\boldsymbol{N} - 1} \sum_{\boldsymbol{m}=1}^{\boldsymbol{N}-1} (\boldsymbol{TPR}(\boldsymbol{p_j}, \boldsymbol{m}+1) - \boldsymbol{TPR}(\boldsymbol{p_j}, \boldsymbol{m})).$$

Если $\boldsymbol{r} < \boldsymbol{z}$, то элемент $\boldsymbol{PSK}\left(\boldsymbol{p_i}, \boldsymbol{p_j}\right) = \boldsymbol{r}/\boldsymbol{z}$. Для такого канала указывается число обработанных состояний, равное $\left(\boldsymbol{N} - \boldsymbol{r}/\boldsymbol{z}\right)$ и число состояний, стоящих в очереди, равное $\boldsymbol{r}/\boldsymbol{z}$. Если $\boldsymbol{r} \geq \boldsymbol{z}$, то элемент $\boldsymbol{PSK}\left(\boldsymbol{p_i}, \boldsymbol{p_j}\right)$ принимается равным 1, а для канала указывается резервное время (недогруженность) канала, равное разности $\boldsymbol{r} - \boldsymbol{z}$.

Пример анализа временных характеристик

В качестве примера взят граф модели Керка, представленный на рис. 2.5.

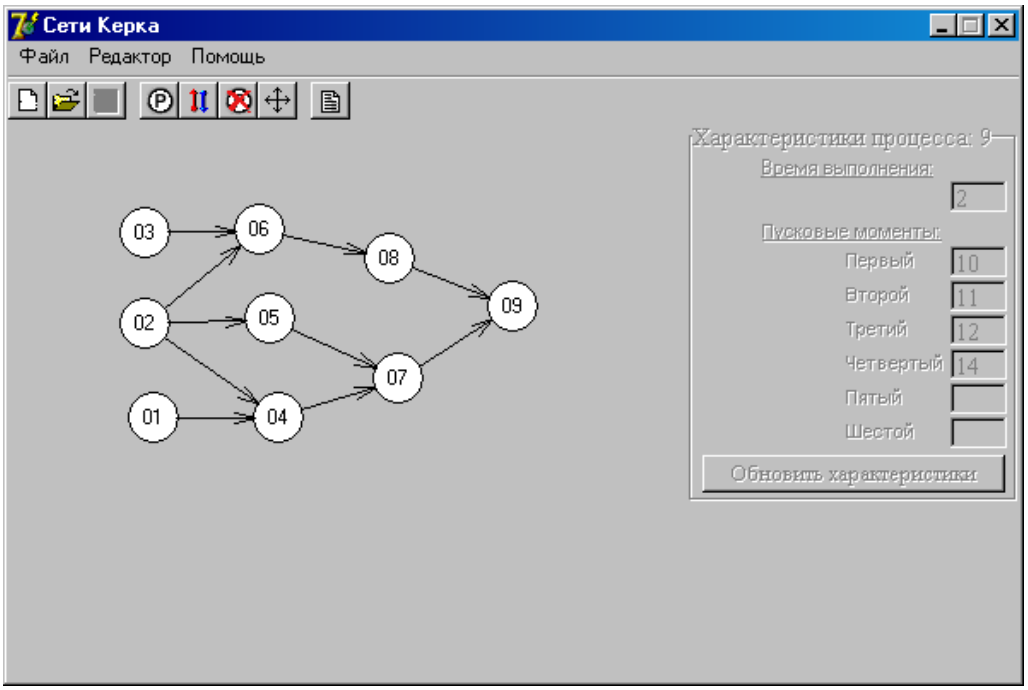


Рис. 2.5. Граф модели Керка

Анализ выполняется с помощью программного средства «Сети Керка», разработанного на кафедре информатики и проектирования систем ТПУ. Матрица пусковых моментов и времена выполнения процессов представлены на рис. 2.6.

Процесс	Время выполнения	1	2	3	4	5	6
1	1	1	4	10	12	0	0
2	1	1	4	7	10	0	0
3	2	1	4	8	12	0	0
4	5	2	6	11	13	0	0
5	2	5	8	16	19	0	0
6	3	2	8	12	16	0	0
7	1	8	9	11	12	0	0
8	1	8	12	16	17	0	0
9	2	10	11	12	14	0	0

Рис. 2.6. Окно с матрицей пусковых моментов

Результаты анализа по третьему пусковому моменту представлены на рис. 2.7.

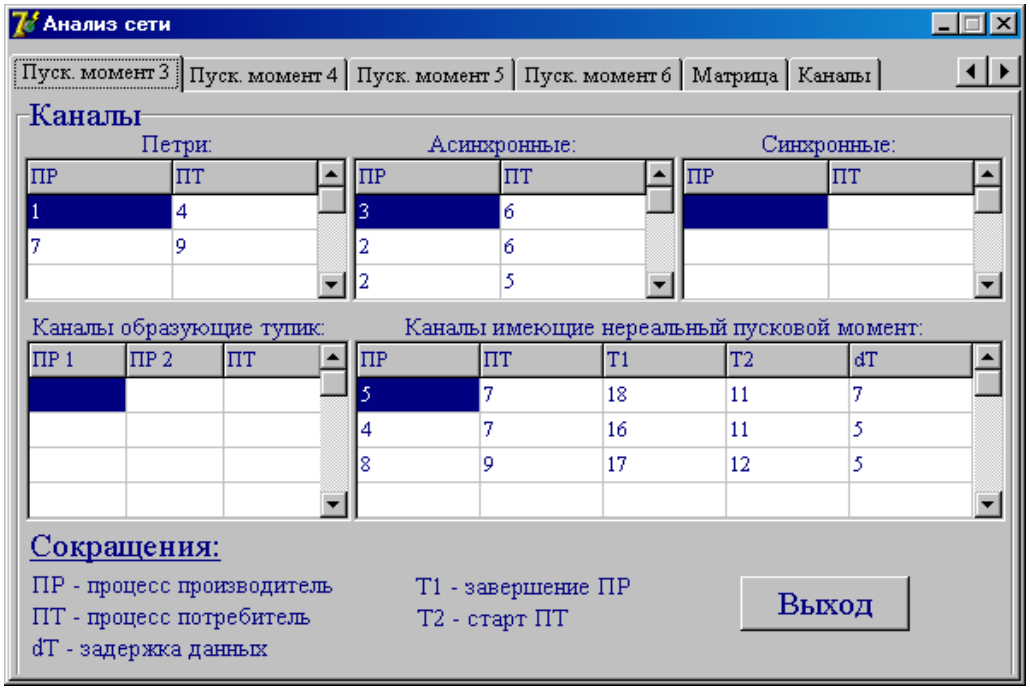


Рис. 2.7. Окно с результатами анализа по 3-му пусковому моменту

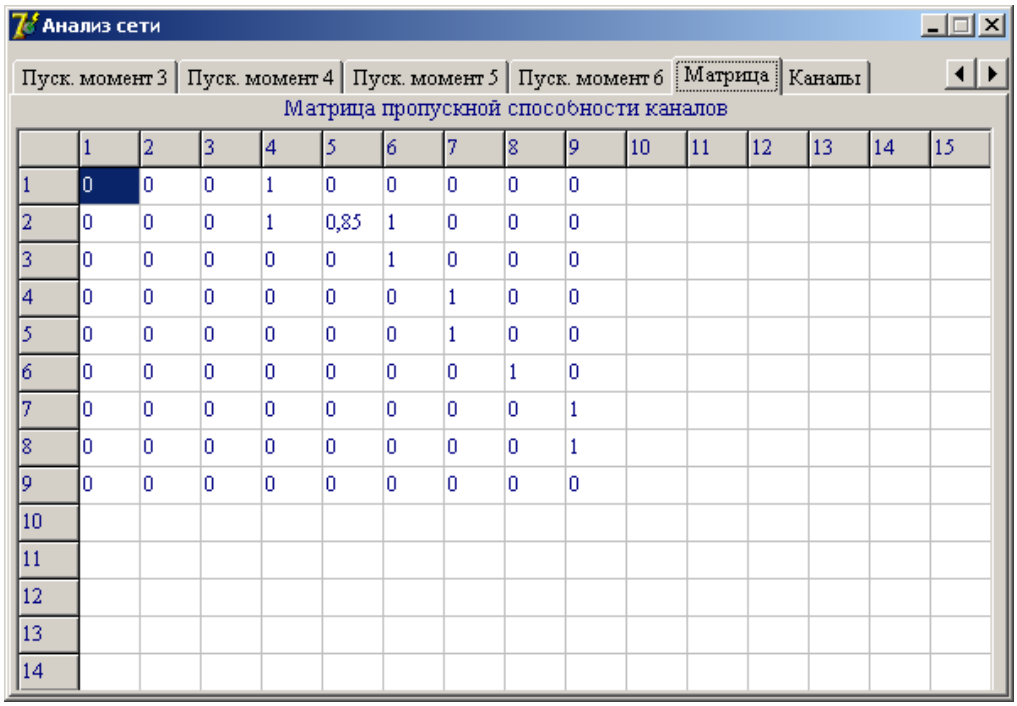


Рис. 2.8. Матрица пропускных способностей

Окно формируется для каждого пускового момента. Приводятся типы каналов, каналы, образующие тупик, каналы, имеющие нереаль-

ный пусковой момент. В данном окне также можно посмотреть матрицу пропускных способностей (рис.2.8) и характеристики каналов (рис. 2.9).

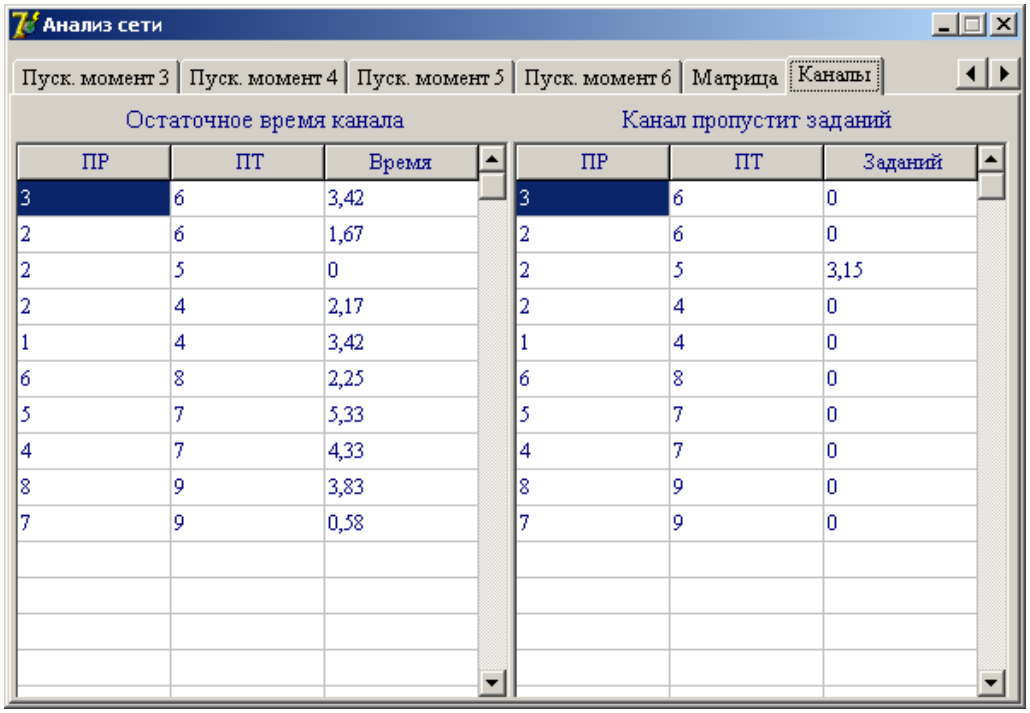


Рис. 2.9. Характеристики каналов

Временная диаграмма выполнения процессов представлена на рис. 2.10.

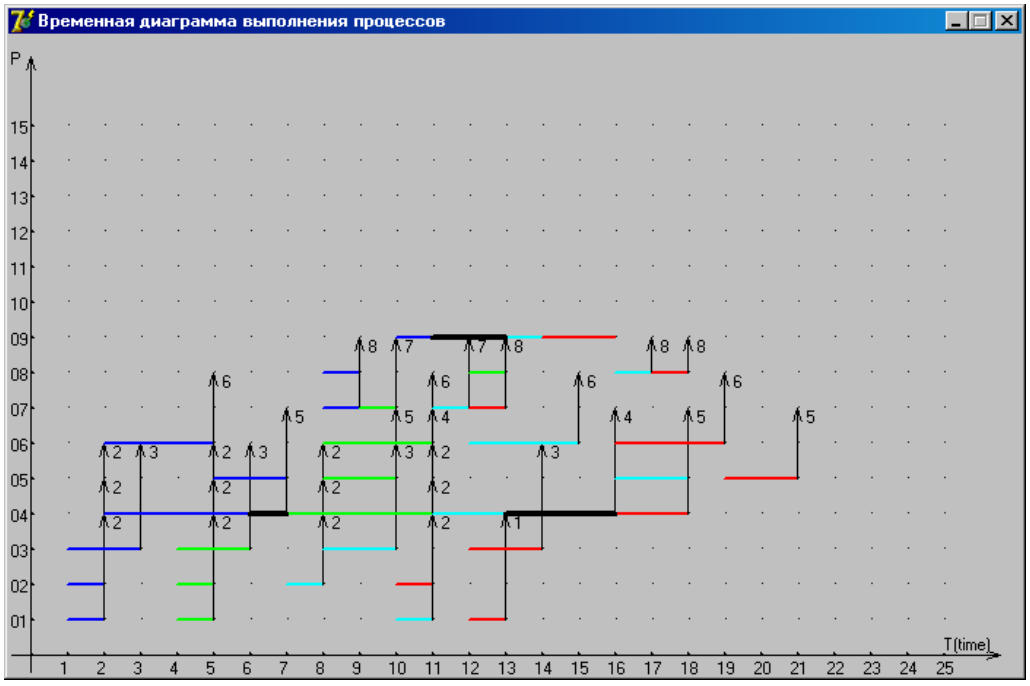


Рис. 2.10. Временная диаграмма выполнения процессов

2.3. Задача выбора числа станций

Анализ на моделях Керка дает некоторое представление о возможных вариантах организации функционирования проектируемой СРВ. Значения временных параметров, полученные на основе анализа временных диаграмм, дают представления о требуемых ресурсах вычислительной системы, в частности, о числе станций, необходимых для реализации программной нагрузки проектируемой СРВ. Число станций, как видно из примера, рассмотренного в разделе 1.5, напрямую зависит также от состава, датчиков и исполнительных механизмов, установленных на окружающей системе, и способности станций подключать определенное число видов датчиков и исполнительных механизмов.

Рассмотрим задачу определения числа станций МВС в условиях, когда совокупность датчиков и исполнительных механизмов и места их установки заданы. Датчики и исполнительные механизмы будем именовать ***терминальными точками***. Таким образом, задачу выбора числа станций для МВС будем рассматривать при известной топологии терминальных точек и заданном наборе конфигураций станций.

Рассматриваемый ниже метод выбора числа станций [13] отличается тем, что вместе с числом станций одновременно определяются и их расположение на объекте, и распределение по ним терминальных точек. Метод также использует замену фактического плана территориального размещения терминальных точек менее детальным плоским планом.

В работе [13] предложена постановка данной задачи как задачи ***линейного математического программирования с булевыми переменными***. Постановка задачи выполнена так, что нелинейную по природе задачу удалось представить как линейную.

Плоский план объекта представляется в виде регулярной координатной решетки с линиями, параллельными осям абсцисс и ординат. Каждому узлу решетки с координатами (i, j) ставится в соответствие величина g_{ij}^{r}, равная числу расположенных в узле (i, j) точек r-го типа. Принимается также, что станции могут размещаться только в узлах решетки и не более чем по одной в каждом узле.

Вводятся следующие переменные:

$$x_{kl} = \begin{cases} 1, & \text{если в узле } (k,l) \text{ размещена станция;} \\ 0, & \text{в противном случае;} \end{cases}$$

$$y_{ij,kl} = \begin{cases} 1, & \text{если точки узла } (i,j) \text{ соединены со} \\ & \text{станцией, размещенной в узле } (k,l); \\ 0, & \text{в противном случае.} \end{cases}$$

В качестве критерия принята минимальная сумма затрат на аппаратуру станций и кабель для подключения точек к станциям. Формальная запись критерия имеет вид

$$\min L = \sum_{(k,l)\in Z} x_{kl} c_s + \sum_{(i,j)\in Z} \sum_{(k,l)\in Z} y_{ij,kl} d_{ij,kl} c_{np} g_{ij}, \qquad (2.1)$$

где Z – множество пар координат всех узлов решетки;

$d_{ij,kl}$ – расстояние между узлами решетки (i,j) и (k,l);

c_s – цена одной станции;

c_{np} – цена единицы длины кабеля, требуемого для соединения станций с терминальными точками;

g_{ij} – число терминальных точек всех видов, отнесенных к узлу (i,j).

Ограничение на число точек r-го вида, подключаемых к станции, можно записать в виде

$$\sum_{(i,j)\in Z} g_{ij}^{r} y_{ij,kl} \le a_r, \qquad \forall r \in R, \qquad \forall (k,l) \in Z. \qquad (2.2)$$

Здесь a_r – предельное число точек r-го вида, которые может подключить станция.

Необходимость подключения точек каждого узла (i,j) к одной из станций может быть представлена в виде

$$\sum_{(k,l)\in Z} y_{ij,kl} = 1, \qquad \forall (i,j) \in Z', \qquad (2.3)$$

где Z' – множество узлов, для которых $g_{ij} \neq 0$.

Задача (2.1) – (2.3) представлена согласно положениям, изложенным в [13]. Более детальный анализ задачи в постановке (2.1) – (2.3) указывает на следующие ее особенности. Все станции принимаются равными по цене и имеют одинаковый вектор подключения, что, как правило, не соответствует реальной действительности. Кроме того, к одной станции должны подключаться все точки узла. Это приводит к сокращению области поиска рационального распределения точек по станциям. Используемая в задаче форма критерия (2.1) предполагает

наличие некоторого баланса между суммарной ценой станций и суммарной ценой кабеля. При значительных различиях в ценах результаты решения задачи могут оказаться несовместимыми с другими требованиями к функционированию системы. Например, с ростом числа станций возникает необходимость учитывать возможные варианты организации связи между ними.

Заметим также, что предлагаемая постановка нелинейной по своей природе задачи в форме линейной получена за счет увеличения размерности. Число переменных $y_{ij,kl}$, объединяющих определение узлов размещения станций (координаты kl) и факт подключения к ним точек узлов (координаты ij), резко возрастает с ростом числа узлов координатной решетки. Так, для решетки размером 20 х 50 линий (1000 узлов), число переменных $y_{ij,kl}$ достигает одного миллиона.

Поэтому для решения данной задачи предлагается другой подход, основанный на разбиении данной задачи на совокупность более простых. При этом каждая станция s-го типа может иметь свой вектор подключения точек различных типов. Соответствующая постановка задачи изложена ниже в разделе 4.1.

2.4. Задача размещения операций обработки прикладных программ и массивов данных

При проектировании МВС распределенной СРВ имеется возможность выбора варианта распределения операций обработки среди процессоров станций и хранения программ, реализующих эти операции, и массивов данных в запоминающих устройствах тех или иных станций [13].

Размещение операций, массивов и программ влияет на загрузку каналов, связывающих станции, а также на загрузку процессоров и запоминающих устройств станций.

Три показателя функционирования ВС распределенной СРВ – загрузка сети передачи данных, загрузка процессоров и загрузка запоминающих устройств станций – определяют три частные задачи размещения операций, массивов и программ. В каждой из задач один из показателей вводится в критерий, а два других должны удовлетворять заданным ограничениям. Как правило, выбирается тот критерий, который соответствует «узкому» месту в проектируемой СРВ. Иногда некоторые характеристики априорно заданы и вводятся автоматически в ограничения. Если критерии равнозначны, то используется итеративная схема решения задач по названным частным критериям.

Критерий минимума загрузки сети передачи данных, как правило, является предпочтительным, так как позволяет уменьшить время, затрачиваемое на обмен данными между станциями по каналам связи, а следовательно, время реализации и запаздывания прикладных функций.

При решении данной задачи для нормального режима функционирования СРВ поиск размещения осуществляется по минимуму средней загрузки сети, определяемой частотами и объемами сетевых передач, необходимых при выполнении периодических прикладных функций.

Для аварийного режима рассчитывается пиковая загрузка сети, которая возникает при одновременном вызове операций тех функций, которые должны исполняться в случае аварии.

Критерий минимума загрузки максимально загруженного процессора также приводит к уменьшению общего времени реализации прикладных функций, благодаря равномерной загрузке процессоров. Здесь также рассчитываются варианты распределения для нормального и аварийного режимов.

Критерий минимума загрузки запоминающих устройств станций означает снижение числа копий массивов данных, распределяемых между станциями. При этом, как правило, увеличивается загрузка каналов связи дополнительными передачами и вследствие этого ухудшаются показатели времени и надежности выполнения прикладных функций.

Рассмотрим вычислительную сеть распределенной СРВ, состоящую из множества $\boldsymbol{V}$ типовых станций, соединенных каналами связи. Имеется множество $\boldsymbol{U}$ операций, использующих множество $\boldsymbol{P}$ прикладных программ ($\boldsymbol{U}=\boldsymbol{U}'\bigcup\boldsymbol{U}''$, где $\boldsymbol{U}'$ и $\boldsymbol{U}''$ – множества операций, составляющих соответственно периодические и непериодические функции). В памяти станций должно быть также размещено множество $\boldsymbol{S}$ массивов, представляющих собой исходные данные и результаты операций обработки.

Заданы размеры σ_s массивов $\boldsymbol{s}\in\boldsymbol{S}$, программ σ_p, $\boldsymbol{p}\in\boldsymbol{P}$ и σ_u рабочей памяти, требуемой для выполнения операции $\boldsymbol{u}\in\boldsymbol{U}$; частота δ_u выполнения каждой операции $\boldsymbol{u}\in\boldsymbol{U}'$. Можно также примерно оценить объем вычислений ρ_u для каждой операции $\boldsymbol{u}\in\boldsymbol{U}$ (например, в элементарных машинных операциях).

Вводятся следующие переменные:

$$x_{uv} = \begin{cases} 1, & \text{если операция } u \in U \text{ выполняется на станции } v \in V, \\ 0, & \text{в противном случае;} \end{cases}$$

$$x_{sv} = \begin{cases} 1, & \text{если массив } s \in S \text{ хранится в памяти станции } v \in V, \\ 0, & \text{в противном случае;} \end{cases}$$

$$x_{pv} = \begin{cases} 1, & \text{если программа } p \in P \text{ хранится в памяти станции } v \in V, \\ 0, & \text{в противном случае.} \end{cases}$$

При постановке частных задач размещения должен быть учтен тип сети передачи данных. Для распределенных СРВ характерны следующие принципиально отличающиеся друг от друга типы сети:

а) ***магистраль*** с несколькими возможными протоколами доступа к ней станций;

б) ***парные связи*** между станциями;

в) ***иерархия магистралей***, когда имеет место несколько магистралей нижнего уровня и магистраль верхнего уровня, по которой организована связь между магистралями нижнего уровня.

Поток данных от станции в магистральной сети типа (а)

Для такой сети в каждый момент времени передача данных может осуществляться только одной станцией. Поэтому для определения средней загрузки магистрали в единицу времени суммируются усредненные во времени потоки данных φ_v, передаваемых каждой станцией $v \in V$.

Поток φ_v складывается из потоков данных, образуемых вследствие того, что выходные массивы некоторых операций, выполняемых в станции v, хранятся в памяти других станций, и из потоков, обусловленных тем, что некоторые массивы (программы) хранятся в памяти станции v, а операции, использующие эти массивы (программы), выполняются на других станциях, в памяти которых эти массивы (программы) отсутствуют:

$$\varphi_v = \sum_{u \in U'} \sum_{s \in S(u)} \delta_u \sigma'_s x_{uv} \Big(\sum_{\substack{j \in V \\ j \neq v}} x_{sj} \Big) + \sum_{s \in S} \sum_{u \in U'(s)} \delta_u \sigma'_s x_{sv} \Big[\sum_{\substack{j \in V \\ j \neq v}} x_{uj} (1 - x_{sj}) \Big] +$$

$$+ \sum_{p \in P} \sum_{u \in U'(p)} \delta_u \sigma'_p x_{pv} \Big[\sum_{\substack{j \in V \\ j \neq v}} x_{uj} (1 - x_{pj}) \Big],$$

где $S(u)$ – множество выходных массивов операции u;

$U'(s)$ – множество операций, использующих при выполнении массив s, $U'(s) \in U'$;

$U'(p)$ – множество операций, выполняемых по программе p, $U'(p) \in U'$;

$\sigma'_s = \sigma_s + \theta$, $\sigma'_p = \sigma_p + \theta$; θ – размер служебной информации, приходящейся на каждую передачу массива.

Если передача осуществляется порциями фиксированной длины и на каждую порцию байт передается q байт служебной информации, то $\sigma'_s = \sigma_s(1+q)$, $\sigma'_p = \sigma_p(1+q)$.

Суммарный объем данных φ_v, которыми должны обмениваться станции в аварийной ситуации, можно рассчитать по формуле, приведенной выше, заменив U', $U'(s)$, $U'(p)$ и δ_u соответственно на U'', $U''(s)$, $U''(p)$ и 1.

Поток данных между станциями в сети с парными связями типа (б)

В общем случае не каждая пара станций непосредственно соединена каналом связи, и тогда данные между такими двумя станциями передаются по цепи из нескольких каналов связи (возможно, не единственной). Для сети с парными связями вводятся булевы переменные, определяющие фиксированные маршруты передачи данных между станциями:

$$y_{z(j,k)} = \begin{cases} 1, \text{ если поток данных между парой станций } (j,k) \\ \quad \text{загружает } z\text{-ю цепь каналов, связывающую } j \text{ и } k; \\ 0, \text{ в противном случае.} \end{cases}$$

При этом $z \in Z$, где Z – множество всех цепей каналов связи в сети.

Некоторые переменные $y_{z(j,k)}$ задаются равными 1, если между станциями j и k может существовать единственная цепь каналов, или равными 0, если станции j и k не соединяются цепью z.

Поток данных между станциями j и k складывается из потоков, которые могут возникнуть по следующим причинам:

- операция $u \in U'$ и ее результат $s \in S(u)$ размещены в разных станциях из пары $j,k \in V$.
- операция $u \in U'$ и входной массив $s \in S'(u)$ [$S(u)$– множество входных массивов операции u] этой операции (или программа p выполнения операции u) размещены в разных станциях из пары $j,k \in V$ и при этом копия этого входного массива (или программы) не хранится в станции, в которой выполняется операция u

Тогда с учетом введенных булевых переменных количество данных ψ_{gh}, передаваемое в среднем в единицу времени по каналу связи между парой станций $g,h \in V$, будет

$$\Psi_{gh} = \sum_{z \in Z} \sum_{(j,k) \in V} a_z(g,h) y_{z(j,k)} \times \sum_{u \in U'} \left\{ \sum_{s \in S(u)} \delta_u \sigma'_s (x_{uj} x_{sk} + x_{uk} x_{sj}) + \right.$$
$$+ \sum_{s \in S'(u)} \delta_u \sigma'_s \left[x_{uj} x_{sk} (1 - x_{sj}) + x_{uk} x_{sj} (1 - x_{sk}) \right] +$$
$$\left. + \delta_u \sigma'_p \left[x_{uj} x_{pk} (1 - x_{pj}) + x_{uk} x_{pj} (1 - x_{pk}) \right] \right\},$$

где $a_z(g,h)$ – элемент матрицы соответствия каналов связи цепям каналов, равный 1, если канал (g,h) входит в z-ю цепь, и 0 в противном случае.

При расчете объема ψ_{gh} данных, передаваемых в аварийном режиме, U' и δ_u заменяются соответственно на U'' и 1.

Усредненная во времени загрузка γ_v процессора станции $v \in V$ обусловливается частотами и объемом вычислений, необходимых для выполнения в этой станции операций

$$\gamma_v = \sum_{u \in U'} \delta_u \rho_u x_{uv} .$$

Расчет объема ρ_v вычислений, которые необходимо выполнять в станции v в аварийной ситуации, производится путем замены U' и δ_u, соответственно на U'' и 1.

Загрузка λ_v запоминающего устройства станции $v \in V$ определяется размерами массивов данных, программ и рабочей памяти, необходимой для выполнения операций, размещенных в этой станции:

$$\lambda_v = \sum_{u \in U} \sigma_u x_{uv} + \sum_{s \in S} \sigma_s x_{sv} + \sum_{p \in P} \sigma_p x_{pv} \ .$$

Критерий минимума загрузки сети

Критерий размещения операций, массивов и программ по минимуму загрузки сети передачи данных для магистральной связи между станциями записывается следующим образом:

$$\min_X \sum_{v \in V} \varphi_v \ , \tag{2.4}$$

где X – множество переменных задачи.

Сети типа (б) с парными связями между станциями, как правило, строятся на каналах одинаковой пропускной способности. С целью достижения равномерной загрузки каналов критерий оптимальности для таких сетей формулируется как минимум максимальной загрузки канала связи:

$$\min_X \max_{(g,h) \in V} \Psi_{gh} . \tag{2.5}$$

Для аварийного режима φ_v и Ψ_{gh} в критериях (2.4) и (2.5) заменяются соответственно на σ_v и σ_{gh}.

Ограничения

При решении задачи должен быть соблюден ряд ограничений.

Свободная емкость памяти L каждой станции должна быть достаточной для выполнения операций и хранения массивов данных и программ, размещенных в станции:

$$\sum_{u \in U} \sigma_u x_{uv} + \sum_{s \in S} \sigma_s x_{sv} + \sum_{p \in P} \sigma_p x_{pv} \le L, \qquad \forall v \in V \ . \tag{2.6}$$

Быстродействие R процессора каждой станции должно быть достаточным для выполнения с соответствующей частотой всех операций, размещенных в станции:

$$\beta_r \sum_{u \in U'} \delta_u \rho_u x_{uv} \le R, \qquad \forall v \in V \ , \tag{2.7}$$

где β_r – коэффициент запаса, учитывающий неравномерность загрузки процессора и выбираемый на основании экспертных оценок.

Каждая операция должна выполняться в одной из станций

$$\sum_{v \in V} x_{uv} = 1, \qquad \forall u \in U . \tag{2.8}$$

Копия каждого массива и каждой программы должна храниться в памяти по меньшей мере одной из станций:

$$\sum_{v \in V} x_{sv} \geq 1, \qquad \forall s \in S ; \tag{2.9}$$

$$\sum_{v \in V} x_{pv} \geq 1, \qquad \forall p \in P . \tag{2.10}$$

Для сети с парными связями вводятся ограничения

$$\sum_{z \in Z} y_{z(j,k)} = 1, \qquad \forall j,k \in V, \qquad j \neq k , \tag{2.11}$$

обуславливающие передачу данных между каждой парой станций по одной из цепей каналов связи, соединяющих эти станции.

Задача размещения операций, массивов и программ для магистральной сети типа (в)

При минимизации загрузки сети типа (в) с несколькими магистралями поиск оптимального размещения осуществляется в два этапа:

Первый этап. Задача размещения решается в постановке сформулированной выше для сети типа (б), причем операции, массивы и программы распределяются между группами станций, объединенных разными магистралями;

каналам связи при решении задачи на этом этапе соответствуют каналы, соединяющие отдельные магистрали с помощью межсетевых интерфейсов;

ограничения по памяти и быстродействию формируются по соответствующим суммарным характеристикам групп станций;

Второй этап. В каждой группе станций, объединенных отдельной магистралью, осуществляется размещение в станциях операций, массивов и программ, распределенных в эту группу в результате решения задачи на первом этапе.

При этом размещение осуществляется согласно постановке задачи, сформулированной выше для сетей типа (а).

Критерий равномерной загрузки процессоров

Если при выборе размещения проектировщик желает минимизировать загрузку процессора станции, максимально загруженной выпол-

нением операций, то есть равномерно распределить операции между процессорами станций, то задача решается по критерию:

$$\min_{X} \max_{v \in V} \gamma_v \qquad (2.12)$$

(для аварийного режима вместо γ_v подставляем ρ_v) при соблюдении ограничения по памяти (2.6), а также ограничений на загрузку сети передачи данных. Это ограничение записывается для сети типа (а) в виде

$$\alpha_w \beta_w \sum_{v \in V} \varphi_v \le w, \qquad (2.13)$$

а для сети типа (б):

$$\alpha_{w'} \beta_{w'} \psi_{gh} \le w', \qquad \forall (g,h) \in V, \qquad g \ne h, \qquad (2.14)$$

где w и w' – соответственно пропускная способность магистрали и канала парной связи;

α_w – коэффициент запаса, учитывающий метод доступа, принятый в данной сети (например, при организации доступа с помощью передачи маркера пропускная способность сети используется примерно на 30%; в этом случае выбирают α_w=3,3);

β_w – коэффициент запаса, учитывающий неравномерность загрузки канала связи и выбираемый с помощью экспертных оценок.

При решении задачи должны быть соблюдены ограничения (2.8) – (2.11).

Критерий минимальной загрузки запоминающих устройств

При минимизации суммарной загрузки запоминающих устройств станций поиск оптимального распределения операций, массивов и программ осуществляется по критерию

$$\min_{X} \sum_{v \in V} \lambda_v$$

при ограничении (2.7) по быстродействию процессоров, (2.12), (2.13) по пропускным способностям каналов и ограничений (2.8) – (2.11).

Метод линеаризации

Сформулированные выше булевы задачи размещения (2.4), (2.6) – (2.10); (2.5) – (2.11); (2.6), (2.8) – (2.14); (2.15), (2.7) – (2.13) – нелиней-

ные по критерию или ограничениям, но могут быть сведены к эквивалентным линейным булевым задачам.

Суть метода линеаризации состоит в следующем. Каждое слагаемое вида $\prod_{i \in Q} x_i$, входящее в целевую функцию, заменяется булевой переменной z_Q. При этом, если данное слагаемое входит в целевую функцию со знаком «+», то оно должно удовлетворять условию

$$z_Q \geq \sum_{i \in Q} x_i + 1 - |Q|, \tag{2.16}$$

а если со знаком «-», то условию

$$z_Q \leq (\sum_{i \in Q} x_i) / |Q|, \tag{2.17}$$

где $|Q|$ – число элементов в Q.

Можно показать, что решение задачи по исходной и линеаризованной моделям эквивалентно. Так, если слагаемое $\prod_{i \in Q} x_i$ входит в целевую функцию со знаком «+» и $\forall i \in Q[x_i = 1]$, то с учетом (2.16) получим $z_Q = 1$. Если же хотя бы одна переменная $x_i = 0$, то z_Q может быть равна 0 или 1. Но в этом случае минимизация целевой функции приводит к $z_Q = 0$. Если слагаемое входит в целевую функцию со знаком «-» и хотя бы одна из переменных $x_i = 0$, то с учетом (2.17), $z_Q = 0$, а если $\forall i \in Q[x_i = 1]$, то z_Q может быть равна 1 или 0, но тогда минимизация целевой функции приводит к $z_Q = 1$.

Если слагаемое вида $\prod_{i \in Q} x_i$ входит в ограничение, то оно заменяется булевой переменной z_Q, на которую накладываются оба ограничения (2.16) и (2.17). При этом, если хотя бы одна переменная из x_i равна 0, то с учетом ограничений (2.16) и (2.17) $z_Q = 0$; если все переменные x_i равны 1, то $z_Q = 1$.

Задача размещения в постановках (2.5) – (2.11); (2.6), (2.8) – (2.14) имеет минимаксный критерий вида $\min_X \max_{v \in V} \theta_v$, где θ_v – загрузка процессора или канала связи; X – множество переменных задачи. Введением целочисленной переменной ξ такой, чтобы выполнялись условия

$\theta_v \le \xi$, $\forall v \in V$, задача сводится к эквивалентной задаче минимизации с критерием вида $\min_x \xi$.

Недостатком метода введения дополнительных переменных и ограничений является рост размерности задачи при ее линеаризации.

Например, для распределения 25 программных модулей по 15 процессорам потребовалось 2500 переменных и 8000 ограничений [13].

2.5. Оптимизация времени выполнения прикладных функций

Задача минимизации времени выполнения периодических прикладных функций

Время выполнения прикладных функций зависит от числа станций и размещения в них операций, программ и массивов. Когда основная часть прикладных функций реализуется в рамках регламентированного периодически повторяющегося цикла, быстродействие СРВ (время реакции системы), можно характеризовать временем выполнения этого цикла.

Под циклом прикладных функций (ПФ) понимают упорядоченное во времени выполнение на ВС операций всех периодических ПФ – от сбора и первичной обработки информации от внешних источников до вывода управляющих воздействий и отображений информации. В общем случае разные операции обработки могут выполняться в пределах цикла различное число раз, определяемое частотой их исполнения.

Из-за наличия информационных связей увеличение числа станций свыше некоторого значения приводит к увеличению времени передачи данных по каналам связи и соответственно времени выполнения цикла ПФ. Таким образом, существует оптимальное число станций, при котором время выполнения цикла ПФ минимально. Поэтому целесообразно ставить задачу оптимального распределения операций, программ и массивов среди станций, число которых может принимать практически любое значение.

Полученное оптимальное число станций может не совпасть с выбранным ранее числом станций. В этом случае проектировщик должен решить оправдано ли увеличение затрат на дополнительные станции или каналы связи ради достигаемого при этом повышения быстродействия СРВ. После этого задача поиска распределения, оптимального по времени выполнения цикла, может быть решена еще несколько раз с

последовательным ужесточением ограничения на число станций снизу или сверху.

Рассмотрим распределенную СРВ, реализуемую с применением станций, подключенных к общей магистрали передачи данных с пропускной способностью $\boldsymbol{w}$. Каждая станция имеет память емкостью $\boldsymbol{L}$ и процессор с быстродействием $\boldsymbol{R}$.

Для реализации всего цикла периодических ПФ требуется множество $\boldsymbol{U'}$ операций обработки, выполнение которых осуществляется набором $\boldsymbol{P}$ прикладных программ. Входные и выходные массивы данных всех операций составляют множество $\boldsymbol{S}$.

Известны размеры массивов, программ, объемы вычислений и рабочей памяти, необходимые для выполнения операций. Ранее принятые обозначения данных характеристик сохранены. Задано также число η_u выполнения каждой операции $\boldsymbol{u} \in \boldsymbol{U'}$ в одном цикле ПФ.

Совокупность ПФ представлена на рис. 2.11 в виде ориентированного графа функциональной структуры. Вершины графа, соответствующие операциям, упорядочены в ***ярусно-параллельной форме*** [13].

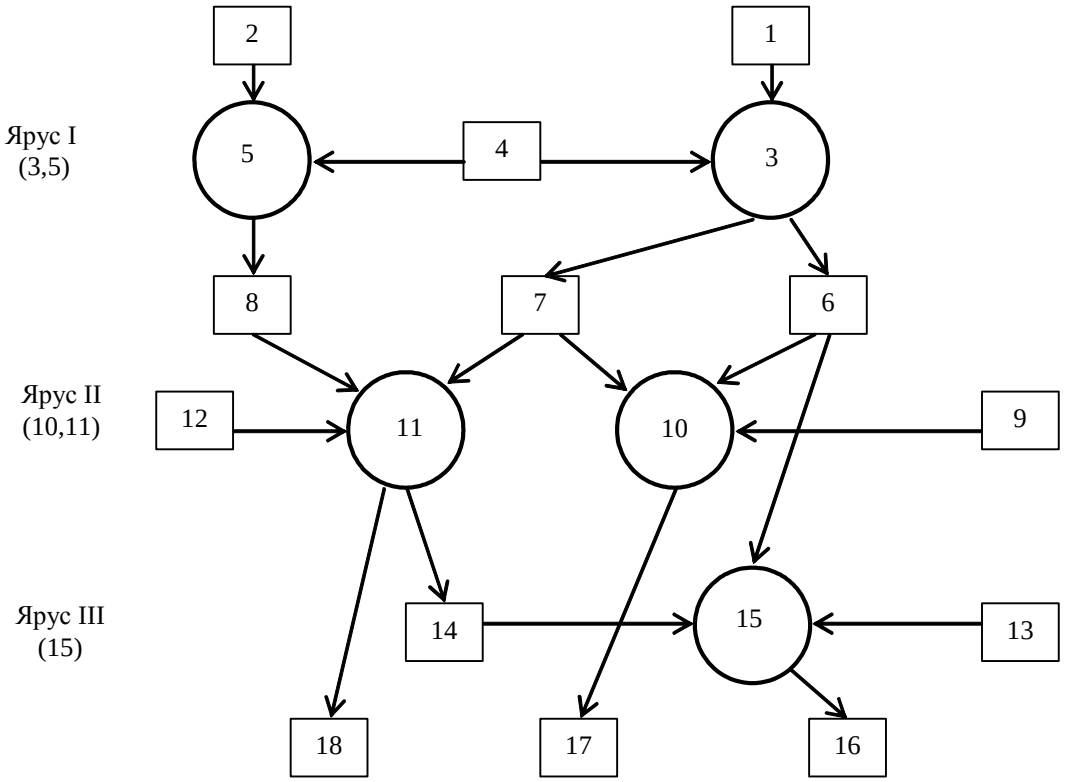


Рис. 2.11. Функциональный граф в ярусно-параллельной форме

Первый ярус образует множество тех и только тех операций, которые используют только данные, соответствующие входным терминальным вершинам. Ярус i образуется множеством тех и только тех операций, которые пользуются выходными массивами хотя бы одной операции яруса $i-1$ и не пользуются выходными массивами операций из ярусов с номерами больше i.

Точный расчет времени выполнения цикла ПФ требует построения временной диаграммы расписания выполнения программ и передач данных по магистрали. Пример расписания для функционального графа (рис. 2.11) и распределения операций массивов и программ (ст.1: 1, 3, 4, 6, 7, 9, 10, 13, 15, 16, 17; ст.2: 2, 4, 5, 8, 11, 12, 14, 18) приведен на рис. 2.12(а).

Расчет времени выполнения цикла T' путем построения временной диаграммы является слишком трудоемким.

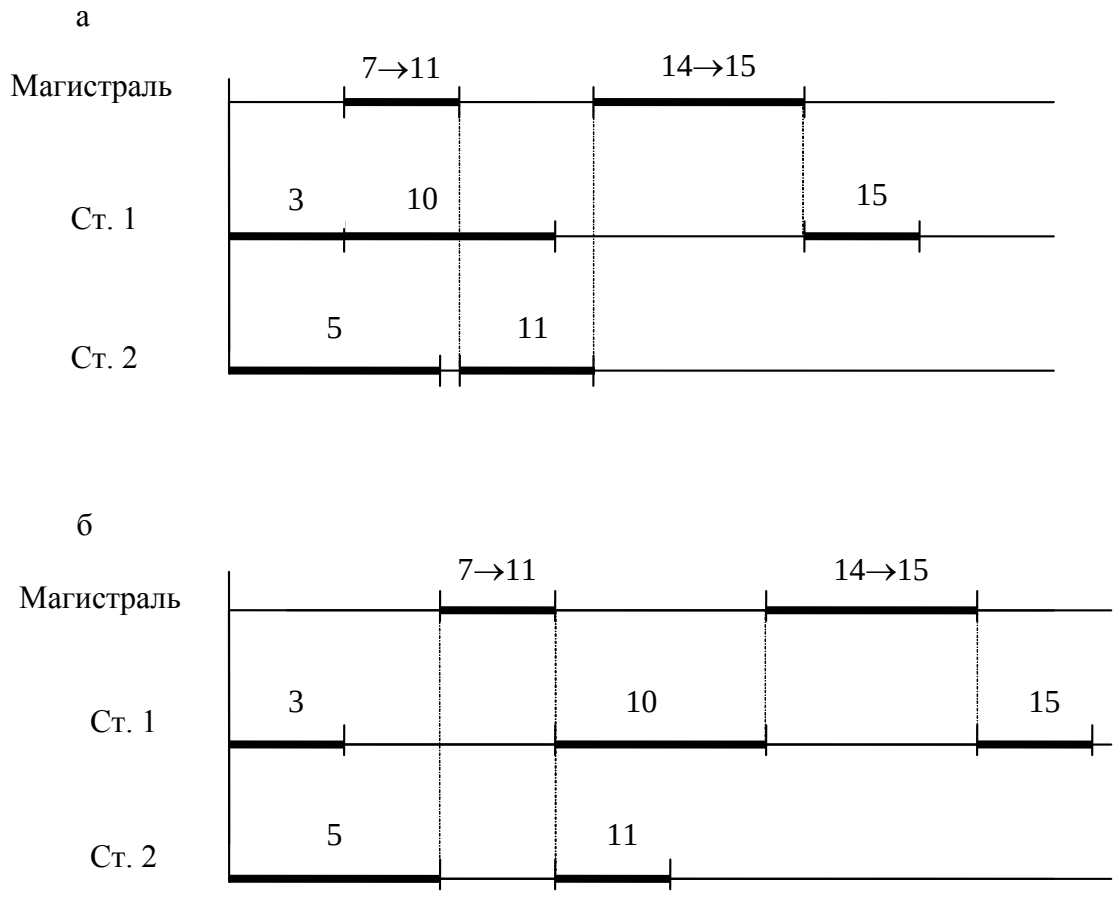


Рис. 2.12: а – временная диаграмма расписания;
б – упрощенная диаграмма

Поэтому для упрощения расчетов в целевую функцию модели вместо величины T' вводят оценочную величину T'', которую рассчитывают, исходя из того, что параллельно в разных станциях могут выполняться операции только одного яруса и передачи данных по магистрали от операций данного яруса к операциям других ярусов осуществляются только после выполнения всех операций данного яруса. Пример расписания с учетом данного допущения приведен на рис. 2.12, б.

Таким образом, $T'' = \sum_{i \in J} T_i^p + T^t$,

где T_i^p – время обработки операций яруса $i \in J$;

T^t – суммарное время загрузки магистрали при выполнении одного цикла функций.

Решения задач минимизации T' и T'' практически во всех случаях будут эквивалентными.

Булевы переменные задачи x_{uv} (а также x_{sv}, x_{pv}) равны 1, если операция $u \in U$ (массив $s \in S$, программа $p \in P$) размещена в станции $v \in V$, и 0 в противном случае. Мощность множества V станций определяется проектировщиком, исходя из указанных выше неформальных соображений.

Если число станций n точно известно заранее и задача распределения операций, массивов и программ решается между известным числом станций, то переменные x_{uv}, x_{sv}, x_{pv} для $v > n$ задаются равными 0. При этом ограничения (2.8) – (2.10) должны выполняться.

Обозначим через $U'(i)$ множество операций i-го яруса, тогда

$$T_i^p = \max_{v \in V} \frac{1}{R} \sum_{u \in U'(i)} (x_{uv} \eta_u \rho_u).$$

Для вычисления суммарного времени, занимаемого передачей данных по магистрали между станциями в одном цикле прикладных функций, воспользуемся формулой определения средней загрузки магистрали в единицу времени, заменив частоту δ_u на число η_u и введя в знаменатель величину w пропускной способности магистрали:

$$T^t = \sum_{v \in V} \left\{ \sum_{u \in U'} \sum_{s \in S(u)} \eta_u \sigma'_s x_{uv} \left(\sum_{\substack{j \in V \\ j \neq v}} x_{sj} \right) + \sum_{s \in S} \sum_{u \in U'(s)} \eta_u \sigma'_s x_{sv} \left[\sum_{\substack{j \in V \\ j \neq v}} x_{uj} (1 - x_{sj}) \right] + \right.$$

$$+\sum_{p\in P}\sum_{u\in U'(p)}\eta_u\sigma'_p x_{pv}\left[\sum_{\substack{j\in V\\ j\neq v}} x_{uj}(1-x_{pj})\right]\Bigg\}/w\,.$$

Таким образом, критерий задачи минимизации длительности цикла прикладных функций записывается следующим образом:

$$\min_X\left\{\sum_{i\in I}(T_i^{\,p})+T^t\right\}.$$

При решении задачи учитываются ограничения по памяти станций (2.6). После решения оптимальное число станций определяется по максимальному среди переменных x, равных 1, значению индекса v.

Поставленная нелинейная булева задача имеет критерий вида

$$\min_X\Big(\sum_{i\in I}\max_{v\in V}\gamma_v(i)+T^t\Big),$$

где $\gamma_v(i)$ – загрузка v-й станции операциями i-го яруса, и сводится к эквивалентной задаче минимизации введением целочисленной переменной ξ и условий $\gamma_v(i)\le\xi,\quad \forall v\in V,\quad \forall i\in I$.

Точное решение задачи осуществляется путем ее линеаризации методом, описанным выше.

Разработаны также эвристические алгоритмы, которые заключаются в повторении размещения операций, массивов и программ для ряда реальных значений n числа станций.

