УДК 004.4'23 DOI: 10.34832/NIIR.2022.9.2.002

МЕТОДЫ И МОДЕЛИ КОРРЕКЦИИ АРХИТЕКТУРЫ БОРТОВОЙ ПРОГРАММНОЙ СИСТЕМЫ

С.В. Назаров, Московский научно-исследовательский телевизионный институт, главный научный сотрудник, д.т.н.; nazarov@mniti.ru
А.Г. Барсуков, Московский научно-исследовательский телевизионный институт, заместитель генерального директора, к.т.н.; mniti@mniti.ru

Аннотация. *Развитие бортового оборудования космических, летательных и других объектов характеризуется постоянным увеличением числа решаемых задач, повышением их сложности, расширением его интеллектуальных и адаптивных возможностей. Это приводит к усложнению бортовой вычислительной системы (БВС). На решение задач, возлагаемых на БВС, накладываются жесткие временные ограничения. Выполнение этих требований возможно путем организации параллельных вычислительных процессов. Актуальной становится задача разработки архитектуры бортовой программной системы (БПС), максимально соответствующей структуре БВС. В данной статье рассматриваются методы и модели коррекции исходной архитектуры программной системы (ПС) и оценки качества возможных ее вариантов, позволяющие принять решение о соответствии архитектуры ПС структуре БВС.*

Ключевые слова: *архитектура, бортовая программная система, расслоение, коррекция, оптимизация, качество.*

Введение. Известно, что с появлением вычислительного машиностроения термин «архитектура» первоначально применялся только к аппаратному обеспечению (архитектура ЭВМ, вычислительных комплексов, сетей и др.). С развитием информационных технологий применение этого термина существенно расширилось. Сейчас активно используются такие понятия, как архитектура программного обеспечения, архитектура операционной системы, архитектура системы сетевых протоколов, архитектура данных и др. В связи с ростом сложности компьютерных систем и с необходимостью их рассмотрения на различных уровнях детализации появились такие понятия, как архитектура процессора, архитектура набора команд и т.д.

Начало архитектуре программного обеспечения как концепции было положено в научно-исследовательских работах Э. Дейкстры в 1968 г. и Д. Парнаса в 1970-х годах. Это были первые работы, посвященные архитектуре программных систем. В них была подчеркнута критическая важность разработки правильной структуры ПС (термин «архитектура» тогда еще не использовался). На сегодняшний день пока нет единого мнения в отношении определения термина «архитектура программного обеспечения». Многие специалисты считают эту дисциплину совокупностью положений без четких правил о «правильном» пути создания системы, а само проектирование архитектуры – смесью науки и искусства. Популярность изучения этой области возросла с начала 1990-х годов вместе с работами по исследованию архитектурных стилей (шаблонов) и формальных методов проектирования, а также по созданию языков описания и документирования архитектуры. Результатами подобных исследований являются популярные монографии, например [1].

Первым стандартом программной архитектуры явился стандарт, ныне именуемый как IEEE 1471: ANSI/IEEE 1471 – 2000: Рекомендации по описанию преимущественно программных систем. Он был принят в 2007 г. под названием ISO/IEC 42010:2007 [2]. Как ни странно, не-

смотря на многолетнее развитие соответствующей дисциплины, до сих пор нельзя дать краткий и ясный ответ на вопрос о том, что такое программная архитектура. Общепринятого определения не существует. На сайте Института программной инженерии (Software Engineering Institute, SEI) [3], посвященном практическим аспектам архитектуры, приводится несколько определений. И хотя отсутствие согласия относительно единого определения не стало существенным препятствием к развитию самой дисциплины, его достижение оказалось весьма трудной задачей.

Классическим источником комплекса архитектурных точек зрения и представлений, построенных в системе координат «вопрос – уровень детализации», является модель Дж. Захмана [4]. Каждое архитектурное представление является результатом ответа на вопросы «как?», «что?», «где?» и т.п. в контексте необходимого уровня абстракции. Например, физическая модель данных (Physical Data Model) является ответом на вопрос «что?» в контексте технологической модели, а логическая модель данных, отвечая на тот же вопрос, находится на один уровень абстракции выше – в контексте системной или логической модели.

Далее в статье рассматриваются архитектурные представления бортовых программных систем и вопросы их архитектурной коррекции и оптимизации. Для формального представления архитектуры ПС предлагается использовать методы теории графов, поскольку ничего лучшего для наглядного представления множества элементов и связей между ними, пожалуй, нет. К этому надо добавить развитый математический аппарат теории графов, позволяющий исследовать, оценивать и модифицировать различные модели архитектуры ПС, представленные графами. Также будут приведены примеры коррекции и оптимизации архитектуры ПС.