Задача сокращения запаздывания выполнения непериодических прикладных функций

Время запаздывания реализации ПФ, то есть промежуток между моментами ее вызова и начала выполнения, определяется ожиданием в очереди к процессорам и каналам связи операций обработки и передач данных, связанных с данной функцией.

Допустимое запаздывание $\hat{T}_j$ в исполнении ПФ обычно задается либо максимальным значением, либо определенным значением с указанием его вероятности.

Если известно $\hat{T}_j$, то задавшись некоторым коэффициентом, можно определить необходимое среднее запаздывание. Необходимо

также проверить соблюдение требования к максимальному времени запаздывания $T_{j\max} \le \hat{T}_j$.

Если ПФ реализуется в рамках жестко заданного периодически выполняемого цикла, максимальное время запаздывания каждой ПФ равно общему времени выполнения всех остальных функций, входящих в цикл.

Если же ПФ присвоены некоторые приоритеты, то максимальное время ожидания выполнения функции $T_{j\max}$ равно суммарному времени выполнения функций с приоритетами выше, чем у данной. При этом в случае, когда ограничения по $T_{j\max}$ ПФ не удовлетворяются, следует вернуться к решению задачи размещения с тем, чтобы минимизировать запаздывание функций в порядке их важности, то есть в порядке нарастания допустимого времени исполнения.

Запаздывание функции f_ι с приоритетом ι в худшем случае будет равно сумме времен исполнения функций с более высокими приоритетами $j < \iota$, то есть

$$\theta_\iota = \sum_{j=1}^{\iota-1} T_j, \qquad (2.18)$$

где T_j – время исполнения функции с приоритетом j.

Запаздывание функции f_1 с высшим приоритетом 1 равно 0.

На время запаздывания ПФ влияет и размещение операций, массивов и программ в станциях сети. Поэтому для сокращения запаздывания ПФ возможен пересмотр ранее полученного варианта решения задачи размещения.

В этом случае при решении задачи размещения сначала минимизируют запаздывание функции f_2, то есть время исполнения функции f_1. Для этого выбирают значения части переменных X^1, отражающих размещение связанных с функцией f_1 операций, массивов и программ. На каждом последующем шаге i минимизируют запаздывание функции f_{i+1}, то есть время выполнения функции f_i, путем определения переменных X^i размещения операций, массивов и программ, связанных с функцией f_i, при известных переменных X^1, X^2, ..., X^{i-1}. Если при этом часть операций, массивов и программ функции f_i и множества функций f_1, f_2, ..., f_{i-1} совпадает, то соответствующие переменные, определенные на предыдущих шагах, не изменяются. Время

выполнения одной функции рассчитывают с помощью представления функционального графа СРВ, в ярусно-параллельной форме и с учетом соответствующих допущений о поярусном выполнении операций обработки и передач данных по магистрали.

Если минимизация максимально возможного запаздывания функции не дает значения $T_{j\max}$ в пределах допустимого $\hat{T}_j$, то можно увеличить или уменьшить число станций, то есть повторить процедуру минимизации запаздывания, не ограничиваясь при этом определенным числом станций. Это осуществляется путем распределения операций, массивов и программ между произвольным числом станций. Если и в этом случае условия на запаздывание не выполняются, то необходимо изменить ограничения на характеристики выбираемой для СРВ аппаратуры обработки и передачи, то есть повысить пропускную способность магистрали или (и) быстродействие процессоров станций.

2.6. Оптимизация использования памяти при выполнении прикладных функций

Сокращение объема памяти, необходимой для выполнения прикладной функции, соответствует общему критерию минимизации ресурсов проектируемой МВС. Помимо данного аспекта рассматриваемая задача включает ***оптимизацию использования регистров процессора***, что в нашем случае является более важным, так как позволяет сократить время выполнения программы, реализующей прикладную функцию.

Объектом исследования является программа, написанная на каком-либо из языков программирования. Программа реализует алгоритм прикладной функции и представлена в виде ***графовой модели алгоритма*** (МГА) [18]. Вершинами МГА являются операторы программы, а дуги соответствуют управляющим и информационным связям между операторами. На МГА определено множество ***компонентов информации*** (КИ), формируемых в вершинах МГА, и КИ, которые являются исходными данными, но не должны храниться до конца выполнения алгоритма. Для каждой КИ известен размер памяти, требуемый для ее размещения в памяти процессора.

Заметим также, что МГА может иметь иерархическую структуру, предполагающую использование в качестве ее вершин отдельные подпрограммы (модули). Иерархическая структура МГА используется для описания больших программ и позволяет при этом сохранять размерность приемлемую для решения рассматриваемой задачи. Предполагается также, что для модулей, входящих в МГА, задача оптимизации

использования памяти может быть решена для каждого модуля автономно.

Основное содержание рассматриваемой задачи заключается в сопоставлении вершин, формирующих КИ, и вершин, после выполнения которых эти КИ становятся ненужными и освобождают память для размещения других КИ. Определение рациональной структуры взаимного совмещения КИ осуществляется исходя из условия сокращения общего объема памяти, затрачиваемой при выполнении проектируемой программы. Метод решения задачи включает выполнение следующих основных этапов [19]:

- построение для МГА графа управляющих и информационных связей;
- анализ информационной структуры МГА;
- построение матрицы парной несовместимости КИ;
- оптимизация структуры совмещения КИ в памяти ЭВМ.

Построение графа управляющих и информационных связей

Отображение МГА в ориентированный граф управляющих и информационных связей (граф **L**) осуществляется для выполнения этапа анализа информационной структуры МГА. Граф **L** характеризуется тем, что в каждой его вершине формируется не более одной КИ и из каждой вершины выходит не более двух управляющих дуг. Этому условию удовлетворяют лишь МГА, вершинами в которых являются только операторы. В общем случае в вершине МГА, описывающей модуль, может формироваться как одна, так и несколько КИ. Кроме того, модули могут иметь более двух управляющих выходов. Поэтому при построении графа модуль интерпретируется подграфом, удовлетворяющим указанному выше условию. Число вершин $n(M)$ подграфа, построенного для модуля M, определяется выражением

$$n(M) = I_M + (U_M - 1),$$

где I_M – число информационных выходов модуля M;

U_M – число управляющих выходов модуля M.

Построение подграфа осуществляется следующим образом. Вершины подграфа выстраиваются в цепь. В начале цепи располагаются вершины, соответствующие информационным выходам. Далее располагаются $(U_M - 1)$ вершин, каждая из которых при $U_M \geq 2$ имеет по два

управляющих выхода, один из которых соответствует одному из управляющих выходов модуля. Оба управляющих выхода последней вершины цепи соответствуют двум оставшимся управляющим выходам модуля. При $U_M = 1$ последняя вершина цепи имеет один управляющий выход. Все информационные входы модуля подаются на последнюю вершину цепи. Пример построения подграфа для модуля с тремя информационными выходами и тремя управляющими выходами представлен на рис. 2.13.

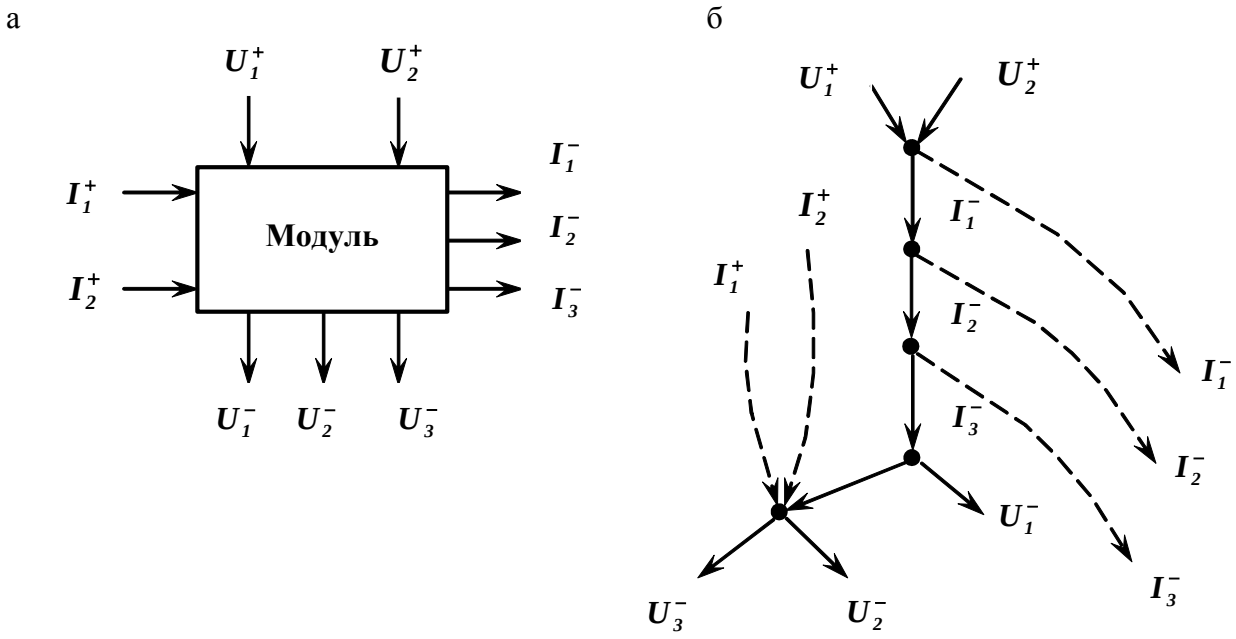


Рис. 2.13. Пример построения графа для модуля:
а – модуль, б – подграф

Таким образом, при построении графа управляющих и информационных связей каждый оператор отображается в одну вершину, а модули МГА – в подграфы, построенные по изложенному выше правилу. Совокупность КИ исходных данных условно принимается в качестве модуля МГА с одним управляющим выходом и информационными выходами, соответствующими данной совокупности КИ. Отображение такого модуля в подграф осуществляется аналогично другим модулям МГА. Граф $L = (X, U)$, в который отображается МГА в соответствии с изложенными выше правилами, является ориентированным мультиграфом без петель, и используемые в последующем понятия маршрута, цепи и цикла рассматриваются применительно к ориентированному графу. Множество вершин X графа L складывается из множества КИ исход-

ных данных, операторов в МГА и множества вершин подграфов, интерпретирующих модули МГА. Множество дуг V графа L включает множество управляющих дуг U и множество информационных дуг W. Таким образом, граф L получается в результате наложения двух графов – управляющего $L_U = (X, U)$ и информационного $L_W = (X, W)$. С учетом правил построения графа $L = (X, V)$ и свойств МГА в графе L существует одна начальная вершина $x_{i_1} \in X$, в которую не входит ни одна управляющая дуга, и конечная вершина $x_{in} \in X$, из которой не выходит ни одна управляющая дуга. Если МГА имеет несколько выходных вершин, то в граф L вводятся дополнительная искусственная вершина x_{i_n} и дополнительные управляющие дуги, которые связывают указанные вершины с вершиной x_{i_n}. На рис. 2.14 *а* представлен пример графа L, полученного для МГА с двумя КИ исходных данных и одним модулем. В графе L сплошными линиями показаны управляющие дуги, а пунктирами – информационные.

Вершины x_{29}, x_{30} соответствуют КИ исходных данных, вершины $x_{14} \div x_{17}$ модулю, а вершины $x_1 \div x_{13}$, $x_{18} \div x_{28}$ соответствуют операторам в МГА.

Анализ информационной структуры МГА

Анализ осуществляется на графе L, в который отображается рассматриваемая МГА, и выполняется с целью установления для каждой КИ совокупности вершин таких, что после прохождения любой из них при выполнении соответствующего алгоритма, информация данных КИ становится ненужной. Установление таких вершин не вызывает трудностей принципиального характера и может быть осуществлено следующим образом. На графе L последовательно формируются все маршруты с началом в вершине x_{i_1} и концом в вершине x_{i_n}. Далее анализируются все информационные дуги, входящие в вершины маршрута, и для них определяются вершины, после выполнения которых соответствующие КИ в алгоритме использоваться не будут. Такие вершины в последующем будем именовать вершинами затирания КИ. Сопоставляя в каждом маршруте положение вершин появления и затирания для соответствующих КИ, получим данные для построения матрицы парной несовместимости КИ, на основе которой осуществляется оптимизация структуры совмещения КИ в памяти ЭВМ.

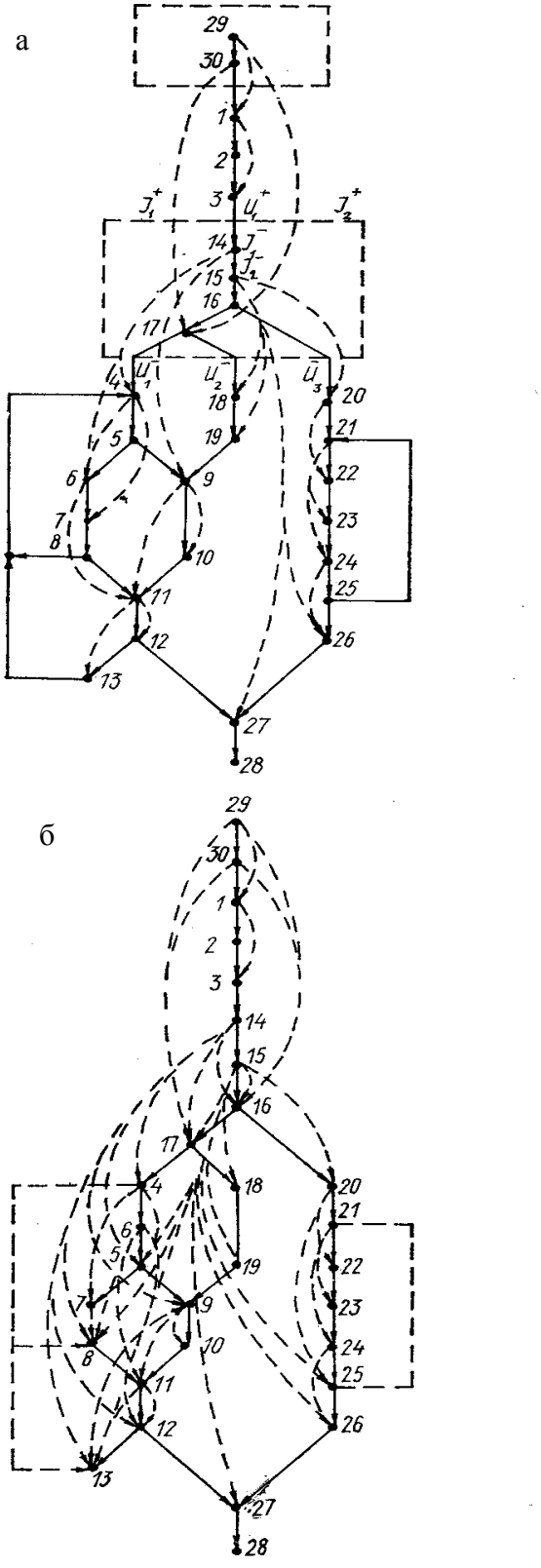


Рис. 2.14: а – граф $\boldsymbol{L}=\left(\boldsymbol{X},\boldsymbol{V}\right)$*;*

б – граф $\boldsymbol{L}'=\left(\boldsymbol{X},\boldsymbol{V}'\right)$*, построенный для графа* $\boldsymbol{L}$

Но такой метод анализа для графов, содержащих несколько сотен вершин, является неприемлемым, так как число маршрутов в этом случае становится недопустимо большим. Кроме того, процесс построения множества уникальных маршрутов связан с необходимостью выполнения сложных правил комбинаторного характера. Поэтому для выполнения этапа анализа предложен метод, который исключает необходимость построения маршрутов и основан на замене графа L новым графом L', в котором отсутствуют циклы, а каждая КИ представлена совокупностью информационных дуг, связывающих вершину появления и вершины затирания, для данной КИ. Построение графа L' достигается выполнением на графе L трех преобразований, представленных на рис. 2.15.

Первое преобразование (рис. 2.15 а) затрагивает КИ, которые используются в вершинах цикла, а формируются в вершинах, не принадлежащих данному циклу. Очевидно, что в этом случае КИ должна храниться в памяти процессора в течение выполнения всего цикла и может затираться лишь после выхода из него. Следовательно, одной из вершин затирания для рассматриваемых КИ является вершина выхода из цикла. Преобразование, таким образом, включает выполнение следующих операций: выявление в графе L циклов; установление для каждого из них вершин, соответствующих началу и концу цикла; введение дополнительных информационных дуг, входящих в вершину конца цикла.

Второе преобразование (рис.2.15 б) также связано с наличием в графе L циклов. Рассматривается ситуация, когда КИ формируется вершиной цикла, а используется вне его, причем в данном цикле имеется выход, который в последовательности вершин цикла располагается раньше, чем вершина, формирующая КИ. Такие выходы в отличие от выхода из вершины конца цикла именуются в последующем промежуточными выходами из цикла. В этом случае рассматриваемая КИ должна быть сохранена в течение выполнения вершин всего цикла. Это условие будет выполнено, если в качестве вершины появления данной КИ искусственно принять вершину начала цикла. Исходя из этого второе преобразование заключается в выполнении следующих операций: среди циклов, выявленных в первом преобразовании, выделяются циклы, имеющие промежуточные выходы; вершины, порождающие КИ, используемые вне цикла и расположенные после вершин с промежуточными выходами, переносятся в начало цикла и встраиваются в последовательную цепь после начальной вершины цикла; информационные дуги, входящие в переносимые вершины, подаются на вершины, предшествующие переносимым в последовательности вершин цикла.

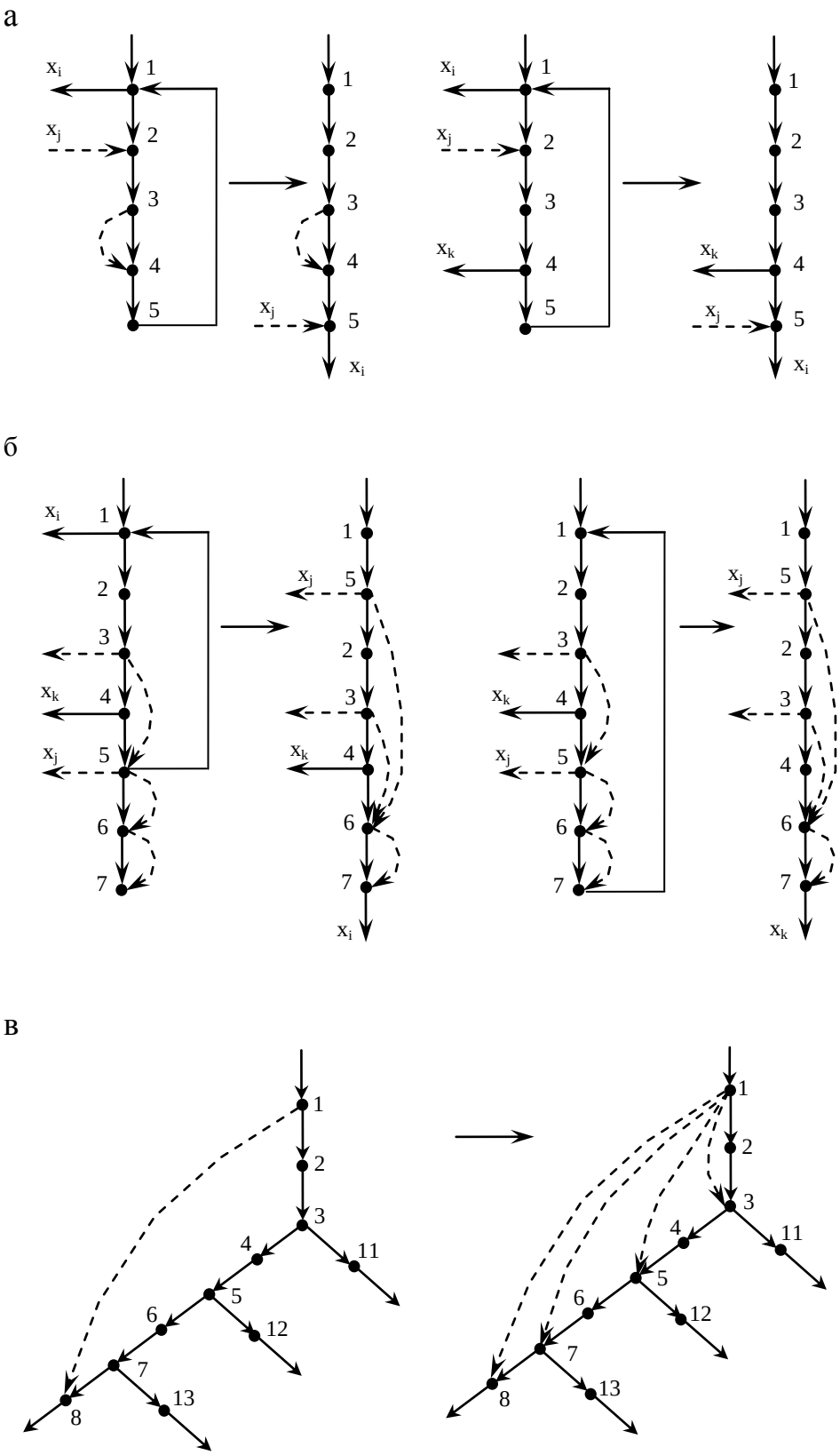


Рис. 2.15. Преобразования на графе ***L***

Третье преобразование (рис. 2.15 в) выполняется в графе L после того, как на нем выполнены первые два преобразования. В данном случае для каждой информационной дуги среди вершин всех цепей, построенных между вершинами начала и конца информационной дуги, проверяется наличие вершин с двумя управляющими выходами. Если такие вершины имеются, то вершина, из которой выходит информационная дуга, связывается информационными дугами с данными вершинами. Выполнение указанного преобразования, в результате которого вводятся дополнительные информационные дуги, позволяет отобразить на графе L продолжительность хранения соответствующей КИ для любых последовательностей прохождения вершин при выполнении алгоритма. Для рассматриваемого примера указанные последовательности соответствуют цепям, проходящим через вершину x_1 и любую из вершин x_{11}, x_{12}, x_{13}.

В графе, полученном после выполнения перечисленных преобразований, осуществляется разрыв циклов путем исключения в каждом из них управляющей дуги, связывающей вершины конца и начала цикла. Полученное при этом множество информационных дуг обозначим W', а множество оставшихся управляющих дуг V'. Таким образом, граф $L' = (X, V')$ получается наложением графов $L'_U = (X, U')$ и $L'_W = (X, W')$. Граф L', полученный для графа L (рис. 2.14 а), показан на рис. 2.14 б.

Построение матрицы парной несовместимости КИ

Граф L' имеет такую структуру информационных дуг, которая на любой цепи графа $L'_U = (X, U')$ отражает информацию о вершинах появления и затирания КИ, формируемых в вершинах цепи. На основе этой информации может быть построена матрица парной несовместимости КИ $Q = \|q_{ij}\|$, $i, j = 1, 2, ..., n$, где n – число вершин в графе L', порождающих КИ. Множество таких вершин обозначим K.

В основе метода построения матрицы Q лежит понятие доминирования вершин графа L'. Вершина $x_i \in K$ доминирует над вершиной $x_j \in K$, если в последовательности вершин одной из цепей графа L'_U, построенных из вершины x_{i_1}, вершина x_i располагается раньше вершины x_j. Информацию о доминировании вершин представим в виде матрицы доминирования $D = \|d_{ij}\|$, $i, j = 1, 2, ..., n$, $i \neq j$,

где n'' – число вершин, в которых заканчиваются информационные дуги, а в цепях после данных вершин нет других вершин, порождающих КИ. Множество таких вершин обозначим через K''. Элемент d_{ij} матрицы D равен единице, если вершина x_i доминирует над вершиной x_j, в противном случае $d_{ij} = 0$. Принимается также, что $d_{ii} = 0$.

Построение матрицы D осуществляется на основе так называемой матрицы локального доминирования $D^* = \|d^*_{ij}\|$, $i, j = 1, 2, \ldots, n'$. Матрица D^* строится следующим образом. Для каждой вершины $x_i \in K$ на графе L'_u строится цепь с началом в этой вершине. Цепь строится до первой вершины $x_j \in K'$, $K' = K \cup K''$. Если при построении цепи встречается вершина с двумя выходами, то построение цепей продолжается по каждому выходу, пока не встретятся вершины из множества K. В соответствии с определением доминирования принимается, что вершина x_i доминирует над вершиной x_j. Так, например, для графа L' (рис. 2.14 б) из вершины x_6 построение цепей продолжается до вершин x_{11} и x_9, а для вершины x_{20} до вершины x_{21}. Следовательно, вершина x_6 доминирует над вершинами x_{11}, x_9, а вершина x_{20} над вершиной x_{21}. Соответствующие элементы $d^*_{6,11}$, $d^*_{6,9}$ и $d^*_{20,21}$ принимаются равными единице и заносятся в матрицу D^*.

Элементы d_{ik} матрицы D доопределяются на основе элементов d^*_{ij} с помощью следующего правила: если вершина x_i доминирует над вершиной x_j, а x_j доминирует над x_k, то вершина x_i доминирует над вершиной x_k (в матрице D, построенной для графа L' (рис. 2.14 б), элементы d^*_{ij} матрицы D^* отмечены звездочкой).

Заметим, что в построенной таким образом матрице D фигурируют вершины множества K'. Остальные вершины $X \setminus K'$ не учитываются, так как информация об их доминировании для построения матрицы Q оказывается избыточной. Для наглядности номера строк и столбцов в матрице D соответствуют номерам вершин множества K'.

Исключение из рассмотрения вершин $X \setminus K'$ в значительной степени сокращает объем вычислений. Это приводит к необходимости корректировки информационных дуг графа L', которая возникает вследствие того, что концами информационных дуг могут быть и вершины $x_i \notin K'$. Поэтому каждая такая информационная дуга должна

Матрица доминирования ($\boldsymbol{D}$)

	$x_i \backslash x_j$	29	30	1	14	15	4	6	11	9	20	21	24	12	13	26	27	25
n	29	0	1*	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	30	0	0	1*	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	1	0	0	0	1*	1	1	1	1	1	1	1	1	1	1	1	1	1
	14	0	0	0	0	1*	1	1	1	1	1	1	1	1	1	1	1	1
	15	0	0	0	0	0	1*	1	1	1*	1*	1	1	1	1	1	1	1
	4	0	0	0	0	0	0	1*	1	1	0	0	0	1	1	0	1	0
	6	0	0	0	0	0	0	0	1*	1*	0	0	0	1	1	0	1	0
	11	0	0	0	0	0	0	0	0	0	0	0	0	1*	1*	0	1	0
	9	0	0	0	0	0	0	0	1*	0	0	0	0	1	1	0	1	0
	20	0	0	0	0	0	0	0	0	0	0	1*	0	0	0	1	1	1
	21	0	0	0	0	0	0	0	0	0	0	0	1*	0	0	1	1	1
	24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1*
n"	12	0	0	0	0	0	0	0	0	0	0	0	0	0	1*	0	1*	0
	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1*	0
	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1*	1	0

быть заменена на одну или несколько дуг, концами в которых являются вершины множества $\boldsymbol{K'}$. Операция замены осуществляется аналогично операции определения локального доминирования с той лишь разницей, что вместо вершин, порождающих КИ, рассматриваются вершины, в которые входят информационные дуги соответствующих КИ. Так, при замене дуги $\varpi_{ij} \in \boldsymbol{W'}$ независимо от того, принадлежит вершина $\boldsymbol{x_j}$ к множеству $\boldsymbol{K}$ или нет, цепи строятся из вершины $\boldsymbol{x_i}$ до встречи с первой вершиной $\boldsymbol{x_k} \in \boldsymbol{K'}$. Например, дуга $\varpi_{4,7}$ (рис. 2.14 б) заменяется на дуги $\varpi_{4,1}, \varpi_{1,4}$. Информационные дуги с концами в вершинах из множества $\boldsymbol{K''}$ корректировке не подлежат. Полученный таким образом список информационных дуг оформляется в виде матрицы $\boldsymbol{W} = \|\varpi_{ij}\|, \quad \boldsymbol{i} = 1,2,...,\boldsymbol{n}, \quad \boldsymbol{j} = 1,2,...,\boldsymbol{n'}$.

Матрица парной несовместимости КИ $\boldsymbol{Q} = \|\boldsymbol{q_{ij}}\|, \quad \boldsymbol{i, j} = 1,2,...,\boldsymbol{n}$ строится на основе матриц $\boldsymbol{W}$ и $\boldsymbol{D}$. Элемент $\boldsymbol{q_{ij}}$ матрицы $\boldsymbol{Q}$ определяется по следующему выражению:

$$\boldsymbol{q_{ij}} = \begin{cases} 1, & \text{если} \quad \mathrm{d_{ij}}(\varpi_{i1}\boldsymbol{d_{j1}} + \varpi_{i2}\boldsymbol{d_{j2}} + ... + \varpi_{in'}\boldsymbol{d_{jn'}}) > 0, \\ 0 & \text{в противном случае.} \end{cases}$$

Матрица информационных дуг ($\boldsymbol{W}$)

x_j / x_i	29	30	1	14	15	4	6	11	9	20	21	24	12	13	26	27	25
29	0	0	0	1	0	1	0	0	1	1	0	0	0	0	0	0	0
30	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	1	1	1	1	1	0	0	1	1	0	0	0
15	0	0	0	0	0	1	0	1	1	1	1	0	1	1	1	1	1
4	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
9	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
21	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

Матрица парной несовместимости ($\boldsymbol{Q}$)

x_j / x_i	29	30	1	14	15	4	6	11	9	20	21	24	r(x)
29	1	1	1	1	1	1	1	0	0	0	0	0	11
30	1	1	1	1	1	1	1	0	0	0	0	0	20
1	1	1	1	0	0	0	0	0	0	0	0	0	20
14	1	1	0	1	1	1	1	1	1	0	0	0	18
15	1	1	0	1	1	1	1	1	1	1	1	1	12
4	1	1	0	1	1	1	1	0	1	0	0	0	9
6	1	1	0	1	1	1	1	1	1	0	0	0	7
11	0	0	0	1	1	0	1	1	1	0	0	0	7
9	0	0	0	1	1	1	1	1	1	0	0	0	18
20	0	0	0	0	1	0	0	0	0	1	1	1	13
21	0	0	0	0	1	0	0	0	0	1	1	1	10
24	0	0	0	0	1	0	0	0	0	1	1	1	25
	7	7	3	8	11	7	8	5	6	4	4	4	170

Диагональные элементы $\boldsymbol{q_{ii}}$ в матрице $\boldsymbol{Q}$ принимаются равными единице. Нетрудно видеть, что матрица $\boldsymbol{Q}$ является симметричной поэтому после определения всех элементов $\boldsymbol{q_{ij}}$ над диагональю матрица $\boldsymbol{Q}$ приводится к симметричной. Строкам матрицы $\boldsymbol{Q}$ приписаны объемы памяти $\boldsymbol{r(x_i)}$, необходимые для размещения соответствующих КИ. Внизу для каждого столбца приведено число нулевых элементов $\boldsymbol{q_{ij}}$.

В общем случае матрица парной несовместимости $\boldsymbol{Q}$ включает КИ размером в одно слово и КИ, требующие для размещения объем па-

мяти $r(x_i)$ больше одного слова. Таким образом матрица Q может быть разбита на две матрицы Q_1 и Q_r. Матрица Q_1 описывает парную несовместимость единичных КИ, а матрица Q_r описывает парную несовместимость КИ с $r(x_i)$ больше одного слова. Для КИ, составляющих матрицу Q_1, решается задача оптимальной загрузки регистров, а для КИ матрицы Q_r задача оптимизации структуры совмещения КИ.

Оптимизация загрузки регистров

Матрицу Q_1 можно интерпритировать как матрицу смежности графа $G(X,U)$ с множеством вершин X, равным совокупности единичных КИ, и множеством звеньев U, соответствующим единичным значениям элементов q_{ij} матрицы Q_1. Тогда задача оптимальной загрузки регистров сводится к задаче раскраски вершин графа G минимальной совокупностью цветов [20]. При этом каждый цвет соотносится с определенным регистром, в котором могут храниться вершины (соответствующие КИ), окрашенные данным цветом.

Если число цветов превышает число доступных регистров, то выбираются вершины для хранения в ОЗУ. Выбор вершин для размешения в ОЗУ может производиться, в частности, по критерию минимальной частоты обращения к соответствующим КИ при выполнении программы.

Оптимизация структуры совмещения КИ

Данная задача определяет такое размещение КИ в ОЗУ процессора, которое, будучи получено из условия сокращения занимаемого объема памяти, вместе с тем не противоречит матрице Q_r, то есть является допустимым. Алгоритм оптимизации оперирует с матрицей Q и значениями $r(x_i)$ для $x_i \in K$. На первой итерации алгоритма формируется совокупность вершин $A_1 \subseteq K$, которые соответствуют совместимым КИ. Первой в A_1 включается вершина x_{i_1}, которой соответствует величина $\max_i \sum_{j+1}^{n} q_{ij}$. Далее среди вершин, совместимых с вершинами $x_i \in A_1$, выбирается вершина x_{i_2} с максимальной суммой элементов q_{ij}. Последующее расширение совокупности A_1 осуществляется аналогично и продолжается до тех пор, пока возможность расширения имеется.

КИ, соответствующие вершинам совокупности A_1, размещаются в памяти ЭВМ, начиная с относительного адреса r_0^*, равного единице. Далее определяется величина

$$r_1 = \min_{x_i \in A_1} r(x_i),$$

которая принимается в качестве нового значения относительного адреса r_1^*, $r_1^* = r_1$. Вершины $x_i \in A_1$, для которых $r(x_i) = r_1$, исключаются из дальнейшего рассмотрения и осуществляется корректировка: $\forall x_i \in A_1 \left[r(x_i) := r(x_i) - r_1 \right]$. На оставшемся после исключения множестве вершин выполняется вторая итерация алгоритма по формированию совокупности A_2.

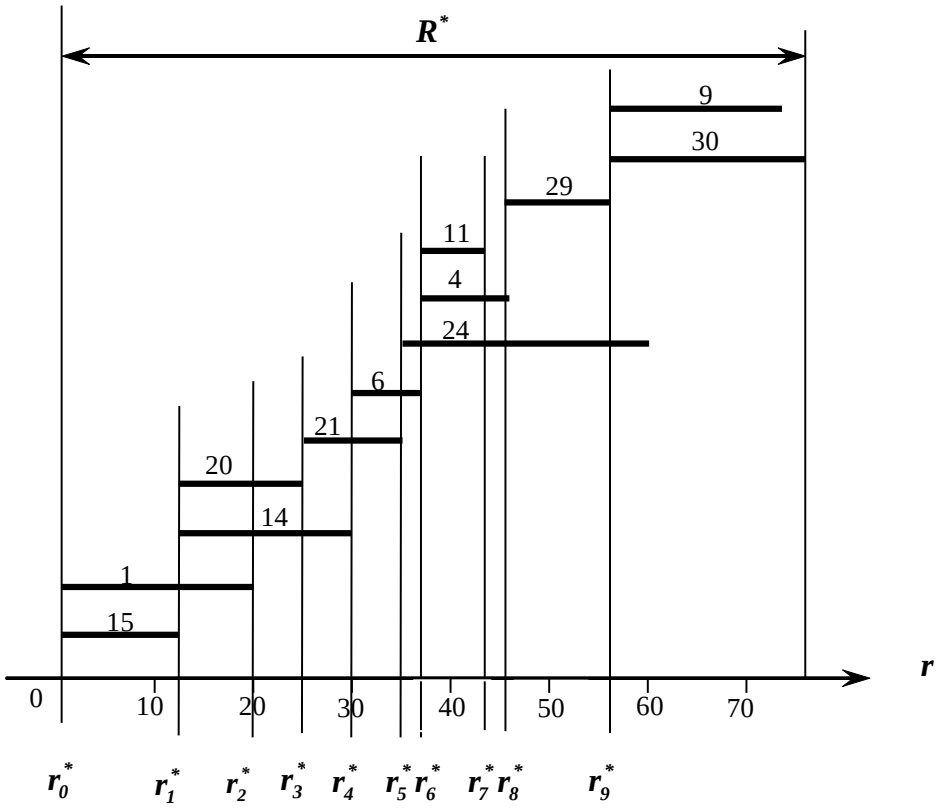


Рис. 2.16. Вариант размещения КИ

Формирование A_2 начинается с включения в нее вершин, оставшихся в совокупности A_1. После этого осуществляется расширение совокупности A_2 аналогично тому, как это делалось в первой итерации.

КИ вершин, включенных в совокупность A_2 при ее расширении, размещаются, начиная с относительного адреса r_1^*. Новое значение относительного адреса переносится в точку $r_1^* = r_1 + r_2$, $r_2 = \min_{x_i \in A_2} r(x_i)$. Корректировка величин $r(x_i)$ для $x_i \in A_2$ осуществляется на величину r_2.

Последующие итерации по формированию совокупностей A_3, A_4, ... и размещению соответствующих КИ в памяти ЭВМ выполняются аналогично. Алгоритм заканчивает работу, когда для всех КИ будут определены относительные адреса. Вариант размещения КИ, полученный в соответствии с изложенным выше алгоритмом для матрицы Q, изображен на рис. 2.16.

Оценка качества решения задачи распределения памяти

Качество полученного варианта решения задачи распределения памяти можно оценить следующим образом. Матрицу Q_r можно рассматривать как матрицу связности некоторого обыкновенного графа H, построенного на множестве вершин K. В графе H выделим множество всех максимальных полных подграфов $\{H_k\}$, $k = 1, 2, ..., p$, где p – множество выделенных подграфов [21]. Для каждого подграфа H_k определим величину R_k:

$$R_k = \sum_{x_i \in S_k} r(x_i),$$

где S_k – множество вершин в графе H_k. Тогда величина $R^* = \max_k \{R_k\}$ является минимально возможным объемом памяти, необходимым для размещения КИ, соответствующих вершинам множества K. Действительно, КИ вершин каждого из подграфов H_k могут быть размещены в памяти ЭВМ только последовательно. Поэтому при размещении всех других КИ с учетом их возможных совмещений величина R^* может либо быть увеличена, либо остаться неизменной. Для рассматриваемого примера максимальный полный подграф, соответствующий величине R^*, включает вершины $\{x_{15}, x_{14}, x_6, x_4, x_{29}, x_{30}\}$. Как видно из рис. 2.16 все остальные КИ удалось разместить без увеличения R^*, то есть в данном случае полученное решение является оптимальным.

Вопросы для контроля усвоения знаний

1. Перечислить состав основных компонентов СРВ подлежащих проектированию.
2. Дать определение математических моделей и задач анализа.
3. Раскрыть содержание задачи идентификации.
4. Дать формальную постановку задачи синтеза СРВ. Решение задач синтеза через задачу анализа.
5. Дать характеристику известных типов моделей и их применения при проектировании СРВ.
6. Дать определение моделей Керка и процессов, описывающих программную нагрузку СРВ.
7. Дать определение каналов в сетях Керка и их типов.
8. Объяснить назначение функции канала и привести примеры ее использования для различных типов каналов.
9. Рассмотреть пример применения модели Керка для регулятора цифрового управления.
10. Пояснить взаимосвязь вариантов временных диаграмм работы регулятора и вариантов архитектуры ВС.
11. Раскрыть содержание основных шагов алгоритма анализа временных характеристик на сетях Керка.
12. На примере работы программного средства «Сети Керка» пояснить типы каналов, тупиковые ситуации, задержки данных в каналах, пропускную способность каналов, временную диаграмму.
13. Раскрыть суть постановки задачи определения числа станций как задачи линейного математического программирования.
14. Перечислить и дать содержательную характеристику критериям решения задачи распределения операций, массивов и программ.
15. Перечислить совокупность переменных в задаче распределения.
16. Перечислить типы сетей, используемых в распределенных СРВ.
17. Пояснить выражение для расчета потока данных, передаваемых станцией, в сети типа (а).
18. Пояснить выражение для расчета данных между станциями в сети типа (б).
19. Пояснить критерий минимума загрузки сети для сетей типа (а) и типа (б).
20. Перечислить и дать пояснения ограничений в задаче распределения операций, массивов и программ.
21. Раскрыть содержание метода решения задачи распределения операций, массивов и программ для сети типа (в).

22. Пояснить критерий минимума загрузки запоминающих устройств при решении задачи распределения.
23. Пояснить критерий равномерной загрузки процессоров при решении задачи распределения.
24. Раскрыть суть метода линеаризации для определенного вида задач нелинейного математического программирования.
25. Раскрыть связь между числом станций, вариантом распределения операций массивов и программ и временем выполнения ПФ.
26. Дать определение ярусно-параллельной формы представления функционального графа.
27. Пояснить цикл выполнения ПФ на временной диаграмме.
28. Почему вводится оценочная величина цикла выполнения ПФ, и как она рассчитывается?
29. Пояснить выражение для расчета длительности цикла выполнения ПФ.
30. Дать постановку задачи минимизации времени выполнения цикла ПФ.
31. Раскрыть содержание методики сокращения запаздывания непериодических ПФ.
32. Пояснить связь между оптимизацией загрузки регистров программы и временем ее выполнения.
33. Раскрыть методику построения графа управляющих и информационных связей для графовой модели алгоритма.
34. Каким свойством обладает граф $\boldsymbol{L'}$, полученный в результате преобразования графа $\boldsymbol{L}$?
35. Пояснить суть первого преобразования графа $\boldsymbol{L}$.
36. Пояснить суть второго преобразования графа $\boldsymbol{L}$.
37. Пояснить суть третьего преобразования графа $\boldsymbol{L}$.
38. Дать определение матрицы парной несовместимости КИ.
39. Дать определение понятия доминирования вершин в графе $\boldsymbol{L'}$ и раскрыть методику построения матрицы доминирования.
40. Дать определение матрицы информационных дуг графа $\boldsymbol{L'}$.
41. Пояснить выражение для определения элементов матрицы парной несовместимости.
42. Раскрыть содержание методики оптимизации совмещения КИ.
43. Как оценивается качество решения задачи оптимизации использования памяти?
44. К какой задаче теории графов сводится задача оптимизации загрузки регистров?

3. ТЕХНОЛОГИЯ МОДУЛЬНОГО ПРОЕКТИРОВАНИЯ. ОБЩИЕ ПОЛОЖЕНИЯ

3.1. Принципы модульного проектирования

Модульное проектирование СРВ отражает определенную технологию проектирования, в соответствии с которой любая часть системы синтезируется из более мелких фрагментов (модулей). Модульное проектирование хорошо согласуется с ***эволюционным подходом***, при котором приходится строить и анализировать совокупность последовательно улучшаемых моделей СРВ. При проектировании сложных систем цепь последовательных улучшений моделей может оказаться достаточно длинной. В этих условиях использование традиционных схем [5] имитационного моделирования становится проблематичным, так как затраты на разработку одного варианта имитационной модели часто оказываются соизмеримыми с затратами на разработку программного обеспечения проектируемой системы. Поэтому для успешной реализации эволюционного подхода, использующего имитационные модели, необходимо дальнейшее развитие модульной технологии проектирования и повышение ее эффективности, в первую очередь, на стадиях построения и трансформации моделей[22].

Принципы. Исследования показали, что значительного продвижения при решении задач, возникающих в процессе последовательной трансформации моделей СРВ при эволюционном подходе к проектированию, можно достигнуть, если модельное представление СРВ будет базироваться на двух основополагающих принципах [23]:

- дискретность модели;
- графовая форма представления модели.

Принцип дискретности позволяет уйти от традиционных текстовых форм представления алгоритмов, описывающих прикладные функции СРВ и архитектуры МВС, а задачи трансформации таких моделей свести к задачам композиции дискретных элементов. Графовая форма в этом случае является естественной формой отображения дискретных элементов (модулей) и отношений между ними в представлении алгоритмов. Кроме того, полученная таким образом графовая модель алгоритма удобна для внесения в модель любых трансформаций ручным путем. Если модули графовой модели к тому же имеют иерархическую структуру, то в этом случае появляется возможность легко изменять уровень детализации описания модели СРВ. Представления алгоритмов в соответствии с данными принципами получили название структурно-

графических представлений [23]. Подробнее описание данной формы представления моделей будет изложено в главе 4.

Структурно-графические представления. Среди различных форм структурно-графических представлений наиболее удобными для построения моделей являются графы потоков данных (ГПД) [25]. В ГПД два вида вершин – фрагменты алгоритмов и данные. Дуги в графе связывают фрагменты и данные, указывают на отношение фрагментов к потреблению и формированию тех или иных данных.

При визуальном представлении ГПД фрагменты по аналогии с сетями Петри изображаются планками, а данные кружками. Планками в ГПД могут являться программные имитаторы алгоритмов или их фрагментов, либо непосредственно программы, реализующие данные фрагменты и алгоритмы. Кроме того, это могут быть программные имитаторы отдельных компонентов архитектуры МВС, а также программные имитаторы отдельных компонентов окружающей системы. В общем случае фрагменты алгоритмов и программ или их имитаторы, соответствующие планкам ГПД, будем именовать модулями.

Заметим также, что методы построения и анализа моделей СРВ не зависят от содержания модулей ГПД. Более того, в одной модели могут использоваться модули разных типов.

Технология модульного проектирования широко применяется при построении и анализе модульных структур радиоэлектронной аппаратуры [24], информационно-телекоммуникационных систем различного назначения [25], систем управления техническим оборудованием, техническими системами, подвижными объектами [26], систем связи, измерительных систем, систем учета в сетях энергоресурсов, систем мониторинга окружающей среды, систем гидрометеослужбы, гибких автоматизированных производственных систем и многих других приложений, в которых проектируемая система может быть представлена совокупностью взаимосвязанных и совместно функционирующих модулей.

Центральное место в этой технологии отводится определению модулей или их наборов, из которых строится система. Совокупность связанных между собой модулей образует модульную структуру. Для проектирования важным является то, что путем несложных операций модульные структуры легко трансформируются, что позволяет вести поиск рациональных в некотором смысле модульных структур.

3.2. Основные свойства модульных структур

Поиск путей повышения производительности труда при проектировании сложных динамических систем приводит нас, в частности, к попытке конструировать такие системы из уже имеющихся или специально создаваемых достаточно крупных модулей. При этом естественным является стремление оперировать как можно более крупными модулями и по возможности меньшим числом их видов. Эти два требования являются противоречивыми, так как сокращение числа видов модулей легче достигается при уменьшении мощности модулей и, напротив, увеличение мощности модулей предопределяет рост числа их видов. Это противоречие во многом удается преодолеть за счет введения иерархической структуры модулей.

Иерархия модулей предполагает наличие набора базовых модулей и модулей других более высоких рангов. Каждый не базовый модуль в данной иерархии представляет собой композицию из модулей более низких рангов, в том числе и базовых. Наличие такой иерархии позволяет применять более крупные модули при построении графовых моделей, имеющих относительно высокую степень регулярности и, напротив, для нерегулярных графовых моделей могут использоваться модули более низких рангов или базовые модули. Очевидно, что говорить об ограниченном наборе типов модулей на каждом уровне иерархии можно относительно некоторого класса графовых моделей, описывающих заданный класс проектируемых систем. Класс систем характеризуется общностью принципов организации и функционального назначения, представления обрабатываемой информации и совокупностью выполняемых преобразований над этой информацией.

Размер имеющейся выборки графовых моделей в значительной степени определяет подход к решению задач модульной структуризации. Так, при наличии малых выборок методы модульной структуризации должны в большей степени учитывать характеристики графовых моделей данной выборки. Напротив, при наличии представительной выборки целесообразнее формировать набор модулей с ориентацией использования его в графовых моделях всего класса систем, представляемого имеющейся выборкой [24].

Конструктивная и функциональная избыточность модулей

Приведем ряд основных конструктивных параметров и технических характеристик модулей. Мощность модуля i-го ранга будем определять как число содержащихся в нем модулей (i-1)-го ранга. При этом

необходимо различать номинальную мощность модулей i-го ранга p_i^0 и установленную мощность j-го модуля i-го ранга p_{ij}. Номинальная мощность указывает на максимально возможное число модулей (i-1)-го ранга, которые можно установить (включить) в модуль i-го ранга. Отклонение установленной мощности от номинальной характеризует заполнение j-го модуля i-го ранга, которое можно оценить коэффициентом конструктивной избыточности j-го модуля i-го ранга K_{ij}^k,

$$K_{ij}^k = (p_i^0 - p_{ij}) / p_i^0 .$$

В случае, если часть установленной мощности не задействована, то имеет место функциональная избыточность, которая может быть оценена коэффициентом функциональной избыточности j-го модуля i-го ранга K_{ij}^ϕ:

$$K_{ij}^\phi = (p_{ij} - p'_{ij}) / p_i^0 ,$$

где p'_{ij} – число задействованных модулей (i-1)-го ранга в j-м модуле i-го ранга.

Конструктивная избыточность для выборки графовых моделей, выраженная в модулях первого ранга, может быть определена коэффициентом K_k:

$$K_k = \frac{1}{N_\xi P_\xi} \sum_{i=2}^{\xi} \sum_{j=1}^{N_i} (p_{i-1}^0 - p_{i-1,j}) P_{i-1} ,$$

где N_i, $i = 2,3,...,\xi$ – общее число модулей i-го ранга в анализируемой выборке графовых моделей;

P_i, $i = 2,3,...,\xi$ – мощность модулей i-го ранга, выраженная в модулях первого ранга,

$$P_1 = 1, \quad P_2 = p_1^0, \quad P_3 = p_1^0 \cdot p_2^0, \quad ..., \quad P_\xi = p_1^0 \cdot p_2^0 \cdot ... \cdot p_{\xi-1}^0 .$$

Функциональная избыточность для выборки графовых моделей, выраженная в модулях первого ранга, определяется коэффициентом K_ϕ:

$$K_\phi = \frac{1}{N_\xi P_\xi} \sum_{i=2}^{\xi} \sum_{j=1}^{N_i} (p_{i-1,j} - p'_{i-1,j}) P_{i-1} .$$

Унификация и универсализация модулей

Процесс унификации и универсализации модулей в общем случае приводит к увеличению числа межмодульных связей, так как повыше-

ние уровня унификации и универсализации может быть достигнуто в том числе и за счет автономного включения в модуль нескольких совокупностей модулей более низкого ранга. Уменьшение у модулей допустимого числа внешних выводов f_i^0, $i = 2,3,...,\xi$ автоматически приводит к росту конструктивной избыточности и снижению возможностей унификации и универсализации. Уровень связности между модулями i-го ранга можно оценить коэффициентом связности K_i^c, который определяется отношением числа связей между модулями i-го ранга Z_i к числу связей между модулями (i-1)-го ранга Z_{i-1}. Коэффициент K_i^c характеризует, таким образом, относительные плотности внутримодульных связей модулей двух соседних рангов. Эти плотности тесно взаимосвязаны, так как увеличение одной из них приводит к уменьшению другой.

Необходимо заметить, что совместное решение задач модульной структуризации, включающих задачу декомпозиции графовых моделей на минимально связанные модули, и задачи определения рационального числа рангов модулей и их номинальной мощности на каждом ранге приводит к большим трудностям. К тому же зависимость коэффициентов связности от числа рангов и номинальной мощности модулей является немонотонной. Поэтому в подавляющем большинстве случаев решение даже более простой задачи модульной структуризации по критерию минимума числа межмодульных связей выполняется в условиях, когда мощность модулей и допустимое число внешних выводов заданы [24,27].

Среди основных параметров модулей можно также выделить их повторяемость или частоту использования в анализируемой выборке графовых моделей с допустимой избыточностью. Повторяемость модулей v-го вида i-го ранга определим коэффициентом повторяемости K_{iv} как отношение частоты использования модулей v-го вида i-го ранга в анализируемой выборке графовых моделей с допустимым значением функциональной избыточности K_ϕ к общему числу модулей на i-м ранге N_i.

Универсальность модуля можно оценить количеством различных функций, на которые может быть настроена структура модуля при выполнении условий на допустимую функциональную избыточность. При этом предполагается, что универсальные модули могут быть построены на основе одной или нескольких автономных универсальных структур. Очевидно, что при увеличении числа сравнительно мелких автономно включаемых в модуль структур, универсальность модуля увеличивает-

ся. При этом может ухудшиться значение конструктивной избыточности за счет наличия ограничения на число внешних выводов модулей. В качестве оценки уровня универсальности набора модулей введем коэффициент универсальности K_s,

$$K_s = \left(1/\sum_{i=1}^{\xi} H_i \right) \sum_{i=1}^{\xi} \sum_{v=1}^{H_i} \xi_{iv},$$

где ξ_{iv} – число вариантов применения модуля v-го вида i-го ранга;

H_i – число видов модулей i-го ранга.

Уровень регулярности и унификации графовых моделей. Применительно к графовым моделям анализируемой выборки может быть введен параметр, характеризующий уровень их регулярности. Значение уровня регулярности можно оценить коэффициентом K_s^p, который получается при условии, что анализ графовых моделей выборки осуществляется путем декомпозиции их на структуры, содержащие по $\gamma = 2, 3, ..., p$ вершин. Величина коэффициента K_s^p определяется по выражению

$$K_s^p = \frac{1}{p-1} \sum_{\gamma=2}^{p} (N_\gamma^0 - N_\gamma^*) / N_\gamma^0,$$

где N_γ^0 – общее число структур, содержащих по γ вершин, требуемых для покрытия всех вершин графовых моделей выборки;

N_γ^* – минимальное число видов структур, содержащих по γ вершин, требуемых для покрытия всех вершин графовых моделей выборки.

Коэффициентом K_s^y, определяющим относительное содержание повторяющихся структур выявленных в результате анализа выборки графовых моделей S, можно характеризовать уровень унификации, достигнутый для данной выборки,

$$K_s^y = \frac{1}{N_1} \sum_{\nu=1}^{S_\nu} d_\nu,$$

где N_1 – число модулей первого ранга в выборке S;

d_ν – суммарная мощность повторяющихся структур ν-го вида в выборке S, $\nu = 1, 2, ..., S_\nu$;

S_ν – число видов повторяющихся структур в выборке S.

Коэффициенты уровня регулярности и функциональной унификации являются важными характеристиками выборки схем и наряду со

значениями номенклатуры повторяющихся структур позволяют прогнозировать качество набора модулей до его проектирования.

3.3. Активные модели

Модульной структурой может быть описана любая динамическая система, состоящая из отдельных компонентов, функционирующих параллельно и так, что результат работы одного компонента оказывает влияние на функционирование другого. При этом взаимное влияние должно быть представимо передачей сообщений (данных). В этих условиях можно строить модульную структуру в виде графовой модели независимо от физической природы компонентов системы. Для этого необходимо лишь описать алгоритм функционирования соответствующего компонента и представить модулем. Физическая природа функционирования компонента учитывается при разработке алгоритма его имитации. Влияние результата срабатывания компонента на функционирование других компонентов должно отражаться через входные данные и условия запуска соответствующих модулей.

Построенная таким образом модульная структура состоит из модулей, имитирующих функционирование соответствующих компонентов системы, входные данные и данные, определяющие условия запуска модулей. Такая модель может быть представлена в форме ГПД, которая в явном виде отражает информационное взаимодействие модулей. Непосредственно на ГПД имеется возможность имитировать работу прикладных функций СРВ без учета динамики протекания процессов.

Имитировать функционирование СРВ в динамике с организацией параллельной работы прикладных функций возможно двумя путями. В первом из них необходимо разработать имитирующую программу, которая бы учитывала совместную работу всех компонентов встроенной и окружающей систем по реализации прикладных функций проектируемой СРВ. Этот подход нацелен на создание макетного образца встроенной и окружающей систем и отработки на нем прикладных функций и проектных решений по поиску приемлемого варианта проектируемой системы. Это традиционная схема имитационного моделирования, использование которой в данном случае ограничено большой трудоемкостью.

Второй путь нацелен на разработку методов, которые на основе условий динамики функционирования проектируемой СРВ преобразуют соответствующий ГПД в модель, способную самостоятельно запускаться и работать в виде программы и осуществлять комплексную имитацию работы СРВ в динамике. Такие модели названы ***активными***.

Понятие ***активных моделей*** введено автором как дальнейшее развитие технологии модульного проектирования и определяет ***третий принцип модельного представления систем*** в этой технологии. Согласно данному принципу активность модели определяет такое описание функционирования системы, которое может использоваться в роли программы для специально разработанной ***программной виртуальной машины*** (ПВМ). Другими словами, активная модель – это модель, обладающая свойствами программы для выполнения на ПВМ. Свойство активности модель системы приобретает после того, как ее наделяют способностью выполняться на ПВМ. Активная модель составляется таким образом, чтобы правила ее запуска и выполнения на ПВМ соответствовали функционированию проектируемой системы в модельном представлении.

Таким образом, использование ***принципов дискретности, графовой формы представления и активности*** составляют основу технологии модульного проектирования и позволяет реализовать эволюционный подход к проектированию.

Общая характеристика частных моделей

При обсуждении проблем моделирования на основе активных моделей исследованию подлежат частные модели основных компонентов встроенной и окружающей систем. Поэтому процесс построения активной модели для комплексного моделирования системы включает получение следующих частных моделей:

- модель представления топологии окружающей системы;
- граф потока данных алгоритма функционирования окружающей системы;
- граф потока данных алгоритмов прикладных функций, выполняемых встроенной системой;
- модель архитектуры вычислительных средств встроенной системы.

Модель топологии окружающей системы помимо визуального представления структуры компонентов содержит информацию для принятия решений о размещении терминальных точек для встроенной системы – датчиков и исполнительных механизмов. Данная модель строится визуальным методом путем композиции компонентов из библиотеки базовых компонентов, которая ориентирована на конкретную предметную область создаваемых систем [28].

Модель алгоритма функционирования окружающей системы строится в форме ГПД. Алгоритм имитирует совместную работу ком-

понентов системы и их взаимодействие. Результатом работы алгоритма является информация о динамике состояний отдельных компонентов и системы в целом. На основе этой информации формируются потоки информации с датчиков для встроенной системы [29].

Алгоритмы прикладных функций составляют основную часть программной нагрузки на встроенную систему. На начальном этапе моделирования при представлении программной нагрузки в виде ГПД в качестве модулей принимаются локальные алгоритмы прикладных функций. Функционирование ГПД программной нагрузки и ГПД алгоритма имитации работы окружающей системы должно осуществляться совместно. Если потоки информации с датчиков имитируются специальными средствами, то ГПД программной нагрузки может функционировать автономно. Информацию с датчиков можно накапливать также при автономном функционировании ГПД окружающей системы на определенном интервале времени. Полученная информация может быть использована при автономной работе ГПД программной нагрузки [30].

Модель архитектуры встроенной многопроцессорной вычислительной системы (МВС) строится в условиях, когда для выполнения программной нагрузки требуется более одного процессора. В этом случае МВС создается на основе совокупности станций, объединенных в локальную сеть. По аналогии с моделью окружающей системы используется визуальный метод представления архитектуры МВС. При построении модели МВС используется библиотека базовых компонентов архитектур и библиотека базовых архитектур МВС [31].

Состав активной модели

Активная модель системы разрабатывается на основе перечисленных выше частных моделей [32]. Основу активной модели составляют ГПД алгоритма функционирования окружающей системы и ГПД прикладных функций встроенной системы. Модели топологии окружающей системы и архитектуры МВС используются при построении активных моделей для решения задач настройки соответствующих ГПД. Кроме того, данные модели содержат информацию о схеме коммуникаций, используемую непосредственно в процессе моделирования. Для удобства представления данной информации строится коммуникационный граф, который содержит в компактной форме информацию о схеме связей между компонентами архитектуры.

Результатом настройки ГПД алгоритма функционирования окружающей системы и прикладных функций встроенной системы на параметры частных моделей является ***ГПД-программа*** для виртуальной ма-

шины. Данная ГПД-программа в совокупности с виртуальной машиной образует активную, выполняемую на виртуальной машине, модель. Процесс выполнения такой модели на виртуальной машине осуществляется под управлением ***виртуальной операционной системы*** и соответствует имитации функционирования проектируемой СРВ.

ГПД-программа включает:

- множество ***вычислительных процессов реального времени (РВ-процессов)***;
- множество ***каналов и канальных функций взаимодействия*** РВ-процессов;
- множество команд для выполнения на виртуальной машине.

Таким образом, активная модель рассматривается как совокупность ***ГПД-программы***, виртуальной машины и разработанной для нее ***виртуальной операционной системы***.

Заметим также, что методы построения активных моделей, включая разработку теории и практических методик модельного представления перечисленных компонентов, должны быть инвариантными относительно ГПД, описывающих функционирование конкретных СРВ.

3.4. Этапы модульной технологии проектирования

Процесс эволюционного проектирования СРВ в соответствии с модульной технологией осуществляется в три этапа. Схема функционального взаимодействия данных этапов в процессе проектирования представлена на рис. 3.1.

Рис. 3.1. Функциональная схема проектирования СРВ

Первый этап моделирования соответствует уровню аналитических расчетов. Входной информацией являются частные модели: модель топологии окружающей системы, ГПД алгоритма функционирования СРВ, ГПД прикладных функций СРВ, модель архитектуры МВС. Аналитические расчеты включают решение задач по оптимизации топологии размещения станций МВС и определения плана подключения к ним терминальных точек по критерию минимума суммы длины расстояний между терминальными точками и станциями. На данном этапе используется также методика приближенной оценки времени выполнения алгоритмов прикладных функций для последовательного и параллельного представления соответствующих ГПД.

Результаты расчетов позволяют принять решения по исходному варианту архитектуры МВС, значениям параметров основных ресурсов станций и локальной сети.

Этап функционального моделирования

Второй этап моделирования несет основную нагрузку по определению приемлемого варианта архитектуры МВС для выполнения заданной совокупности прикладных функций СРВ. То есть на данный этап приходится основная доля трансформаций, выполняемых в эволюционном цикле проектирования СРВ. Исходя из этого, предлагаемая ниже стратегия моделирования нацелена на многократные трансформации модели архитектуры МВС и согласования с ними моделей прикладных функций. Поэтому используемые на данном этапе модели прикладных функций и методы моделирования по возможности упрощены до такой степени, чтобы трудоемкость выполнения последовательности трансформаций не привела к необходимости сокращения области поиска решений при выборе приемлемого варианта архитектуры МВС.

Таким образом, предлагаемые на данном этапе методы моделирования позволяют снять значительную долю неопределенности относительно варианта архитектуры МВС способного выполнить совокупность прикладных функций в установленное время. Полученные при этом модели архитектуры МВС и алгоритмов прикладных функций следует рассматривать как исходные для комплексного моделирования программной нагрузки СРВ с учетом динамических свойств окружающей системы.

Этап комплексного моделирования СРВ в динамике

Варианты моделей архитектуры МВС и прикладных функций, полученные по результатам исследований на этапе функционального моделирования рассматриваются как исходные для построения комплексной модели программной нагрузки и проведения моделирования с учетом условий реального времени. Решение данных задач является основным содержанием этапа комплексного моделирования СРВ в динамике. По содержанию данные задачи перекликаются с рассмотренными в предыдущей главе и могут рассматриваться как продолжение процесса эволюции моделей. Однако учет условий реального времени, продиктованных динамическими свойствами окружающей системы, приводит к существенному увеличению сложности моделей и принципиально иным механизмам управления взаимодействием параллельных процессов при моделировании.

Предполагается, что область трансформаций моделей на данном этапе эволюции в сравнении с предшествующим этапом будет невелика. Основные цели исследований на данном этапе заключаются в следующем:

- доопределение модели программной нагрузки проектируемой СРВ с учетом условий реального времени и динамических свойств окружающей системы;
- уточнение требуемых значений параметров ресурсов МВС при относительно стабильной архитектуре;
- экспериментальная проверка корректности модели программной нагрузки по условиям реального времени путем моделирования ее выполнения на модели МВС.

Методы достижения данных целей сопровождаются введением новых понятий, таких как:

- ***вычислительные процессы реального времени (РВ-процессы)***;
- ***каналы взаимодействия процессов и функции каналов***;
- ***активная динамическая модель программной нагрузки СРВ***.

Данные понятия имеют принципиально важное значение не только для рассматриваемых задач, но и для разработки на этой основе в последующем формальных методов анализа корректности моделей и методов синтеза управляющих программ при проектировании распределенных СРВ.

Вопросы для контроля усвоения знаний

1. Пояснить принципы дискретности и графовой формы представления моделей как основы технологии модульного проектирования.
2. Дать определение ГПД как одной из форм структурно-графических представлений алгоритмов.
3. Перечислить основные свойства модульных структур.
4. Дать понятие активных моделей как воплощения третьего принципа технологии модульного проектирования.
5. Перечислить и дать определения частных моделей.
6. Определить состав компонентов активной модели.
7. Раскрыть существо различий между моделированием на основе активных моделей и традиционным подходом к имитационному моделированию.
8. Раскрыть содержание и показать взаимосвязь основных этапов модульной технологии проектирования.
9. Пояснить основное назначение этапа моделирования на функциональном уровне.
10. Назвать основные цели исследований на этапе комплексного моделирования СРВ в динамике.

4. ЧАСТНЫЕ МОДЕЛИ КОМПОНЕНТОВ ПРОЕКТИРУЕМЫХ СИСТЕМ

4.1. Модель топологии размещения компонентов МВС

При проектировании распределенных систем реального времени на базе микропроцессорных станций приходится строить МВС, станции в которых объединяются в локальную сеть. При определении топологии размещения компонентов МВС наибольший интерес представляют станции, непосредственно связанные с датчиками и исполнительными механизмами объекта управления, которые именуются терминальными точками МВС. Каждая станция может быть связана с ограниченным числом терминальных точек. Терминальные точки различаются по видам и, следовательно, станции имеют ограничения на подключение точек по каждому виду. Места установки станций и точек могут быть заранее фиксированы либо нет.

Количество станций в МВС может определяться по числу точек с учетом их видов. В этом случае число станций считается фиксированным. Если число станций определяется исходя из экономического критерия, учитывающего затраты на станции и линии связи по подключению точек, то при решении такой задачи число станций не может быть фиксированным. Таким образом, задачи оптимизации топологии размещения компонентов ВС могут рассматриваться в разных условиях.

Рассматривается два варианта задачи. В первом из них все точки имеют один вид, места их расположения на топологическом поле фиксированы и заданы координатами. Все станции могут подключать одинаковое число терминальных точек и размещаться в местах их расположения. Принимается также, что терминальная точка, на месте расположения которой размещена станция, подключается к данной станции. Кроме того, число станций принимается фиксированным и определяется как отношение общего числа точек к числу точек, которое можно подключить к одной станции. Второй вариант отличается тем, что на топологическом поле терминальные точки могут быть разных типов, а станции подключать разные совокупности таких точек.

Задача заключается в поиске мест расположения станций и плана подключения к ним терминальных точек так, чтобы сумма длин расстояний между точками и станциями была минимальной. Данная задача сформулирована как линейная задача математического программирования с булевыми переменными и была изложена в разделе 2.3. В такой постановке совмещены задачи определения числа станций, их располо-

жения на объекте и распределения терминальных точек по станциям. Основным недостатком такой постановки является большая размерность задачи. В работе [33] решение данной задачи предлагается осуществлять в два этапа. На первом из них определяются места расположения станций, а на втором непосредственно осуществляется распределение терминальных точек по станциям. Такой подход представляется более предпочтительным, так как упрощает задачу и позволяет более гибко изменять полученные решения, при согласовании их с решениями других задач проектирования СРВ.

Определение мест расположения станций (1−й вариант задачи)

Рассмотрим задачу определения мест расположения станций. При этом все терминальные точки принимаются равноценными и, следовательно, станции также могут быть приняты одинаковыми. Разумеется, что такие условия нереалистичны для проектирования СРВ. А, для других приложений такая ситуация является вполне возможной. Кроме того, решение данной и последующей задачи распределения терминальных точек по станциям имеет важное познавательное значение.

Введем следующие обозначения:

$\boldsymbol{E} = \{\boldsymbol{e_j}\}$ – множество точек, заданных координатами $(\boldsymbol{x_j}, \boldsymbol{y_j})$ на топологическом поле;

$\boldsymbol{S} = \{\boldsymbol{s_i}\}$ – множество станций;

$\boldsymbol{r}$ – число точек, которое может быть подключено к одной станции.

Для определения мест расположения станций на топологическом поле выделяется подмножество точек $\boldsymbol{E_p} \subset \boldsymbol{E}$, $|\boldsymbol{E_p}| = |\boldsymbol{S}|$. Точки $\boldsymbol{e_t} \in \boldsymbol{E_p}$ именуются полюсами. Для выделения полюсов предлагается использовать эвристическое правило, согласно которому полюса на топологическом поле располагаются относительно равномерно.

Алгоритм определения мест расположения полюсов заключается в следующем. Первым полюсом принимается точка $\boldsymbol{e_t}$, которая максимально удалена от всех других точек множества $\boldsymbol{E} \setminus \boldsymbol{e_t}$. Точка $\boldsymbol{e_t}$ включается в множество $\boldsymbol{E_p^*}$. Второй полюс выбирается среди точек $\boldsymbol{E} \setminus \boldsymbol{E_p^*}$. С этой целью для всех точек определяется расстояние до ближайшего полюса и среди них выбирается точка с максимальным значением минимального расстояния.

В общем случае выбор очередного полюса можно записать следующим образом. Обозначим d_{tt_p} − расстояние от точки $e_t \in E \setminus E_p^*$ до точки $e_{t_p} \in E_p^*$. Тогда условие выбора очередного полюса имеет вид

$$d_{tt_p}^* = \max_{E \setminus E_p} \min_{E_p} d_{tt_p}. \qquad (4.1)$$

Здесь $d_{tt_p}^*$ − расстояние от точки $e_t \in E \setminus E_p^*$ до ближайшего полюса в множестве E_p^*, выбранной для размещения очередного полюса и соответствующей максимальному, в сравнении с другими точками множества $E \setminus E_p^*$, удалению от полюсов E_p^*. Терминальная точка e_t, выбранная по условию (3.1), включается в множество E_p^*. Выбор точек для размещения полюсов продолжается до получения множества $E_p^* = E_p$, мощность которого равна числу размещаемых станций.

Задача распределения терминальных точек по станциям

Для решения задачи распределения терминальных точек по станциям (полюсам) множества E_p введем матрицу $C = \|c_{ij}\|_{m \times n}$, где c_{ij} − расстояние между полюсом $p_i \in E_p$ и точкой e_j; m − число полюсов, $m = |E_p|$; n − число распределяемых точек, $n = |E|$. Введем также переменную $x_{ij} = 1$, если точка e_j подключается к полюсу p_i, в противном случае $x_{ij} = 0$. В принятых обозначениях задача распределения точек по полюсам (станциям) по критерию минимальной суммы длин связей при подключении точек к станциям запишется в виде:

$$\sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \Rightarrow \min; \qquad (4.2)$$

$$\sum_{j=1}^{n} x_{ij} \le r_i, \quad i = 1, 2, \ldots, m; \qquad (4.3)$$

$$\sum_{i=1}^{m} x_{ij} = 1, \quad j = 1, 2, \ldots, n. \qquad (4.4)$$

Здесь r_i − число мест для подключения точек к станции в i − м полюсе. Если все станции могут подключать одинаковое число терминальных точек, то в выражении (4.3) величина $r_i = r$ для всех $i = 1, 2, \ldots, m$.

Задача определения числа станций
(2 - й вариант задачи)

Введем ряд обозначений. Пусть на топологическом поле размещена совокупность терминальных точек $\boldsymbol{E} = \bigcup_{r} \boldsymbol{E}_r$, где $\boldsymbol{E}_r$ – терминальные точки $\boldsymbol{r}$-го типа, $\bigcap_{r} \boldsymbol{E}_r = \varnothing$.

Для каждой станции $\boldsymbol{s}$-го типа введем вектор подключения точек $\boldsymbol{G}_s = \{\boldsymbol{g}_r\}$, $\boldsymbol{r} = 1, 2, ..., \mathrm{R}$, где $\boldsymbol{R}$ – общее число типов точек на топологическом поле. Тогда способность станций подключать точки согласно векторам $\boldsymbol{G}_s$, $\boldsymbol{s} = 1, 2, ..., \mathrm{S}$ представим матрицей

$$\boldsymbol{A} = \|\boldsymbol{a}_{rs}\|_{\boldsymbol{R}\times\boldsymbol{S}}; \qquad \boldsymbol{r} = 1, 2, ..., \mathrm{R}; \qquad \boldsymbol{s} = 1, 2, ..., \boldsymbol{S}.$$

здесь $\boldsymbol{a}_{rs}$ − число точек $\boldsymbol{r}$-го вида, которые могут быть подключены к станции $\boldsymbol{s}$-го типа.

Введем переменную $\boldsymbol{x}_s$, $\boldsymbol{s} = 1, 2, ..., \boldsymbol{S}$, которая определяет число станций $\boldsymbol{s}$-го типа, необходимых для подключения точек топологического поля. Тогда задача определения минимального числа станций разных типов, необходимых для подключения всех терминальных точек с учетом введенных обозначений заключается в следующем:

$$\min \boldsymbol{L} = \sum_{s=1}^{S} \boldsymbol{x}_s, \tag{4.5}$$

$$\sum_{s=1}^{S} \boldsymbol{a}_{rs} \boldsymbol{x}_s \geq \boldsymbol{b}_r, \qquad \boldsymbol{r} = 1, 2, ..., \mathrm{R}, \tag{4.6}$$

$\boldsymbol{x}_s$ – целое для всех $\boldsymbol{s} = 1, 2, ..., \mathrm{S}$. (4.7)

Здесь $\boldsymbol{b}_r$ −число точек $\boldsymbol{r}$-го типа на топологическом поле, $\boldsymbol{b}_r = |\boldsymbol{E}_r|$.

Задача (4.5)–(4.7) является целочисленной задачей линейного программирования. По своей специфике данная задача относится к ставшей широко известной задачей покрытия [34]. Для нее предложен достаточно эффективный метод решения [35].

Результатом решения задачи (4.5)–(4.7) является вектор $\boldsymbol{X}^* = \{\boldsymbol{x}_s^*\}$, $\boldsymbol{s} = 1, 2, ..., \boldsymbol{S}$. Компоненты $\boldsymbol{x}_s^*$ вектора $\boldsymbol{X}^*$ определяют оптимальное число станций, необходимое для подключения множества терминальных точек системы. Следует заметить, что оптимальность вектора $\boldsymbol{X}^*$ следует рассматривать только относительно состава терминальных точек. Вместе с тем, последующие решения других задач могут привести к необходимости увеличения числа станций. Заметим также, что критерий (4.5) записан исходя из равноценности станций. Если цена

станций различна, то необходимо ввести величины c_s, $s = 1, 2, ..., S$, определяющие стоимость станции s-го типа. Критерий (4.5) в этом случае запишется в виде

$$\min L = \sum_{s=1}^{S} c_s x_s .$$

Задача распределения терминальных точек по заданному числу станций

Данную задачу можно рассматривать как продолжение предыдущей. Исходными данными является множество терминальных точек E, разбитое на подмножества E_r, $r = 1, 2, ..., R$ и вектор $X^* = \{x_s^*\}$, $s = 1, 2, ..., S$, определяющий число станций s-го типа, необходимых для распределения точек множества E. Для каждой s-й станции известен вектор подключения точек $G_s = \{g_r\}$, $r = 1, 2, ..., R$.

Формирование совокупности полюсов

На этом этапе определяются места, в которых могут быть размещены станции. Для решения данной задачи используется алгоритм определения полюсов, который работает в условиях, когда терминальные точки одного вида [28]. Полюса выделяются на местах расположения точек и являются потенциальными местами для размещения станций.

В нашем случае $R > 1$, а число станций определяется вектором X^*, полученным при решении задачи (4.5) – (4.7). Поэтому предлагается определять совокупность полюсов E_r^p, $r = 1, 2, ..., R$, отдельно для каждого множества точек E_r. При этом число полюсов принимается одинаковым для всех r и равным $\left|E_r^p\right| = \sum_{s \in S} x_s^*$. В результате выполнения алгоритма получим совокупность множеств полюсов $\{E_r^p\}$, $r = 1, 2, ..., R$. Места расположения полюсов E_r^p совмещаются с местами расположения точек r-го типа. В итоге получается общее число полюсов $\left|E^p\right|$, равное произведению $\left|E_r^p\right| \cdot R$.

Формирование вариантов подключения точек к станциям

Для каждой станции с $x_s^* \geq 1$ и вектором подключения G_s формируется множество вариантов подключения $V_s = \{v_{sj}\}, \quad j = 1,2,...,n_s$, $n_s = |V_s|$. Алгоритм формирования вариантов v_{sj} заключается в следующем.

Последовательно просматриваются полюса в точках $e_i \in E^p$, $i = 1,2,...,|E^p|$. Если точка e_i может быть подключена к станции с вектором G_s, то станция размещается в точке e_i и принимается, что точка e_i подключена к станции и входит в формируемый вариант v'_{sj}. Последующие точки $e_t \in E \setminus v'_{sj}$, $t = 1,2,...,T$ включаются в вариант v'_{sj}, если они не нарушают условия подключения согласно вектору G_s и удалены от станции на минимальное расстояние. Формирование совокупности вариантов $\{v_{sj}\}$ для станции s-го типа заканчивается, если все точки $e_i \in E^p$ просмотрены. В итоге формируется множество вариантов $V_s = \{v_{sj}\}$, $j = 1,2,...,n_s$. Аналогично формируются множества вариантов для станций с другими векторами подключения G_s.

Выбор вариантов подключения

Полученное таким образом множество вариантов $V = \bigcup_{s \in S} V_s$ составляет основу для решения задачи выбора предпочтительных вариантов и размещения соответствующих станций. В качестве оценки варианта v_{sj} вводится величина c_{sj} – сумма расстояний между точками, входящими в вариант.

Принадлежность точек $e_t \in E$ вариантам v_{sj} представим в виде матрицы $A = \|a_{tsj}\|$, $t = 1,2,...,T'$. Строки матрицы A соответствуют элементам множества E, а столбцы – вариантам множества V. Элемент $a_{tsj} = 1$, если точка e_t принадлежит варианту v_{sj}, в противном случае $a_{tsj} = 0$. Здесь величина T' определяет мощность множества точек e_t, вошедших в варианты $\{v_{sj}\}$, $T' \leq |E|$.

Введем переменную x_{sj},

$$x_{sj} = \begin{cases} 1, & \text{если вариант } v_{sj} \text{ входит в решение,} \\ 0, & \text{в противном случае.} \end{cases}$$

Тогда задачу выбора вариантов v_{sj} по критерию минимальной суммы их оценок c_{sj} можно записать в виде:

$$\min L = \sum_{s=1}^{S} \sum_{j=1}^{n_s} c_{sj} x_{sj}; \tag{4.8}$$

$$\sum_{s=1}^{S} \sum_{j=1}^{n_s} a_{tsj} x_{sj} \geq 1; \qquad t = 1,2,...,T'; \tag{4.9}$$

$$\sum_{j=1}^{n_s} x_{sj} = x_s^*; \qquad s = 1,2,...,S. \tag{4.10}$$

Задача (4.8) – (4.10) является линейной задачей математического программирования с булевыми переменными.

Определение «центров» размещения станций

Варианты v_{sj}^*, выбранные в результате решения задачи (4.8) – (4.10), включают совокупности точек различных типов согласно векторам подключения G_s. Обозначим совокупность точек варианта v_{sj}^* множеством E_{sj}. В одной из точек множества E_{sj}, являющейся полюсом, расположена станция s-го типа. В общем случае расположение станции в множестве E_{sj} относительно критерия минимальной суммы расстояний от станции до точек $e_t \in E_{sj}$ не является наилучшим. Для определения «центра» в множестве E_{sj} и размещения в нем станции используется следующая процедура.

В множестве E_{sj} каждая точка e_t представлена координатами (x_t, y_t). Координаты точки с минимальным суммарным удалением от точек E_{sj} определяются выражениями:

$$x_c = \sum_{e_t \in E_{sj}} x_t / |E_{sj}|; \qquad y_c = \sum_{e_t \in E_{sj}} y_t / |E_{sj}|.$$

Точка t_c с координатами (x_c, y_c) принимается в качестве «центра» множества E_{sj} и в нее размещается станция. Данная процедура выполняется для всех вариантов v_{sj}^*.

При решении задачи (4.8) – (4.10) качество варианта v_{sj} оценивалось величиной c_{sj}, которая не зависит от места размещения станции в данном варианте. В действительности, после размещения станции в «центре» множества E_{sj}, вариант v_{sj} следует оценивать суммой расстояний от «центра» до всех точек E_{sj}. Обозначим эту величину c°_{sj} и сформулируем следующее предположение. Если для двух вариантов v_{sj_1} и v_{sj_2}, $\left|E_{sj_1}\right| = \left|E_{sj_2}\right|$ выполняется условие $c_{sj_1} \geq c_{sj_2}$, то условие $c^{\circ}_{sj_1} \geq c^{\circ}_{sj_2}$ также должно выполняться. Данная гипотеза проверялась экспериментально. С этой целью на топологическом поле случайным образом генерировались два множества точек E_{sj_1} и E_{sj_2} одинаковой мощности. В каждом из множеств определялся «центр» и сопоставлялись условия $c_{sj_1} \geq c_{sj_2}$ и $c^{\circ}_{sj_1} \geq c^{\circ}_{sj_2}$. На множестве проведенных экспериментов гипотеза всегда подтверждалась.

Улучшение распределения точек по станциям

После определения центров t_c, $c = 1,2,...,m$ и перемещения в них станций появляется возможность улучшить распределение терминальных точек по станциям. Кроме того, процедура формирования вариантов и решение задачи (4.8) – (4.10) не гарантируют покрытие всех точек множества E. В связи с этим на заключительном этапе решается задача распределения точек множества E по станциям, по критерию минимальной суммы расстояний от точек до станций.

Представим множество точек $E = \{E_z\}$, $z = 1,2,...,R$, совокупностью множеств, содержащих точки z-го типа, и пронумеруем точки в множествах $E_z = \{e^z_t\}$, $t = 1,2,...,t_z$, $t_z = |E_z|$.

Пронумеруем также множество центров $\{t_c\}$, $c = 1,2,...,m$, и компонентов векторов подключения станций, размещенных в этих центрах, $G_c = \{a_{cr}\}$, $c = 1,2,...,m$, $r = 1,2,...,r_c$. Здесь a_{cr} – число мест подключения точек r-го типа в станции, размещенной в c-м центре, а r_c – число типов точек, которые могут быть подключены к станции в центре t_c.

Информацию о расстоянии между точками e^z_t и центрами t_c представим матрицей $D = \|d_{crtz}\|$, где d_{crtz} – расстояние от центра t_c с

местом подключения r-го типа до точки e_t^z z-го типа. Введем переменную x_{crtz}:

$$x_{crtz} = \begin{cases} 1, & \text{если точка } e_t^z \text{ подключается к станции} \\ & \text{в центре } t_c \text{ по месту подключения} \\ & z\text{-го типа;} \\ 0, & \text{в противном случае.} \end{cases}$$

В принятых обозначениях задача распределения точек по станциям запишется в виде

$$\min \sum_{c=1}^{m}\sum_{r=1}^{r_c}\sum_{t=1}^{t_z}\sum_{z=1}^{n} d_{crtz}x_{crtz}; \qquad (4.11)$$

$$\sum_{z=1}^{n}\sum_{t=1}^{t_z} x_{crtz} \le a_{cr}, \quad c = 1,2,...,m, \quad r = 1,2,...,r_c; \qquad (4.12)$$

$$\sum_{c=1}^{m}\sum_{r=1}^{r_c} x_{crtz} = 1, \quad t = 1,2,...,t_z, \quad z = 1,2,...,n. \qquad 4.13)$$

Ограничение (4.12) разрешает подключение точек к станции согласно ее вектору подключения. Ограничение (4.13) обеспечивает подключение каждой точки к одной из станций.

Задача (4.11) – (4.13) занимает промежуточное положение между задачей о назначении и транспортной задачей. Так, если вектор подключения точек к станции представить булевским и, соответственно, пронумеровать места подключения каждой точки, то задачу (4.11) – (4.13) можно записать в виде задачи назначения точек на доступные по типу места подключения. И, напротив, если топологическое поле представить совокупностью клеток координатной сетки и каждую клетку с расположенными в ней точками разных типов рассматривать в качестве пункта потребления, а места подключения станции – пунктами производства, то задача преобразуется в классическую задачу транспортного типа. Известно, что и транспортная задача и задача о назначениях имеют эффективные алгоритмы решения [32].

Постановка задачи распределения терминальных точек разного типа с учетом сокращения размерности топологического поля

Рассмотрим постановку данной задачи при условии, что точки различаются по видам, а каждой станции s-го вида соответствует вектор подключения точек $G_s = \{g_r\}$, $r = 1, 2, \ldots, R$. Здесь g_r – число точек r-го вида, которые могут быть подключены к станции s-го вида, R –

число видов точек. Множество точек $\boldsymbol{E}$ в этом случае является объединением непересекающихся подмножеств точек $\boldsymbol{r}$-го вида $\boldsymbol{E_r}$, $\boldsymbol{E} = \bigcup_{r} \boldsymbol{E_r}$.

Для сокращения размерности рассматриваемой задачи множество $\boldsymbol{E_r}$ разобьем на подмножества, компактно расположенные на топологическом поле, $\boldsymbol{E_r} = \bigcup_{j} \boldsymbol{E_{jr}}$, $\boldsymbol{E_{jr}} \neq \varnothing$. Компактность подмножества $\boldsymbol{E_{jr}}$ можно оценить отношением суммарной длины связей (расстояний) $\boldsymbol{L_{jr}}$ между точками множества $\boldsymbol{E_{jr}}$ к мощности данного множества $\left|\boldsymbol{E_{jr}}\right|$, $\boldsymbol{L_{jr}} \Big/ \left|\boldsymbol{E_{jr}}\right|$.

Чем меньше величина $\boldsymbol{L_{jr}} \Big/ \left|\boldsymbol{E_{jr}}\right|$, тем более компактным является подмножество $\boldsymbol{E_{jk}}$. В частности, подмножество $\boldsymbol{E_{jr}}$ может включать лишь одну точку. В этом случае величина отношения является наименьшей и равной нулю. При разбиении множества $\boldsymbol{E_r}$ на подмножества $\boldsymbol{E_{jr}}$, их компактность можно ограничить сверху значением отношения $\boldsymbol{L_{jr}} \Big/ \left|\boldsymbol{E_{jr}}\right|$, которое назначается экспертным путем. Для упрощения разбиения множеств $\boldsymbol{E_r}$ на подмножества $\boldsymbol{E_{jr}}$ на топологическое поле можно наложить регулярную сетку с некоторым шагом δ. Если $\boldsymbol{j}$-я ячейка сетки содержит совокупность точек $\boldsymbol{r}$-го вида $\boldsymbol{E_{jr}}$, то данное множество принимается компактным. Таким образом, все точки топологического поля разбиваются на подмножества точек $\boldsymbol{r}$-го вида в $\boldsymbol{j}$-й ячейке $\boldsymbol{E_{jr}}$. При этом учитываются только ячейки с $\boldsymbol{E_{jr}} \neq \varnothing$. Пустые ячейки сетки, для которых $\boldsymbol{E_{jr}} = \varnothing$, исключаются из рассмотрения и не нумеруются.

Задача распределения терминальных точек по полюсам в данном случае имеет следующий вид:

$$\sum_{s=1}^{m} \sum_{j=1}^{n} \sum_{r=1}^{R} \boldsymbol{c_{sjr}} \boldsymbol{x_{sjr}} \Rightarrow \mathbf{min;} \qquad (4.14)$$

$$\sum_{j=1}^{n} \sum_{r=1}^{R} \boldsymbol{x_{sjr}} = \boldsymbol{g_r}, \quad \mathrm{r} = 1,\ 2,\ ...,\ \mathrm{R}; \qquad (4.15)$$

$$\sum_{i=1}^{m} \sum_{r=1}^{R} \boldsymbol{x_{sjr}} = \boldsymbol{b_{jr}}, \quad \mathrm{j} = 1,\ 2,\ ...,\ \mathrm{n},\ \mathrm{r} = 1,\ 2,\ ...,\ \mathrm{R}. \qquad (4.16)$$

Здесь c_{sjr} – расстояние от центра ячейки сети, содержащей s-ю станцию с возможностью подключения g_r точек, до ячейки с компактным множеством E_{jr};

x_{sjr} – число точек множества E_{jr} подключенных к s-й станции;

b_{jr} – число точек в множестве E_{jr}.

Задача (4.14) – (4.16) относится к классу задач транспортного типа, для которой выполняется условие $\sum_{i=1}^{m}\sum_{r=1}^{R} g_r = \sum_{j}^{n}\sum_{r}^{R} b_{jr}$.

Заметим также, что в выражении (4.14) величина $c_{sjr} = \infty$, если компактное множество E_{jr} связывается со станцией, для которой $g_r = 0$. Для решения задачи (4.14) – (4.16), так же как и для задачи (4.2)–(4.4), может быть использован один из методов решения задач транспортного типа, например, метод потенциалов.

Пример топологического поля

Пример решения задачи определения полюсов и распределения по ним точек приведен на рис. 4.1

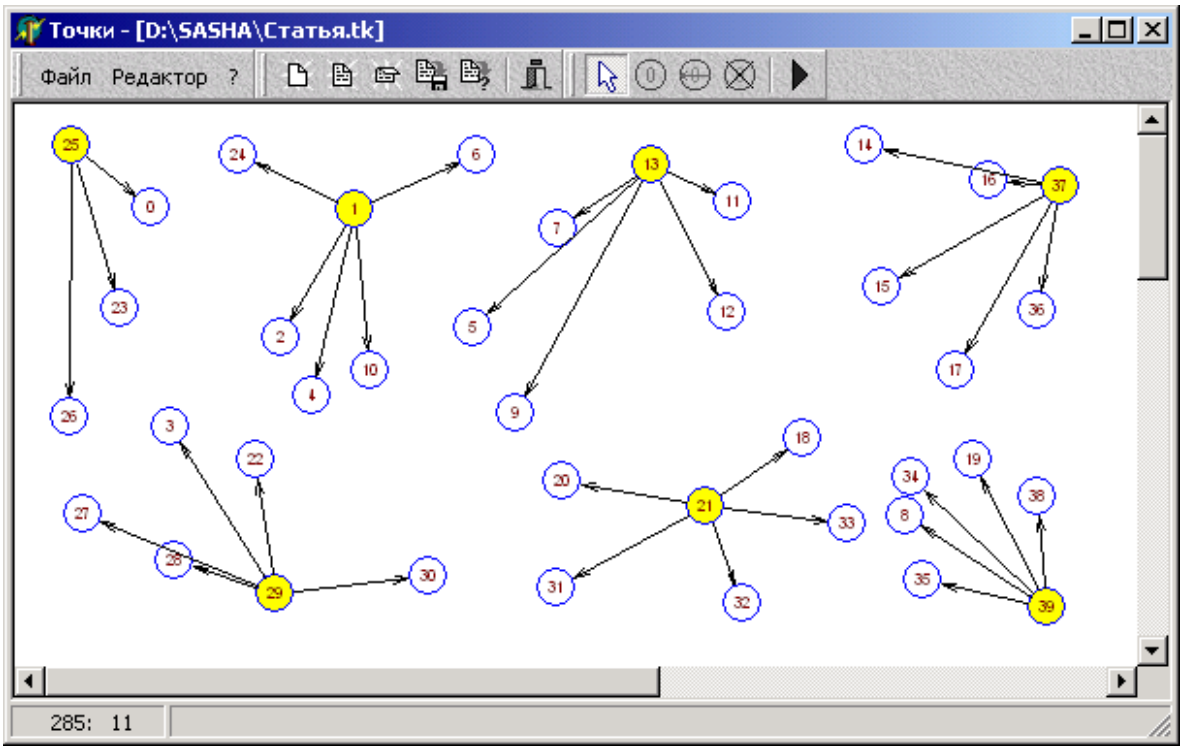


Рис. 4.1. Пример топологического поля

Расположение полюсов на рисунке соответствует варианту, полученному после выполнения первого этапа алгоритма. Здесь предполагается также, что точки, на которых расположились полюса, подключаются к данным полюсам. После решения задачи распределения точек по полюсам места размещения станций уточняются. С этой целью среди точек, подключенных к конкретной станции, определяется точка с минимальным суммарным расстоянием от нее до всех остальных точек. В выбранной таким образом точке размещается соответствующая станция.

Предложенный метод решения задач определения мест размещения станций и распределения терминальных точек по станциям может использоваться как базовый при разработке методов решения данных задач с другими исходными условиями. Заметим также, что предложенный алгоритм определения мест размещения полюсов согласно выражению (4.1) разработан для условия $\boldsymbol{R}=1$. Для условия $\boldsymbol{R}>1$ выбор мест расположения станций осуществляется на основе решения задачи (4.5) – (4.7), в которой определяется число станций, и далее решения задачи (4.8) – (4.10) и задачи (4.11) – (4.13) по выбору вариантов подключения точек, определения мест расположения станций и распределения точек по станциям.

Отметим также, что для условия $\boldsymbol{R}>1$ необходимо проведение дополнительных исследований по оценке эффективности применения предложенных алгоритмов. Необходимо исследовать также задачу формирования компактных множеств $\boldsymbol{E}_{jr}$ и влияние вариантов решения данной задачи на качество решения задачи распределения терминальных точек по станциям.

4.2. Томография структурных свойств алгоритмов прикладных функций СРВ

Отображение ***структурных свойств алгоритмов, прикладных функций*** можно определить известным термином "томография". Данный термин применительно к алгоритмам и программам уместно использовать для определения действий по отображению ряда важнейших структурных свойств алгоритмов – ***структурных компонентов семантики алгоритмов*** [18, 22, 23]. Среди важнейших структурных свойств алгоритмов выделяются следующие:

- ***состав элементов структуры*** (операторов, совокупностей операторов), которые в последующем будем именовать ***модулями***;
- ***межмодульные связи по управлению***;
- ***информационные межмодульные связи***.

Перечисленные свойства относятся к структурообразующим, но не являются конструктивными и не могут непосредственно использоваться при определении свойств полезных для анализа характеристик. Производными от этих свойств являются другие в большей степени конструктивные свойства, такие как:

- ***вложенность модулей***;
- ***параллельность***;
- ***связность***;
- ***функциональность***.

Свойство вложенности может характеризовать относительную глубину свертки или декомпозиции алгоритма на модули. ***Параллельность*** можно оценить относительным числом уровней обработки данных. Для оценки ***связности*** с точки зрения возможности локализации вычислительных процессов можно использовать ***коэффициенты связности*** для различных ***уровней свертки модулей***. Коэффициент связности можно определить как отношение числа связей между модулями на i-м уровне свертки к числу межмодульных связей на $(i-1)$ -м уровне. Функциональность структуры алгоритма отражает функциональную законченность модулей на различных уровнях свертки, возможность их диагностики, локализации изменений. Свойства ***функциональности*** модулей могут оцениваться на основе ***гиперграфовых моделей***, предложенных в [38]. Ряд свойств модульных структур были рассмотрены ранее в разделе 3.2.

Перечисленные свойства определяют основные задачи томографии алгоритмов и программ и в целом предмет теоретических исследований в этой области. Предмет исследований включает определения и детальное изучение перечисленных свойств, введение метрик для их количественных оценок и поиск эффективных методов вычисления данных оценок.

Структурно - графические представления алгоритмов

Известно, что структурные свойства программ наиболее полно в явном виде могут быть представлены графовыми моделями [18, 23]. Поэтому в основу методов томографии алгоритмов и программ положены их структурно-графические представления [36, 37]. Такие представления в отличие от графовых моделей [39] создают условия для более широких возможностей по отображению структурных свойств алгоритмов и используют современные возможности визуального представления графической информации и программирования.

Методы томографии алгоритмов и программ являются инструментом для визуальной и количественной оценки структурных свойств в статике. Вместе с тем для принятия решений при проектировании СРВ необходимо знание таких характеристик, которые отражают динамику алгоритма, оценивают алгоритм при его функционировании. В первую очередь это характеристики, которые оценивают время выполнения алгоритма или его отдельных фрагментов. Не менее важными являются также характеристики, которые непосредственно или косвенно влияют на временные оценки работы алгоритма и рационального использования ресурсов МВС.

Объектом анализа являются алгоритмы прикладных функций, составляющих ***программную нагрузку*** проектируемой СРВ. Сюда относятся алгоритмы сбора информации, алгоритмы ее обработки, алгоритмы принятия решений по выработке управляющих воздействий, отображения информации о состоянии окружающей системы и другие локальные алгоритмы контроля и управления. Будем также предполагать, что для всех алгоритмов написаны программы на языке высокого уровня, например, на Паскале.

Заметим, что при написании программ архитектура МВС неизвестна и поэтому данные программы следует рассматривать как автономные, реализующие функции соответствующих алгоритмов без учета условий реального времени. При анализе представленной таким образом программной нагрузки следует исходить из того, что значения характеристик будут существенно зависеть от того, в какой ***вычислительной среде*** будут выполняться данные программы. Здесь под вычислительной средой понимаются технические средства вычислительной системы, определяемые составом устройств и структурой связей между ними, и правила организации вычислений при выполнении программной нагрузки.

Наличие зависимости характеристик от вычислительной среды очевидно для программной нагрузки, представленной совокупностью алгоритмов. Что касается локальных программ, то и здесь такая зависимость легко обнаруживается. Так, если организовать параллельное выполнение программы, то время вычислений может существенно сократиться. Разное время вычислений может быть получено и для однопроцессорной вычислительной системы с различными значениями ресурсов. Поэтому в последующем при анализе характеристик будем помнить, что оценка их значений всегда должна производиться применительно к конкретной вычислительной среде.

Среди характеристик, которые непосредственно влияют на оценки предпочтительности решений при проектировании СРВ, выделены ха-

рактеристики, связанные с ***временными оценками***, с ***загрузкой устройств МВС***, с ***частотами выполнения отдельных фрагментов алгоритма***. Полезность первых двух характеристик не вызывает сомнений. Знание частот выполнения отдельных фрагментов в совокупности с оценкой времени их выполнения позволяет принимать решения о внесении изменений в данные фрагменты, например, путем распараллеливания, более удачной программной реализации, снижения или повышения точности вычислений и другими средствами.

Минимизация времени выполнения программной нагрузки

Оценка ***минимального времени выполнения программной нагрузки на МВС*** является четвертой характеристикой. Данная оценка позволяет установить за какое, по возможности минимальное, время может быть выполнен алгоритм или программная нагрузка в целом на МВС с заданным набором архитектур и типов микропроцессоров. Для получения этой оценки необходимо выявить факторы, влияющие на ее значение.

Прежде всего, это возможность параллельного выполнения. Очевидно, что можно осуществить ***максимально возможное распараллеливание***, не связывая это с числом процессоров в МВС [30]. В то же время неясно, на каком уровне детализации представления алгоритмов следует вести распараллеливание. Будем предполагать, что распараллеливание выполнено на уровне всех переменных алгоритма и, соответственно, операторов. Максимально возможное распараллеливание, в частности, не означает, что на каждую параллельную ветвь будет выделен процессор. К тому же это далеко не всегда приводит к снижению времени выполнения алгоритма. Однако наличие максимального распараллеливания обеспечивает возможность вести поиск решения на всей допустимой области.

Поиск максимально возможного распараллеливания на уровне операторов приводит к задаче большой размерности. Поэтому на практике распараллеливание производится на уровне более крупных фрагментов алгоритмов. Размер фрагментов в каждом конкретном случае устанавливается в зависимости от объема программной нагрузки и размерности, приемлемой для решения задачи распараллеливания.

Вторым фактором является планирование вариантов использования ресурсов при выполнении программной нагрузки. ***Планы использования ресурсов*** включают ***распределение памяти для хранимых данных и программ, и распределение функций по процессорам МВС***. Планирование использования ресурсов во многом предопределяет рав-

номерность загрузки устройств МВС и в целом снижение загрузки коммуникаций локальной сети, используемой в МВС.

При поиске оптимального или приемлемого плана использования ресурсов памяти и процессоров, распределение функций алгоритмов, программ и данных должно осуществляться в комплексе. Такая необходимость обусловлена тесным взаимным влиянием вариантов распределения всех трех компонентов – функций (алгоритмов или их фрагментов), программ и данных. Это резко усложняет задачу и поэтому в практических приложениях можно рассчитывать лишь на приближенные решения.

Третьим фактором является архитектура МВС или более полно – ***вычислительная среда***, в которой будет выполняться программная нагрузка. Вычислительная среда включает элементы технических средств, входящих в МВС, с описанием типов устройств и их характеристик, структуру связей между устройствами и правила их взаимодействия. Разработка вычислительной среды должна быть согласована с программной нагрузкой, что предполагает, в частности, отсутствие так называемых узких мест. Под узким местом понимается какой-либо ресурс, использование которого относительно других ресурсов имеет существенно большую интенсивность. Ликвидацию узких мест, то есть выравнивание загрузки устройств МВС, можно достигнуть подбором характеристик производительностей соответствующих устройств или изменением варианта распределения функций, программ и данных.

Приведенные выше три фактора тесно связаны друг с другом и при минимизации времени выполнения программной нагрузки должны учитываться совместно. Сложность возникающей при этом задачи делает невозможным ее комплексное решение. Поэтому при определении времени выполнения программной нагрузки соответствующие задачи приходится решать последовательно.

Активные модели и анализ характеристик алгоритмов

Ранее в разделе 3.3 отмечалось, что для анализа алгоритмов прикладных функций СРВ и определения характеристик используется имитация выполнения данных алгоритмов на виртуальной машине. Так как в рассматриваемом варианте виртуальная машина реализована программно, то в последующем ее будем именовать как программная виртуальная машина (ПВМ). Имитация работы СРВ заключается в выполнении модели алгоритмов прикладных функций на модели МВС с помощью ПВМ. Модели, способные выполняться на ПВМ, в отличие от традиционных (пассивных) [5], ***названы активными***.

Активные графовые модели выступают в роли программ для ПВМ, а их выполнение на ПВМ имитирует работу соответствующего алгоритма. В этой схеме напрашивается вариант, в котором ПВМ выступает в роли виртуального процессора, используемого в МВС. Если при этом функции устройств МВС представить алгоритмами, имитирующими их функционирование в форме пригодной для восприятия ПВМ, то мы получим ***активную имитационную модель МВС*** [32, 40].

Преимущества такого подхода очевидны. При наличии библиотеки имитаторов компонентов возможных архитектур МВС можно путем композиции имитаторов и виртуальных процессоров строить модели МВС разнообразной архитектуры. Качество таких моделей будет определяться качеством имитаторов и их функциональной полнотой относительно состава устройств МВС, алгоритмов их функционирования, характеристик используемых локальных вычислительных сетей.

Важным достоинством такого подхода является возможность легко строить активные модели различного уровня детализации. Для этого достаточно иметь набор соответствующих имитаторов и в зависимости от стадии эволюции моделей в процессе проектирования СРВ использовать соответствующие имитаторы. Очевидно, что на начальных стадиях эволюции можно довольствоваться низким уровнем детализации, что позволяет избежать больших затрат времени на моделирование. Полученная таким образом модель МВС легко трансформируется в процессе эволюции и согласования с моделью программной нагрузки.

4.3. Графовые модели алгоритмов прикладных функций СРВ

Графовые модели алгоритмов и программ рассматриваются как одна из форм структурно - графических представлений. Среди таких представлений можно выделить следующие группы. Первая из них соответствует графовым моделям, получаемым в результате преобразования программ в графовую форму представления [36]. Вторая группа соответствует представлению алгоритма совокупностью фрагментов в соответствии со схемой построения модели сверху вниз [18, 23]. Третья группа соответствует языковым формам представления алгоритмов [41, 42]. Здесь возможны варианты, когда разрабатываются наборы элементарных алгоритмических функций, а графовая форма несет информацию о структурной компоненте семантики, связывая алгоритмические функции в графовую модель алгоритма. В работах [43, 44] графовая форма совмещается с элементарными алгоритмическими функция-

ми, реализованными на каком-либо из известных языков программирования или ассемблере.

В последующем развиваются первая и вторая формы представления. Первая в ситуациях, когда требуется определить временные характеристики локальных алгоритмов или их фрагментов. В ситуациях, когда алгоритмы не описаны на языке программирования, используется вторая форма представления. Оценки времени выполнения алгоритмов и их фрагментов в этом случае принимаются приближенными и, следовательно, такие модели могут использоваться для предварительных приближенных оценок прикладных функций и программной нагрузки СРВ. Заметим также, что для первой и второй форм представления принципиально важным является возможность использования одной формы графовой модели. Поэтому в последующем изложении данного раздела основное внимание сосредоточено на разработке такой формы графовой модели, которая в наибольшей мере соответствует целям задач, решаемых при моделировании СРВ, и требованиям эволюционного подхода к проектированию.

Требования к моделям прикладных функций

В общем случае построение моделей прикладных функций СРВ осуществляется в условиях наличия соответствующих алгоритмов, реализованных без учета временной специфики СРВ и знания архитектуры МВС, на которой будут выполняться данные алгоритмы. Исходя из этих предпосылок модели прикладных функций и программной нагрузки согласно основному своему назначению должны отвечать двум требованиям.

Первое из них сводится к тому, чтобы модель можно было легко трансформировать в процессе эволюции и согласования с архитектурой МВС. Выполнение данного требования дает возможность адаптации программ СРВ к ее функциональным особенностям, продиктованным ограничениями реального времени и спецификой вычислительной среды.

Второе требование обусловлено тем, что модель должна быть активной, то есть выступать в роли программы для виртуальной машины, и выполняться точно так же, как программная нагрузка может выполняться на инструментальной ЭВМ. Выполнение данного требования делает возможным реализацию на виртуальной машине имитационного способа моделирования работы СРВ.

Виртуальная машина, в частности программная виртуальная машина (ПВМ), в общем случае вводится в состав крупных программных

систем для придания им свойств и возможностей, которые нельзя обеспечить для пользователя при реализации системы непосредственно на инструментальной ЭВМ. В зависимости от ориентации программной системы меняются язык, функции и структура ПВМ. В нашем случае ПВМ используется в моделирующей системе, в которой объектом моделирования является программная нагрузка на МВС. Поэтому языком ПВМ должен быть язык представления алгоритмов в форме программ. В таком случае естественным для представления алгоритмов является использование любого языка программирования, например языка Паскаль или Си. В этом случае программную нагрузку можно выполнять на инструментальной ЭВМ и оценивать характеристики. При использовании такого варианта модели осуществить ее эволюцию не представляется возможным. Эволюционный путь согласования моделей программной нагрузки и архитектур МВС выдвигает ряд требований:

- для анализа очередного нового варианта модели архитектуры МВС модель программной нагрузки не должна перепрограммироваться;
- модель программной нагрузки должна легко трансформироваться и формальными методами согласовываться с архитектурой МВС в единую динамическую модель СРВ;
- должна быть обеспечена возможность адекватного отображения на программную нагрузку трансформаций модели.

Очевидно, что текстовые языки высокого уровня не отвечают указанным требованиям. Поэтому при разработке языка ПВМ принята стратегия, согласно которой алгоритмы программной нагрузки могут быть представлены на языке высокого уровня, например, на языке Паскаль с последующим преобразованием их в графовую модель, пригодную для выполнения трансформаций и для восприятия ПВМ и пользователем.

Графовая модель строится на основе представления Паскаль – программы в виде иерархической структуры фрагментов. Построение графовой модели включает решение ряда задач:

- структурный анализ Паскаль – программ и формирование фрагментов;
- свертка фрагментов с учетом их вложенности;
- хранение и отображение графовой модели на экране монитора.

Для выполнения графовой модели на ПВМ каждый фрагмент автономно транслируется в программу для получения загрузочного модуля. Каждый модуль соответствует минимальному дискретному элементу, используемому в моделирующей системе. Таким образом, модель

алгоритма в виде иерархической структуры фрагментов полностью соответствует принципу дискретности. На нижнем уровне этой структуры расположены базовые фрагменты – команды ПВМ, выше располагаются фрагменты более высоких уровней свертки.

Виды графовых форм представления алгоритмов

Графовая форма представления алгоритмов в виде иерархической структуры включает три вида графов. Первый вид представляет собой дерево процедур и функций, к каждой ветви которого привязано дерево структурных операторов соответствующей процедуры или функции. Пример таких деревьев представлен на рис. 4.2. Второй вид – это граф управляющих связей. Вершинами в нем являются фрагменты алгоритма, а дуги соответствуют управляющим связям.

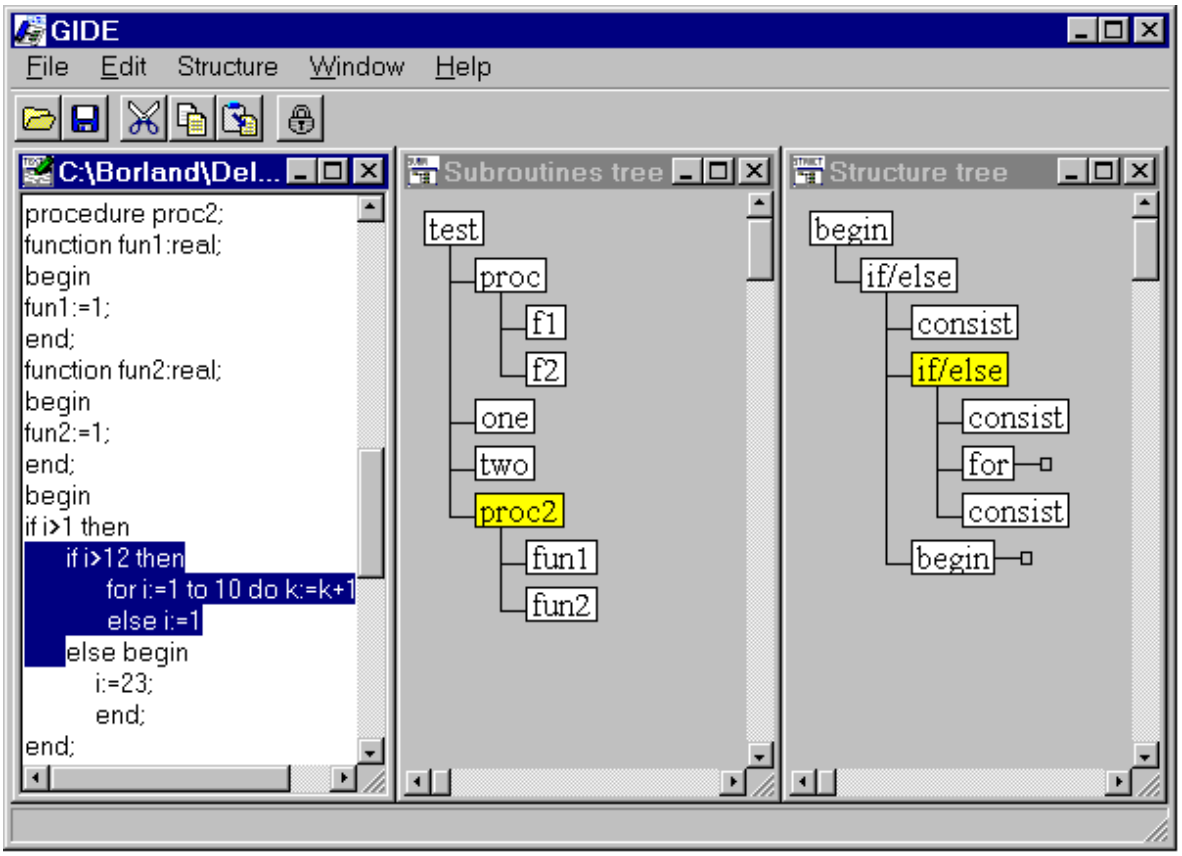


Рис.4.2. Граф процедур и функций

На рис. 4.3 представлен пример алгоритма в форме графа управляющих связей. Все вершины графа (операторы) поименованы. Это позволяет наглядно представить процесс свертки фрагментов алгоритма, в результате которых алгоритм преобразуется в иерархическую структуру фрагментов (модулей).

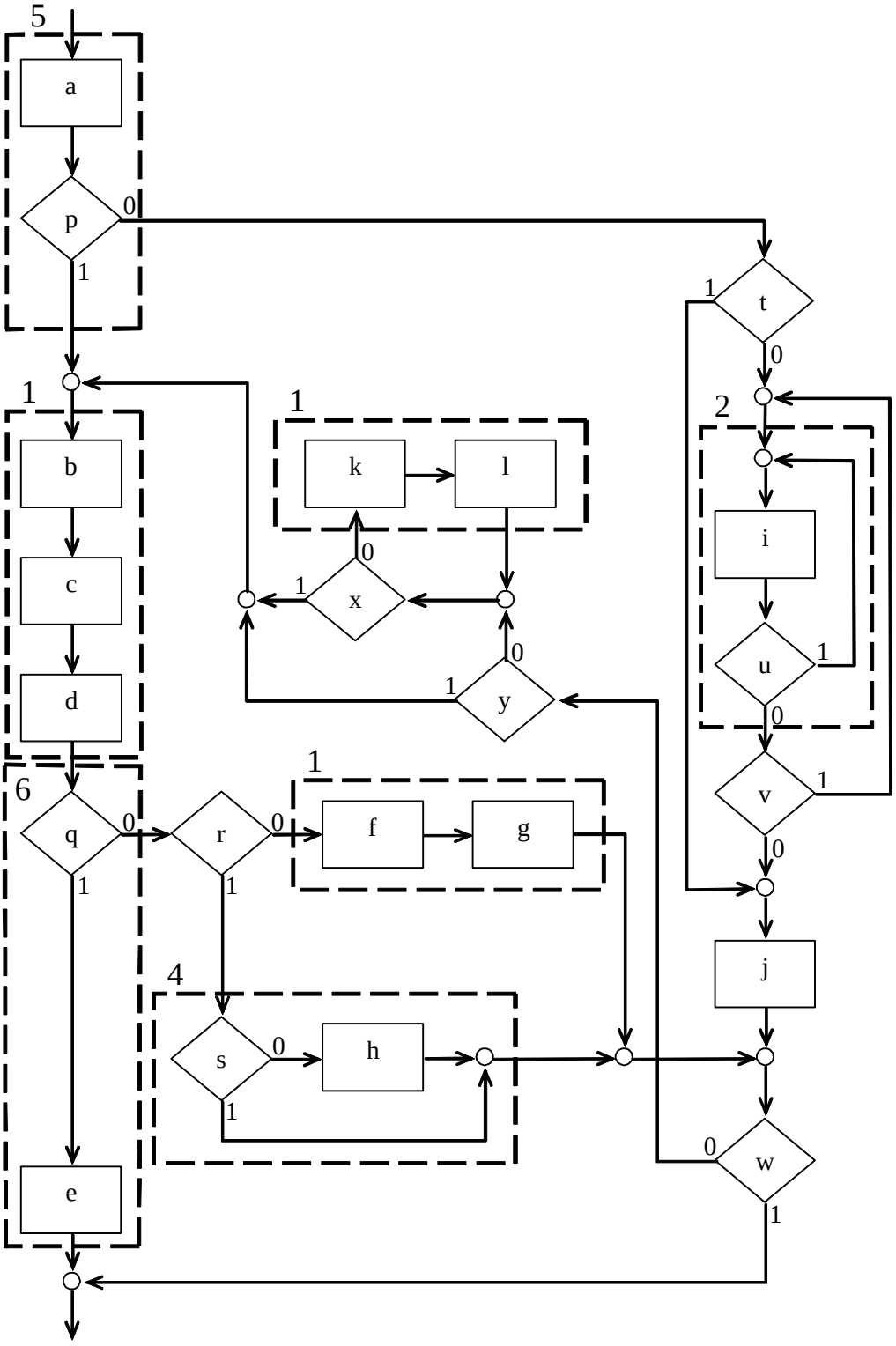


Рис. 4.3. Пример алгоритма в форме графа управляющих связей

Примеры базовых фрагментов алгоритмов и результаты выполнения операций свертки над ними представлены на рис. 4.5.

Процесс полной свертки алгоритма (рис. 4.3) представлен на рис. 4.4 a, b, c, d, e, f. При этом фрагменты на каждой итерации свертки {a, b, c, d, e, f } выделены пунктирными линиями и пронумерованы в соответствии с нумерацией фрагментов, принятой на рис. 4.5.

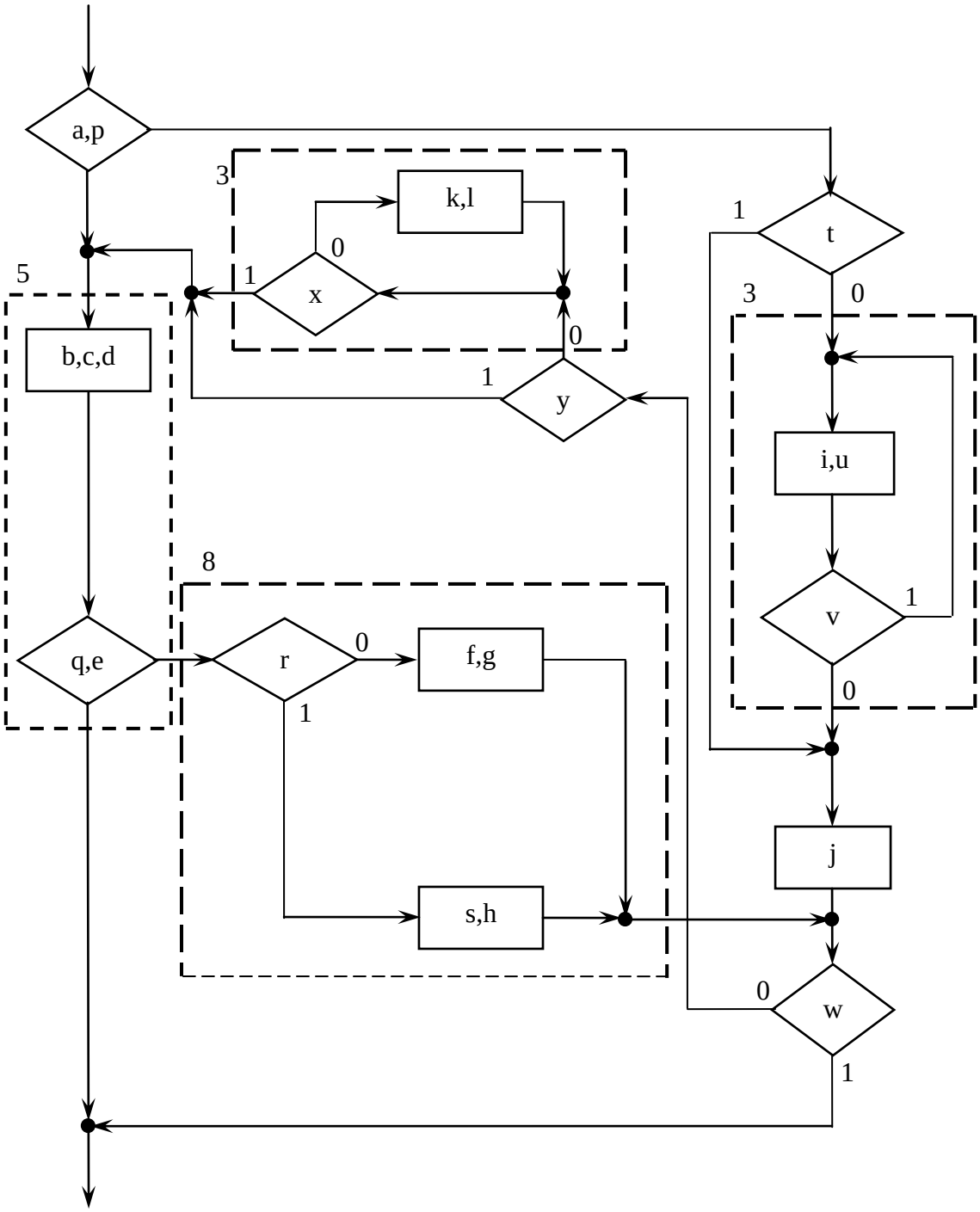


Рис. 4.4: а – первая итерация свертки

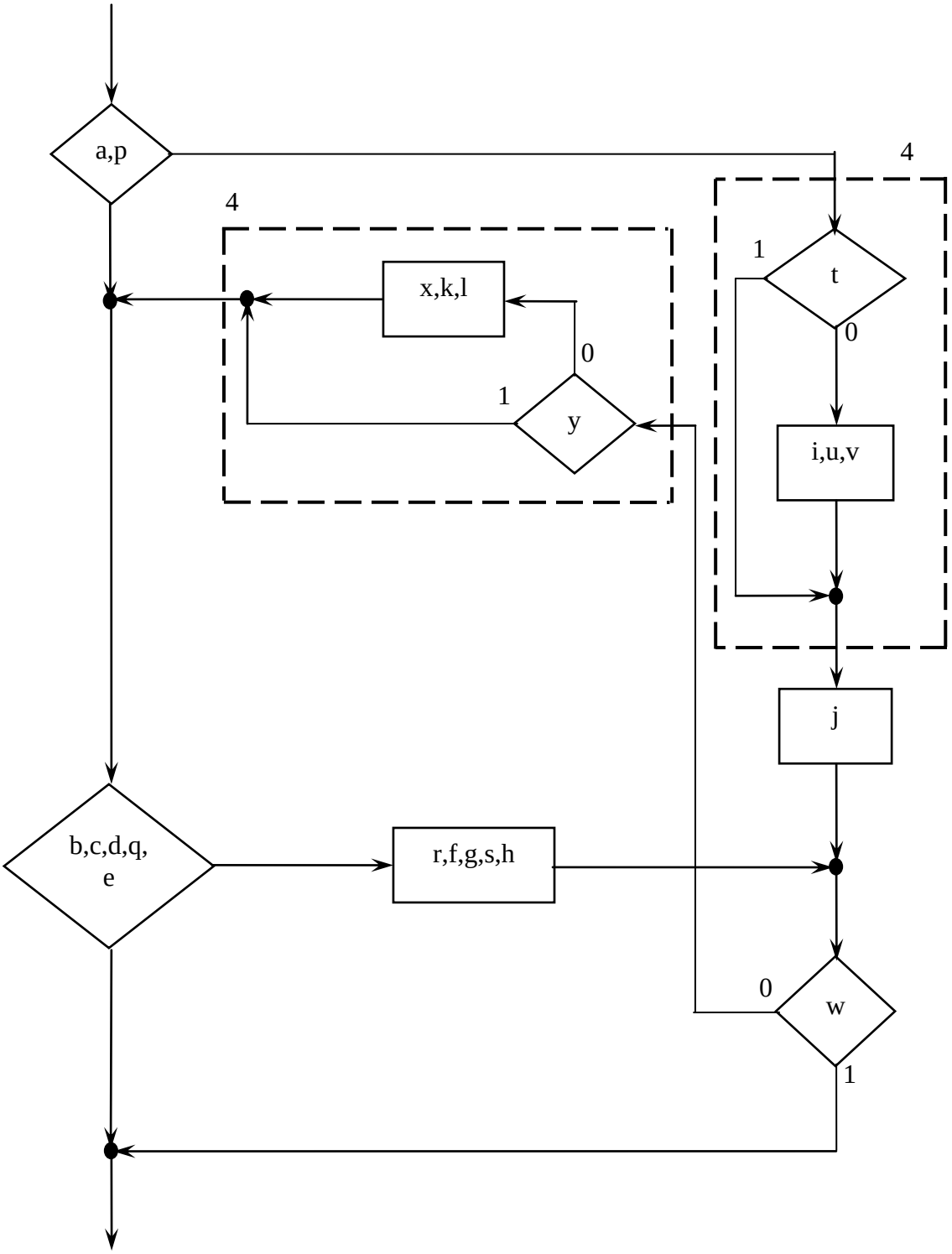


Рис. 4.4: b – вторая итерация свертки

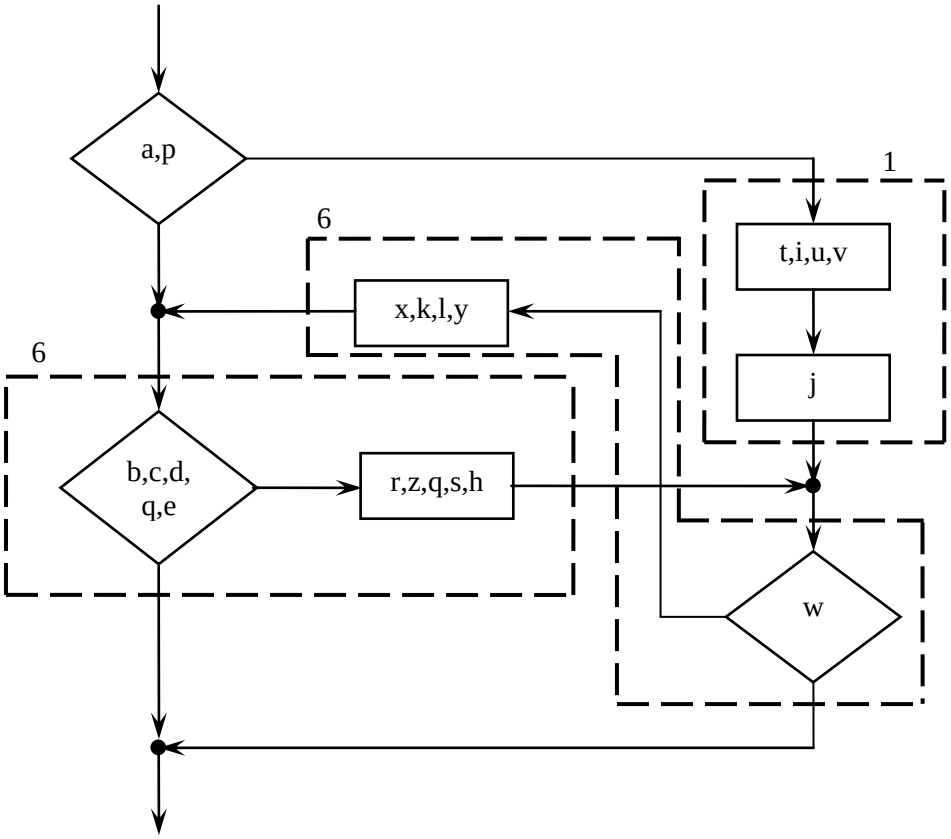


Рис. 4.4: c – третья итерация свертки

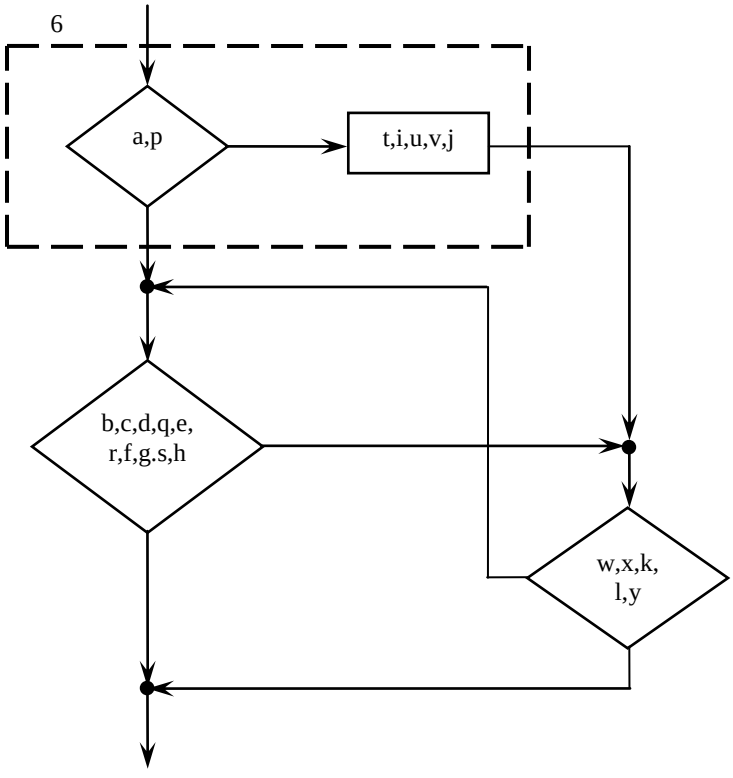


Рис. 4.4: d – четвертая итерация свертки

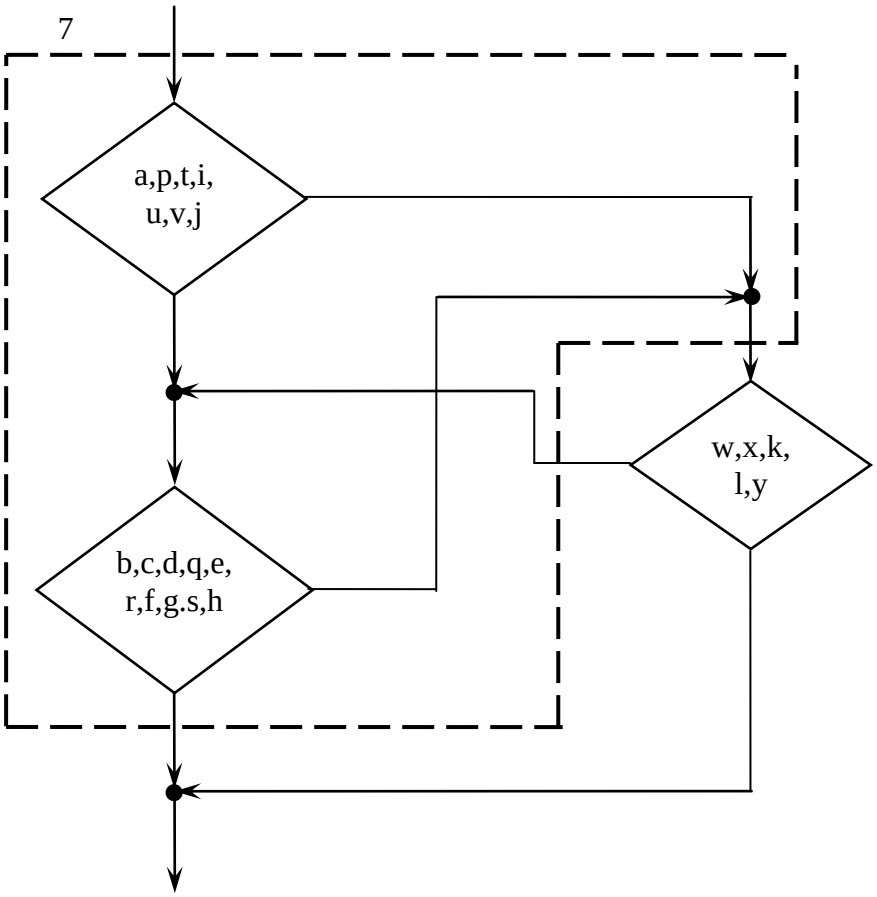


Рис. 4.4: e – пятая итерация свертки

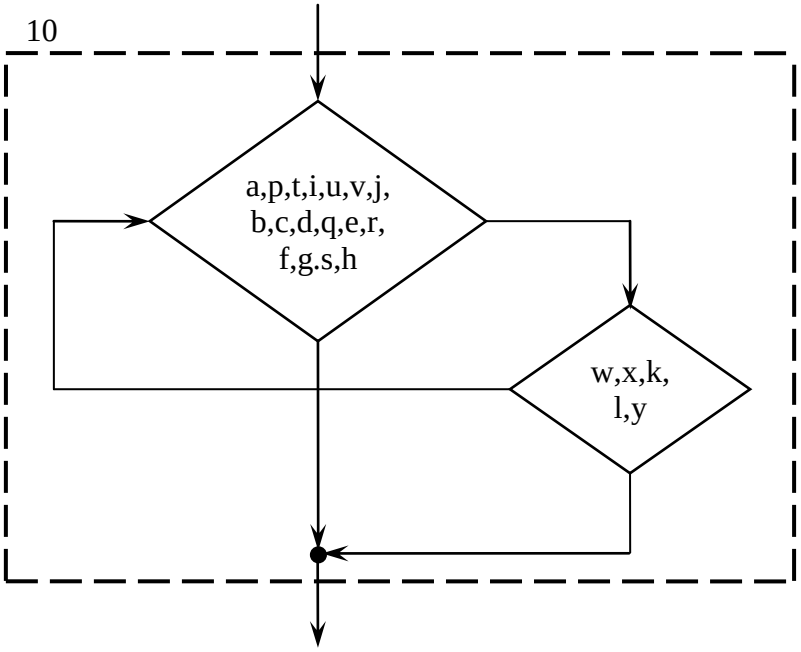


Рис. 4.4: f – шестая итерация свертки

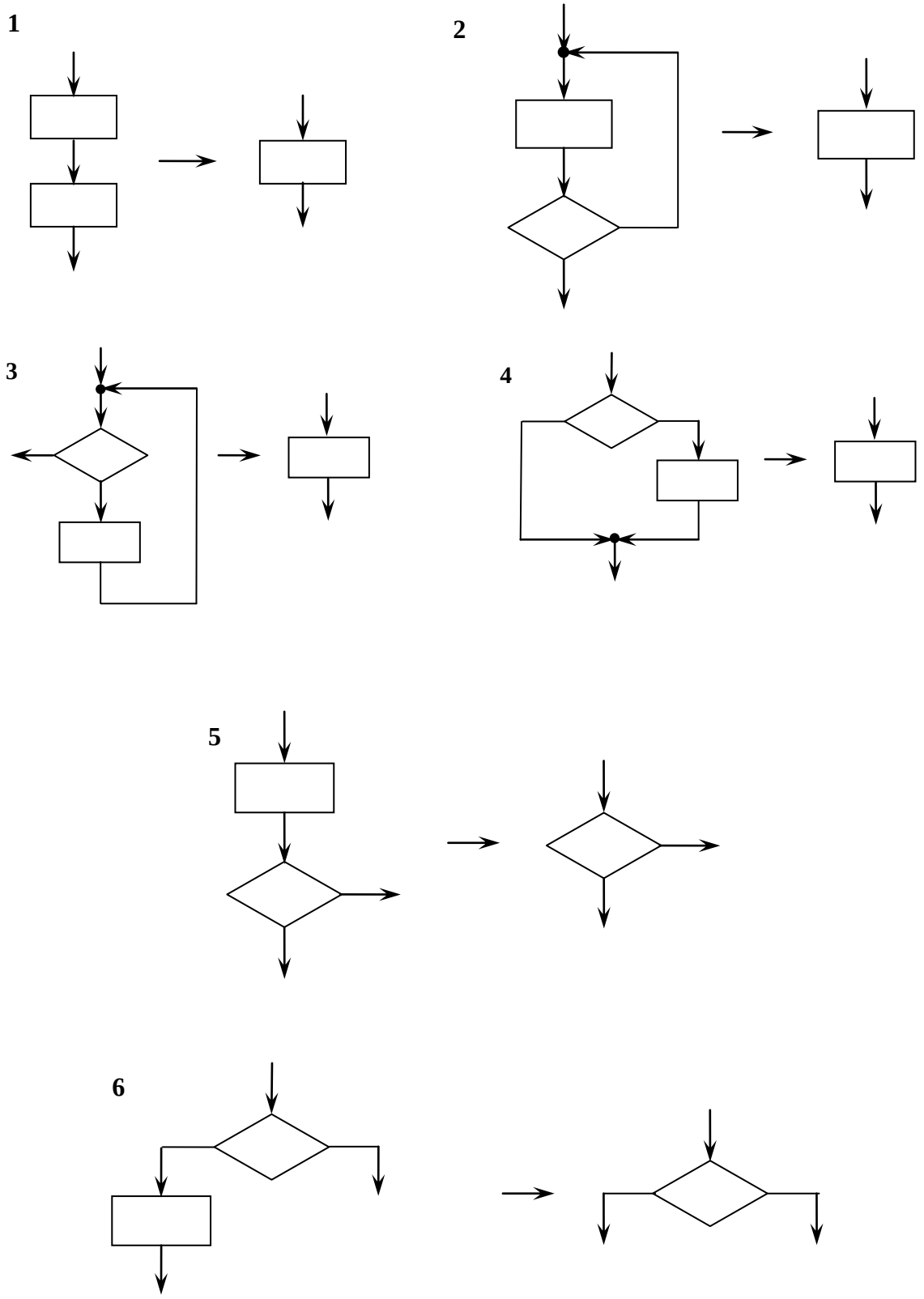


Рис. 4.5. Примеры фрагментов алгоритмов для выполнения операций свертки (фрагменты 1 – 6)

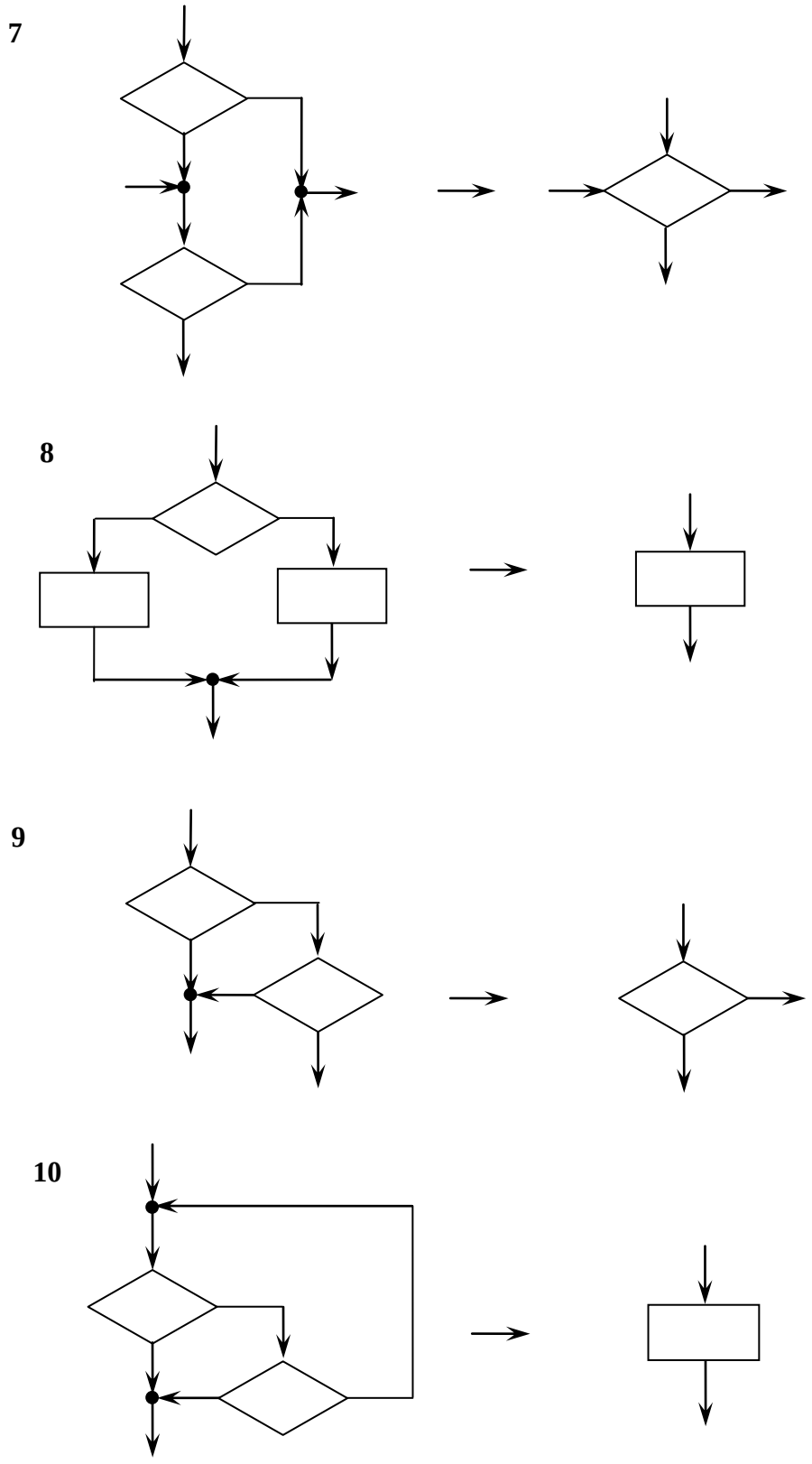


Рис. 4.5. Примеры фрагментов алгоритмов для выполнения операций свертки (фрагменты 7 –10)

Таким образом, процесс свертки графа управляющих связей (рис. 4.3) позволяет преобразовать его в граф с одной вершиной (рис. 4.4 f) либо в граф, представленный на одной из итераций свертки {a, b, c, d, e, f }. Заметим также, что процесс свертки не является однозначным, так как на каждой итерации может быть использован тот или иной фрагмент алгоритма.

Третий вид графа раскрывает взаимосвязи фрагментов и данных и именуется графом потока данных (ГПД) (рис. 4.6). В функциональном графе (рис. 4.6 а) два вида вершин – фрагменты и данные. Фрагменты, по аналогии с сетями Петри, изображаются планками, а данные кружками. В информационном графе (рис. 4.6 б) отображаются связи только между данными. Дуги в данном графе связывают используемые и формируемые данные. В методах анализа в основном будет использоваться функциональный граф потока данных.

Процедуры свертки для графов всех видов полностью согласованы. Это означает, что переход в графе управляющих связей на более высокий или низкий уровень свертки автоматически приводит к соответствующим изменениям в графе потока данных. Граф управляющих связей используется для визуального анализа программной нагрузки, разбиения на фрагменты и анализа их входных и выходных данных, а также многих других сервисных приложений.

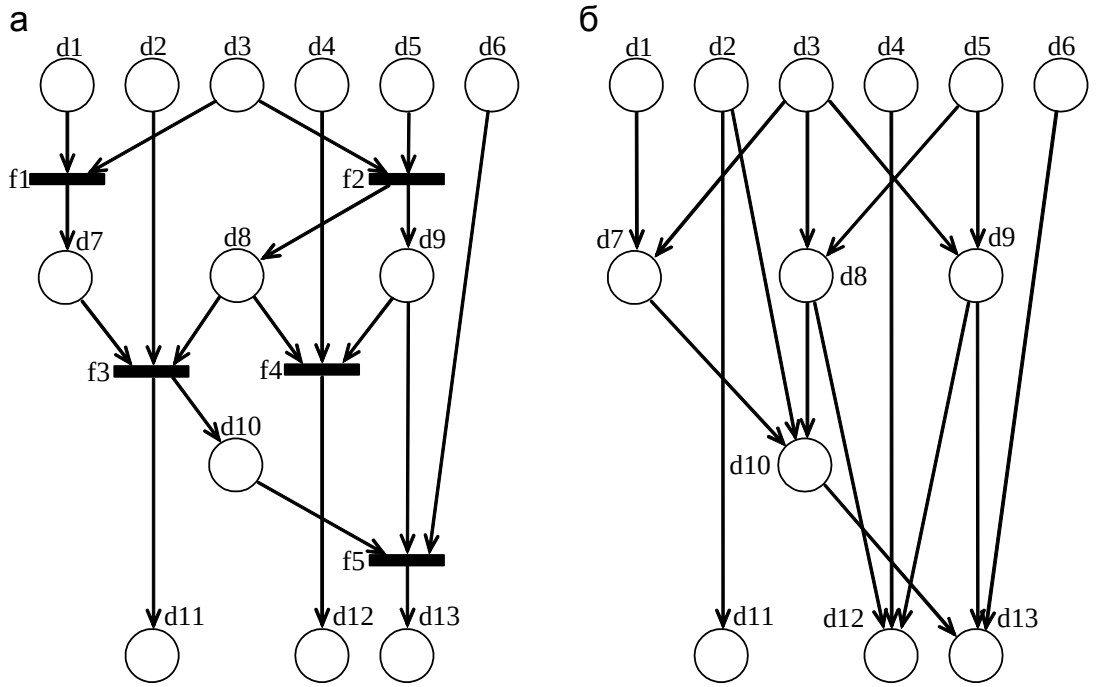


Рис. 4.6. Граф потока данных: а – функциональный; б – информационный

Основным носителем модельного представления алгоритмов программной нагрузки, с которым работает пользователь, является граф потока данных. Над ним выполняются основные процедуры по эволюции программной нагрузки и согласования ее с архитектурой МВС. Заметим, что используемые при этом уровни свертки устанавливаются пользователем в зависимости от числа алгоритмов программной нагрузки и их сложности. При необходимости более детального анализа конкретного алгоритма может приниматься более низкий уровень свертки.

Функциональный граф потока данных (ГПД) может быть получен двумя способами. Первый из них соответствует получению функционального ГПД на основе дерева процедур и функций Паскаль–программы. Второй способ соответствует построению функционального ГПД на основе схемы декомпозиции алгоритмов прикладных функций сверху вниз. В этом случае алгоритм прикладной функции представляется совокупностью фрагментов (алгоритмических модулей), для каждого из которых определяются входные и выходные данные и выполняемые функции. Совокупность таких модулей оформляется библиотекой и может использоваться в последующем при представлении новых алгоритмов в форме функциональных ГПД.

4.4. Модель архитектуры многопроцессорной вычислительной системы

В локальных вычислительных сетях устройства, выполняющие функции и обладающие ресурсом памяти, принято называть станциями. По аналогии с этим микроЭВМ, входящие в МВС, также будем именовать станциями. Построение вычислительных систем на базе типовых станций нашло широкое распространение при проектировании СРВ. При этом рассматриваются три принципиально отличающиеся друг от друга типа сети:

- магистраль с несколькими возможными протоколами доступа к ней станций;
- парная связь между станциями;
- несколько магистралей нижнего уровня и магистраль верхнего уровня, по которой организована связь между магистралями нижнего уровня.

Сети первого типа характеризуются тем, что в каждый момент времени передача данных может осуществляться только одной станцией.

В сетях второго типа в общем случае не все пары станций непосредственно связаны между собой. В этом случае данные передаются

между станциями по цепи, включающей несколько станций. Такая цепь для конкретной сети не всегда является единственной.

Что касается сетей третьего типа, то они могут рассматриваться как комбинация сетей первого и второго типов. Если магистрали нижнего уровня рассматривать в качестве станций, то магистраль верхнего уровня связывает такие станции в сеть по второму типу. Станции, связываемые магистралью нижнего уровня, образуют сеть первого типа.

Для построения модели архитектуры МВС разработаны имитаторы компонентов архитектуры, которые оформлены в библиотеку. Модель исходного варианта архитектуры МВС изображается в форме рисунка путем композиции отдельных компонентов из библиотеки компонентов архитектур. Каждая компонента архитектуры сопровождается необходимыми параметрами и имитаторами, настроенными на данные параметры. Таким образом, при изображении рисунка архитектуры МВС из имитаторов соответствующих компонентов автоматически генерируется модель архитектуры МВС. Инструментальное программное средство, реализующее данные задачи, разработано в форме «Редактора архитектур».

Редактор предоставляет пользователю возможности по функциональному синтезу архитектуры и описанию характеристик ее отдельных компонентов в интерактивном режиме. Необходимость такого синтеза возникает при решении задач анализа программной нагрузки, выполняемой на заданной архитектуре МВС. Поскольку предполагается, что в процессе эволюции архитектура может многократно трансформироваться, основным требованием к редактору является возможность визуального построения и описания характеристик таких архитектур удобным для пользователя способом. Также очевидна необходимость эффективного инструмента для выбора из множества архитектур той, на которой в текущий момент будут выполняться алгоритмы прикладных функций СРВ. Данный круг задач и призван решать предлагаемый редактор.

Программно редактор реализован на языке Borland Delphi 4.0 для операционной системы Windows 95/NT, что делает программу инвариантной по отношению к устройствам инструментальной ЭВМ, дает большие возможности по ведению активного диалога с пользователем, используя стандартные элементы управления, а также наделяет редактор рядом свойств характерных для Windows-программ. На рис. 4.7 представлен пример архитектуры, построенной с помощью редактора.

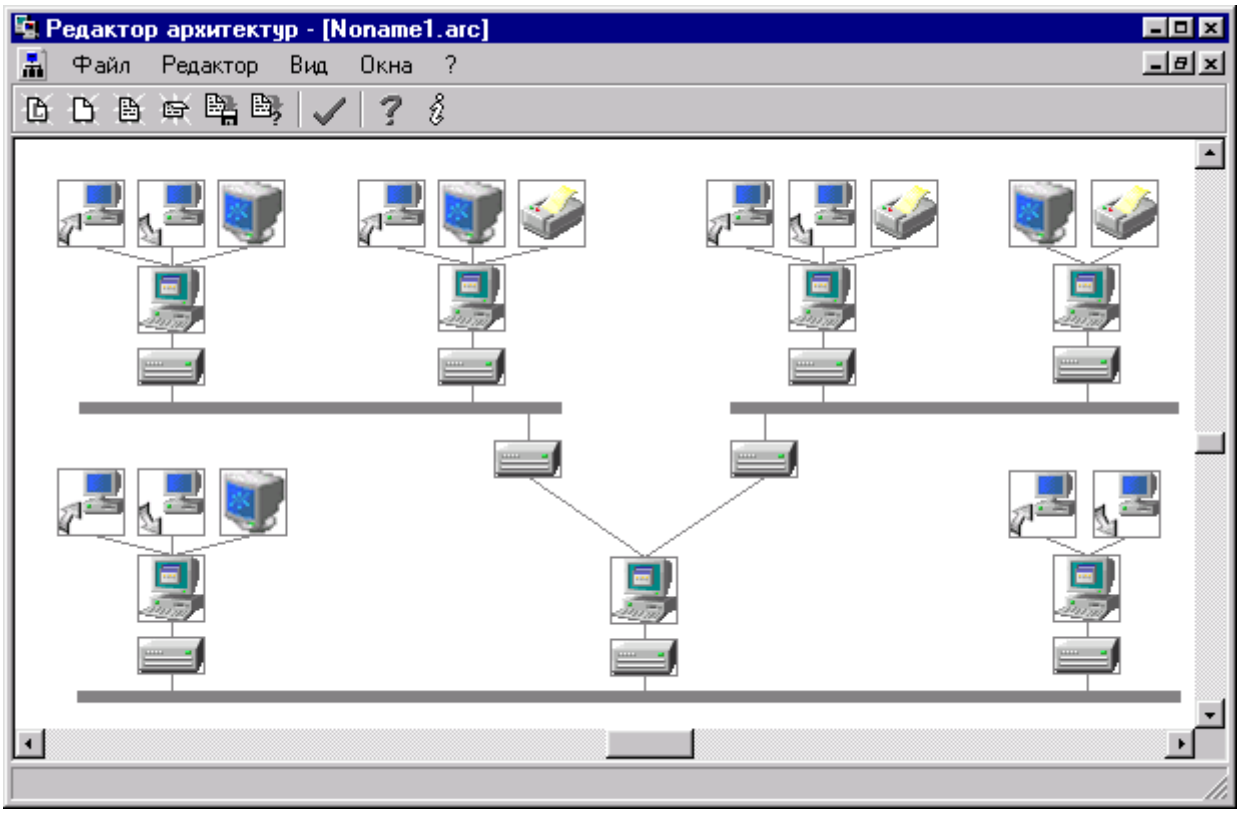


Рис. 4.7. Пример построения архитектуры

Ниже перечислены визуальные элементы "Редактора архитектур" описывающие инструментальные средства проектируемой архитектуры МВС и используемые для ее визуального синтеза. Для каждого элемента приведен перечень возможных параметров, определяющих его характеристики, и краткое их описание.

1. Станция (рабочая станция в сетевом варианте архитектуры):
   - Имя. Состоит из префикса "Станция" и следующего далее порядкового номера, обеспечивающего уникальность имени в пределах редактируемого описания архитектуры;
   - Тип ЦПУ. Выбор типа ЦПУ используемого в рабочей станции из предложенного ряда;
   - Математический процессор. Определение наличия в системе математического процессора для работы с вещественными числами;
   - Тактовая частота. Определение тактовой частоты работы процессора;
   - Время простоя. Определение процессорного времени, затрачиваемого на выполнение работы, не связанной с обработкой основной программы, например: обработка прерываний от таймера, клавиатуры, выполнение фоновых задач и т.д. Данная характеристика

выражается в процентном отношении к общему времени работы процессора;
- Размер ОЗУ. Определение размера оперативной памяти доступной для работы алгоритмов СРВ;
- Выборка ОЗУ. Определение времени выборки при доступе к оперативной памяти;
- Размер диска. Определение размера дисковой памяти доступной для работы алгоритмов СРВ;
- Скорость диска. Определение скорости дисковых операций ввода/вывода.

2. Адаптер (сетевой адаптер, реализующий протокол обмена в сети):
   - Имя. Состоит из префикса "Адаптер" и следующего далее порядкового номера;
   - Размер буфера. Определение размера буфера ввода/вывода доступного для формирования очереди пакетов;
   - Размер пакета. Определение размера пакета принятого для работы в сети;
   - Скорость. Определение скорости обмена информацией между адаптером и рабочей станцией.
3. Шина (определяет топологию сети):
   - Имя. Состоит из префикса "Шина" и следующего далее порядкового номера;
   - Скорость. Определение скорости передачи на описываемом участке сети;
   - Время простоя. Определение сетевого времени, затрачиваемого на выполнение работы, не связанной с пересылкой полезной информации. Данная характеристика выражается в процентном отношении к общему времени работы сети.
4. Устройства ввода/вывода (устройства, следящие за изменениями в окружающей системе в процессе функционирования СРВ и устройства формирования управляющих воздействий на исполнительные механизмы системы соответственно):
   - Имя. Состоит из префикса "Ввод/Вывод" и следующего далее порядкового номера;
   - Порт. Определяет адрес порта инструментальной ЭВМ, к которому подключено данное устройство. Предусмотрена возможность ввода значений в шестнадцатеричном виде, используя префикс – $;

- Интерфейс. Определяет интерфейс описываемого устройства. Возможны следующие значения: параллельный, последовательный, непосредственное соединение;
- Скорость. Определяет скорость обмена информацией между устройством и инструментальной ЭВМ. В случае непосредственного соединения данный параметр игнорируется, поскольку время обмена равно времени выполнения машинных команд in/out.

5. Дисплей (устройство отображения информации):
   - Имя. Состоит из префикса "Дисплей" и следующего далее порядкового номера;
   - Тип. Определяет тип видеоадаптера.
6. Принтер (печатающее устройство):
   - Имя. Состоит из префикса "Принтер" и следующего далее порядкового номера;
   - Порт. Определяет имя порта инструментальной ЭВМ, к которому подключено данное устройство;
   - Скорость порта. Определяет скорость обмена информацией с инструментальной ЭВМ через выбранный порт;
   - Размер буфера. Определяет размер буфера, имеющегося у печатающего устройства;
   - Скорость работы. Определяет скорость выводы информации печатающим устройством.
7. Соединение. Определяет соединение между перечисленными выше функциональными элементами архитектуры. При задании соединения автоматически проводится его контроль на соответствие принятым правилам, что необходимо для однозначной интерпретации построенной архитектуры.

Следует отметить, что скоростные характеристики устройств не описывают их динамических характеристик. Данные характеристики будут задаваться в последствии выбором имитаторов технических устройств. Временные оценки работы устройств позволяют системе оценить ресурс архитектуры по быстродействию, а также отдельных ее компонентов, в частности, для активной помощи пользователю в решении задачи распределения функций, программ и данных по ресурсам выбранной архитектуры МВС.

4.5. Модель алгоритма функционирования окружающей системы

Построению модели окружающей системы предшествует решение задач по определению состава активных компонентов, состава и типов датчиков и исполнительных механизмов и оптимизации топологии общей структуры системы с учетом размещения станций МВС. Должны быть определены также условия поступления внешних воздействий сигналов управления, временных параметров. Для систем с переменной структурой должен быть определен сценарий внесения изменений в структуру окружающей системы.

Модель алгоритма функционирования окружающей системы строится в форме ГПД. Модули ГПД соответствуют алгоритмам функционирования активных компонентов окружающей системы и алгоритмам выполнения вспомогательных функций, необходимых для построения модели. Модули вспомогательных функций, в свою очередь, делятся на два вида.

Первый вид определяет модули, которые работают по неизменному алгоритму, формируя воздействия на другие модули или данные для ГПД прикладных функций, то есть являются генераторами.

Модули второго вида находятся в зависимости от воздействий других модулей, вычисляют показания датчиков и формируют воздействия на другие модули.

Модули-генераторы могут формировать заявки на обслуживание, сигналы на изменения параметров модулей, сигналы на включение и выключение модулей. Последнее используется для моделирования окружающих систем с переменной структурой, то есть систем, в которых имеет место динамизм 2-го рода. Сигналы об изменении структуры системы можно рассматривать как заявки специального содержания, извещающие о выходе из строя отдельного компонента либо, наоборот, о его восстановлении. В этих случаях модули-генераторы запускают модули с функциями «восстановления» совокупности связей с соответствующими модулями.

Сигналы модуля-генератора на изменение структуры окружающей системы могут также подключать новые модули с заданными значениями параметров и совокупностью связей с другими модулями либо исключать модули без возможности их восстановления.

Рассмотрим более полно варианты модулей для построения моделей алгоритмов функционирования окружающей системы. Используется четыре вида модулей:

- модули имитаторы компонентов системы (К);

- модули для имитации вспомогательных функций системы (Ф);
- модули для генерации моментов наступления событий (Г);
- модули для имитации системного таймера (Т).

В ГПД модули могут иметь три вида изображения, представленные на рис. 4.8.

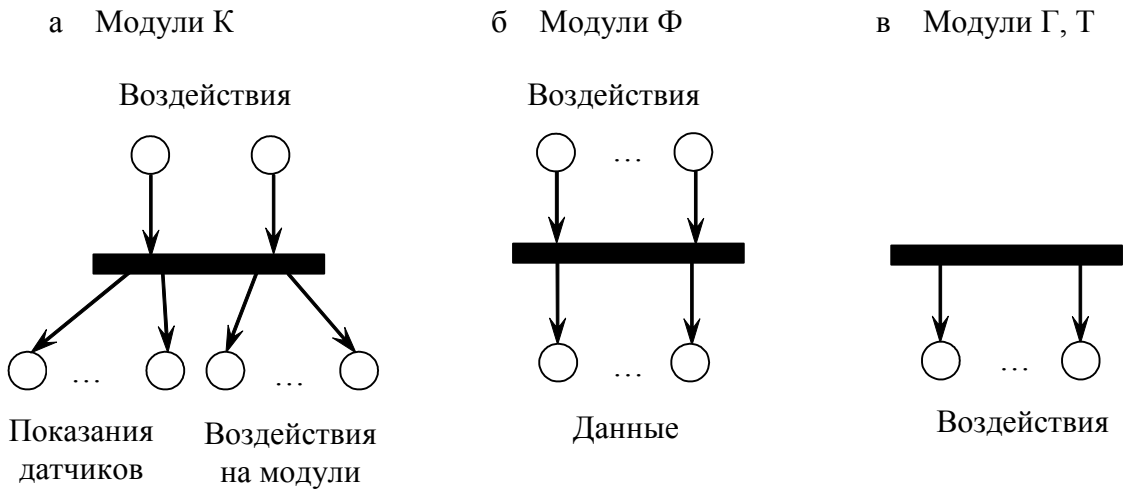


Рис. 4.8. Варианты изображения модулей

Модули имитаторы компонентов окружающей системы (К) представлены на рис. 4.8 а). Основной функцией данного модуля является формирование показаний датчиков, которые поставлены в зависимость от функционирования данного компонента. Входными воздействиями модуля К являются команды на изменения состояний компонента, его параметров и команды на изменения алгоритма функционирования. На выходе модуля К помимо показаний датчиков могут формироваться воздействия по изменению состояний параметров и алгоритмов функционирования других компонентов.

Модули для имитации вспомогательных функций (Ф) (рис. 4.8 б) формируют данные, используемые в качестве входных для модулей в ГПД прикладных функций встроенных систем. Модули Ф по выполняемым функциям можно разделить на три группы:

- формирование событий, например, формирование значений параметров заявок;
- анализ состояний встроенной системы, включая функции тестирования, контроля работоспособности компонентов, выявления предаварийных состояний;

- редактирование структуры встроенной системы, то есть редактирование ГПД алгоритма функционирования встроенной системы.

Модули генерации моментов наступления событий (Г) (рис. 4.8 в) формируют сигналы запуска других модулей. Модули Г работают по жесткому алгоритму генерации случайных величин, распределенных по заданному закону. Как правило, модули Г не имеют входов, но при необходимости могут предусматривать входы для настройки параметров либо входы запуска от других модулей.

Модули для имитации системного таймера (Т) (рис.4.8 в). Данный модуль формирует сигналы для запуска других модулей в моменты времени, определяемые регламентом работы окружающей системы. Модуль Т чаще всего реализует неизменные во времени правила формирования сигналов. В этом случае модуль Т не использует входы. В случае, если для вычисления момента времени требуется дополнительная информация, то используются модули Т с входами, либо модули Ф.

Пример, иллюстрирующий применение перечисленных выше модулей, представлен на рис. 4.9.

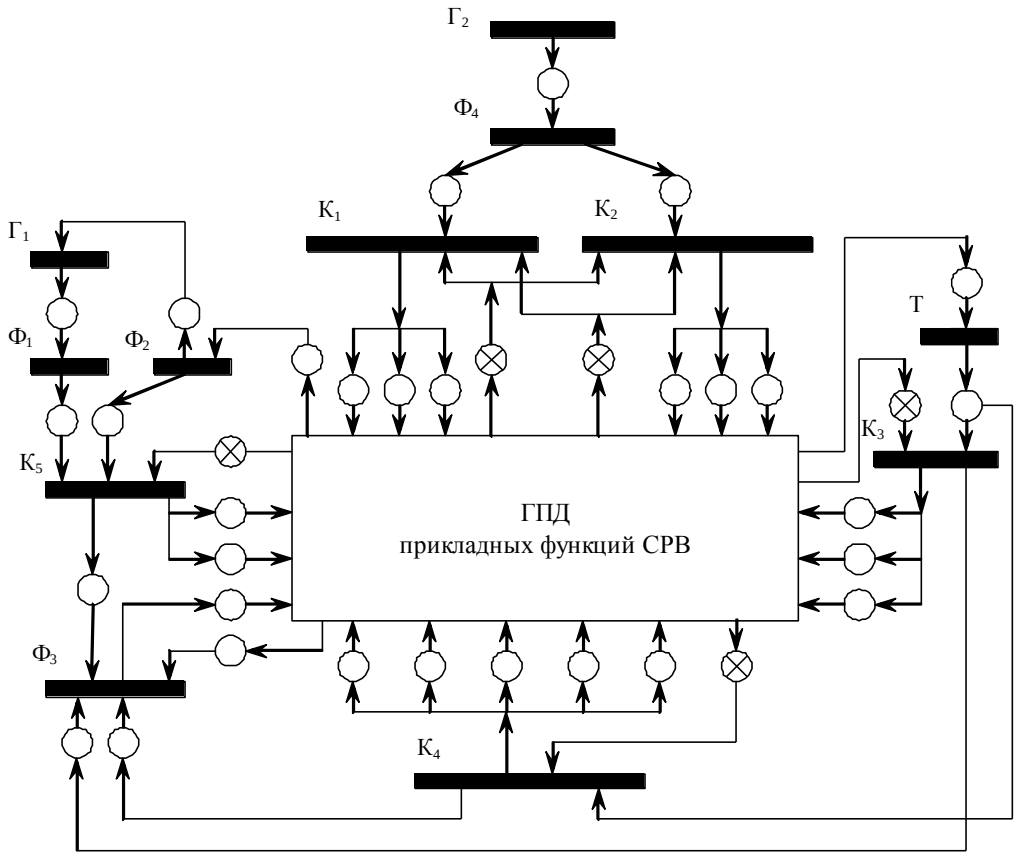


Рис. 4.9. ГПД алгоритма функционирования окружающей системы

На рис. 4.9 изображена модель алгоритма функционирования окружающей системы в форме ГПД. Встроенная система и ее связь с окружающей системой представлена в виде ГПД прикладных функций. Окружающая система включает пять активных компонентов – модулей $К_1$ – $К_5$. При этом модуль $К_5$ подключается к окружающей системе, изменяя ее структуру. Информацию о необходимости подключения формирует ГПД прикладных функций. Соответствующий сигнал запускает модуль $Ф_2$, который осуществляет подключение модуля $К_5$ и запускает модуль $Г_1$. Модули $Г_1$ и $Ф_1$ осуществляют загрузку оборудования, имитируемого модулем $К_5$. Загрузку однородного оборудования, имитируемого модулями $К_1$ и $К_2$, осуществляют модули $Г_2$ и $Ф_4$. Модули $К_3$ и $К_4$ запускаются по таймеру модулем Т, который использует дополнительную информацию для вычисления момента времени запуска. Управление модулями $К_1$ – $К_5$ производится через исполнительные механизмы управляющими воздействиями, которые формирует ГПД прикладных функций.

Анализ состояния оборудования, имитируемого модулями $К_3$, $К_4$, $К_5$, выполняет модуль $Ф_3$. Информация о состоянии поступает на ГПД прикладных функций. Запуск модуля $Ф_3$ производится по сигналу, формируемому в ГПД прикладных функций.

Моделирование алгоритма функционирования окружающей системы осуществляется на основе полученного ГПД и имеет некоторую специфику в сравнении с моделированием на основе ГПД, построенных для прикладных функций. Вопросы построения активных моделей и выполнения их на ПВМ излагаются в последующих разделах. Здесь лишь рассмотрим некоторую специфику процесса моделирования. Главное отличие состоит в том, что в данном случае нет необходимости имитировать выполнение ГПД на МВС, так как она не является объектом проектирования. Принимается, что выполнение ГПД может осуществляться на одном процессоре, для которого ресурсы по быстродействию и памяти не ограничиваются. В этих условиях можно считать, что каждый модуль ГПД выполняется на своем процессоре, а время на передачу данных между модулями принимается равным нулю.

Основным результатом моделирования является вычисление показаний датчиков. При циклическом обновлении показаний необходимо организовать работу ГПД так, чтобы показания датчиков обновлялись своевременно. Это означает, что циклы обновления показаний датчиков должны быть установлены исходя из технологических условий функционирования окружающей системы и согласованы с условиями поступления данных на модули прикладных функций встроенной системы.

Для датчиков с ациклическим обновлением показаний должны выдерживаться законы распределения моментов времени поступления таких сигналов. Моделирование моментов времени обновления показаний датчиков должно соответствовать пропорции между системным (модельным) и реальным временем. Такая пропорция устанавливается при комплексном моделировании совместного функционирования встроенной и окружающей систем.

Вопросы для контроля усвоения знаний

1. Раскрыть сущность задачи определения топологии станций и терминальных точек.
2. Сформулировать правило выбора полюсов на топологическом поле.
3. Дать постановку задачи распределения терминальных точек одного вида по станциям.
4. Дать определение компактного множества терминальных точек.
5. Привести формализованную постановку задачи распределения терминальных точек по станциям с учетом разбиения их на компактные множества.
6. Записать постановку задачи определения числа станций как задачи покрытия.
7. С какой целью формируется множество вариантов подключения точек к станциям?
8. Записать постановку задачи распределения точек по заданному числу станций.
9. В чем основной недостаток метода решения задачи распределения точек через формирование множества вариантов подключения?
10. Перечислить основные и производные структурные свойства алгоритмов. Определить термин «томография алгоритмов».
11. Пояснить проблематику анализа характеристик алгоритмов.
12. Пояснить проблемы оценки минимального времени выполнения программной нагрузки на МВС.
13. Пояснить необходимость поиска максимального распараллеливания алгоритмов.
14. Перечислить и пояснить основные факторы, влияющие на время выполнения программной нагрузки.
15. Дать общее смысловое определение активной модели.
16. Перечислить формы структурно-графических представлений алгоритмов.

17. Перечислить основные требования к моделям программной нагрузки.
18. Перечислить виды графовых форм представления алгоритмов.
19. Зачем нужны операции свертки? Привести примеры фрагментов алгоритмов для операций свертки.
20. Пояснить основные задачи редактора архитектуры МВС.
21. Зачем нужна модель окружающей системы?
22. Определить основные функции модулей имитаторов для построения моделей окружающих систем.
23. Пояснить специфику моделирования функционирования окружающей системы.

5. МЕТОДИКА ПРИБЛИЖЕННОЙ ОЦЕНКИ ВРЕМЕНИ ВЫПОЛНЕНИЯ АЛГОРИТМОВ ПРИКЛАДНЫХ ФУНКЦИЙ НА ОСНОВЕ АНАЛИТИЧЕСКИХ РАСЧЕТОВ

5.1. Сокращение времени выполнения программной нагрузки на основе ее распараллеливания

Описание программной нагрузки

Рассмотрим более подробно исходные сведения об алгоритмах программной нагрузки, представленных в форме функциональных ГПД. Примем следующие условия взаимодействия окружающей системы с МВС встроенной системы. Входные данные (информация с датчиков) $\boldsymbol{X}^0 = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_l, ..., \boldsymbol{x}_L\}$ поступают на информационные входы МВС циклически. Для каждого входа $\boldsymbol{x}_l \in \boldsymbol{X}^0$ задается время цикла поступления δ_i из множества $\Delta = \{\delta_1, \delta_2, ..., \delta_i, ..., \delta_V\}$, $\boldsymbol{L} \geq \boldsymbol{V}$. Выходные данные (управляющие воздействия) $\boldsymbol{Y}^0 = \{\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_z, ..., \boldsymbol{y}_Z\}$ должны поступать на исполнительные устройства (или устройства отображения информации) с временами циклов обновления из множества $\Theta = \{\Theta_1^*, \Theta_2^*, ..., \Theta_h^*, ...\Theta_H^*\}$ или ациклически, по мере их формирования, но с ограничением времени их получения. Задано $\boldsymbol{k}$ циклов вывода (обновления) управляющих воздействий с временами $\{\Theta_1^*, \Theta_2^*, ..., \Theta_j^*, ..., \Theta_k^*\}$, распределенными в порядке возрастания величины Θ_j^*. Интервалы времени δ_i поступления входных данных $\boldsymbol{X}^0$ могут быть не синхронны с циклами Θ_j^*.

Множество циклов обновления $\{\Theta_j^*\}$, $\boldsymbol{j} = 1, 2, ..., \boldsymbol{k}, \boldsymbol{k}+1, \boldsymbol{k}+2$, разбивает множество $\boldsymbol{Y}^0$ на непересекающиеся подмножества $\boldsymbol{Y}_j$, $\boldsymbol{j} \in \{1, 2, ..., \boldsymbol{k}, \boldsymbol{k}+1, \boldsymbol{k}+2\}$ так, что $\bigcup \boldsymbol{Y}_j \bigcup \boldsymbol{Y}_{k+1} \bigcup \boldsymbol{Y}_{k+2} = \boldsymbol{Y}^0$. Подмножество $\boldsymbol{Y}_j$ включает информационные выходы $\boldsymbol{y}_z \in \boldsymbol{Y}^0$, $\boldsymbol{z} = 1, 2, ..., \boldsymbol{Z}$ с циклом обновления Θ_j^*. Подмножество $\boldsymbol{Y}_{k+1}$ включает $\boldsymbol{y}_z \in \boldsymbol{Y}^0$, формируемые ациклически, но с ограничением на время их формирования. Под-

множество Y_{k+2} включает $y_z \in Y^0$, формируемые ациклически, но требований на время их формирования не накладывается.

Функциональный ГПД представлен двудольным графом $R = (V, S)$, где V – множество вершин, а S – множество дуг [45]. Множество V разбито на два непересекающихся подмножества D и F, $D = \{d_q\}$, $q = 1, 2, ..., Q$, $F = \{f_m\}$, $m = 1, 2, ..., M$. Дуги множества S не могут связывать вершины внутри множеств D или F, то есть граф R является двудольным. Для любой дуги $s \in S$ возможен вариант $s = (d_q, f_m)$, либо $s = (f_m, d_q)$. Кроме того, вершины $d_q \in D$ и $f_m \in F$ могут быть связаны дугой лишь в одном направлении. Множество D соответствует множеству данных ГПД, а множество F – множеству фрагментов (алгоритмов) в ГПД. Множество дуг S соответствует информационным связям в ГПД – входным, если дуга связывает вершину данных и фрагмент, и выходным, если дуга связывает фрагмент и вершину данных. Очевидно, что множество вершин данных инцидентных только входным дугам в ГПД соответствует множеству X^0, а множество вершин данных инцидентных только выходным дугам в ГПД соответствует множеству Y^0. Заметим также, что граф R может быть несвязным. Это возможно в тех случаях, когда множество F описывает автономные прикладные функции СРВ.

Введенные обозначения дают более полное представление об информации, которой мы располагаем, приступая к решению задач построения модели СРВ и ее анализа. Проведя дополнительные исследования можно получить ряд необходимых дополнительных сведений как по архитектуре, так и алгоритмам программной нагрузки. Это могут быть объемы данных, времена выполнения фрагментов (алгоритмов), временные затраты на пересылку данных согласно принятой архитектуре и ряд других сведений. Наличие таких сведений позволяет сформулировать рассматриваемые задачи определения характеристик как оптимизационные. Но в этом случае мы наталкиваемся на отсутствие подходящих алгоритмов для решения данных задач. Вопросы адекватности также оказываются непреодолимой преградой. Поэтому формальные процедуры могут рассматриваться как предварительные решения для снятия самых больших неопределенностей на начальных этапах эволюции архитектуры МВС и программной нагрузки. Более точные решения будут получены в последующем методами моделирования.

В данном случае в целях получения приближенных оценок времени выполнения прикладных функций предлагается функциональ-

ный ГПД представить совокупностью параллельных процессов. Тогда проверку условий взаимодействия окружающей системы с МВС можно выполнить относительно полученных процессов. На основе данного анализа можно получить также предварительную оценку числа процессоров в МВС, необходимых для выполнения рассматриваемого ГПД.

Проектирование параллельных процессов

В принятых обозначениях задача проектирования параллельных процессов сводится к декомпозиции графа $\boldsymbol{R}=(\boldsymbol{V},\boldsymbol{S})$ на подграфы. Если каждому фрагменту (алгоритму) $\boldsymbol{f_m}\in\boldsymbol{F}$ поставить в соответствие значение τ_m – время выполнения фрагмента $\boldsymbol{f_m}$, то декомпозицию можно попытаться осуществить с учетом ограничений на время получения информационных выходов.

Разобьем граф $\boldsymbol{R}$ на $\boldsymbol{k}+2$ подграфа $\boldsymbol{R_j}=(\boldsymbol{V_j},\boldsymbol{S_j})$, так, что $\forall \boldsymbol{j}\left[\boldsymbol{Y_j}\subset\boldsymbol{V_j},\ \left(\boldsymbol{V_j}\cap\boldsymbol{X}^0\right)\neq\varnothing,\ \left(\boldsymbol{V_j}\cap\boldsymbol{V_\rho}\right)=\varnothing,\ \cup\boldsymbol{V_j}=\boldsymbol{V}\right],\ \boldsymbol{j},\ \rho=1,2,...,\boldsymbol{k},\boldsymbol{k}+1,\boldsymbol{k}+2.$ Множество $\boldsymbol{S_j}\in\boldsymbol{S}$ включает дуги множества $\boldsymbol{S}$, связывающие вершины из множества $\boldsymbol{V_j}$. Вариант разбиения функционального ГПД на два параллельных процесса представлен на рис. 5.1.

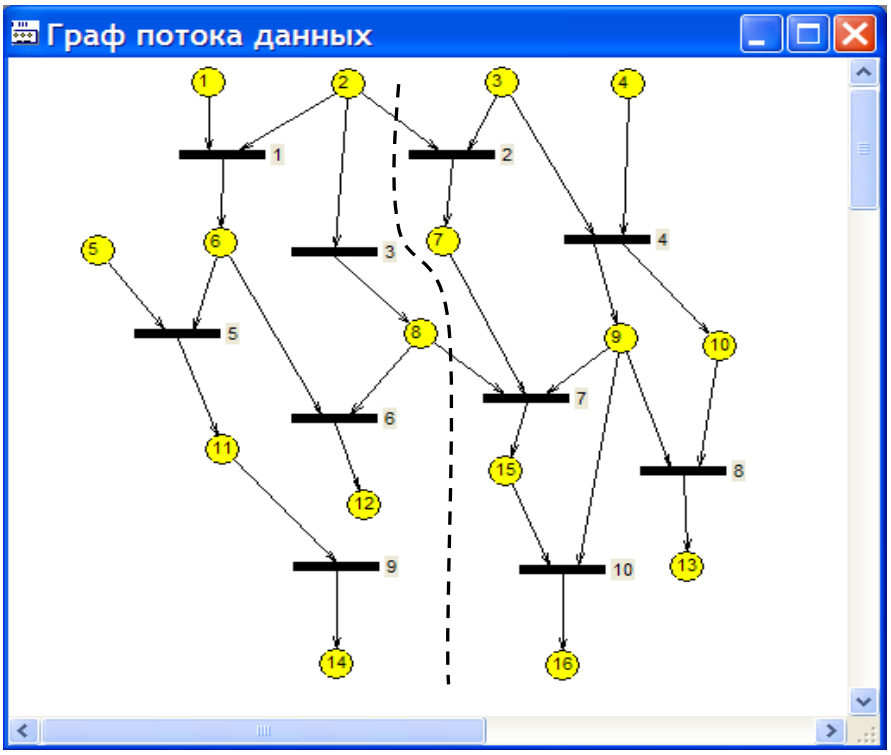


Рис. 5.1. Вариант разбиения ГПД на два подграфа (процесса)

Вариант однопроцессорной ВС

Каждому подграфу $\boldsymbol{R_j}$ соответствует подмножество фрагментов (алгоритмов) $\boldsymbol{F_j \subset F}$, которые должны обрабатывать данные из множества $\boldsymbol{V_j}$ за цикл Θ_j^*. Следовательно, суммарное время выполнения на одном процессоре $\boldsymbol{j}$-го процесса с множеством фрагментов $\boldsymbol{F_j}$ должно удовлетворять ограничению:

$$\sum_{f_m \in F_j} \tau_m = \Theta_j \le \Theta_j^*, \ \boldsymbol{j = 1,2,...,k}.$$

Ограничение на время выполнения на одном процессоре фрагментов всех подмножеств $\boldsymbol{F_j}$ имеет вид:

$$\sum_{j=1}^{k} \nu_j \Theta_j = \boldsymbol{T} \le \Theta_k^*, \ \boldsymbol{T} < \Theta_k^* \text{ при } \boldsymbol{F_{k+1}} \ne 0, \qquad (5.1)$$

где ν_j – указывает, сколько раз цикл Θ_k^* покрывает целым числом раз цикл Θ_j^*.

Принимается, что $\nu_j \Theta_j^* \ge \Theta_k^*$. Например, если $\Theta_j^* = 3$, а $\Theta_k^* = 14$, то $\nu_j = 5$.

Условие $\boldsymbol{T} < \Theta_k^*$ обеспечивает резерв времени у процессора для выполнения асинхронных процессов, в первую очередь для подмножества фрагментов $\boldsymbol{F_{k+1}}$.

Критерием качества решения данной задачи может быть принят минимум суммарных задержек времени при обработке входных сигналов $\boldsymbol{X^0}$ и формирования управляющих воздействий $\boldsymbol{Y^0}$. Заметим, что использование данного критерия на этапе формирования процессов и проведения аналитических расчетов возможно, если полностью пренебречь условиями запуска процессов или по крайней мере согласовать их запуск с временами циклов обновления Θ_j. В этом случае возможен перенос модуля $\boldsymbol{f_m}$ из одного процесса в другой и, как следствие, изменение частоты его выполнения ν_m. В реальных системах условия запуска процессов жестко регламентируются, поэтому данный критерий можно применять, если задачи проектирования процессов и формирования условий их запуска решать совместно. Качество сформированных процессов может оцениваться также уровнем их связности или, что то же самое минимальной разностью $\boldsymbol{S_\Delta}$:

$$S_{\Delta} = |S| - \sum_{j=1}^{k} |S_j|.$$

Здесь $|S|$ и $|S_j|$ обозначают мощности соответствующих множеств S и S_j. Возможны также и другие критерии, которые будут рассмотрены на этапах моделирования.

Многопроцессорный вариант ВС

Если в выполнении процессов F_j участвуют несколько процессоров, в том числе и разных типов, то необходимо знать значение τ_{mp} – время выполнения фрагмента (алгоритма) $f_m \in F_j$ на процессоре p. В этом случае каждый процесс F_j необходимо распараллелить так, чтобы время выполнения всех процессов на МВС в соответствии с ГПД удовлетворяло ограничениям реального времени.

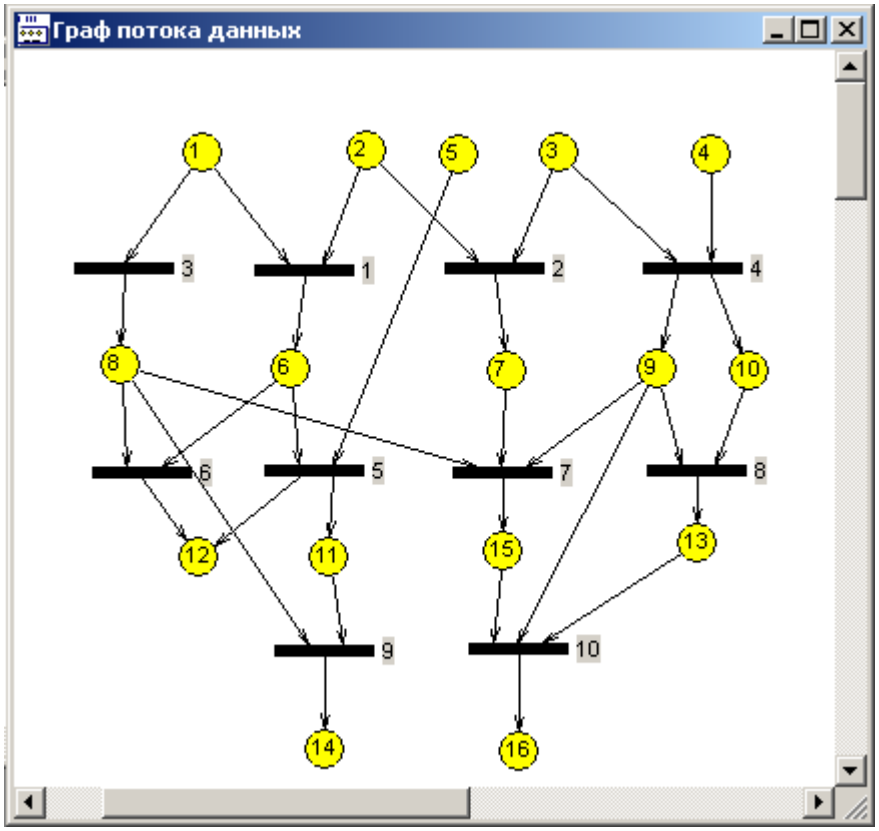


Рис. 5.2. ГПД в ярусно-параллельной форме

Распараллеливание ГПД выполняется путем преобразования его в ярусно-параллельную форму [46]. Выполнение данной операции отно-

сится к процедурам, реализуемым при томографии алгоритмов и программ. Пример ГПД в ярусно-параллельной форме представлен на (рис. 5.2). Первый ярус включает множество тех модулей, которые используют только входные данные. К i-му ярусу относятся модули, которые пользуются выходными данными хотя бы одного модуля $(i-1)$-го яруса и не могут использовать выходные данные модулей яруса i и выше.

Задача распараллеливания решается для всех подграфов R_j, представляющих подмножество фрагментов (алгоритмов) F_j соответствующего процесса. При необходимости осуществляется переход на более низкий уровень свертки фрагментов, и задача распараллеливания может решаться для соответствующего подграфа R_j или отдельного фрагмента на данном уровне свертки. При многопроцессорном выполнении процесса F_j модули $f_m \in F_j$ разбиваются на подмножества F_{jp}, каждое из которых обрабатывается соответствующим процессором p. В этом случае время завершения Θ_j^p определяется как максимальное время завершения обработки модулей F_{jp} в цикле Θ_j^* среди всех процессоров, участвующих в выполнении модулей процесса F_j. Если процессор p имеет максимальное время завершения Θ_j^p, то эту величину можно определить по выражению:

$$\Theta_j^p = \sum_{f_m \in F_{jp}} \tau_{mp}.$$

По аналогии с однопроцессорным вариантом должно выполняться условие $\Theta_j^p \le \Theta_j^*$.

Время выполнения на МВС всех фрагментов множества F определяется в этом случае по выражению:

$$T^n = \sum_{j=1}^{k+1} \nu_j \Theta_j^p = \sum_{j=1}^{k+1} \nu_j \sum_{f_m \in F_{jp}} \tau_{mp}, \qquad (5.2)$$

где $F_{jp} \subset F_j$ – подмножество фрагментов, выполняющихся на процессоре p, который имеет наибольшее время завершения обработки в цикле Θ_j^*.

Критерием качества решения задачи распараллеливания в этом случае, как правило, принимается достижение наиболее раннего завершения выполнения процессов.

Ограничение на время выполнения на МВС по аналогии с (5.1) запишется в виде:

$$\boldsymbol{T}^{n} \le \Theta_{\boldsymbol{k}}^{*}. \qquad (5.3)$$

В ограничениях (5.1) и (5.3) резерв времени для выполнения фрагментов из $\boldsymbol{F}_{\boldsymbol{k}+2}$ не учитывается, так как ограничений на время $\Theta_{\boldsymbol{k}+2}^{*}$ формирования информационных выходов $\boldsymbol{Y}_{\boldsymbol{k}+2}$ не накладывается. Предполагается, что выполнение фрагментов из множества $\boldsymbol{F}_{\boldsymbol{k}+2}$ будут выполнять процессоры, время завершения обработки которых не является максимальным.

Для применения данной методики необходимо знать величины $\tau_{\boldsymbol{m}}$ фрагментов $\boldsymbol{f}_{\boldsymbol{m}} \in \boldsymbol{F}$. В условиях, когда алгоритм программно реализован, величины $\tau_{\boldsymbol{m}}$ можно определить экспериментально. С этой целью программа алгоритма выполняется на инструментальной ЭВМ и фиксируется время выполнения каждого фрагмента. Полученное время пересчитывается для процессора, используемого в МВС. Процедуру выполнения программы можно повторить многократно для различных исходных данных. Полученные при этом значения $\tau_{\boldsymbol{m}}$ можно представить интервалом или средней величиной.

В работах [42, 47] предлагается более точный способ получения оценок $\tau_{\boldsymbol{m}}$. Здесь программа исследуемого алгоритма транслируется в программу для виртуальной машины, которая настраивается на соответствующий процессор. Команды виртуальной машины имеют реализацию в кодах различных процессоров и для них подсчитано время выполнения команды. Это позволяет при выполнении очередной команды на виртуальной машине в кодах инструментальной ЭВМ заносить в таймер виртуальной машины время выполнения соответствующей команды для исследуемого процессора. Заметим также, что алгоритм может быть непосредственно описан на входном языке виртуальной машины [42]. В этом случае программная реализация алгоритма и трансляция в программу для виртуальной машины не производится, а оценки величин $\tau_{\boldsymbol{m}}$ получаются еще более точными, так как сокращается цепь преобразований алгоритма из одной формы представления в другую.

Известны также и другие методы оценки времени выполнения программ, основанные на аналитических расчетах [48]. В этом случае программа представляется графом управляющих связей, на котором определяются наиболее продолжительные по времени маршруты, соответствующие выполнению алгоритма. При этом для логических опера-

торов приходится назначать вероятности логических выходов, что порождает большие сомнения в адекватности таких моделей.

5.2. Выбор архитектуры локальной сети МВС с минимальным временем передачи данных

Постановка задачи

Исходными для решения данной задачи являются сведения о совокупности ***микропроцессорных станций***, размещенных на ***топологическом поле*** объекта управления, и программной нагрузке, представленной в виде ГПД, реализующего прикладные функции. Пример топологического поля с размещенными на нем шестью станциями представлен на рис.5.3.

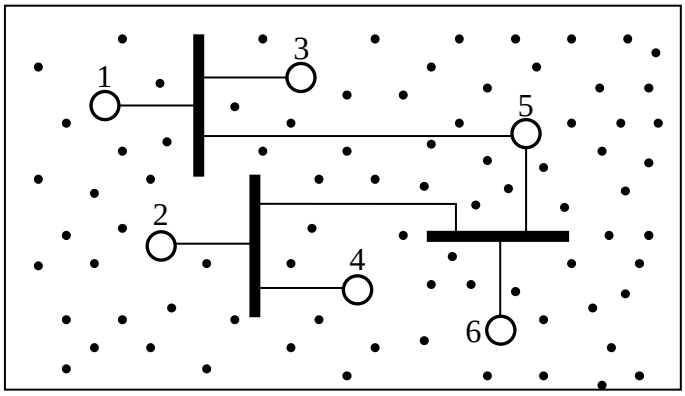


Рис.5.3. Топологическое поле

Предполагается также, что число станций и места их размещения определены, а множество ***терминальных точек***, также показанных на рис. 5.3, распределено по станциям.

Пример ГПД представлен на рис. 5.4. Кружки в ГПД соответствуют данным, а планки соответствуют операциям, выполняемым прикладными функциями СРВ. Данные и операции в ГПД помечены цифрами, обозначающими номера станций, на которых будут размещены данные и выполнены соответствующие операции.

Будем также считать, что возможные варианты базовых структур ***локальных сетей*** заданы и сформированы в библиотеку. Примеры вариантов таких структур для шести станций приведены на рис. 5.5. Данные варианты получены для рассматриваемого примера на основе ***базовых сетей***, построенных на основе одной, двух и трех ***магистралей***.

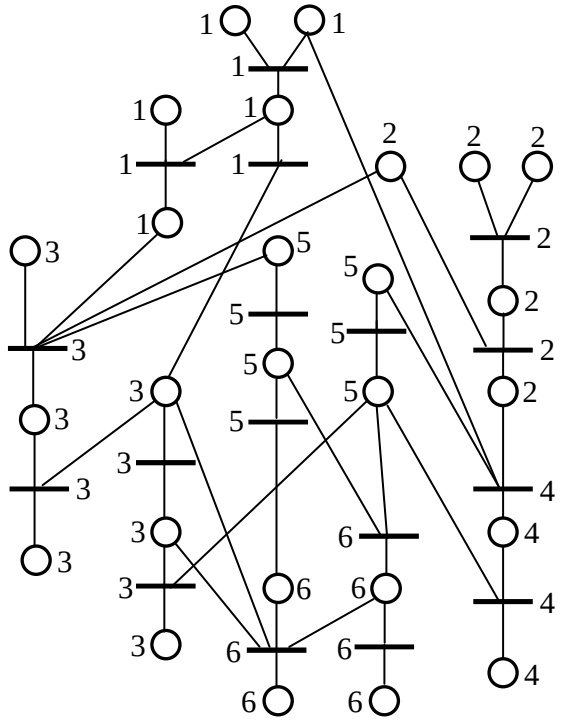


Рис.5.4. Граф потока данных

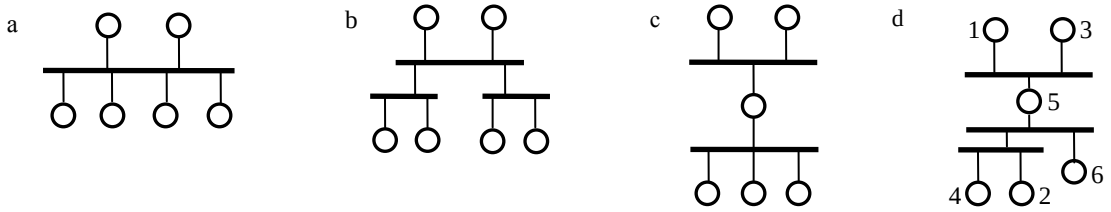


Рис.5.5. Варианты базовых сетей

При наличии указанных исходных данных ***рассматриваемая задача заключается в выборе структуры базовой сети и варианта подключения станций к магистралям сети так, чтобы по возможности большая часть данных могла передаваться между станциями сети параллельно.*** Это позволит сократить общее время на передачу данных в сети при выполнении ГПД на станциях вычислительной системы.

Методика решения задачи основана на выявлении параллельных передач данных для возможных вариантов подключения станций к магистральной сети. С этой целью выполняется совокупность операций по построению следующих объектов:

- ***граф передачи данных*** между станциями сети;
- ***матрица наличия конфликтов*** при доступе к магистралям сети;
- ***диаграмма совмещения*** параллельных передач данных.

Граф передачи данных

Граф передачи данных $\boldsymbol{G=(S,R,P)}$ строится на основе ГПД и включает множество вершин $\boldsymbol{S=\{s_i\}}$ и ребер $\boldsymbol{R=\{r_{ij}\}}$. Вершина $\boldsymbol{s_i}$ соответствует $\boldsymbol{i}$-й станции, а наличие ребра $\boldsymbol{r_{ij}}$ соответствует тому, что фрагменты ГПД, размещенные в станциях $\boldsymbol{i}$ и $\boldsymbol{j}$, связаны между собой. Каждому ребру $\boldsymbol{r_{ij}}$ графа $\boldsymbol{G}$ ставится в соответствие величина $\boldsymbol{p_{ij}\in P}$, которая определяет объем данных, передаваемых между станциями $\boldsymbol{i}$ и $\boldsymbol{j}$ за определенный интервал времени работы ГПД (функционирования СРВ). Величины $\boldsymbol{p_{ij}}$ суммируют объемы передаваемых данных между фрагментами $\boldsymbol{i}$ и $\boldsymbol{j}$ ГПД с учетом частоты выполнения операций, с которыми связаны эти данные. В ГПД частота выполнения каждой операции зависит от условия запуска процесса, включающего данную операцию, и продолжительности сеанса моделирования СРВ.

Пример графа $\boldsymbol{G}$, построенного для ГПД (рис. 5.4), представлен на рис. 5.6.

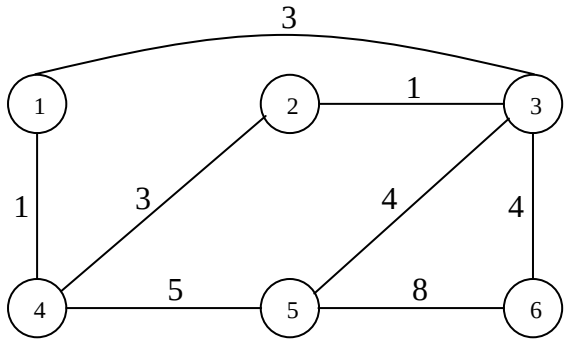


Рис.5.6. Граф передачи данных $\boldsymbol{G}$

Ребра $\boldsymbol{r_{ij}}$ графа соответствуют наличию связей между соответствующими фрагментами $\boldsymbol{i}$ и $\boldsymbol{j}$ в ГПД. Веса ребер (величины $\boldsymbol{p_{ij}}$) взяты произвольно.

Матрица наличия конфликтов

Построение матрицы наличия конфликтов осуществляется на основе графа G и варианта подключения станций для выбранной базовой сети. Методику построения матрицы покажем на рассматриваемом примере. В качестве базовой структуры сети выберем вариант, представленный на рис. 5.5 d. На этом же рисунке показан один из возможных вариантов подключения станций для данной сети. Размерность матрицы определяется числом ребер графа G. Множество ребер графа G {(1,3), (1,4), (2,3), (2,4), (3,5), (3,6), (4,5), (5,6)} обозначим соответствующими кодовыми номерами, сохранив в них номера станций. Получим множество ребер (13, 14, 23, 24, 35, 36, 45, 56). Так, например, ребро r_{24} означает наличие связи между станциями 2 и 4 с объемом передач $p_{24} = 3$. Элемент q_{vk} матрицы наличия конфликтов $Q = \|q_{vk}\|$, $v, k \in (13,14,23,24,35,36,45,56)$ определяется следующим образом,

$$q_{vk} = \begin{cases} 1, \textit{если пары станций ребер v и k при передаче данных в сети} \\ (\textit{рис.}\ 5.5\,d)\ \textit{имеют конфликт по доступу к магистрали;} \\ 0, \textit{в противном случае.} \end{cases}$$

	13	14	23	24	35	36	45	56
13		1	1		1	1		
14	1		1	1	1	1	1	1
23	1	1		1	1	1	1	1
24		1	1				1	
35	1	1	1			1		
36	1	1	1		1		1	1
45		1	1	1		1		1
56		1	1			1	1	

Рис.5.7. Матрица конфликтов Q

Так, например, элемент $q_{36,45} = 1$, так как при передаче данных между станциями 3 и 6 и станциями 4 и 5 имеет место конфликт за доступ к магистрали. Напротив элемент $q_{13,56} = 0$, так как при передаче данных используются разные магистрали. Построенная таким образом матрица Q представлена на рис. 5.7.

Диаграмма совмещения параллельных передач данных

На основе матрицы $\boldsymbol{Q}$ строится диаграмма совмещения параллельных передач данных. Для удобства построения диаграммы матрица $\boldsymbol{Q}$ принимается в качестве матрицы связности вершин графа. Соответствующий граф $\boldsymbol{Q}$ представлен на рис. 5.8. Заметим, что в графе $\boldsymbol{Q}$ вершины 14 и 23 исключены, так как все элементы соответствующих строк и столбцов матрицы равны 1 и, следовательно, отсутствует возможность для параллельной передачи данных.

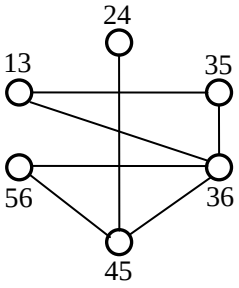


Рис.5.8. Граф $\boldsymbol{Q}$

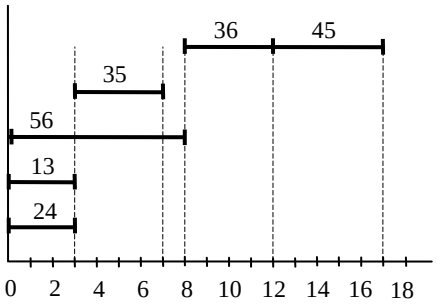


Рис.5.9. Диаграмма совмещения данных

Для построения диаграммы в графе $\boldsymbol{Q}$ (рис. 5.8) последовательно выделяются ***максимальные пустые подграфы*** и объемы передач соответствующих вершин совмещаются на диаграмме. Так, на первой итерации, совмещены вершины 24, 13 и 56 (рис. 5.9). На второй итерации совмещается вершина 35 и остаток объема вершины 56. Далее на диаграмме размещаются вершины 36 и 45.

Из диаграммы следует, что суммарный объем передач данных с учетом совмещения передач составляет 19 единиц (17 единиц по диаграмме и 2 единицы от исключенных ребер $\boldsymbol{r}_{14}$ и $\boldsymbol{r}_{23}$). Общий объем передаваемых данных согласно графу $\boldsymbol{G}$ равен 29 единицам. Таким образом, для выбранной локальной сети последовательная цепь передач данных сокращается на 10 единиц объема, что соответственно приводит к сокращению времени на передачу данных в локальной сети. Сеть с вариантом подключения станции согласно рис. 5.5 d, показана на топологическом поле (рис. 5.3).

В заключение отметим, что для поиска наилучшего варианта, доставляющего наибольшее совмещение параллельных передач данных, необходимо перебрать множество возможных вариантов подключения станций для различных базовых сетей. Разработка приемлемого алгоритма организации такого перебора является самостоятельной задачей. На данном этапе исследований удалось формализовать переход от ре-

зультатов распределения операций и данных по станциям к выбору варианта архитектуры локальной сети и на примере показать возможность сокращения затрат времени на передачу данных в выбранной сети.

Вопросы для контроля усвоения знаний

1. Определить условия взаимодействия встроенной и окружающей систем.
2. Дать определение функционального графа потока данных.
3. Изложить методику разбиения графа потока данных на процессы.
4. Раскрыть содержание анализа возможности использования однопроцессорной ВС.
5. Раскрыть смысл критериев качества решения задачи проектирования процессов.
6. Изложить методику преобразования графа потока данных в ярусно-параллельную форму.
7. Раскрыть содержание анализа многопроцессорного варианта ВС.
8. Изложить известные подходы к оценке времени выполнения фрагментов.
9. Пояснить задачу выбора локальной сети. Исходные данные и формулировка задачи.
10. Предложить пример и на нем изложить методику построения графа передач данных.
11. Построить матрицу наличия конфликтов (исходные данные взять произвольно).
12. Построить диаграмму совмещения параллельных передач данных (исходные данные взять произвольно).

6. МОДЕЛИРОВАНИЕ СРВ НА ФУНКЦИОНАЛЬНОМ УРОВНЕ

6.1. Согласование моделей прикладных функций с моделью архитектуры МВС

Модель прикладных функций программной нагрузки проектируемой СРВ представлена графом потока данных $\boldsymbol{G=(V,S)}$, разбитым на подграфы $\boldsymbol{R_j=(V_j,S_j)}$ согласно методике, изложенной в разделе 5. Каждому подграфу $\boldsymbol{R_j}$ соответствует $\boldsymbol{j}$-й вычислительный процесс с циклом обновления выходных данных (управляющих воздействий) равным величине Θ_j^*.

Для согласования модели, представленной функциональным ГПД, и модели МВС необходимо спланировать размещение фрагментов (модулей) по процессорам МВС, программ и данных по имеющимся в МВС ресурсам памяти.

Критерии качества решения рассматриваемой задачи определяются исходя из влияний, которые она оказывает на загрузку каналов, связывающих станции, а также на загрузку процессоров и запоминающих устройств станций. Названные характеристики определяют три частные задачи размещения фрагментов, программ и данных. В каждой из задач один из показателей принимается в качестве критерия, а два других должны удовлетворять заданным ограничениям. Как правило, выбирается тот критерий, который соответствует узкому месту в проектируемой СРВ. Иногда некоторые характеристики могут быть заданы и вводятся автоматически в ограничения. Если критерии принимаются равнозначными, то названные задачи решаются последовательно для каждого частного критерия.

Критерий минимума загрузки сети передачи данных, как правило, является предпочтительным, так как позволяет уменьшить время, затрачиваемое на обмен данными между станциями, а следовательно, время выполнения прикладных функций.

Решение данной задачи осуществляется по минимуму средней загрузки сети, определяемой частотами и объемами сетевых передач, необходимых при выполнении модулей ГПД. Для аварийных режимов рассчитывается пиковая нагрузка сети.

Критерий минимума загрузки запоминающих устройств станций приводит к снижению числа копий данных, распределяемых между

станциями. При этом, как правило, увеличивается загрузка каналов связи дополнительными передачами и, вследствие этого, ухудшаются показатели времени и надежности выполнения модулей ГПД.

Постановка задачи распределения модулей, программ и данных

Рассмотрим вычислительную сеть распределенной СРВ, состоящую из множества S станций, соединенных магистралью. Имеется множество F модулей, использующих множество P, $|P| \le |F|$, программ. В памяти станций должно быть размещено множество D данных (входная и выходная информация и промежуточные результаты) согласно ГПД. Заданы размеры данных r_d, $d \in D$, программ r_p, $p \in P$, рабочей памяти r_f, необходимой для выполнения модулей $f \in F$. Известна также частота ν_f выполнения каждого модуля $f \in F$. Желательно, пусть и приближенно, знать объем вычислений h_f для каждого модуля $f \in F$, например, в элементарных машинных операциях. Напомним также, что могут быть заданы значения τ_f времени выполнения модуля f.

Учитывая, что рассматриваемая задача носит существенно комбинаторный характер, эффективные алгоритмы ее решения следует искать на пути разработки комбинаторных методов. Ниже предлагается приближенный метод решения данной задачи, учитывающий ее комбинаторные свойства. В принятых выше обозначениях решением данной задачи являются три матрицы:

- матрица $X_{FS} = \|x_{fs}\|$, где $x_{fs} = 1$, если модуль $f \in F$ выполняется на станции $s \in S$ и $x_{fs} = 0$ в противном случае;
- матрица $X_{PS} = \|x_{ps}\|$, где $x_{ps} = 1$, если программа $p \in P$ хранится в памяти станции $s \in S$ и $x_{ps} = 0$ в противном случае;
- матрица $X_{DS} = \|x_{ds}\|$, где $x_{ds} = 1$, если данные $d \in D$ размещены в памяти станции $s \in S$ и $x_{ds} = 0$ в противном случае.

Так как программы и данные используют одни и те же запоминающие устройства станций, то для удобства множество программ включим в множество данных и в последующем будем рассматривать только расширенную матрицу X_{DS}. Объединение программ и данных можно отобразить на ГПД, если дополнить его вершинами по числу программ

и связать эти дополнительные вершины с вершинами функций, использующих соответствующие программы.

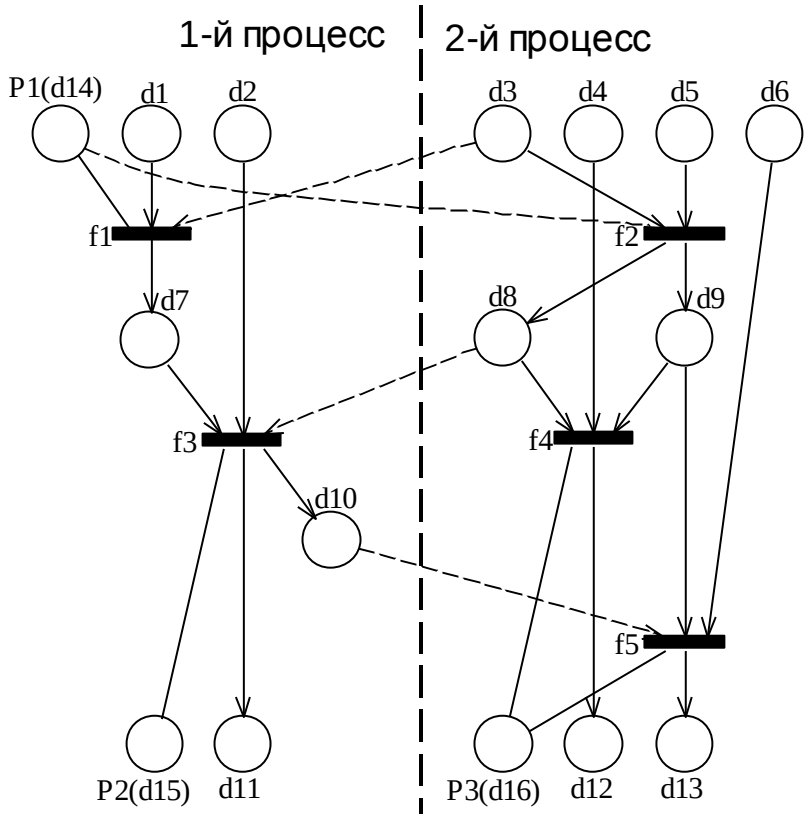


Рис. 6.1. Функциональный ГПД из двух параллельных процессов с дополнением трех программ

На рис. 6.1 пунктирными линиями показаны связи, которые оказались разорванными при декомпозиции ГПД на два подграфа. Второй процесс также может быть представлен двумя процессами, например, если разорвать связи $(\boldsymbol{d}_9, \boldsymbol{f}_5)$, $(\boldsymbol{d}_{10}, \boldsymbol{f}_5)$, $(\boldsymbol{f}_5, \boldsymbol{p}_3)$. Если принять, что модель программной нагрузки, изображенной на рис. 6.1, будет выполняться на трех станциях, то задачу размещения модулей и данных (включая программы) можно представить схемой на рис. 6.2.

На рис. 6.2 под номерами $\boldsymbol{d}_{14}$, $\boldsymbol{d}_{15}$, $\boldsymbol{d}_{16}$ показаны данные, соответствующие программам $\boldsymbol{p}_1$, $\boldsymbol{p}_2$, $\boldsymbol{p}_3$. Данная схема представляет собой двудольный граф $\boldsymbol{G}$ с множеством вершин $\boldsymbol{F}$ и $\boldsymbol{D}$. Вершина $\boldsymbol{f}_j \in \boldsymbol{F}$ связывается с вершиной $\boldsymbol{d}_i \in \boldsymbol{D}$ ребром $(\boldsymbol{f}_j, \boldsymbol{d}_i)$, если в ГПД имеется связь $(\boldsymbol{f}_j, \boldsymbol{d}_i)$. Существенным для данного графа является то, что каждая

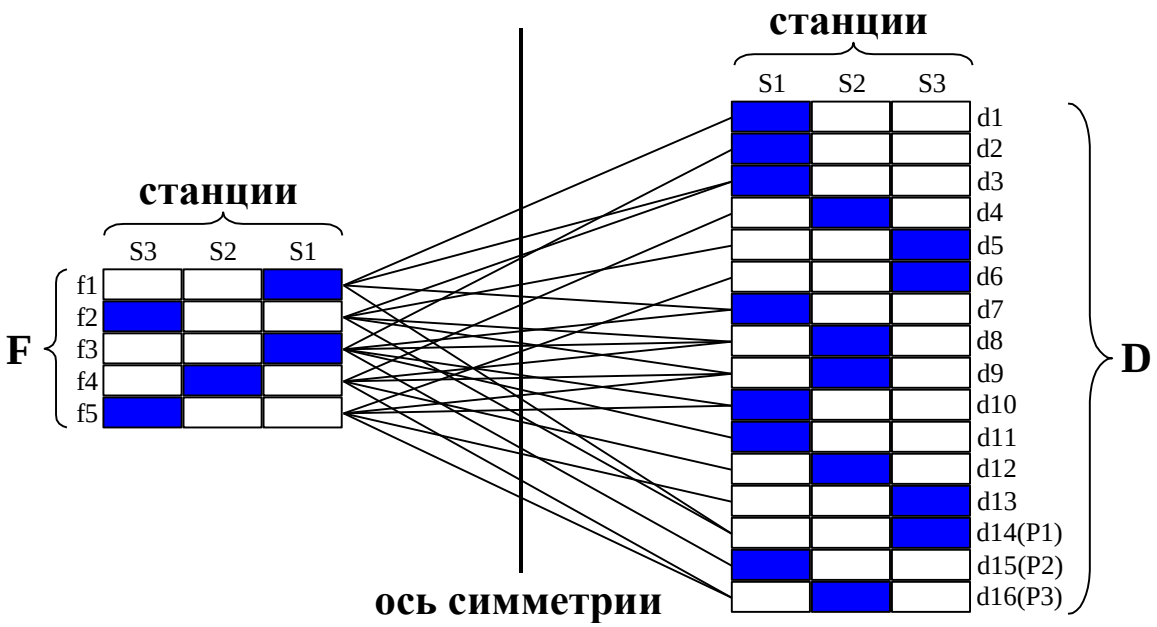


Рис. 6.2. Иллюстрация к задаче распределения модулей и данных

вершина может иметь несколько состояний. Число состояний определяется числом станций. Для рассматриваемого примера число станций равно трем. Состояние вершины определяется номером станции, в которую распределена данная вершина.

Введем понятие длины ребра графа G. Длину ребра (d_i, f_j) можно определить через состояния вершин. Вершина d_i удалена от оси симметрии на расстояние $c(d_i)$ равное ее состоянию $s(d_i)$. Аналогично для вершин f_j расстояние $c(f_j)$ определяется состоянием $s(f_j)$. Тогда ребро (d_i, f_j) имеет длину c_{ij} равную 1, если $|s(d_i) - s(f_j)| > 0$, и $c_{ij} = 0$, если $|s(d_i) - s(f_j)| = 0$. Другими словами, ребро (d_i, f_j) графа G имеет нулевую длину, если вершины d_i и f_j удалены от оси симметрии на одинаковые расстояния. Во всех других случаях длина ребра (d_i, f_j) равна 1.

С учетом принятых определений рассматриваемая задача сводится к определению состояний вершин графа G, при котором сумма длин ребер была бы минимальной.

Ребра графа G могут иметь веса, учитывающие, например, размеры данных r_d и частоты их использования модулем f. Вес m_{df} ребра

(d, f) определим как произведение размера данных r_d на частоту выполнения модуля ν_f, то есть $m_{df} = r_d \cdot \nu_f$. В этом случае критерием решения задачи является минимальная сумма взвешенных длин ребер.

Ограничения задачи в принятых обозначениях имеют вид:

$$\sum_{s \in S} x_{fs} = 1, \text{ для всех } f \in F; \quad (6.1)$$

$$\sum_{f \in F} x_{fs} \cdot \nu_f \cdot h_f \le H_s, \text{ для всех } s \in S; \quad (6.2)$$

$$\sum_{s \in S} y_{ds} = 1, \text{ для всех } d \in D; \quad (6.3)$$

$$\sum_{d \in D} y_{ds} \cdot r_d \le R_s, \text{ для всех } s \in S. \quad (6.4)$$

Здесь H_s – допустимый объем вычислений для процессора станции s; R_s – допустимый размер памяти в запоминающем устройстве станции s.

Переменные x_{fs} и y_{ds} соответствуют введенным ранее переменным x_{fs} и y_{ds}. В новых обозначениях переменная $x_{fs} = 1$, если вершина f пребывает в состоянии s. Аналогично $y_{ds} = 1$, если вершина d находится в состоянии s. Во всех других случаях переменные x_{fs} и y_{ds} равны нулю.

Ограничение (6.1) разрешает распределить каждую функцию только в одну станцию. Аналогично ограничение (6.3) обеспечивает размещение данных в памяти одной станции.

Ограничения (6.2) и (6.4) разрешают загрузить процессор и запоминающие устройства станций на величины, не превышающие значений H_s и R_s, соответственно.

Рассмотрим простой эвристический алгоритм решения данной задачи. На первом шаге, например с помощью венгерского алгоритма [21, 45], на графе G определяется максимальное паросочетание с учетом весов ребер. Второй шаг алгоритма выполняет распределение вершин ребер найденного паросочетания по станциям так, чтобы длины данных ребер были нулевыми. При этом проверяются условия (6.2) и (6.4). Если какое - либо из условий нарушается, то выбирается следующая станция. Выбор станций осуществляется также с учетом результатов распараллеливания. Модули одного процесса желательно распределять в одну станцию.

На третьем шаге распределяются оставшиеся вершины так, чтобы по возможности большее число ребер получили нулевую длину. Это достигается при условии, если вершины d, связанные с модулем f, рас-

пределенным в станцию s, также распределить в станцию s. Алгоритм заканчивает работу, когда все вершины распределены и при этом условия (6.1) – (6.4) выполняются.

Оптимизационная постановка задачи

Данную задачу можно сформулировать как задачу математического программирования с булевыми переменными. При этом воспользуемся принятыми выше обозначениями. Граф G представим матрицей смежности вершин $Q = \|q_{fd}\|_{F \times D}$, где $q_{fd} = 1$, если вершины f и d связаны ребром (f_j, d_i), $q_{fd} = 0$ в противном случае. Решение данной задачи заключается в поиске такого варианта размещения модулей, программ и данных, который включает ребра нулевой длинны с максимальным суммарным весом. С учетом принятых обозначений, сформулированный таким образом критерий оптимальности можно представить в следующем виде:

$$\max L = \sum_{f=1}^{F} \sum_{d=1}^{D} q_{fd} \left(x_{f1} \cdot y_{d1} + x_{f2} \cdot y_{d2} + ... + x_{fs} \cdot y_{ds} \right) m_{fd} . \qquad (6.5)$$

Задача (6.5), (6.1) – (6.4) является нелинейной по критерию задачей математического программирования с булевыми переменными, и для ее решения нет эффективных алгоритмов. Для преодоления данной проблемы можно воспользоваться методом линеаризации [13]. Применительно к нашему случаю линеаризацию можно провести следующим образом. Пусть $x_{fs} \cdot y_{ds} = z_{fds}$, тогда критерий оптимальности может быть записан в следующем виде:

$$\max \sum_{f=1}^{F} \sum_{d=1}^{D} q_{fd} \left(z_{fd1} + z_{fd2} + ... + z_{fds} \right) m_{fd} . \qquad (6.6)$$

При этом, если слагаемое $x_{fs} \cdot y_{ds}$ в целевой функции имеет знак "+", то оно должно удовлетворять условию:

$$x_{fs} + y_{ds} - z_{fds} \le 1, \qquad (6.7)$$

а если со знаком "–", то условию

$$x_{fs} + y_{ds} - z_{fds} \ge 0 . \qquad (6.8)$$

В нашем случае слагаемое $x_{fs} \cdot y_{ds}$ всегда положительно, поэтому в задачу включается только условие 6.7. Применение линеаризации приводит к многократному росту размерности данной задачи. Поэтому для ее решения более предпочтительными являются комбинаторные методы. В частности, данную задачу можно свести к задаче разрезания

графа на минимально связанные подграфы и использовать для ее решения существующие алгоритмы. Данный подход будет изложен в седьмом разделе.

6.2. Моделирование выполнения прикладных функций и оценка их характеристик

Выполнение активной модели

Процесс моделирования сводится к выполнению активных моделей прикладных функций на модели архитектуры МВС в соответствии с планом использования ресурсов, полученным при решении задачи размещения модулей, программ и данных. Выполнение модели, представленной функциональным ГПД, производится на виртуальной машине, которая воспринимает ГПД как свою программу. Процесс моделирования включает решение двух основных задач:

- управление выполнением ГПД на модели МВС согласно плану использования ресурсов;
- отображение и анализ результатов моделирования.

Алгоритм управления выполнением ГПД является операционной системой виртуальной машины. Основные функции алгоритма управления заключаются в следующем:

- выбор очередной команды для выполнения на виртуальной машине;
- синхронизация выполнения команд по наличию данных;
- имитация работы устройств МВС согласно плану использования ресурсов;
- учет приоритетов выполнения команд;
- управление таймером виртуальной машины.

В качестве очередной команды может быть принят модуль ГПД (программа или имитатор модуля), имитатор компоненты архитектуры, системные функции алгоритма управления моделированием. Синхронизация выполнения команд заключается в проверке условий готовности команды к выполнению по наличию данных, наступлению соответствующего момента времени и соблюдению отношения приоритетности.

Построение временной диаграммы

Процесс моделирования сопровождается построением временной диаграммы. Пример временной диаграммы, построенной для ГПД

(рис. 6.1) и плана распределения функций, программ и данных (рис. 6.2) представлен на рис. 6.3.

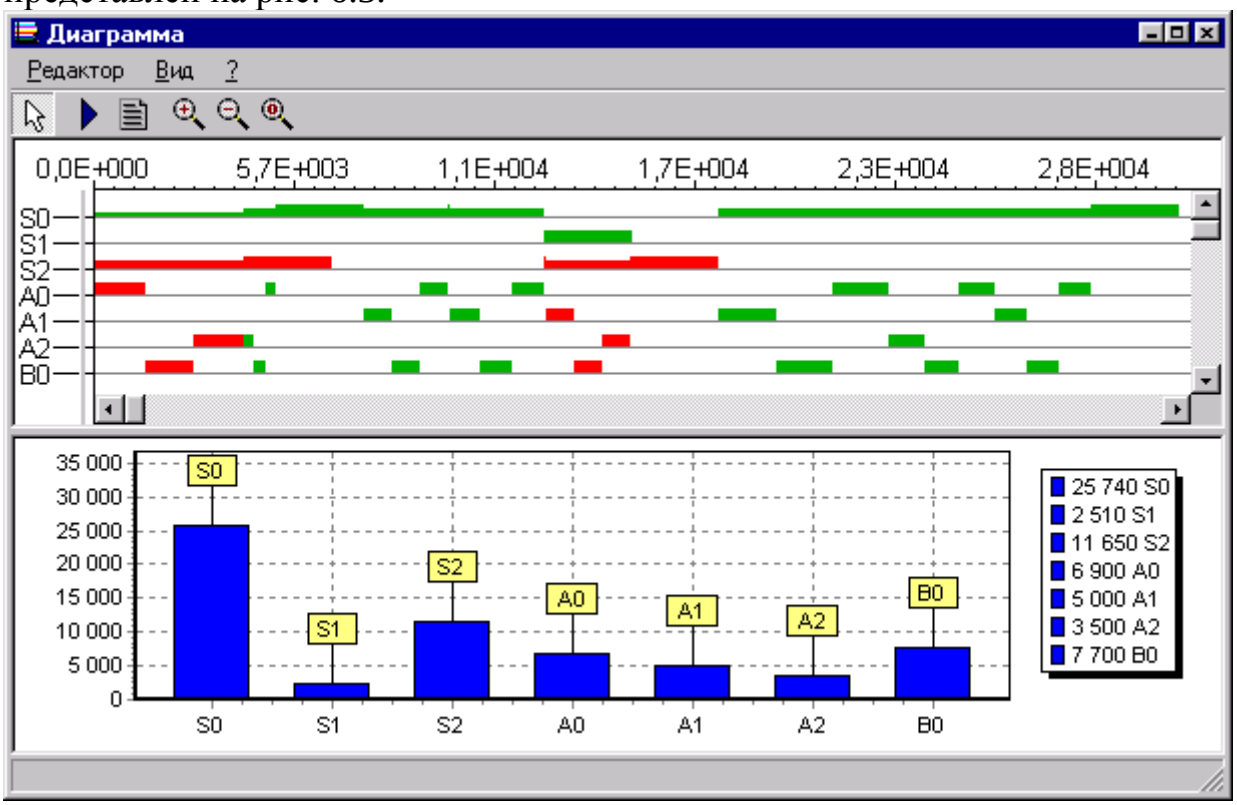


Рис. 6.3. Пример временной диаграммы

Оси временной диаграммы соответствуют активным компонентам модели архитектуры МВС. На осях откладываются временные отрезки работы соответствующих устройств МВС при выполнении команд виртуальной машины. После выполнения очередной команды для какого-либо вычислительного процесса информация о времени ее выполнения отображается на соответствующей временной оси. Для команд, которые соответствуют модулям ГПД, время τ_f известно. Для имитаторов, описывающих работу соответствующего устройства, время их работы подсчитывается имитатором и после этого отображается на временной диаграмме.

Время работы устройства также может задаваться как фиксированная величина либо выбираться по определенным правилам, запрограммированным в имитаторе.

Временная диаграмма по желанию пользователя может включать временные оси лишь для отдельных устройств. При этом число осей для отображения может изменяться в большую или меньшую сторону, как перед началом моделирования, так и после его завершения. После моделирования временную диаграмму можно просматривать, изменяя мас-

штабирование более детально и по отдельным, интересующим пользователя, устройствам.

Определение характеристик

Определение характеристик выполняется по временной диаграмме после окончания моделирования. Основными характеристиками являются следующие:

- время выполнения прикладной функции в целом или ее фрагментов на процессорах различных платформ;
- коэффициенты загрузки устройств МВС при выполнении алгоритмов;
- частоты выполнения отдельных фрагментов алгоритмов.

Последняя характеристика имеет важное значение при детальном анализе работы алгоритма с точки зрения поиска возможных вариантов сокращения времени выполнения алгоритма. При высокой частоте даже относительно небольшое сокращение времени выполнения соответствующего фрагмента может привести к желаемому сокращению общего времени завершения работы данного алгоритма и прикладной функции.

Коэффициенты загрузки устройств определяются как отношение суммарного времени работы устройства к общему модельному времени. Значения данных коэффициентов существенно зависят от варианта размещения модулей, программ и данных по ресурсам МВС. Поэтому характеристики по загрузке устройств и частотам выполнения важно знать для принятия решений по эволюции моделей. В конечном итоге мы должны получить вариант МВС с приемлемой избыточностью по ресурсам и способный выполнить прикладные функции проектируемой СРВ с учетом установленных ограничений на время.

Оценки времени выполнения прикладных функций являются основными при оценке варианта архитектуры МВС. Для такой оценки важно знать минимально возможное время выполнения заданной совокупности прикладных функций при ее максимальном распараллеливании. Если данная оценка превышает ограничения реального времени, то очевидно, что необходимо вносить изменения в МВС или в прикладные функции. Следует, однако, иметь в виду, что максимальное распараллеливание далеко не всегда доставляет минимальное время завершения выполнения прикладных функций. Это объясняется тем, что при увеличении числа станций, участвующих в выполнении прикладных функций, проявляется тенденция к росту объемов информации, передаваемой между станциями.

6.3. Методика выбора предпочтительного варианта архитектуры МВС

Используемые методы моделирования отличаются низким объемом вычислений и являются достаточно эффективным инструментом направленным на максимальное снижение неопределенности относительно архитектуры МВС и алгоритмов прикладных функций. Это достигается многовариантным анализом моделей, что в свою очередь позволяет на данном этапе пройти большую часть пути в процессе эволюции исходных вариантов моделей МВС и ГПД. Многовариантный анализ позволяет более полно установить характер зависимости основных параметров: число станций в МВС и время задержки выполнения прикладных функций. Если в данный анализ ввести экономический фактор, то можно сформулировать задачу выявления предпочтительного с позиции экономического критерия варианта архитектуры МВС и заданного варианта ГПД.

Введем функцию $R_z = f_1(T_z)$, где R_z–экономические потери от снижения качества управления в зависимости от роста времени задержки выполнения прикладных функций T_z. Функцией $R_s = f_2(V_s)$ введем зависимость дополнительных затрат от роста числа станций в МВС. Общий вид данных функций представлен на рис. 6.4.

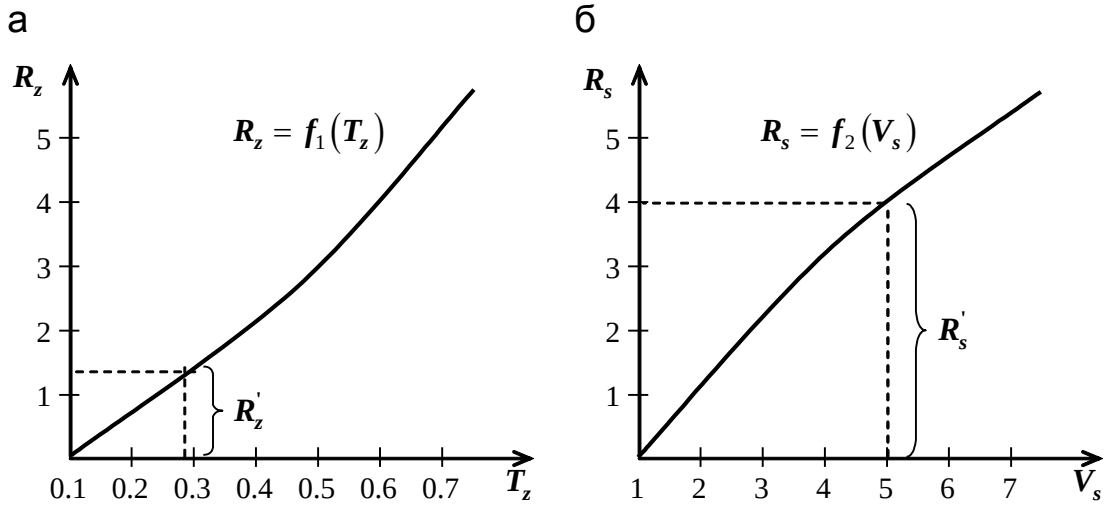


Рис. 6.4. а – функция $R_z = f_1(T_z)$; б – функция $R_s = f_2(V_s)$

Начало координат при построении функции f_1 соответствует некоторому минимальному значению T_z, которое получается при максимальном распараллеливании и для каждой параллельной ветви ГПД вы-

делен свой процессор. Функция f_2 построена таким образом, что начало координат соответствует ситуации, когда архитектура МВС включает одну станцию.

Экспериментальным путем на основе результатов многовариантного анализа можно построить функцию $T_z = f_3(V_s)$. На рис. 6.5 а показан общий вид функции f_3.

Имея функции f_1, f_2, f_3, можно с помощью простых геометрических построений получить интересующую нас функцию $R = f(V_s, T_z)$. Здесь R – экономические потери в зависимости от числа станций V_s и времени завершения выполнения прикладных функций T_z. На рис. 6.5 б показан механизм графического построения данной функции. Из рисунка видно, что функция f имеет минимум, который соответствует МВС с $V_s \cong 3$. Величина T_z при этом равна 0.44 условным единицам времени. Заметим, что числовые значения на рис. 6.4 и рис. 6.5 являются условными и приведены для удобства восприятия.

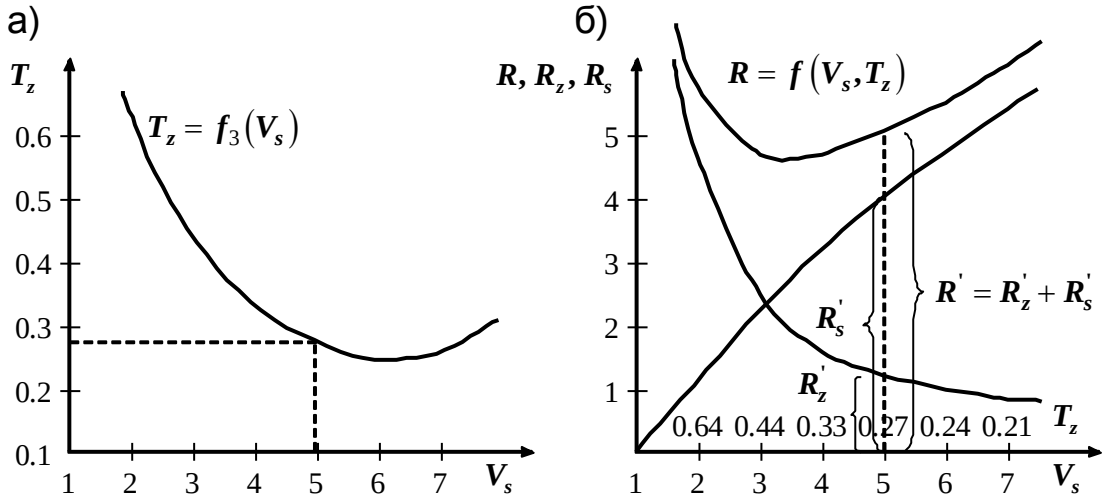


Рис. 6.5: а – функция $T_z = f_3(V_s)$, б –построение функции $R = f(V_s, T_z)$

Следует отметить, что величина T_z зависит не только от величины V_s, а и от многих других значений величин ресурсов МВС и плана их использования, полученного в результате решения задачи распределения модулей, программ и данных. Поэтому предложенную методику следует рассматривать как демонстрацию общего подхода к выбору приемлемого варианта архитектуры МВС. В частности, можно ввести некоторый обобщенный показатель мощности ресурсов МВС и исполь-

зовать его вместо величины V_s. По - видимому, это будет в большей мере оправдано для последующих более поздних этапов эволюции, использующих более точные модели.

Заметим также, что весьма полезными могут оказаться зависимости $T_z = f_r(V_r)$, в которых вместо числа станций V_s используется другой ресурс V_r, например: производительность процессора, ресурс по памяти, по пропускной способности сети, ресурс по внешней памяти и скоростям доступа и многие другие. Наличие таких частных функций f_r, экспериментальное получение которых не вызывает трудностей для рассматриваемого уровня детализации моделей, создает условия для принятия обоснованных решений в процессе эволюции моделей архитектуры МВС и алгоритмов прикладных функций проектируемой СРВ.

Вопросы для контроля усвоения знаний

1. Привести критерии решения задачи размещения модулей, программ и данных.
2. Записать постановку задачи размещения модулей, программ и данных.
3. Изложить эвристический алгоритм решения задачи размещения модулей, программ и данных.
4. Дать оптимизационную постановку задачи размещения модулей, программ и данных.
5. Изложить алгоритм управления моделированием.
6. Определить содержание временной диаграммы и методики ее анализа.
7. Определить понятие предпочтительного варианта архитектуры МВС.
8. Изложить методику выбора предпочтительного варианта архитектуры МВС.

7. КОМПЛЕКСНОЕ МОДЕЛИРОВАНИЕ СРВ В УСЛОВИЯХ РЕАЛЬНОГО ВРЕМЕНИ

7.1. Вычислительные процессы реального времени (РВ-процессы)

Вычислительный процесс реального времени является одним из основных понятий, с которым приходится оперировать при моделировании программной нагрузки СРВ в условиях реального времени. Понятие вычислительного процесса использовалось нами ранее при моделировании прикладных функций. Такие процессы рассматривались как функционально законченная часть ГПД прикладной функции. Основное внимание при этом уделялось исследованию возможностей распараллеливания ГПД и представлению его совокупностью параллельных процессов. Это позволило оценить минимально возможное время выполнения прикладной функции на МВС. При этом имитация функционирования прикладных функций могла осуществляться как автономно для каждой из них, так и совместно, но практически без учета условий запуска процессов и их взаимодействия при функционировании в составе программной нагрузки СРВ. Такие процессы можно отнести к классу простых вычислительных процессов. Процессы, которые мы будем рассматривать ниже, отнесем к классу вычислительных процессов реального времени и в последующем будем именовать ***РВ-процессами***.

Взаимодействие процессов

Следует отметить, что РВ-процессы к настоящему времени изучены недостаточно. По-видимому, наибольшее продвижение в этой области сделано в работах [14, 15]. Вместе с тем, условия запуска и взаимодействия РВ-процессов не укладываются в традиционные модельные схемы взаимодействий: ***синхронные, асинхронные и существенно асинхронные***. Последний тип взаимодействия введен в работах [15, 16] и предполагает наличие селекции передаваемых данных в зависимости от моментов их получения. Такие схемы [17] характеризуются тем, что накладывают жесткие ограничения на условия запуска и взаимодействия, что не позволяет строить модели, адекватно описывающие функционирование программной нагрузки с учетом динамических свойств окружающей системы. В этих условиях вопросы корректного учета динамики окружающей системы, то есть описания поведения системы во времени, приобретают особо важное значение.

Рассмотрим более подробно традиционные схемы организации взаимодействия процессов, например, в [49]. В таких схемах рассматриваются два способа организации параллельного выполнения процессов – синхронный и асинхронный. В первом из них запуск процессов синхронизируется по какому-либо регулярному, внешнему по отношению к процессу сигналу. Во втором способе процессы запускаются в недетерминированные моменты времени. Обычно – это некоторые предусловия, которые выполняются в результате работы предшествующих во времени процессов или наступления событий, порождаемых окружающей системой.

Разумеется, что к асинхронному способу можно отнести все многообразие условий взаимодействия за исключением тех случаев, когда процессы запускаются одновременно, то есть синхронно. Но даже в этой простой схеме классификации возникает неопределенность. Например, к какому способу следует отнести ситуацию, когда процессы запускаются в один момент времени, а при этом внешний по отношению к процессам сигнал синхронизации не является регулярным. Это означает, что процессы запускаются синхронно, но интервалы времени между очередными запусками не являются постоянной величиной. Неоднозначной является и такая ситуация, когда синхронизация запуска процессов производится по регулярному сигналу, но в интервале между сигналами процессы запускаются неравное число раз. Можно привести и ряд других примеров, для которых взаимодействие не укладывается в известные схемы.

Это показывает, что известные модели взаимодействия процессов требуют существенного развития. С этой целью предлагается определение РВ-процессов, которое более полно учитывает условия совместного функционирования вычислительных процессов в реальном времени.

Определение РВ-процессов

Вычислительный процесс строится на основе одного модуля либо информационно - связанной совокупности модулей, представленной в виде ГПД. РВ-процесс $\boldsymbol{p}$ представим следующей совокупностью записей:

$$\boldsymbol{p} = \{\boldsymbol{G(M,D,R)},\ \boldsymbol{Z(D_x)},\ \boldsymbol{Z(D_u)},\ \boldsymbol{\pi(D_x)}\}.$$

Здесь $\boldsymbol{G(M,D,R)}$ − алгоритм, выполняемый процессом и заданный в форме ГПД. Множество $\boldsymbol{M}$ включает модули алгоритма. Совокупность данных $\boldsymbol{D}$ состоит из двух подмножеств $\boldsymbol{D_x}$ и $\boldsymbol{D_u}$. $\boldsymbol{D_x}$ −

множество информационных входов модулей, $\boldsymbol{D_x = \bigcup D_{x_j}}$, $\boldsymbol{D_{x_j}}$ – множество информационных входов $\boldsymbol{j}$-го модуля. $\boldsymbol{D_u}$ – ***множество информационных выходов*** модулей, $\boldsymbol{D_u = \bigcup D_{u_j}}$, $\boldsymbol{D_{u_j}}$ – множество информационных выходов $\boldsymbol{j}$-го модуля. Множество $\boldsymbol{R}$ определяет информационные связи между модулями и данными.

$\boldsymbol{Z(D_x)}$ – ***условия поступления*** входных данных;

$\boldsymbol{Z(D_u)}$ – ***условия обновления*** выходных данных;

$\boldsymbol{\pi(D_x)}$ – ***приоритеты*** на обработку входных данных.

В условиях, когда РВ-процесс включает более одного модуля, часть вершин множества $\boldsymbol{D}$ являются ***промежуточными***, то есть одновременно принадлежат множествам $\boldsymbol{D_x}$ и $\boldsymbol{D_u}$, $\mathrm{D_x} \bigcap \mathrm{D_u} = \mathrm{D_t} \neq \varnothing$. На рис. 7.1 показан ГПД с множествами:

$$\boldsymbol{D_x = \{d_1, d_4, d_3\},\ D_u = \{d_2, d_3, d_5\},\ D_t = \{d_3\},\ M = \{m_1, m_2\},}$$

$$\boldsymbol{R = \{(d_1, m_1), (m_1, d_2), (m_1, d_3), (d_4, m_2), (m_2, d_5)\}.}$$

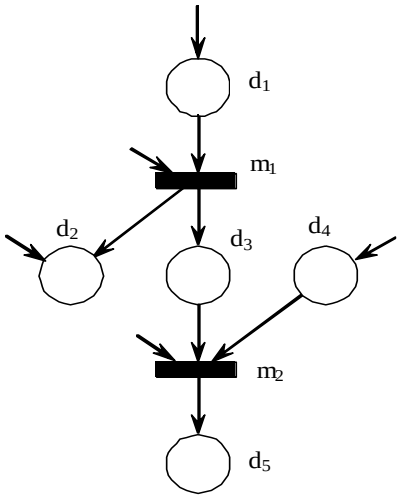


Рис. 7.1. Граф потока данных РВ-процесса

Условия поступления входных данных

Для вершин $\boldsymbol{d \in D_t}$ условия поступления $\boldsymbol{Z(d_x)}$ и обновления $\boldsymbol{Z(d_u)}$ отсутствуют. Напротив, для внешних входных вершин $\boldsymbol{D_x^* = D_x \setminus D_t}$ и для внешних выходных вершин $\boldsymbol{D_u^* = D_u \setminus D_t}$ условия поступления $\boldsymbol{Z(D_x^*)}$ и обновления $\boldsymbol{Z(D_u^*)}$ должны быть определены.

Условия поступления $\boldsymbol{Z(D_x^*)}$ входных данных $\boldsymbol{D_x^*}$ определяются на следующей области $\boldsymbol{Z_x^* = \{Ц, Д, В, У\}}$, где

Ц – ***циклическое поступление*** с фиксированным временем цикла ΔT;

Д – ***детерминированная последовательность*** моментов времени поступления, задаваемая в явном виде, либо функцией параметров с указанием их начальных значений;

В – ***вероятностное поступление*** с заданным законом распределения вероятности моментов времени поступления;

У – ***поступление сигнала при выполнении некоторого условия***, проверяемого модулем ГПД программной нагрузки, либо модулем алгоритма функционирования встроенной системы (сигнал программного прерывания).

Условия обновления выходных данных

Обновленные выходные данные могут поступать на вход нескольких процессов с разными условиями передачи данных. В этом случае вариант обновления выходных данных для процесса p_i должен учитывать всю специфику передачи этих данных другим процессам. С этой целью при определении условий обновления состояний выходных данных выполняется процедура согласования вариантов обновления на всем множестве процессов.

Варианты обновления состояний $Z(D_u^*)$ выходных данных D_u^* определяются на множестве $Z_u^* = \{ОС, ЦО, ЦП, КП, ТО, ТП, ВП\}$, где

$ОС$ − обновление текущего состояния ($С - обновление$);

$ЦО$ − циклическое обновление состояния с временем цикла ΔT ($Ц - обновление$);

$ТО$ − обновление состояния (получение нового) в течение времени ΔT ($Т - обновление$);

$ЦП$ − циклическое присоединение $К$, $К = const$, состояний по счетчику ($Ц - присоединение$);

$КП$ − присоединение $К$, $К = const$, состояний по счетчику ($К - присоединение$);

$ТП$ − присоединение состояний, полученных за время Δt, $\Delta t = const$, ($Т - присоединение$);

$ВП$ − присоединение $В$, $В \neq const$, состояний, вложенных в интервал времени, определяемый K_{p_i}, $K_{p_i} = const$, запусками процесса p_i ($В - присоединение$).

Последние четыре вида обновления формируют множества состояний. Каждое состояние при необходимости может сопровождаться порядковым номером и моментом времени получения. При $Ц$ − присоединении, в отличие от $К$ − присоединения, порядковые номера состояний k и моменты времени получения t связаны выражением $k = t/\Delta T$, ΔT − время цикла. Поэтому для состояния можно хранить только один параметр k или t. Кроме того, состояния в сформированном множестве всегда получаются за фиксированное время $\Delta t = k\Delta T$. При $К$ − присоединении величина Δt, определяющая время получения очередных $К$ состояний, является переменной.

При обновлении по типу присоединения вновь полученное состояние приписывается к формируемому множеству, а состояние с наиболее ранним моментом времени получения или наименьшим порядковым номером исключается из данного множества.

Заметим также, что обновление по типу $Т$ − присоединения может выполняться в двух вариантах. Последовательность формирования множеств состояний по обоим вариантам для семи интервалов Δt показана на рис. 7.2. В первом варианте множества состояний, представленные порядковыми номерами, обновляются в каждом интервале Δt путем приписывания новых состояний и удаления такого же числа состояний с минимальными порядковыми номерами. Так, в интервале Δt_3 получены состояния 8 и 9 и, соответственно, исключены состояния 4 и 5.

Вариант 1	1,2,3	4,5,6,7	6,7,8,9	6,7,8,9	7,8,9,10	11,12, 13,14	12,13, 14,15	…
Вариант 2	1,2,3	4,5,6,7	8,9		10	11,12, 13,14	15	…
	Δt_1	Δt_2	Δt_3	Δt_4	Δt_5	Δt_6	Δt_7	… t

Рис. 7.2. Варианты $Т$ − присоединения

Во втором варианте множество состояний в каждом интервале Δt формируется заново. Из рис. 7.2 видно, что в интервале Δt_4 новые состояния не формировались. Поэтому в протоколе взаимодействия процессов при необходимости могут использоваться состояния, полученные на предыдущем интервале Δt, то есть состояния $\{8, 9\}$.

Результатом обновления входных данных процесса p_i по типу присоединения является множество состояний, которое обозначим

как D_i. В зависимости от типа обновления мощность данного множества для $Ц$ − присоединения и $К$ − присоединения является постоянной, $|D_i| = const$, а для $Т$ − присоединения и $В$ − присоединения $|D_i| \neq const$. В соответствии с этим ***селекция состояний*** может производиться для фиксированной мощности множества D_i и для динамической, изменяющейся во времени. При этом селекция состояний из множества D_i может осуществляться по их порядковым номерам либо по моментам времени получения. В последнем случае чаще всего селекция состояний производится до некоторого момента времени или после него.

Приоритеты

Область определения приоритетов π включает ***статические*** или ***динамические*** приоритеты, которые назначаются вершинам множества D_x^*. По условиям назначения приоритеты различаются также на ***абсолютные*** и ***относительные***. Абсолютные приоритеты πx_j назначаются информационным входам x_j в виде чисел натурального ряда. При этом, меньшим числам соответствуют более высокие приоритеты. Относительные приоритеты задают отношение приоритетности для пары информационных входов с использованием операций больше, меньше или равенство. Запись $\pi x_{j_1} > \pi x_{j_2}$ означает, что приоритет входа x_{j_1} выше приоритета входа x_{j_2}. При этом абсолютные значения приоритетов входов x_{j_1} и x_{j_2} не указываются.

Статические абсолютные и ***статические относительные*** приоритеты остаются неизменными на всем интервале моделирования. ***Динамические абсолютные*** и ***динамические относительные*** приоритеты назначаются на ограниченный отрезок времени. При этом имеет место два способа назначения. В первом из них назначение новых приоритетов для некоторых входов осуществляется специальным модулем ГПД в ходе его выполнения. Переназначение приоритетов происходит всякий раз, когда запускается в работу данный модуль. Во втором способе динамическое назначение приоритетов выполняется на основе анализа ситуации, складывающейся в окружающей системе, в ходе ее функционирования. Переназначение приоритетов в этом случае осуществляют модули ГПД, описывающие алгоритм функционирования окружающей системы.

7.2. Взаимодействие РВ-процессов

Для организации совместной работы РВ-процессов необходимо определить существо взаимодействия. Традиционно взаимодействие определяется как операция передачи данных от процесса-производителя p_i к процессу-потребителю p_j. В общем случае процесс p_i может формировать совокупность выходных данных

$$D_i = \left\{ d_{i_1},\ d_{i_2},\ ...,\ d_{i_v},\ ...,\ d_{i_{V_i}} \right\},\quad v = 1,\ 2,\ ...,\ V_i\ ,$$

каждое из которых независимо друг от друга может передаваться другим процессам. Выходное данное $d_{i_v} \in D_i$ формируется при каждом запуске процесса p_i. Таким образом, каждому k − му запуску процесса p_i в момент времени t_k соответствует значение выходного данного $d_{i_V}(k)$, привязанное к k − му порядковому номеру запуска процесса p_i либо это же значение $d_{i_V}(t_k)$, привязанное к моменту времени t_k.

Величину $d_{i_V}(k)$ или $d_{i_V}(t_k)$ будем именовать ***состоянием процесса*** p_i по выходному данному d_{i_V}, полученному после запуска процесса p_i в момент времени t_k.

Рассматривая взаимодействие процессов p_i и p_j будем считать, что передается одно или несколько состояний процесса p_i по данному d_{i_v}. Передача нескольких состояний осуществляется на основе их селекции из некоторого множества $D_{i_v} = \left\{ d_{i_v}(k) \right\},\quad k = 1,2,...,K_{p_i}$ состояний, формируемого в результате K_{p_i} запусков процесса p_i.

Величина K_{p_i} может задаваться тремя способами. В первом способе значение K_{p_i} задается пользователем и определяет число запусков процесса p_i, для которых состояния $d_{i_v}(k)$ сохраняются. Во втором способе задается интервал времени, на котором контролируются все запуски процесса p_i и получаемые при этом состояния сохраняются. В этом случае число состояний в формируемом множестве D_{i_v} может меняться во времени. Третий способ отличается от второго тем, что интервал времени определяется как функция от числа запусков некоторого процесса p_j, взаимодействующего с процессом p_i. В этом случае необходимость формирования множества состояний D_{i_v}, на котором осу-

ществляется селекция состояний для передачи другим процессам, обусловлена зависимостью множества D_{i_v} от числа запусков процесса p_j. Таким образом, возникает потребность во ***взаимном контроле*** числа запусков взаимодействующих процессов p_i и p_j. Данную ситуацию рассмотрим на примере рис. 7.3.

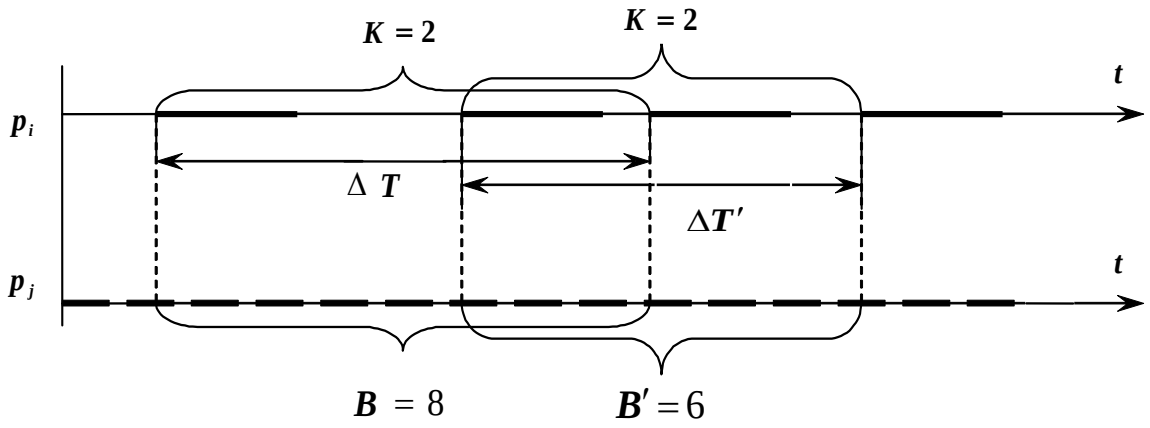


Рис. 7.3. Пример взаимного контроля числа запусков процесса

Пример на рис. 7.3 иллюстрирует контроль запусков процесса p_j на интервале времени ΔT, определяемом двумя запусками процесса p_i, $K = 2$. В интервал ΔT вкладывается 8 запусков процесса p_j, $B = 8$. На примере показано также, что для интервала $\Delta T'$, определяемого двумя следующими запусками процесса p_i, величина $B' = 6$. Следовательно, число запусков процесса p_j, контролируемое на интервале времени, определяемом двумя запусками процесса p_i, изменяется во времени, то есть величина B в данном случае не является постоянной. Если процесс p_i будет запускаться циклически с фиксированным временем цикла Δt, то очевидно, что величина B будет постоянной во времени.

Таким образом, применение механизма взаимного контроля запусков процессов p_i и p_j для формирования множества состояний D_{i_v}, приводит к необходимости связывать определение форм взаимодействия параллельных процессов с условиями передачи данных и условиями запуска процессов. Условия запуска процессов могут также оказывать влияние на алгоритм (протокол) взаимодействия процессов при передаче данных. Поэтому, в последующем, при разработке методов организации взаимодействия процессов будем учитывать два связанных между собой вида взаимодействия:

- по условиям запуска процессов;
- по условиям передачи данных.

В общем случае взаимодействие процессов p_i и p_j по условиям запуска зависит от следующих параметров:

- условия запуска процессов;
- условия обновления состояний выходных данных;
- условия селекции состояний выходных данных для передачи процессу потребителю;
- условия взаимного контроля запусков взаимодействующих процессов и фиксации множества D_{i_v}.

Взаимодействие по условиям запуска

Для описания взаимодействия процессов p_i и p_j по условиям запуска введем совокупность параметров $T = \left\{\Delta t_i, t_{i_k}, t_{i_0}, \tau_i, \tau_{ij}\right\}$:

Δt_i − цикл работы процесса p_i;

t_{i_k} − k - й момент запуска процесса p_i;

t_{i_0} − момент начального запуска процесса p_i;

τ_i − время выполнения процесса p_i;

τ_{ij} − время получения данных процессом p_i для использования процессом p_j.

В комбинации условий запуска процессов p_i, p_j предполагается, что на первом месте располагается процесс производитель p_i, на втором месте – процесс потребитель p_j. Рассмотрим комбинации условий запуска с учетом временных параметров совокупности T запуска и выполнения процессов.

Комбинация ЦЦ:

- синхронный циклический запуск при соблюдении условий $\Delta t_i = \Delta t_j, t_{i_0} = t_{j_0}$;
- синхронный ВК − запуск при соблюдении условий $\Delta t_i < \Delta t_j,$ $\Delta t_i B = \Delta t_j K$; B, K − целые положительные числа;
- синхронный КВ−запуск при соблюдении условий $\Delta t_i > \Delta t_j,$ $\Delta t_i K = \Delta t_j B$;

- асинхронный запуск при условии $\Delta t_i \neq \Delta t_j$, параметры B и K не контролируются;
- последовательный запуск при соблюдении условий $\Delta t_i = \Delta t_j$, $2\Delta t_i \geq \tau_i + \tau_j$, $t_{j_k} = t_{i_k} + \tau_i$;
- последовательный запуск по наличию данных при соблюдении условий $t_{j_k} = t_{ik} + \tau_{ij}$, $\Delta t_i = \Delta t_j$, $2\Delta t_i \geq \tau_{ij} + \tau_j$.

Комбинация ЦД:

- асинхронный запуск при отсутствии контроля параметров t_{i_0}, t_{j_0}, B, K;
- асинхронный ВК − запуск при условии контроля $m_{p_i}\left(K_{p_j}\right)$ состояний процесса p_i за время получения K состояний процесса p_j, $0 \leq m_{p_i}\left(K_{p_j}\right) \leq B$;
- последовательный запуск при условии, что функция последовательности моментов запуска процесса p_j имеет вид $t_{j_k} = t_{i_k} + \tau_i$;
- последовательный запуск по наличию данных при условии, что функция последовательности моментов запуска процесса p_j имеет вид $t_{j_k} = t_{i_k} + \tau_{ij}$.

Комбинация ЦВ:

- асинхронный запуск при отсутствии контроля параметров B, K;
- асинхронный ВК − запуск при условии контроля $m_{p_i}\left(K_{p_j}\right)$ состояний процесса p_i за время получения K состояний процесса p_j, $0 \leq m_{p_i}\left(K_{p_j}\right) \leq B$.

Комбинация ЦУ:

- асинхронный запуск при отсутствии контроля параметров B, K;

- асинхронный ВК − запуск при условии контроля $\boldsymbol{m}_{p_i}\left(\boldsymbol{K}_{p_j}\right)$ состояний процесса $\boldsymbol{p_i}$ за время получения $\boldsymbol{K}$ состояний процесса $\boldsymbol{p_j}$, $0 \le \boldsymbol{m}_{p_i}\left(\boldsymbol{K}_{p_j}\right) \le \boldsymbol{B}$;
- последовательный запуск при условии, что условием запуска процесса $\boldsymbol{p_j}$ является завершение выполнения процесса $\boldsymbol{p_i}$ и при этом $\tau_j \le \boldsymbol{t}_{i_k} - \boldsymbol{t}_{i_{k-1}} = \Delta \boldsymbol{t}_i$;
- последовательный запуск по наличию данных при условии, что условием запуска процесса $\boldsymbol{p_j}$ является получение данных процессом $\boldsymbol{p_i}$ для использования процессом $\boldsymbol{p_j}$, $\boldsymbol{t}_{j_k} = \boldsymbol{t}_{i_k} + \tau_{ij}$.

Комбинация ДЦ, ВЦ, УЦ:

- асинхронный запуск при отсутствии контроля параметров $\boldsymbol{K}, \boldsymbol{B}$;
- асинхронный КВ − запуск при условии контроля $\boldsymbol{m}_{p_j}\left(\boldsymbol{K}_{p_i}\right)$ состояний процесса $\boldsymbol{p_j}$ за время получения $\boldsymbol{K}$ состояний процесса $\boldsymbol{p_i}$, $0 \le \boldsymbol{m}_{p_j}\left(\boldsymbol{K}_{p_i}\right) \le \boldsymbol{B}$.

Комбинация ДД:

- синхронный запуск при условии $\boldsymbol{t}_{i_0} = \boldsymbol{t}_{j_0}$ и равенстве последовательностей моментов детерминированного запуска процессов $\boldsymbol{p_i}$ и $\boldsymbol{p_j}$ или параметров функций, описывающих данные последовательности;
- асинхронный запуск при отсутствии контроля параметров последовательностей;
- последовательный запуск при условии, что функция последовательности моментов запуска процесса $\boldsymbol{p_j}$ имеет вид $\boldsymbol{t}_{j_k} = \boldsymbol{t}_{i_k} + \tau_i$;
- последовательный запуск по наличию данных при условии, что функция последовательности моментов запуска процесса $\boldsymbol{p_j}$ имеет вид $\boldsymbol{t}_{j_k} = \boldsymbol{t}_{i_k} + \tau_{ij}$.

Комбинации ДВ, ВД, ВВ, УД, УВ:

- асинхронный запуск.

Комбинация ДУ, ВУ, УУ:

- асинхронный запуск при условии отсутствия контроля параметров моментов запуска;
- последовательный запуск при условии, что условием запуска процесса $\boldsymbol{p_j}$ является завершение выполнения процесса $\boldsymbol{p_i}$ и при этом $\tau_j \le \boldsymbol{t}_{i_k} - \boldsymbol{t}_{i_{k-1}} = \Delta \boldsymbol{t}_i$;
- последовательный запуск по наличию данных при условии, что условием запуска процесса $\boldsymbol{p_j}$ является получение данных процессом $\boldsymbol{p_i}$ для использования процессом $\boldsymbol{p_j}$.

На основе анализа параметров запуска процессов, обновления состояний, селекции и взаимного контроля запусков сформулируем совокупность вариантов взаимодействия процессов по условиям их запуска:

- ***синхронный запуск*** (***C***);
- ***синхронный запуск через K состояний процесса потребителя***, процесс производитель укладывается ***B*** раз в ***K*** циклов процесса потребителя (***CBK***);
- ***синхронный запуск через K состояний процесса производителя***, процесс потребитель укладывается ***B*** раз в ***K*** циклов процесса потребителя (***CKB***);
- ***асинхронный запуск*** (***A***);
- ***асинхронный запуск с контролем B состояний процесса производителя***, полученных за время получения ***K*** состояний процесса потребителя (***ABK***);
- ***асинхронный запуск с контролем B состояний процесса потребителя***, полученных за время получения ***K*** состояний процесса производителя (***AKB***);
- ***последовательный запуск***, процесс потребитель запускается по завершению работы процесса производителя (***П***);
- ***запуск по наличию данных***, процесс потребитель запускается как только процесс производитель получит данные для процесса потребителя (***ПД***).

Варианты взаимодействий в зависимости от условий запуска представим таблицей.

Виды взаимодействий процессов p_i и p_j по условиям запуска	Комбинации условий запуска процессов p_i и p_j
Синхронный запуск (С)	ЦЦ, ДД
Синхронный ВК − запуск (СВК)	ЦЦ
Синхронный КВ − запуск (СКВ)	ЦЦ
Асинхронный запуск (А)	ЦЦ, ЦД, ЦВ, ЦУ, ДЦ, ВЦ, УЦ, ДД, ДВ, ВД, ВВ, УД, УВ, ДУ, ВУ, УУ
Асинхронный ВК − запуск (АВК)	ЦД, ЦВ, ЦУ
Асинхронный КВ − запуск (АКВ)	ДЦ, ВЦ, УЦ
Последовательный запуск (П)	ЦЦ, ЦД, ЦУ, ДД, ДУ, ВУ.УУ
Последовательный запуск по наличию данных (ПД)	ЦЦ, ЦД, ЦУ,ДД, ДУ, ВУ, УУ

В левой колонке таблицы в скобках указаны краткие обозначения видов взаимодействия процессов по условиям запуска.

Взаимодействие по условиям передачи данных

При взаимодействии процессов p_i и p_j по передаче данных от p_i к p_j рассматриваются выходные данные, формируемые процессом p_i в виде определенной совокупности состояний. Процесс p_i в зависимости от условий обновления выходных данных может формировать одно или несколько состояний. При этом возможны два варианта взаимодействия процессов p_i и p_j по условиям передачи данных.

Первый вариант соответствует жесткой схеме взаимно-однозначного соответствия между запуском процессов и передачей данных. В этом случае всегда передается одно состояние. Взаимодействие по такому условию передачи данных названо ***PP − взаимодействием***.

Во втором варианте передача одного или нескольких состояний может происходить в любое время и не связывается с запуском процессов. Взаимодействие по такой схеме относится к ситуации, когда передаются все данные, полученные процессом производителем к данному моменту времени в соответствии с принятым условием обновления. Такой вид взаимодействия назван ***VV − взаимодействием***.

Для случаев, когда по условиям обновления процесс производитель формирует совокупность состояний, передача данных может осуществляться с учетом параметра вида селекции состояний S. Различается два вида селекции:

- ***селекция*** одного или нескольких состояний ***по порядковым номерам*** их получения;
- ***селекция*** состояний ***по времени получения***.

В последнем случае селекция одного состояния осуществляется по конкретному моменту времени получения. Селекция многих состояний может осуществляться по различным правилам. Чаще всего выбор многих состояний производится до некоторого момента времени или после него.

Множества состояний, на которых производится селекция, могут иметь фиксированные размеры или изменяемые во времени (динамические размеры). В соответствии с этим взаимодействие по передаче данных с учетом селекции состояний для фиксированного множества состояний названо ***SF - взаимодействием***, а для динамического множества состояний ***SD - взаимодействием***.

Заметим также, что условия обновления в значительной мере предопределяют перечисленные выше виды взаимодействий по передаче данных. Возможные варианты соответствия между условиями обновления и видами взаимодействий представлены на рис. 7.4. Отношение соответствия на рис. 7.4 показано ребрами, связывающими соответствующие условия обновления и виды взаимодействий.

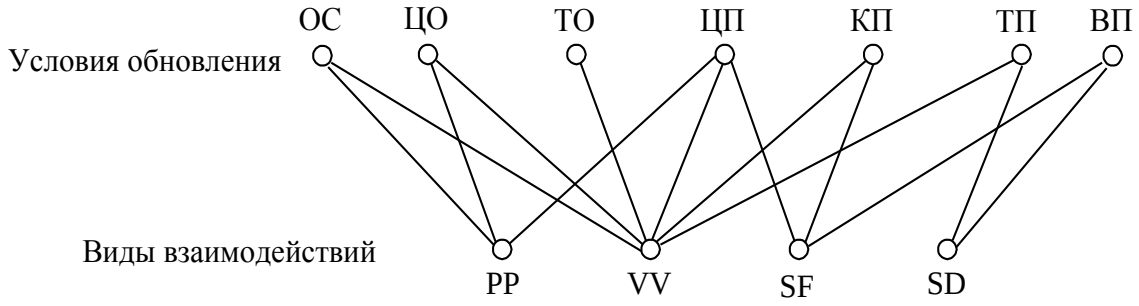


Рис. 7.4. Варианты соответствия между условиями обновления и видами взаимодействий

На основе типов взаимодействий по условиям запуска и типов взаимодействий по условиям передачи данных определяются варианты соответствия между ними:

$$\boldsymbol{Z(p_i, p_j) \leftrightarrow S(p_i, p_j)}.$$

Здесь $Z(p_i, p_j) = \{C, СВК, СКВ, А, АВК, АКВ, П, ПД\}$ – совокупность типов взаимодействий по условиям запуска;

$S(p_i, p_j) = \{PP, VV, SF, SD\}$ – совокупность типов взаимодействий по условиям передачи данных.

Варианты соответствия одноканальных взаимодействий процессов по схеме производитель (процесс p_i) – потребитель (процесс p_j) приведены на рис. 7.5.

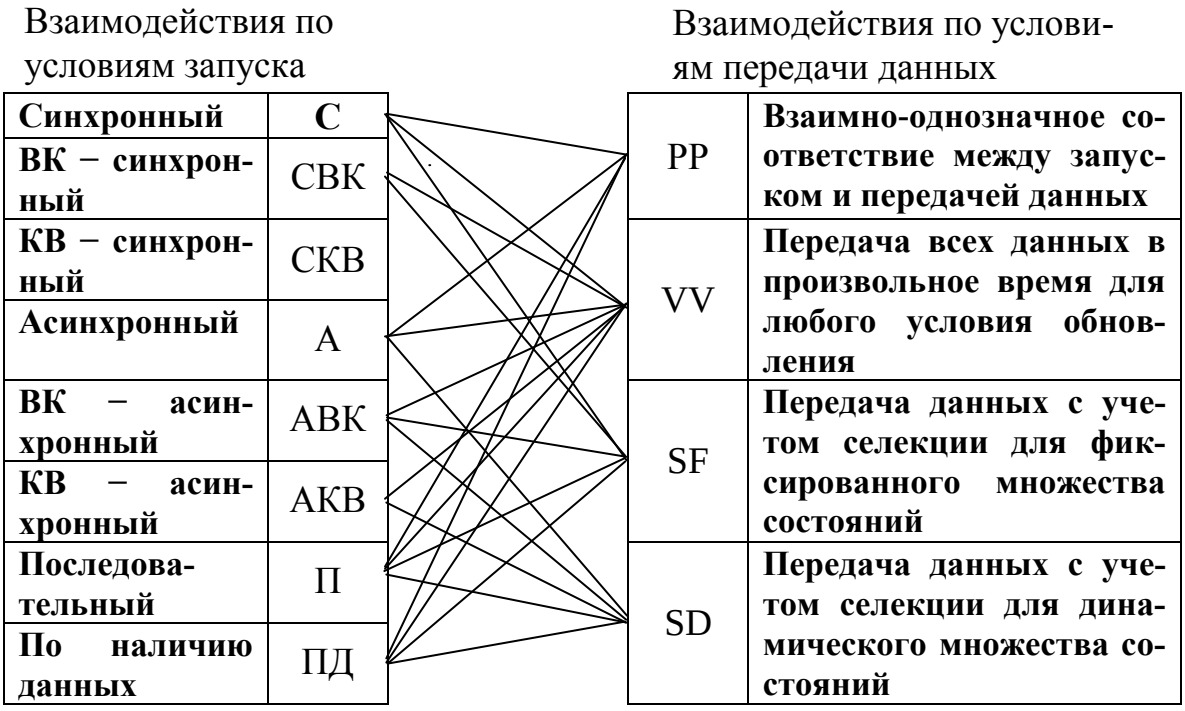


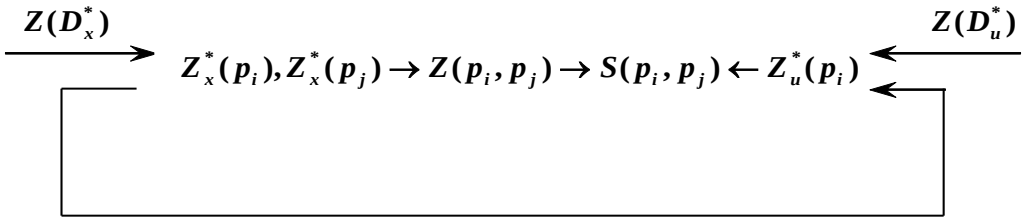
Рис.7.5. Варианты соответствия одноканальных взаимодействий процессов

7.3. Канальная функция взаимодействия процессов

Отношения между условиями запуска $Z(p_i, p_j)$ и условиями передачи данных $S(p_i, p_j)$ завершают анализ согласованности параметров, определяющих канальную функцию $f(K_{ij})$ взаимодействия процессов p_i и p_j. Канальная функция $f(K_{ij})$ определяется как совокупность отношений на множестве значений рассмотренной совокупности параметров:

$$Z = \left\{Z(D_x^*), Z(D_u^*), Z_x^*(p_i), Z_x^*(p_j), T, K, B, S, \pi, Z(p_i, p_j), S(p_i, p_j), Z_u^*(P_i)\right\}.$$

Для определения канальной функции $f(K_{ij}) = F(Z)$ выделим основные параметры множества Z и отношения между ними представим следующей схемой:



Данная схема определяет последовательность действий алгоритма по установлению существования оператора F в построении канальной функции.

На первом этапе алгоритма осуществляется анализ условий поступления $Z(D_x^*)$ и определение условий запуска процессов p_i и p_j – параметры $Z_x^*(p_i), Z_x^*(p_j)$.

Второй этап алгоритма включает анализ параметров $T, K, B, S, \pi, Z_x^*(p_i), Z_x^*(p_j)$, на основе которых выявляется тип взаимодействия процессов по условиям запуска $Z(p_i, p_j)$. Заметим, что значения параметров $T, K, B, S, \pi, Z_x^*(p_i), Z_x^*(p_j)$ могут варьироваться и порождать несколько приемлемых типов взаимодействий $Z(p_i, p_j)$. В последующем рассмотрению подлежит вся полученная совокупность $Z(p_i, p_j)$.

На третьем этапе согласно рис. 7.5 формируется множество типов взаимодействий по условиям передачи данных $S(p_i, p_j)_x$, приемлемых для совокупности $Z(p_i, p_j)$, полученной на втором этапе алгоритма.

На четвертом этапе на основе анализа условий обновления выходов $Z(D_u^*)$, параметров $Z_x^*(p_i), Z_x^*(p_j), S, K, B$ определяется условие обновления состояний $Z_u^*(p_i)$ процесса p_i.

На пятом этапе согласно рис. 7.4 формируется множество типов взаимодействий по условиям передачи данных $S(p_i, p_j)_u$, приемлемых для одного или нескольких условий обновления $Z_u^*(p_i)$, полученных на четвертом этапе.

Шестой этап осуществляет проверку условия:

$$S(p_i, p_j)_x \cap S(p_i, p_j)_u \neq \varnothing$$

Выполнение условия означает, что на данной совокупности значений параметров Z, соответствующих оператору F и определяющих канальную функцию $f(K_{ij})$, существует одна или несколько ***непротиворечивых систем отношений*** между параметрами. Если условие не выполняется, то это означает, что на принятой совокупности значений параметров Z нельзя организовать взаимодействие процессов по каналу K_{ij}. В этом случае нужно корректировать значения параметров и осуществлять поиск непротиворечивой системы отношений.

Систему отношений между значениями основных параметров, выделенных на схеме, представим графом $G = (V, R)$. Множество вершин V соответствует значениям параметров, а множество ребер R – допустимым отношениям между параметрами. Множество $V = V_{Z_x} \cup V_{Z_p} \cup V_{S_p} \cup V_{Z_u}$ объединяет подмножества вершин $V_{Z_x}, V_{Z_p}, V_{S_p}, V_{Z_u}$, которые соответствуют подмножествам параметров $Z_x^* \times Z_x^*, Z(p_i, p_j), S(p_i, p_j), Z_u^*(p_i)$.

Множество $R = R(V_{Z_x}, V_{Z_p}) \cup R(V_{Z_p}, V_{S_p}) \cup R(V_{S_p}, V_{Z_u})$ объединяет три подмножества ребер $R(V_{Z_x}, V_{Z_p}), R(V_{Z_p}, V_{S_p}), R(V_{S_p}, V_{Z_u})$, которые соответствуют допустимым отношениям между значениями параметров согласно представленной схеме.

Каждому каналу на графе G соответствует маршрут с началом в вершине множества V_{Z_x} и концом в вершине V_{Z_u}. Отсутствие маршрута означает, что принятая совокупность значений параметров не позволяет сформировать непротиворечивую систему отношений для соответствующего канала. В этом случае для определения оператора канальной функции необходимо уточнять значения параметров Z.

7.4. Активная модель программной нагрузки

После определения значений параметров Z, фиксирующих соответствующие канальные функции взаимодействия РВ-процессов, модель программной нагрузки, представленная в форме ГПД, наделяется способностью выполняться на ПВМ. С этой целью для каждого модуля составляется паспорт, который содержит значения параметров множества Z, относящихся к данному модулю, и воспринимается командой ПВМ.

Ранее отмечалось, что в ГПД модуль может быть представлен как программой, так и имитатором соответствующего алгоритма. Если модуль представлен программой, то в ГПД непосредственно можно использовать данную программу, либо ее имитатор. В случае, если мы имеем дело с алгоритмическим модулем, то в ГПД может использоваться лишь его имитатор. Независимо от формы представления модуль имеет один паспорт, который сопровождает его в соответствующих библиотеках модулей и библиотеках имитаторов.

Паспорт модуля содержит следующие позиции:

1. Имя модуля <комментарий>;
2. Входы <множество $\boldsymbol{D_x}$ >;
3. Выходы <множество $\boldsymbol{D_u}$ >;
4. Время выполнения <параметр τ >;
5. Объем вычислений <параметр ρ >;
6. Требуемая память <параметр $\boldsymbol{L}$ >;
7. Используемые ресурсы <имя и величина>;
8. Станция приписки <имя>;
9. Имя входа <комментарий>;
10. Условия поступления <параметр $\boldsymbol{Z(D_x^*)}$ >;
11. Условия селекции <параметр $\boldsymbol{S}$ >;
12. Приоритет <параметр π >;
13. Требуемая память <параметр $\boldsymbol{L}$ >;
14. Условия хранения <совокупность записей>;
15. Описание на языке <запись>;
16. Имя выхода <комментарий>;
17. Условия обновления <параметр $\boldsymbol{Z(D_u^*)}$ >;
18. Требуемая память <параметр $\boldsymbol{L}$ >;
19. Условия хранения <совокупность записей>;
20. Описание на языке <запись>.

Некоторые позиции паспорта требуют пояснений. В позициях "входы" и "выходы" указываются соответствующие номера $\boldsymbol{d_i \in D}$, а информация о каждом входе последовательно приводится в позициях 9 – 15, о каждом выходе в позициях 16 – 20. В позиции "время выполнения" (поз. 4) указывается прогнозируемое или определяемое экспериментально время выполнения модуля на инструментальной ЭВМ. Если модуль реализован программно и непосредственно используется при моделировании, то данное значение времени будет уточнено. При использовании имитаторов модуля приведенное значение применяется при вычислении времени для отображения на временной диаграмме и

при планировании использования ресурсов. Объем вычислений (поз. 5) задается числом команд, выполняемых при реализации модуля. Величина требуемой памяти (поз. 6) помимо рабочей памяти, необходимой для выполнения модуля, включает память для размещения программы модуля. По использованию разделяемых ресурсов (поз. 7) принято соглашение о том, что, если модуль использует ресурс (за исключением процессора), то он считается занятым на весь интервал времени выполнения модуля. В позиции 8 указывается номер станции, на которую распределен модуль.

Начиная с позиции 9, поочередно описываются каждый из входов $\boldsymbol{d_i}$, перечисленных в позиции 2. При задании условий поступления входа (поз. 10) указывается один из типов: Ц, В, Д, У. Каждый из типов сопровождается соответствующими параметрами:

- для Ц указывается время цикла;
- для В указывается закон распределения вероятностей поступления входа $\boldsymbol{d_i}$ (сигнала прерывания) и значения параметров данного закона;
- для Д указывается последовательность моментов запуска в явной форме или функцией. При задании функцией должно быть указано также число моментов в последовательности, либо ограничение на время формирования данных согласно заданной функции;
- тип У для входа $\boldsymbol{d_i}$ соответствует взаимодействию процессов по условиям запуска по типу П или ПД, а также выполняет функции программного прерывания при соблюдении определенных условий. Правила формирования сигнала У реализуются либо соответствующим модулем, либо управляющей программой.

Позиция 11 заполняется для данных, которые получены модулями ГПД, и содержит информацию об условиях селекции состояний описываемого входа $\boldsymbol{d_i}$ модулем потребителем. Данная информация должна быть согласована с условиями обновления соответствующего выхода в паспорте модуля производителя (позиция 17). При задании условий селекции указывается, какие состояния входа $\boldsymbol{d_i}$ из общего числа запоминаемых состояний выбираются для модуля потребителя. Множество запоминаемых состояний указывается при описании условий обновления выходов в паспорте модуля производителя. Из этого следует, что условия селекции для модулей потребителей данного $\boldsymbol{d_i}$ определяют условия обновления данного $\boldsymbol{d_i}$ для модуля производителя.

В позиции 14 указывается конкретная память МВС для хранения конкретного входа, и приводятся дополнительные сведения по хране-

нию данных независимо от условий поступления и обновления (наличие копий, состояний для восстановления и т.п.), которые могут использоваться управляющей программой.

Начиная с позиции 16, последовательно описывается каждый выход, указанный в позиции 3. В позиции 17 приводится одно из условий обновления и значения соответствующих параметров. При циклическом обновлении указывается время цикла ΔT. В условиях циклического присоединения дополнительно указывается число присоединяемых состояний K.

Для T - обновления указывается допустимый интервал времени с момента поступления сигнала на прерывание до получения выхода. Условия обновления по типу K - присоединение разрешает обновлять данные после получения K - состояний. При этом обновление K - состояний производится последовательно, начиная с первого, так чтобы каждый из K - состояний были доступны в любой момент времени. T - присоединение отличается от K - присоединения лишь тем, что состояния присоединяются по мере поступления в течение определенного отрезка времени. После чего процесс обновления производится так же как и при K - присоединении.

Условия хранения выходов задаются в позициях 19 аналогично тому, как это делается для входов в позициях 14.

Модель программной нагрузки, представленная совокупностью команд виртуальной машины (ПВМ), приобретает свойство активности и может быть выполнена на ПВМ. Полученная таким образом ГПД-программа наряду с виртуальной машиной и ее операционной системой образует активную модель.

В активной модели ПВМ является программой для инструментальной ЭВМ, операционная система которой реализует следующие основные функции:

- запуск РВ-процессов;
- продвижение РВ-процессов;
- обработка приоритетных прерываний;
- маршрутизация каналов взаимодействия РВ-процессов по каналам связи локальной сети МВС;
- отображение продвижения процессов на временной диаграмме;
- сохранение состояний РВ-процесса;
- обработка директив оператора.

При выполнении активной модели ПВМ выступает в роли виртуального процессора, имитирующего работу процессоров станций МВС. В общем случае МВС может включать процессоры различных типов. В

этом случае виртуальный процессор перед выполнением очередного модуля ГПД настраивается на конкретный тип процессора. Имеется и ряд других вспомогательных функций, которые можно рассматривать как компоненты операционной системы ПВМ.

Запуск РВ-процессов осуществляется согласно условиям поступления, указанным в позициях 10 соответствующих паспортов модулей.

Продвижение выполнения активных РВ-процессов через виртуальный процессор производится в очередной интервал времени Δt. Далее для следующего интервала Δt определяется совокупность активных РВ-процессов и производится их продвижение. При этом факт продвижения РВ-процессов отображается на временной диаграмме.

Задача маршрутизации каналов взаимодействия РВ-процессов решается при настройке команд ПВМ после решения задачи распределения модулей, программ и данных по ресурсам МВС. Оптимальные маршруты взаимодействия процессов отображаются в активной модели совокупностью имитаторов компонентов архитектуры МВС, по которым проходит маршрут. Имитаторы компонентов оформляются так же как и модули ГПД и выполняются на ПВМ при моделировании, имитируя работу каналов связи локальной сети МВС.

7.5. Планирование использования ресурсов

После определения канальных функций и множества команд ПВМ можно считать, что активная модель программной нагрузки, реализующей прикладные функции СРВ, построена. Данная модель с помощью ПВМ может выполняться на модели МВС и при отсутствии плана использования ресурсов. Правило использования ресурсов при этом может быть самым простым, например, когда очередная потребность в ресурсе удовлетворяется первым свободным ресурсом. В этом случае эффективность использования ресурсов может оказаться низкой. Поэтому перед выполнением активной модели необходимо построить план более эффективного использования ресурсов.

Задача распределения модулей, программ и данных в оптимизационной постановке как задача нелинейного математического программирования сформулирована в разделе 6. В условиях многократной трансформации ГПД и модели архитектуры МВС алгоритм решения данной задачи должен быть простым и эффективным. Поэтому в условиях значительной эволюции моделей, характерной для данного этапа, был предложен простой приближенный алгоритм. В данном случае, когда больших трансформаций моделей не предполагается, к алгоритму распределения программной нагрузки по ресурсам МВС предъявляются

другие требования. Приоритетным становится требование нахождения оптимального решения или близкого к оптимальному. Поиск оптимального решения в приведенной постановке для нелинейного варианта задачи приводит к большим трудностям. При линеаризации задачи резко возрастает ее размерность, что также ограничивает применение известных методов решения для таких задач. Поэтому возникает необходимость в более глубоком анализе содержания задачи и выявлении возможных подходов к ее решению, учитывающих конкретную специфику.

Обратимся к функциональному ГПД, изображенному на рис. 7.8. Объектами размещения являются:

- данные, требующие память для их хранения;
- алгоритмы модулей, требующие процессоры для их выполнения;
- программы реализации модулей, требующие память для их хранения.

Архитектура МВС представлена совокупностью станций, объединенных локальной сетью. Каждая станция имеет процессор и память. При распределении данных на станцию используется ресурс памяти. В общем случае модуль может распределяться на две станции – алгоритм модуля, загружая процессор одной станции, и программа модуля – память другой станции. В предположении, что для каждого модуля написана своя программа, можно для упрощения задачи размещать модуль на одной станции. Возможное при этом снижение качества решения задачи будет незначительным. Это обусловлено тем, что число размещаемых на каждую станцию компонентов информации (данных и программ модулей) достаточно велико. Так, для рассматриваемого примера число таких компонентов составляет 71, а число станций, например, 4. Тогда, на каждую станцию размещается в среднем по 6 модулей и по 12 данных. В этом случае ограничение замены программы модуля на данные или наоборот приводит к относительно небольшому сокращению области поиска решений. Таким образом, компонентами распределения в рассматриваемой задаче принимаются модули и данные.

Модули и данные образуют РВ-процессы, которые запускаются и взаимодействуют по определенным правилам. В связи с этим возникает необходимость в определении параметров распределяемых компонентов. С этой целью динамическая модель программной нагрузки преобразуется в статическую форму. Для модулей в этой форме указаны объем памяти и объем вычислений, для данных – объем памяти. Дуги указывают, какие данные потребляются и производятся модулем при его выполнении на процессоре.

Для определения параметров введем расчетный цикл работы динамической модели. Время цикла определяется как наименьший общий

знаменатель времен циклов поступления входов РВ-процессов. Для входов, поступающих по условиям типа ***В***, ***Д***, ***У***, моделируется число моментов поступления входов за время расчетного цикла. Для циклически поступающих входов число моментов поступления получается делением расчетного цикла на время цикла соответствующего входа.

Размер памяти, необходимой для размещения программы модуля, указан в паспорте модуля. Для данных размер памяти вычисляется на основе значений, указанных в паспорте с учетом условий обновления данных и их селекции, то есть числа сохраняемых состояний.

Объем вычислений для каждого модуля определяется как произведение объема вычислений, указанного в паспорте модуля, на число запусков РВ-процесса, к которому принадлежит модуль, за время расчетного цикла. Аналогично, объем передаваемых данных по дугам графа определяется как произведение размера памяти, вычисленного для соответствующего данного, на число запусков РВ-процесса, потребляющего или производящего эти данные.

Таким образом, для решения задачи распределения мы имеем статическую графовую форму динамической модели программной нагрузки, на которой определены объемы вычислений модулей, размер памяти, необходимой для хранения модуля и данного, а также объем данных, передаваемых по дугам графа.

Естественным критерием качества решения задачи распределения является нахождение варианта распределения модулей и данных по станциям, доставляющего минимальный суммарный объем данных, передаваемых в локальной сети МВС. Исходя из названного критерия при решении задачи важно, чтобы модули и данные, между которыми передается большой объем информации, были распределены на одну станцию. Следовательно, решение задачи должно быть таким, чтобы суммарный объем данных передаваемых по дугам, связывающим модули и данные, размещенные в разных станциях, был минимальным. Данное решение соответствует известной задаче разрезания графа на минимально связанные части, которая широко применяется при решении задачи компоновки на конструкторском этапе проектирования вычислительных устройств [38].

Задача разрезания

В известной постановке задача формулируется следующим образом. Требуется разрезать граф $G=(V,S)$ на части $G_i=(V_i,S_i)$, $i=1,2,...,n$, где

n – число частей, на которые разбивается граф;

V_i – множество вершин принадлежащих i – й части;

S_i – множество ребер инцидентных вершинам V_i.

Совокупность частей $B(G_i)$ называется разрезанием графа G, если

$$\forall G_i \in B(G_i)\left[G_i \neq \varnothing\right],\ \bigcup_i G_i = G,\ i = 1,2,...,n;$$

$$\forall G_i, G_j \in B(G_i)\left[G_i \neq G_j\ \&\ V_i \cap V_j = \varnothing\ \&\ S_i \cap S_j = S_{ij}\right],\ i, j = 1,2,...,n.$$

Здесь S_{ij} – множество ребер, попадающих в разрез между частями G_i и G_j. Обозначим $|S_{ij}| = r_{ij}$ и назовем его числом реберного соединения G_i и G_j. Тогда число реберного соединения разрезания графа G определяется величиной r,

$$r = \sum_{i=1}^{n}\sum_{j=1}^{n} r_{ij},\ i \neq j.$$

Традиционным критерием решения задачи разрезания является минимизация числа реберного соединения r при ограничении на число вершин в частях G_i.

Применительно к рассматриваемой задаче распределения модулей и данных имеем статическую графовую форму динамической модели в виде взвешенного графа $G = (V, Z, R)$, где $V = D \cup F$, D – множество вершин данных, F – множество вершин модулей, R – веса ребер множества Z. Вес $r_{gt} \in R,\ g, t = 1,2,...,m$ ребра z_{gt} равен объему данных, передаваемых по ребру за время расчетного цикла. Введем также следующие обозначения:

p_g – размер памяти, необходимый для размещения вершины $v_g \in V$, $g = 1,2,...,m$;

q_g – объем вычислений модуля, соответствующего вершине $v_g \in F$, для вершин $v_g \in D$, $q_g = 0$;

r_{ij} – взвешенное число реберного соединения частей G_i и G_j,

$$r_{ij} = \sum_{z_{g\ell} \in Z_{ij}} r_{gt}.$$

Размер частей G_i определяется ресурсами станций $c_i \in C$ по памяти P_i и общему объему вычислений Q_i. Здесь C – множество станций в МВС. В принятых обозначениях задача разрезания графа G на n частей G_i, $i = 1,2,...,n$, запишется в виде

$$\min\ r = \sum_{i=1}^{n}\sum_{j=1}^{n} r_{ij}\,; \tag{7.1}$$

$$\sum_{v_g \in V_i} p_g \le P_i,\ i = 1,2,...,n; \tag{7.2}$$

$$\sum_{v_g \in V_i} q_g \le Q_i,\ i = 1,2,...,n. \tag{7.3}$$

Разрезание графа G производится на n частей, по числу станций в МВС. При этом должны соблюдаться условия

$$\sum_{v_g \in V} p_g \le \sum_{i=1}^{n} P_i,\ \sum_{v_g \in F} q_g \le \sum_{i=1}^{n} Q_i\,. \tag{7.4}$$

Выполнение условий (7.4) обуславливает возможность решения задачи (7.1) – (7.3), то есть ресурсов МВС должно быть достаточно для размещения всех вершин графа с учетом значений параметров по памяти и объемам вычислений. Как правило, значения P_i, Q_i задаются с некоторым коэффициентом запаса.

Алгоритмы решения задачи разрезания

Среди известных алгоритмов разрезания графа существуют точные алгоритмы, основанные на методах решения задач дискретного программирования, и приближенные. Большее распространение получили приближенные алгоритмы, среди которых выделяются последовательные, итерационные и смешанные алгоритмы.

В приближенных алгоритмах последовательного типа сначала по определенному критерию выбирается вершина графа, затем к ней присоединяются другие вершины до получения первой части. Затем из оставшихся вершин графа формируется вторая часть и последующие до полного разбиения.

Итерационные алгоритмы в качестве исходного берут некоторое разрезание, полученное, например, с помощью одного из последовательных алгоритмов, а затем последовательно в связанных парах частей производится перестановка вершин из одной части в другую так, чтобы улучшалось значение критерия качества.

В смешанных алгоритмах сначала выделяется некоторое исходное множество групп вершин, в нашем случае это могут быть РВ-процессы, которые затем распределяются по частям по критерию (7.1) с учетом ограничений (7.2), (7.3).

Последовательные алгоритмы

Рассмотрим более подробно одну из схем последовательного алгоритма. Из исходного множества вершин с помощью последовательной процедуры формируется первое подмножество вершин V_1. Далее из оставшегося множества $V \setminus V_1$ аналогично формируется второе подмножество V_2 и так далее до полного разбиения множества V на части V_i, $i = 1, 2, ..., n$. При формировании очередного подмножества V_i существенны способы принятия решений по двум вопросам:

- выбор первой вершины, с которой начинается формирование подмножества;
- выбор очередной вершины, когда в формируемое подмножество уже включены несколько вершин.

Возможны различные процедуры выбора первой вершины и дальнейшего формирования подмножества V_i. При выборе таких процедур будем руководствоваться следующими соображениями:

- суммарный вес ребер, соединяющих вершины подмножества V_i должен быть максимален;
- суммарный вес ребер, соединяющих вершины подмножества V_i с остальными вершинами графа должен быть минимален;
- сумма размеров памяти и объемов вычислений вершин подмножества V_i должна соответствовать ограничениям (7.2), (7.3).

Руководствуясь данными соображениями, определим процедуру формирования подмножества V_i следующим образом:

1^{o}. Выбирается первая вершина $v_g \in V$, включается в подмножество V_i и исключается из множества V. Выбор вершины может быть выполнен по трем правилам:

- выбирается вершина $v_{g_1} \in V$ с максимальной оценкой $h_{g_1} = p_{g_1} / P_i + q_{g_1} / Q_i$, если таких вершин несколько, то из них выбирается вершина с максимальным суммарным весом инцидентных ей ребер;
- выбирается вершина с максимальным суммарным весом инцидентных ей ребер;
- выбирается ребро с максимальным суммарным весом, из вершин инцидентных данному ребру выбирается вершина с максимальной оценкой h_{g_1}.

2°. Выбирается вершина $v_{g_2} \in \left(V \setminus v_{g_1}\right)$, которая удовлетворяет ограничениям (7.2), (7.3), и при этом разность суммы весов ребер вершины v_{g_2} с ранее включенными в множество V_i и суммы весов ребер с вершинами $V \setminus V_i$ является наибольшей. Для вершины v_{g_2} это разность Δ:

$$\Delta = r_{g_1 g_2} - \sum_{v_t \in V^*} r_{g_2 t}\ ,\ V = V \setminus \left(v_{g_1}, v_{g_2}\right).$$

Выбранная вершина v_{g_2} включается в V_i и исключается из V.

3°. Аналогично выбираются другие вершины v_g для включения в множество V_i, которые удовлетворяют ограничениям (7.2), (7.3) и имеют максимальную $\Delta = \sum_{v_t \in V_i} r_{gt} - \sum_{v_t \in V \setminus V_i} r_{gt}$.

4°. Процесс формирования новых множеств V_i выполняется по п.п. $1° - 3°$ пока не будут распределены все вершины множества V.

Приведенная процедура последовательного алгоритма дает разрезание графа G на части G_i, каждая из которых размещается на станции C_i. Полученное при этом значение r критерия (7.1) может быть улучшено применением процедуры итерационного типа по перестановке вершин из одной части в другую. Процедура перестановки вершин производится для всех пар частей G_i и G_j, $i \neq j$.

Задача разрезания как задача покрытия

Для улучшения полученного таким образом решения предлагается методика, основанная на расширении области поиска решения за счет формирования расширенного множества вариантов разрезания. Для формирования вариантов воспользуемся алгоритмом декомпозиции графа на связные подграфы [50]. Алгоритм позволяет формировать полную совокупность всех связных подграфов G_i графа G с заданным числом вершин. Число вершин в подграфах определяется ограничениями (7.2), (7.3). Совокупность подграфов может быть расширена за счет вариантов, полученных изложенным выше алгоритмом разрезания последовательного типа. При этом в качестве первой выбираются поочередно все вершины $v_g \in F$. Сформированная совокупность несовпадающих друг с другом частей G_i объединяется с подграфами, полученными путем декомпозиции графа G на связные подграфы.

Общее множество подграфов обозначим через $\boldsymbol{H} = \{\boldsymbol{H}_j\}$, $\boldsymbol{j} = 1, 2, ..., \boldsymbol{n}$. Для каждого подграфа $\boldsymbol{H}_j \in \boldsymbol{H}$ определяется сумма $\boldsymbol{r}_j$ весов ребер, связывающих вершины $\boldsymbol{V}_j$ в подграфе $\boldsymbol{H}_j$. Тогда задачу поиска варианта разрезания можно сформировать как известную задачу покрытия [34, 35].

Введем переменную $\boldsymbol{x}_j = 1$, если подграф $\boldsymbol{H}_j$ входит в искомое разрезание, иначе $\boldsymbol{x}_j = 0$. Принадлежность вершин графа $\boldsymbol{G}$ подграфам $\boldsymbol{H}_j$ представим матрицей $\boldsymbol{A} = \|\boldsymbol{a}_{ij}\|$, $\boldsymbol{i} = 1, 2, ..., \boldsymbol{m}$, $\boldsymbol{j} = 1, 2, ..., \boldsymbol{n}$. Элемент $\boldsymbol{a}_{ij} = 1$, если вершина $\boldsymbol{v}_i \in \boldsymbol{V}$ входит в подграф $\boldsymbol{H}_j$, иначе $\boldsymbol{a}_{ij} = 0$. Задача заключается в выборе совокупности подграфов $\boldsymbol{H}_j$, покрывающих множество вершин $\boldsymbol{V}$ и доставляющих максимальную сумму оценок $\boldsymbol{r}_j$:

$$\mathbf{max}\ \boldsymbol{L} = \sum_{\boldsymbol{j}=1}^{\boldsymbol{n}} \boldsymbol{r}_j\, \boldsymbol{x}_j\,, \tag{7.5}$$

$$\text{при условиях} \quad \sum_{\boldsymbol{j}=1}^{\boldsymbol{n}} \boldsymbol{a}_{ij}\, \boldsymbol{x}_j = 1,\ \boldsymbol{i} = 1, 2, ..., \boldsymbol{m}. \tag{7.6}$$

Здесь $\boldsymbol{m}$ – число вершин в графе $\boldsymbol{G}$. Задача (46), (47) является совместной, так как множество подграфов $\boldsymbol{H}$ включает части $\boldsymbol{G}_i$, полученные в результате разрезания графа $\boldsymbol{G}$. В частности, подграфы, соответствующие частям $\boldsymbol{G}_i$, могут быть несвязными. Заметим также, что для задач большой размерности подграфы $\boldsymbol{H}_j$ с малыми значениями $\boldsymbol{r}_j$ можно исключить из рассмотрения. При этом могут быть некоторые потери в качестве получаемого решения. Задача будет оставаться совместной, если исключаемые подграфы будут относиться к связным подграфам, полученным путем декомпозиции графа $\boldsymbol{G}$ на части. Для решения задачи (7.5), (7.6) имеются приемлемые алгоритмы, например, [51].

Заметим также, что решение рассматриваемой задачи распределения данных и модулей по станциям МВС можно вести на уровне РВ-процессов. В этом случае вершины РВ-процесса стягиваются в одну вершину. В РВ-процессах, рис. 7.8, имеющих параллельные ветви, например, модули 22 и 21 в процессе $\boldsymbol{p}_8$ и модули 24 и 23 в процессе $\boldsymbol{p}_{10}$, ветви стягиваются в отдельные вершины. Таким образом, в результате стягивания вершин ГПД, представленного на рис. 7.8, получается граф, изображенный на рис. 7.6, содержащий 15 вершин. Значения па-

раметров стягиваемых вершин по размерам памяти и объемам вычислений суммируются. Веса дуг между стягиваемыми совокупностями вершин также суммируются. Для полученного графа полностью применим изложенный выше подход к решению задачи (7.1) – (7.3).

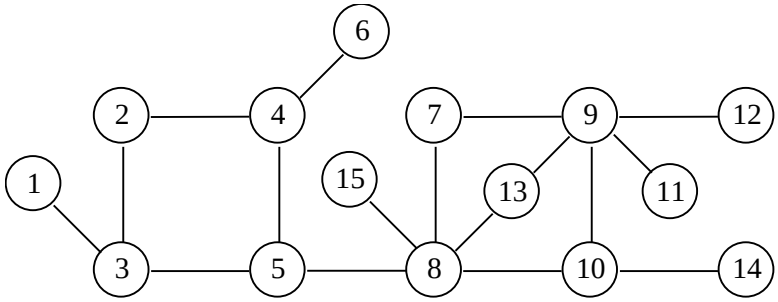


Рис. 7.6. Граф передачи данных между РВ-процессами

Определение рациональных путей передачи данных

Решение задачи распределения данных и модулей выполнялось в предположении, что каналы передачи данных между станциями МВС эквивалентны и, соответственно, время, затрачиваемое на передачу данных между станциями, также одинаково. На практике это положение не всегда выполняется, особенно для МВС со сложной архитектурой, когда между двумя станциями может существовать несколько маршрутов передачи данных. Поэтому после решения задачи распределения можно оценить полученный вариант размещения по суммарному времени передачи данных в сети МВС.

Для получения такой оценки необходимо найти кратчайшие по времени пути между всеми парами станций. Представим архитектуру МВС графом $\boldsymbol{A}=(\boldsymbol{S},\boldsymbol{B})$, где $\boldsymbol{S}$ – множество вершин графа $\boldsymbol{A}$, соответствующих станциям, $\boldsymbol{S}=\{\boldsymbol{s_i}\}$, $\boldsymbol{i}=1,2,...,\boldsymbol{n}$; $\boldsymbol{B}$ – множество ребер графа $\boldsymbol{A}$, соответствующих связям между станциями.

Каждому ребру $(\boldsymbol{s_i},\boldsymbol{s_j})\in\boldsymbol{B}$ поставлены в соответствие веса $\boldsymbol{t_{ij}}$ равные сумме времен передачи одного байта для адаптеров соответствующих станций и шины, к которой эти станции подключены. Задача нахождения путей с наименьшим временем передачи данных соответствует известной задаче теории графов – задаче нахождения кратчайших путей между всеми парами станций. Для решения данной задачи может быть использован алгоритм Флойда [45].

Введем матрицу $\boldsymbol{C} = \left\| \boldsymbol{c_{ij}} \right\|_{n \times n}$ весов ребер графа $\boldsymbol{A}$. Элементы $\boldsymbol{c_{ij}}$ определяются следующим образом:

$$\boldsymbol{c_{ij}} = \begin{cases} 0, & \textit{если } \boldsymbol{i = j}; \\ \infty, & \textit{если нет непосредственной связи между станциями } \boldsymbol{s_i} \textit{ и } \boldsymbol{s_j}; \\ \boldsymbol{t_{ij}}, & \textit{если станции } \boldsymbol{s_i} \textit{ и } \boldsymbol{s_j} \textit{ связаны ребром с весом } \boldsymbol{t_{ij}}. \end{cases}$$

В принятых обозначениях и с учетом того, что $\boldsymbol{t_{ij}} \geq 0$ шаги алгоритма Флойда запишутся в следующем виде:

1^{o}. Положить $\boldsymbol{k} = 0$.

2^{o}. $\boldsymbol{k} = \boldsymbol{k} + 1$.

3^{o}. Для всех $\boldsymbol{i} \neq \boldsymbol{k}$ таких, что $\boldsymbol{c_{ik}} \neq \infty$ и для всех $\boldsymbol{j} \neq \boldsymbol{k}$ таких, что $\boldsymbol{c_{kj}} \neq \infty$ выполнить операцию $\boldsymbol{c_{ij}} = \mathbf{min}\left[\boldsymbol{c_{ij}}, \boldsymbol{c_{ik}} + \boldsymbol{c_{kj}} \right]$.

4^{o}. Если $\boldsymbol{k} = \boldsymbol{n}$, то конец алгоритма. Элементы $\boldsymbol{c_{ij}}$ матрицы $\boldsymbol{C}$ соответствуют кратчайшим путям между вершинами $\boldsymbol{s_i}$ и $\boldsymbol{s_j}$. Если $\boldsymbol{k} < \boldsymbol{n}$, то вернуться на шаг 2^{o}.

Полученная матрица кратчайших путей используется для оценки времени $\boldsymbol{T}$ на обмен данными между станциями:

$$\boldsymbol{T} = \sum_{\boldsymbol{i}=1}^{\boldsymbol{n}} \sum_{\boldsymbol{j}=1}^{\boldsymbol{n}} \boldsymbol{c_{ij}}\, \boldsymbol{m_{ij}},$$

где $\boldsymbol{m_{ij}}$ – объем данных, передаваемых между станциями $\boldsymbol{s_i}$ и $\boldsymbol{s_j}$.

Величина $\boldsymbol{T}$ отражает суммарное время, затрачиваемое на передачу данных в сети МВС за время расчетного цикла согласно полученному варианту распределения данных и модулей по станциям МВС.

7.6. Выполнение активной модели на виртуальной машине

Моделирование работы активной модели программной нагрузки, представленной в виде ГПД, осуществляется на модели МВС с учетом плана размещения алгоритмов по процессорам МВС, а данных по их запоминающим устройствам. Причем алгоритмы могут быть представлены как в виде имитаторов, так и в виде реальных алгоритмов СРВ. Независимо от способа представления каждый алгоритм транслируется в загрузочный модуль, который состоит из двух функциональных частей, оформленных в виде отдельных потоков. Основной поток представляет алгоритм работы перехода. Второй поток создается и запуска-

ется автоматически при запуске перехода на выполнение. Ниже перечислены основные функции вторичного потока:

- управление первичным потоком, приостановка и продолжение его работы в случае необходимости, вызванной процессом управления моделированием;

- подсчет времени затраченного процессором инструментальной ЭВМ на выполнение первичного потока, причем время определяется в составе двух величин – времени выполнения функций Windows и времени выполнения прикладного кода основного потока;

- синхронизация реального времени выполнения алгоритма, представленного первичным потоком, и модельного времени;

- предоставление доступа к данным для чтения и записи в соответствии с ГПД.

Для выполнения описанных выше требований предлагается создать ПВМ, которая должна обладать следующими возможностями:

- запускать и выполнять алгоритмы программной нагрузки как совокупность параллельных процессов;

- синхронизировать работу процессов;

- обеспечивать корректную передачу данных между процессами;

- выполнять квантование времени между процессами, вести контроль за ресурсами МВС в соответствии с планом распределения;

- имитировать работу технических устройств МВС.

Исходя из перечисленных возможностей можно сказать, что виртуальная машина выступает в роли операционной системы для динамической модели проектируемой СРВ. Для обеспечения функциональных возможностей такой операционной системы каждый алгоритм (переход) сопровождается специальным паспортом, описывающим его специфику. Паспорт модуля имеет следующую структуру: ***имя модуля, список входов, список выходов, время выполнения, объем вычислений, требуемая память, используемые ресурсы, станция приписки.*** Для каждого входа: ***имя входа, условия поступления, условия селекции, приоритет, требуемая память, условия хранения.*** Для каждого выхода: ***имя выхода, условия обновления, требуемая память, условия хранения.***

ПВМ базируется на средствах операционной системы Windows NT по управлению процессами и их синхронизации. Возможности управления и синхронизации существенно расширены за счет введения

дополнительных функций, что позволяет реализовать перечисленные возможности. Алгоритмы (переходы) представляются в данном случае в виде отдельных загрузочных модулей.

Моделирование работы параллельных процессов на основе ПВМ включает выполнение двух основных этапов – подготовка к моделированию, в процессе которого выполняется настройка системы, и непосредственно моделирование, в ходе которого осуществляется выполнение модели и строится временная диаграмма работы системы.

Подготовка к моделированию

Данная функция запускается непосредственно перед началом моделирования и автоматически выполняет следующую последовательность действий, направленных на обеспечение процесса моделирования:

- проверка наличия всех программ СРВ, реализующих алгоритм, описанный с помощью ГПД;
- проверка правильности распределения функций, программ и данных по ресурсам МВС;
- распределение памяти для всех данных, представленных в ГПД в виде позиций, инициализация входных данных каждого процесса начальными значениями, представленных в ГПД в виде позиций, имеющих только выходные дуги, и позиций, у которых входные и выходные дуги соединены с переходами, принадлежащими различным процессам и заданы условия их формирования;
- инициализация внутренних объектов, предназначенных для корректного восстановления ПВМ в случае возникновения критической ошибки в процессе моделирования;
- инициализация внутренних объектов, предназначенных для управления процессом моделирования, хранения и отображения временной диаграммы;
- построение списка процессов по графическому представлению ГПД;
- проверка связности процессов исходя из правила – все переходы процесса должны быть связаны через соответствующие позиции ГПД;
- задание параметров моделирования, включающее выбор общего времени моделирования и кванта времени, определяющего дискрет-

ность при проведении моделирования и построения временной диаграммы;

- определение типа и тактовой частоты процессора инструментальной ЭВМ для пересчета времени работы программ, выполняющихся на станциях, имеющих отличные характеристики.

Моделирование работы СРВ

Процесс моделирования работы СРВ включает выполнение следующей последовательности функций.

1. Строится список процессов, которые необходимо запустить в текущий момент времени. Необходимость запуска каждого процесса в отдельности определяется из условия готовности данных. Готовность данных в свою очередь определяется в зависимости от условия их поступления:

- для циклических данных готовность определяется в зависимости от времени цикличности и модельного времени;

- для данных, имеющих детерминированную последовательность моментов времени поступления, готовность определяется в соответствии с указанными моментами;

- для данных, имеющих случайные моменты поступления, готовность определяется исходя из результата вычисления функции, описывающей закон распределения вероятности;

- для данных, имеющих условные моменты поступления, готовность определяется результатом проверки выполнения соответствующего условия.

2. Производится последовательный запуск всех модулей процесса в соответствии с ГПД. Причем перед запуском каждого модуля проверяется, работает ли модуль в текущий момент времени. В случае положительного результата проверки выдается сообщение об ошибке, означающее, что процесс, включающий модуль, не справляется с обработкой входной информации за необходимое время.

3. Проверяется, работает ли в данный момент времени какой - либо из модулей. В случае положительного результата проверки управление передается на следующий шаг, а в противном случае считается, что прошел квант времени и управление передается на шаг 1.

4. Система переводится в режим управления. В этом режиме осуществляется обработка информационных сообщений, поступающих от

работающих модулей. Под обработкой понимается обеспечение функций чтения/записи данных модулями, а также временная синхронизация их работы. Синхронизация обеспечивается благодаря возможности приостановить работу модуля в момент отправки сообщения. Принято также, что каждое сообщение, отправленное модулем, должно быть подтверждено специальным служебным сообщением со стороны ПВМ, до получения которого работа модуля приостанавливается. При получении виртуальной машиной информации о работе в течение принятого кванта времени от каждого работающего модуля осуществляется переход к следующему шагу.

5. На временной диаграмме отображается работа модуля в течение кванта времени в соответствии с полученными временными характеристиками.

6. Осуществляется проверка на окончание процесса моделирования в соответствии с заданным временем моделирования. В случае положительного результата проверки происходит выход из режима моделирования, а в противном случае управление передается на шаг 1.

В процессе моделирования строится временная диаграмма работы системы, по которой можно определить основные временные характеристики и характеристики загрузки устройств МВС. Оси временной диаграммы соответствуют активным компонентам модели архитектуры МВС. На осях откладываются временные отрезки работы соответствующих устройств МВС при выполнении команд виртуальной машины.

Временная диаграмма по желанию пользователя может включать временные оси лишь для отдельных устройств. При этом число осей для отображения может изменяться в большую или меньшую сторону, как перед началом моделирования, так и после его завершения. После моделирования временную диаграмму можно просматривать, изменяя масштабирование более детально и по отдельным, интересующим пользователя, устройствам.

Определение характеристик выполняется по временной диаграмме после окончания моделирования. Основными характеристиками являются следующие:

- время выполнения прикладной функции в целом или ее фрагментов на процессорах различных платформ;
- коэффициенты загрузки устройств МВС при выполнении алгоритмов;
- частоты выполнения отдельных фрагментов алгоритмов.

Последняя характеристика имеет важное значение при детальном анализе работы алгоритма с точки зрения поиска возможных вариантов сокращения времени выполнения алгоритма. При высокой частоте даже относительно небольшое сокращение времени выполнения соответствующего фрагмента может привести к желаемому сокращению общего времени завершения работы данного алгоритма и прикладной функции в целом.

Коэффициенты загрузки устройств определяются как отношение суммарного времени работы устройства к общему моделируемому времени. Значения данных коэффициентов существенно зависят от варианта размещения функций, программ и данных по ресурсам МВС. Поэтому характеристики по загрузке устройств и частотам выполнения важно знать для принятия решений по эволюции моделей.

Безусловно, при моделировании любой системы важным является вопрос адекватности моделей и результатов моделирования. По этому поводу можно сказать следующее. Конечно, система Windows не является операционной системой реального времени и не имеет развитых функций, оценивающих временные характеристики выполняющихся процессов. Тем не менее, Windows NT содержит ряд функций, позволяющих решить задачу оценки времени и создать алгоритм с определенным уровнем погрешности измерения времени выполнения процессов.

Исследования показали, что погрешность, вносимая алгоритмом измерения, становится исчезающей при работе процесса порядка нескольких секунд. Вследствие этого основную погрешность измерения образует неточность коэффициентов, описывающих тип и тактовую частоту процессора, время доступа к памяти и т.д. Однако следует отметить, что данная погрешность является статической и, следовательно, относительная оценка времени выполнения различных вариантов моделирования является в достаточной степени достоверной, что позволяет судить о качестве изменения СРВ при последующих итерациях моделирования.

Данный подход к моделированию позволяет при удовлетворительных результатах моделирования получить практически готовую систему в виде совокупности алгоритмов, представленных отдельными исполняемыми файлами и алгоритмов управления их работой.

7.7. Пример модели программной нагрузки

В качестве примера для построения и анализа модели программной нагрузки взят алгоритм управления системой связи.

Система связи (СС) с встроенной распределенной системой управления состоит из совокупности узлов связи и совокупности пунктов ретрансляции. Узел связи организует связь по нескольким направлениям, каждое из которых имеет ресурсы по отдельным видам каналов связи: проводные, тропосферные, радиорелейные, радиоканалы, космические.

Исследования проводились в двух аспектах:

- создание активной модели для имитации работы СС и построения временной диаграммы загрузки каналов СС для заданного потока заявок;
- анализ текущего состояния системы связи и принятие решений по управлению в реальном времени.

В первом случае на период эксперимента модель СС остается неизменной и отражает существующий вариант СС, модернизируемый либо созданный заново. Основными функциями активной модели СС является имитация заданного потока заявок различного типа по направлениям узлов связи, определения маршрутов заявок и построения временной диаграммы загрузки каналов и направлений. Фрагмент временной диаграммы показан на рис. 7.7. Имитация функционирования СС может осуществляться как в модельном времени, так и в реальном масштабе времени и нацелена в основном на анализ конкретного варианта СС и его функционирования в различных условиях.

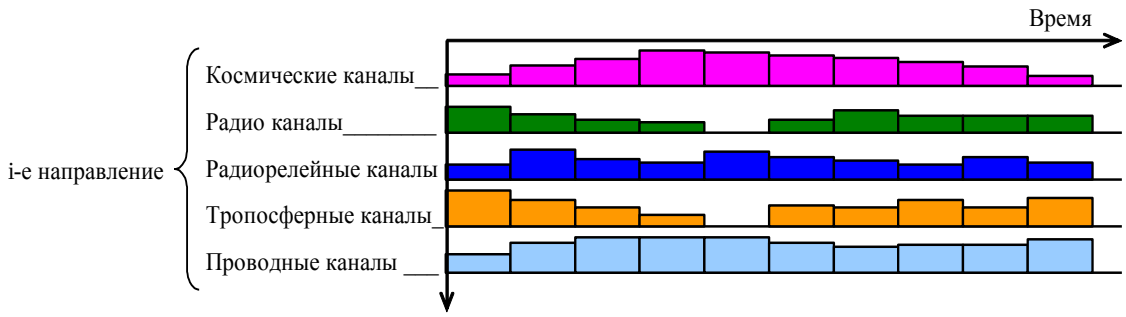


Рис. 7.7. Фрагмент временной диаграммы загрузки каналов связи на ***i*** *– м направлении*

Второй случай рассматривает ситуацию, когда СС меняется в реальном масштабе времени. Изменения происходят случайно и затрагивают параметры структуры СС и ее ресурсов, а также топологию узлов связи и их конфигурацию. В СС могут оперативно вводиться дополнительные ресурсы.

Функции управления в этом случае существенно усложняются и требуют компьютерной поддержки. Ниже приведены предварительные исследования программной нагрузки такой системы и приемлемого варианта архитектуры вычислительной системы.

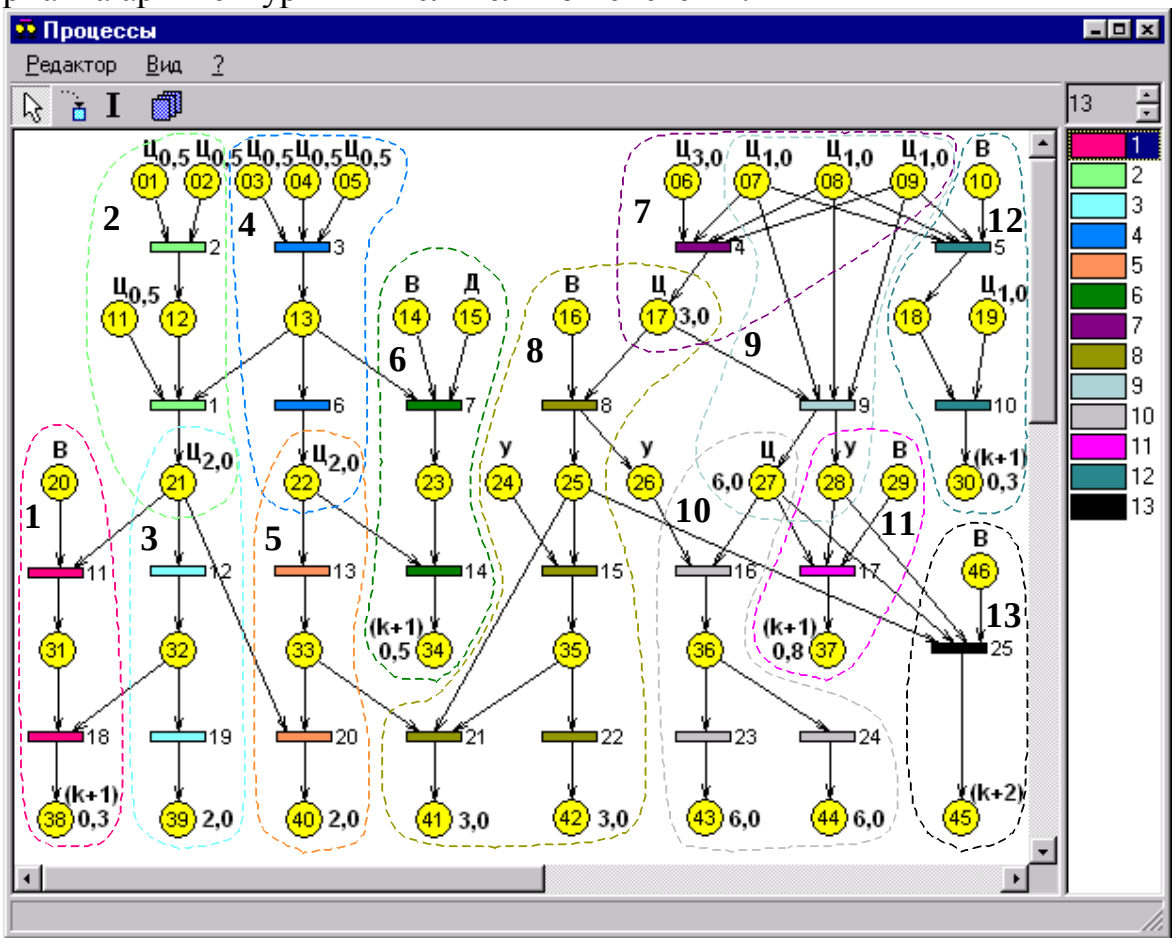


Рис. 7.8. Функциональный ГПД

Модель программной нагрузки представлена в форме ГПД (рис. 7.8) и реализует следующие прикладные функции:

1. (Процесс 1) – регистрация и определение маршрута для спецзаявки типа 1 или 2.
2. (Процессы 2, 3) – регистрация заявок типа 1 и 2 и определение маршрутов для них.
3. (Процессы 4, 5) – регистрация заявок типа 3 и определение для них маршрутов.
4. (Процесс 6) – регистрация и определение маршрута для спецзаявки типа 3.
5. (Процессы 7, 8) – регистрация состояния СС и внесение изменений в модель. Изменение состояния фиксируется от датчиков со-

стояния ресурсов СС и внешней среды, а также от оператора. Параметры внешней среды формируются с учетом сведений, получаемых от геоинформационной системы.

6. (Процессы 9, 10, 11) – определение рациональной конфигурации СС исходя из анализа потока заявок и загрузки каналов, а также параметров внешней среды. Предусматривается также ручная корректировка конфигурации СС и привлечение дополнительных ресурсов.
7. (Процесс 12) – обработка заявки оператора по оценке состояния СС для конкретных направлений.
8. (Процессы 13) – формирование общей сводки о состоянии СС по требованию оператора.

Совокупность информационных входов $\boldsymbol{X}^0$ включает следующие данные:

- $\boldsymbol{X}_1, \boldsymbol{X}_2$ – заявки 2-го типа;
- $\boldsymbol{X}_3, \boldsymbol{X}_4, \boldsymbol{X}_5$ – заявки 3-го типа;
- $\boldsymbol{X}_6$ – данные о топологии узлов связи;
- $\boldsymbol{X}_7, \boldsymbol{X}_8, \boldsymbol{X}_9$ – параметры текущего состояния СС;
- $\boldsymbol{X}_{10}$ – запрос о состоянии СС;
- $\boldsymbol{X}_{11}$ – заявки 1-го типа;
- $\boldsymbol{X}_{14}, \boldsymbol{X}_{15}$ – запрос на определение маршрута для приоритетной заявки 3-го типа;
- $\boldsymbol{X}_{16}$ – данные с датчиков об изменении состояния модели СС;
- $\boldsymbol{X}_{19}$ – данные о готовности к работе активных ресурсов СС;
- $\boldsymbol{X}_{20}$ – запрос на определение маршрута для приоритетной заявки 1-го типа;
- $\boldsymbol{X}_{24}$ – корректировка изменений в состоянии модели СС по предложениям оператора;
- $\boldsymbol{X}_{29}$ – запрос оператора на внесение изменений в конфигурацию СС;
- $\boldsymbol{X}_{46}$ – запрос оператора на составление общей сводки по состоянию СС.

Совокупность информационных выходов $\boldsymbol{Y}^0$ включает следующие данные:

- $\boldsymbol{Y}_{30}$ – данные о состоянии СС;
- $\boldsymbol{Y}_{34}$ – управление по организации маршрута для приоритетной заявки 3-го типа;

- Y_{37} – информация об изменении конфигурации модели СС;
- Y_{38} – управление по организации маршрута для приоритетной заявки 1-го или 2-го типа;
- Y_{39} – управление по организации маршрутов для заявок 1-го и 2-го типов;
- Y_{40} – управление по организации маршрутов для заявок 3-го типа;
- Y_{41}, Y_{42} – информация по изменению текущего состояния модели СС;
- Y_{43}, Y_{44} – информация по изменению рациональной конфигурации модели СС;
- Y_{45} – общая сводка по состоянию СС.

Рассмотрим более подробно условия запуска приведенных РВ-процессов. Первый процесс запускается от входа d_{20} типа B и формирует выход d_{38} по типу T – обновление с ограничением на время обновления равным 0,3 единицы времени. Процесс $p_2 \in P$ имеет три циклических входа d_1, d_2, d_{11} и выход d_{21} по типу циклическое присоединение четырех состояний, $K = 4$. Процесс p_3 запускается входом d_{21}, который одновременно является выходом процесса p_2. Формирование состояний d_{21} процессом p_2 происходит в цикле с $\Delta t = 0{,}5$. Вместе с тем, запуск процесса p_3 производится в цикле с $\Delta t = 2$, определяемом циклом обновления выхода d_{39}, формируемого процессом p_3. Поэтому для выхода d_{21} определено обновление по условию циклического присоединения четырех состояний. Процесс p_4 по условиям запуска и обновления данных полностью совпадает с процессом p_2, а процесс p_5 с процессом p_3.

Процесс p_6 запускается двумя входами: d_{14} с условием поступления B и d_{15} с условием поступления $Д$. Процессы с входом $Д$ всегда запускаются входами B или $У$, а дальше они запускаются согласно детерминированной последовательности моментов времени, задаваемой по входу $Д$. Выходом процесса p_6 является d_{34} по типу T – обновление с ограничением на время равное 0,5 единиц.

Процесс p_7 имеет входы d_7, d_8, d_9 с циклическим поступлением, $\Delta T = 1$ и вход d_6 с циклическим поступлением, но с другим временем цикла $\Delta T = 3$. Запуск процесса осуществляется циклически с $\Delta T = 1$.

Вход $\boldsymbol{d}_6$ в данном случае используется модулем $\boldsymbol{f}_4$ три раза за один цикл. Выходом $\boldsymbol{p}_7$ является $\boldsymbol{d}_{17}$, формируемый циклическим присоединением с $\boldsymbol{K}=3$.

Процесс $\boldsymbol{p}_8$ запускается циклическим входом $\boldsymbol{d}_{17}$ и двумя входами, формирующими сигналы прерывания по типу $\boldsymbol{B}$ (вход $\boldsymbol{d}_{16}$) и типу $\boldsymbol{У}$ (вход $\boldsymbol{d}_{24}$). При этом приоритет входа $\boldsymbol{d}_{16}$ выше приоритета входа $\boldsymbol{d}_{17}$, $\pi(\boldsymbol{d}_{16})>\pi(\boldsymbol{d}_{17})$, а $\pi(\boldsymbol{d}_{24})>\pi(\boldsymbol{d}_{16})$. Выходы $\boldsymbol{d}_{41}$, $\boldsymbol{d}_{42}$ обновляются циклически с $\Delta\boldsymbol{T}=3$, а при запуске от $\boldsymbol{d}_{16}$ и $\boldsymbol{d}_{24}$ по типу $\boldsymbol{T}$ – обновление с $\boldsymbol{T}\le 0{,}2$. Заметим, что вход $\boldsymbol{d}_{24}$ формируется в данном случае управляющей программой. Напротив, в процессе $\boldsymbol{p}_9$ выход $\boldsymbol{d}_{28}$ формируется модулем $\boldsymbol{f}_9$ как сигнал программного прерывания для процесса $\boldsymbol{p}_{11}$. Выход $\boldsymbol{d}_{27}$ формируется как циклическое присоединение с $\boldsymbol{K}=6$.

Процесс $\boldsymbol{p}_{10}$ запускается циклически входом $\boldsymbol{d}_{27}$ и по программному прерыванию входом $\boldsymbol{d}_{26}$, $\pi(\boldsymbol{d}_{26})>\pi(\boldsymbol{d}_{27})$. Выходы $\boldsymbol{d}_{43}$ и $\boldsymbol{d}_{44}$ циклически обновляются с $\Delta\boldsymbol{T}=6$, а при запуске от $\boldsymbol{d}_{26}$ (тип $\boldsymbol{У}$) выходы обновляются по типу $\boldsymbol{T}$ – обновление с временем $\boldsymbol{T}\le 0{,}8$. Процесс $\boldsymbol{p}_{11}$ запускается сигналами прерывания $\boldsymbol{d}_{28}$ (тип $\boldsymbol{У}$) и $\boldsymbol{d}_{29}$ (тип $\boldsymbol{B}$), при этом принято, что $\pi(\boldsymbol{d}_{29})>\pi(\boldsymbol{d}_{28})$.

Запуск процесса $\boldsymbol{p}_{12}$ производится входом $\boldsymbol{d}_{10}$ (тип $\boldsymbol{B}$). Три других входа $\boldsymbol{d}_8$, $\boldsymbol{d}_9$, $\boldsymbol{d}_{19}$ с циклическим поступлением $(\Delta\boldsymbol{T}=1)$ не запускают процесс $\boldsymbol{p}_{12}$, так как выход $\boldsymbol{d}_{30}$ обновляется по типу $\boldsymbol{T}$ – обновление с $\boldsymbol{T}\le 0{,}3$. По данному процессу видно, что не всегда вход, для которого определено условие поступления, участвует в запуске процесса. Это зависит также от условия обновления выходов рассматриваемого процесса. Если имеет место $\boldsymbol{T}$ – обновление по соответствующему выходу, как в примере для выхода $\boldsymbol{d}_{30}$, $\boldsymbol{T}\le 0{,}3$, то запуск процесса производится только по сигналу прерывания. Для сравнения, выходы $\boldsymbol{d}_{43}$, $\boldsymbol{d}_{44}$ процесса $\boldsymbol{p}_{10}$ обновляются циклически с $\Delta\boldsymbol{T}=6$,и наряду с этим имеет место $\boldsymbol{T}$ – обновление с $\boldsymbol{T}\le 0{,}8$. Поэтому процесс $\boldsymbol{p}_{10}$ запускается циклически по входу $\boldsymbol{d}_{27}$ и по сигналу прерывания $\boldsymbol{d}_{26}$. В этом случае в позиции 17 паспорта модуля $\boldsymbol{f}_{23}$, $\boldsymbol{f}_{24}$ для выходов $\boldsymbol{d}_{43}$, $\boldsymbol{d}_{44}$ указывается два условия обновления: циклическое обновление с $\Delta\boldsymbol{T}=6$ и $\boldsymbol{T}$ – об-

новление с $T \le 0{,}8$. При этом предполагается, что модули f_{16}, f_{23}, f_{24} в сравнении с циклическим запуском работают по упрощенному алгоритму, что обеспечивает получение выхода за время $T \le 0{,}8$.

Запуск процесса p_{13} осуществляется двумя входами A (тип Y) и d_{46} (тип a_{is}), при этом $\pi(d_{28}) > \pi(d_{46})$. Условий на обновление выхода $s = 1, 2, ..., k$ не накладывается, поэтому запуск процесса p_{13} по циклическому входу d_{27} не производится.

Выполнение активной модели программной нагрузки осуществлялось на МВС, состоящей из 4− х станций. Моделирование выполнялось на одном расчетном цикле со следующим распределением программной нагрузки по станциям: станция 1 (процессы 1, 2, 3), станция 2 (процессы 4, 5, 6), станция 3 (процессы 7, 8, 12), станция 4 (процессы 9, 10, 11, 13). Временная диаграмма представлена на рис. 7.9.

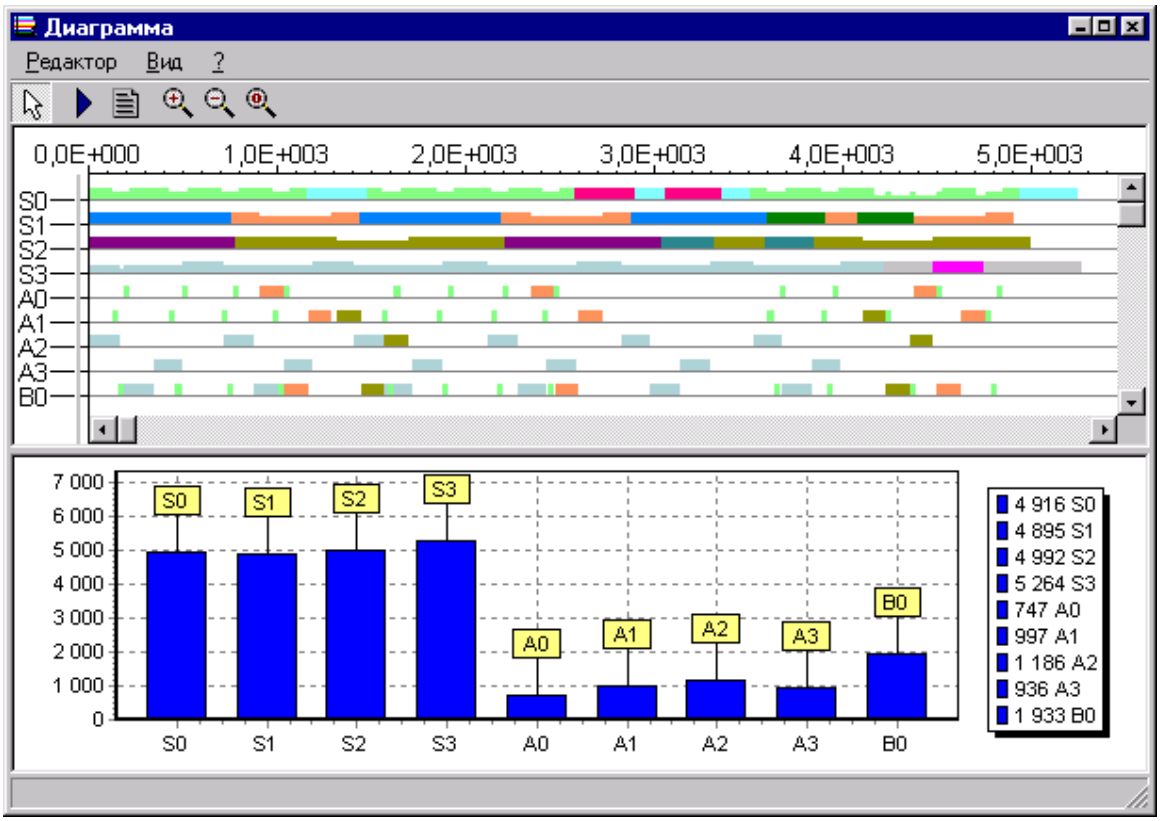


Рис. 7.9. Временная диаграмма управления работой системы связи

Моделирование осуществлялось также для двухпроцессорной и трехпроцессорной архитектуры. При этом в первом случае процессы 1 – 6 выполнялись первым процессором, а остальные вторым. Данная архи-

тектура оказалась неприемлемой. Вариант с тремя процессорами удовлетворял условиям реального времени. При этом возрастали задержки на передачу данных и не оставалось резерва времени для выполнения дополнительных функций, которые появятся при уточнении модели программной нагрузки. Кроме того, анализ программной нагрузки с помощью новой версии программных средств, которая более точно реализует канальные функции, также может привести к росту задержек времени на синхронизацию процессов. Поэтому в качестве варианта для комплексного моделирования и экспериментальной проверки корректности соблюдения условий реального времени принята МВС на базе 4-х станций.

Вопросы для контроля усвоения знаний

1. Что обусловило необходимость введения понятия РВ-процессов?
2. Назвать недостатки известных моделей взаимодействия процессов.
3. Дать формальное определение РВ-процесса.
4. Перечислить условия поступления входных данных.
5. Определить варианты обновления выходных данных.
6. Дать определение видов приоритетов.
7. Раскрыть содержание взаимодействия РВ-процессов.
8. Почему возникает потребность во взаимном контроле числа запусков взаимодействующих процессов?
9. Пояснить смысл взаимодействия процессов по условиям запуска.
10. Перечислить варианты взаимодействия процессов по условиям запуска.
11. Перечислить и дать определения взаимодействий по условиям передачи данных.
12. Раскрыть суть вариантов соответствия одноканальных взаимодействий процессов.
13. Дать определение канальной функции.
14. Пояснить схему отношений основных параметров для определения канальных функций.
15. Изложить алгоритм построения канальной функции.
16. Дать определение ГПД-программы.
17. Раскрыть состав команды виртуальной машины (паспорта модуля).
18. Пояснить основные функции операционной системы виртуальной машины.

19. Дать постановку задачи планирования использования ресурсов МВС.
20. Привести формализованную постановку задачи планирования использования ресурсов как задачи разрезания графов.
21. Дать характеристику известных алгоритмов решения задачи разрезания графов.
22. Изложить методику постановки задачи разрезания графов через задачу покрытия.
23. Изложить методику нахождения рациональных путей передачи данных в МВС.
24. Перечислить и пояснить функции вторичного потока.
25. Перечислить и пояснить операции подготовки к моделированию.
26. Изложить методику организации моделирования.
27. Пояснить основные результаты моделирования.

Заключение

Учебное пособие по содержанию соответствует лекционному курсу по дисциплине «Автоматизированное проектирование распределенных СРВ». Опыт чтения лекций по материалу данного пособия показывает, что несмотря на значительную методическую проработку, усвоение дисциплины вызывает значительные трудности. Эти трудности обусловлены рядом причин.

Первая из них заключается в том, что излагаемые в пособии методы широко используют аппарат математического программирования и теории графов. Этими разделами математики студенты владеют слабо. Поэтому по ходу изложения материала приходится по возможности восполнять этот недостаток знаний.

Вторая значимая причина связана с тем, что основное внимание в пособии уделено методам разработки и анализа модели, которая является исходной в эволюционном цикле проектирования. Процесс эволюции модели, то есть непосредственно процесс проектирования в пособии представлен недостаточно. Ясно, что принятие решений по трансформации модели трудно формализуемо и отнесено к компетенции разработчика системы. Вместе с тем, предложенная форма представления модели в виде модульной структуры, позволяет вносить изменения в модель на основе принятых решений с помощью формальных процедур трансформации. В пособии представлены лишь некоторые фрагменты такого инструментария.

Методы разработки и анализа модели не всегда сопровождаются примерами, что несомненно затрудняет восприятие материала учебного пособия. Расширение числа учебных примеров, которые можно разместить в методических указаниях к выполнению лабораторных и курсовых работ, будет способствовать более глубокому усвоению теоретического материала дисциплины.

Особое место в учебном пособии занимает введенное автором понятие активной модели. В общем случае ***активную модель можно определить как имитационную модель динамической системы представленной модульной структурой, каждый модуль которой оформлен в виде команды виртуальной машины***. В пособии модульные структуры представлены функциональным ГПД, а виртуальная машина реализована программно. В качестве модулей могут быть имитаторы фрагментов алгоритмов программной нагрузки СРВ или имитаторы алгоритмов функционирования любых активных компонентов системы. Число видов модулей не ограничено, так как любой из них мо-

жет быть преобразован в команду виртуальной машины. Моделирование СРВ в этом случае сводится к выполнению модульной структуры на виртуальной машине.

Следует также отметить, что к настоящему времени методы моделирования на основе активных моделей, так же, как и методы проектирования СРВ, находятся в стадии развития. Надеюсь, что предлагаемое учебное пособие привлечет внимание студентов, аспирантов и других заинтересованных специалистов к развитию теории активных моделей и разработки на этой основе более эффективных методов автоматизированного проектирования СРВ.

Библиографический список

1. Каган Б.М. Электронные вычислительные машины и системы. – М.: Энергоатомиздат, 1985. –552 с.
2. Ларионов А.М., Майоров С.А., Новиков Г.И. Вычислительные комплексы, системы и сети. – Л.: Энергоатомиздат, 1987. –288 с.
3. Клейнрок Л. Вычислительные системы с очередями. – М.: Мир, 1979. –600 с.
4. Шрайбер Т.Дж. Моделирование на GPSS: Пер. с англ. – М.: Машиностроение, 1980. –592 с.
5. Альянах И.Н. Моделирование вычислительных систем. – Л.: Машиностроение, 1988. –223 с.
6. Питерсон Дж. Теория сетей Петри и моделирование систем: Пер. с англ. – Мир, 1984. –264 с.
7. Котов В.Е. Сети Петри. – М.: Наука, 1984. –158 с.
8. Лобков С.Н., Фатхи В.А., Климович Г.И., Дуднакова О.В. Стохастическо-детерминированные временные сети Петри как средство описания моделей многопроцессорных вычислительных систем. – УсиМ, №8, 1991. С.60 – 68.
9. Костин А.Е., Савченко Л.В. Модифицированные Е-сети для исследования систем распределенной обработки информации. – Автоматика и вычислительная техника. №6, 1988. С.27 – 35.
10. Баев В.В., Пипетко С.В. Пакет программ моделирования дискретных процессов расширенными сетями Петри.– УсиМ, №8, 1991. С.83– 87.
11. Слепцов А.И., Юрасов А.А. Автоматизация проектирования управляющих систем гибких автоматизированных производств. – К.: Техника, 1986. – 110 с.
12. Робототехника и гибкие автоматизированные производства. В 9-ти кн. Кн. 5. Моделирование робототехнических систем и гибких автоматизированных производств: учебное пособие для втузов. Пантюшин С.В., Назаретов В.М., Тягунов О.А. и др./ под ред. Макарова И.М. – М.: Высшая школа, 1986. – 175 с.
13. Шенброт И.М., Алиев В.М., Проектирование вычислительных систем распределенных АСУ ТП. – М.: Энергоатомиздат, 1989. – 88 с.
14. Мытус Л. Модель программного обеспечения распределенных вычислительных систем управления. – Программирование, №3, 1983. С.46 – 54.
15. Мытус Л., Кяэрамеэс К. Один метод создания семейства специализированных операционных систем реального времени. – УсиМ, 3№, 1983. С. 47 – 52.

16. Мытус Л., Чугунов В.С., Артемьева Н.И. и др. Выбор формального метода спецификации ПО систем управления дискретно-непрерывными производствами. – УСиМ, №2, 1985. С. 11 – 15.
17. Мытус Л. Особенности моделирования программного обеспечения многопроцессорных вычислительных систем. – Изд. АН СССР. Техническая кибернетика, №4, 1985. С. 149 – 156.
18. Погребной В.К. Об одном способе представления алгоритмов в виде графовых моделей. УСиМ, №1, 1983. С.63 – 69.
19. Погребной В.К. Об автоматизации распределения памяти на графовых моделях алгоритмов. УСиМ, №5, 1980. С. 37 – 42.
20. Ершов А.П. Введение в теоретическое программирование. М.: Наука, 1977. – 288 с.
21. Оре О. Теория графов. – М.: Наука, 1980. – 336 с.
22. Погребной В.К. Об автоматизации модульного проектирования программного обеспечения АСУ ТП. – УСиМ, №1, 1978. С.25 – 34.
23. Погребной В.К., Погребной Д.В. Методы анализа алгоритмов, функционирующих в системах реального времени. – Кибернетика и вуз, вып. 28, Томск, ТПИ, 1994. С.98 – 106.
24. Погребной В.К., Ямпольский С.З. Проектирование унифицированного набора блоков для заданного класса функциональных схем. – Известия ТПИ, т.269, Изд-во ТГУ, Томск, 1976. –С.91 – 94.
25. Сонькин М.А. Информационные технологии в задачах построения микропроцессорных систем с передачей информации по радиоканалу. В кн.: Трансферные технологии в информатике. Научно-технический сборник. Вып. 1. Томск: Изд. ТПУ, 1999. –С. 12–18.
26. Сонькин М.А., Диденко С.В. Способ построения аппаратно-программных средств контроля подвижных объектов. В кн.: Математическое и программное обеспечение проектирования систем. Научно-технический сборник. Вып. 2. Томск: Изд. ТПУ, 2002.–С.141–147.
27. Погребной В.К. Распределение типовых структур вычислительных устройств по блокам унифицированного набора. – Изв. ТПИ, т.269, Изд-во ТГУ, Томск, 1976. – С.1000–112.
28. Погребной А.В. Оптимизация топологии компонентов вычислительной системы при проектировании систем реального времени. В сб. Математическое и программное обеспечение проектирования систем. Вып. 2. Томск: Изд-во ТПУ, 2002. –С.53–55.
29. Погребной А.В. Построение модели для топологически распределенной динамической системы. – Кибернетика и вуз, вып. 30 , Томск, Изд-во ТПУ, 2003. – С. 82–86.

30. Pogrebnoy V.K., Pogrebnoy D.V. The theory and methods of modeling of distributed real-time system on the basis of structural-graphic representations of models. Proceedings of KORUS'98. – P. 217–220.
31. Погребной Д.В. Визуальное проектирование архитектуры многопроцессорной вычислительной системы реального времени. В кн.: Математическое и программное обеспечение САПР. Вып. 1 Томск: ТПУ, 1997. – С.17–22.
32. Погребной В.К. Активные модели систем. Определение и область применения. В сб. Математическое и программное обеспечение проектирования систем. Вып. 2. Томск.: Изд-во ТПУ, 2002, –С.4–19.
33. Pogrebnoy V.K., Pogrebnoy A.V. Efficient placement of stations of topologically distributed multiprocessing computing systems. Proceedings of KORUS'2004. – P.
34. Погребной В.К. Покрытие схем вычислительных устройств блоками унифицированного набора. – Известия ТПИ, т.211, изд-во ТГУ, Томск, 1970. –С.81–87.
35. Погребной В.К. Решение одной задачи дискретного программирования с булевыми переменными. – В кн.: Кибернетика и вуз, Томск: Изд-во ТГУ, 1971, №4.
36. Погребной В.К., Демин А.Ю., Погребной Д.В. Томография и структурно-графическое представление программ. В кн.: Математическое и программное обеспечение САПР. Вып.1, Томск, Изд-во ТПУ, 1997. –С.8–16.
37. Демин А.Ю. Формальное описание структурно-графического представления программного обеспечения. В кн.: Математическое и программное обеспечение проектирования систем. Вып.2. Томск: изд-во ТПУ, 2002. –С.67–75.
38. Корниенко А.В., Погребной В.К. Модель и алгоритм разбиения схем вычислительных устройств на функциональные блоки. – УсиМ, 1976, №5. –С.94–98.
39. Погребной В.К. Об автоматизации распределения памяти на графовых моделях алгоритмов. – УсиМ, 1980, №5. –С.37–42.
40. Погребной Д.В. Моделирование работы параллельных процессов на основе виртуальной машины. В кн.: Математическое и программное обеспечение проектирования систем. Вып.2. Томск: Изд-во ТПУ, 2002. –С.105–109.
41. Погребной В.К. Построение и исследование графовых моделей алгоритмов управления в АСУ. В кн.: Автоматизация проектирования систем управления. – М.: Статистика. 1978. –С.68–99.

42. Погребной В.К. Программирование на программно - реализованной вычислительной машине. В кн.: Вопросы программирования и автоматизации проектирования. Изд-во ТГУ, 1983. –С.39–66.
43. Вельбицкий И.В. Р-технология программирования. – К.: Техника, 1984. –269 с.
44. Михалевич В.С., Вельбицкий И.В. Информатика и промышленная технология программирования. – Киев. ИК АН УССР, 1984. –33с.
45. Кристофидес Н. Теория графов. Алгоритмический подход. – М.: Мир, 1978. –432 с.
46. Трахтенгерц Э.А. Введение в теорию анализа и распараллеливания программ ЭВМ в процессе трансляции. – М.: Наука, 1991. –256 с.
47. Погребной В.К. ЭФ-технология моделирования и автоматизированного проектирования систем реального времени. – УСиМ, №4, 1988. –С.23–30.
48. Липаев В.В. Проектирование программных средств: учебное пособие для вузов. – М.: Высшая школа, 1990. –303 с.
49. Терехов А.Н. RTST-технология программирования встроенных систем реального времени. Записки семинара кафедры системного программирования «CASE – средства RTST++». Вып.1. – СПб. Изд-во С-Петербургского университета, 1998.
50. Погребной В.К. О декомпозиции графов на классы изоморфных подграфов. – Вопросы программирования и автоматизации проектирования, №4, Томск, ТГУ, 1979. –С.82–96.
51. Дегтярев Ю.И. Методы оптимизации: учебное пособие для вузов. – М.: Сов. Радио, 1980. –272 с.

Оглавление