Особенности бортовых вычислительных систем. Специализированные БВС имеют разнообразное применение. В данной статье рассматриваются системы, которые используются для управления и иных целей (стабилизации, защиты, безопасности и др.) в подвижных объектах космического, воздушного и наземного характера, например, летательных аппаратах (дронах, ракетах боевого применения различных классов, самолетах и др.), транспортных средствах и т.п. Как правило, наборы программ, реализуемых в таких объектах современными БВС, можно представить совокупностью информационно-связанных подпрограмм (заданным набором задач – ЗНЗ).

Это позволяет представлять решаемые задачи в виде нагруженного графа и, соответственно, использовать для их анализа методы теории графов, сетевого планирования и управления. Обычно ЗНЗ обладают достаточным внутренним параллелизмом, который может быть удобно использован при реализации задач многопроцессорными (многоядерными) бортовыми системами. Это особенно важно, поскольку БВС, как правило, учитывая специфику управляемых объектов, работают в режиме реального времени с ограничениями на время выполнения ЗНЗ.

Использование многопроцессорных (многоядерных) вычислителей часто характеризуется тем, что архитектура аппаратных средств многопроцессорной системы фиксируется на этапе проектирования и остается неизменной на время выполнения программ. Это означает, что разработчик должен выбирать соответствующую систему для каждого предполагаемого приложения. С этой целью разработчик должен разбить приложение на части, рассмотреть возможности параллелизма при выполнении приложения и выбрать соответствующую систему для своего приложения. Ограничением такого подхода является то, что у существующих многопроцессорных систем отсутствует адаптивность на стадии разработки и во время выполнения программы [5].

Программируемая разработчиком логическая матрица предлагает более гибкое решение, потому что с ее помощью аппаратные средства можно переконфигурировать с новыми функциями и использовать повторно с различными приложениями. Некоторые поставщики программируемых пользователем логических матриц предлагают специальную функцию, называемую динамическим и частичным переконфигурированием [6, 7]. Это означает, что часть аппаратных

средств системы может быть изменена во время выполнения программы, а оставшаяся часть остается действующей и неизменной.

Рассмотрим модели архитектуры бортовых программных систем, учитывая указанные выше особенности задач (внутренний параллелизм, временные ограничения), решаемых в бортовых системах, и возможности бортовых вычислительных средств (жесткие структуры, переконфигурация, адаптация и др.).

Модели формального представления архитектуры БПС. Архитектура – это базовая организация системы, воплощенная в ее компонентах, их отношениях между собой и с окружением, а также принципы, определяющие проектирование и развитие системы [8–10]. Архитектура программной системы определяет ее структуру, точнее, несколько структур, каждая из которых включает в себя элементы и взаимосвязи между ними. Элементы могут быть вычислительными объектами, связанными потоком управления, или бизнес-объектами, связанными семантическими ограничениями.

В целом процесс проектирования архитектуры состоит из систематической декомпозиции элементов верхнего уровня на совокупности более мелких элементов. Наилучшим математическим аппаратом для формального представления архитектуры, исходя из этого определения, следует считать теорию графов. Математический аппарат теории графов позволяет исследовать, оценивать и модифицировать различные модели архитектуры программных систем, представленные графами.

Среди множества возможных архитектурных схем, отметим положительные моменты многослойной архитектуры БПС. Прежде всего заметим, что эта архитектура не подразумевает какое-то обязательное количество слоев, их может быть три, четыре, пять и больше. Это хорошо соответствует процессу декомпозиции элементов вышележащего уровня на совокупности нижележащих элементов программы. При реализации элементов каждого слоя может быть занято переменное количество процессорных единиц (что очень важно).

Переход от реализации некоторого слоя программных элементов к следующему слою может потребовать увеличения или уменьшения числа процессорных элементов. В этом плане конфигурируемая матрица процессорных элементов наиболее эффективна. Завершение некоторого программного элемента текущего слоя позволяет использовать освободившийся процессорный элемент для выполнения программного элемента следующего слоя.

Кроме того, к положительным сторонам многослойной архитектуры можно отнести:

- более простую реализацию по сравнению с другими архитектурами;
- более высокий уровень абстракции благодаря разделению ответственностей между слоями;
- защита одного слоя от изменений, вносимых в другие слои, благодаря изолированию слоев;
- повышение управляемости программного обеспечения за счет слабой связности.

Рассмотрим возможные варианты многослойной архитектуры. Если в варианте, показанном на рис. 1, каждый слой имеет доступ к элементам только одного нижележащего слоя, то разработчик должен иметь в виду только предыдущий слой. Хотя с точки зрения проектирования этот вариант кажется привлекательным, он может оказаться неэффективным.

Например, если некоторое средство, представляемое слоем $L(2)$, потребуется в слое $L(i)$, то каждый из слоев $L(3)$, $L(4)$, ..., $L(i-1)$ должен обеспечить это средство. Это значит, что запрос данного средства слоем $L(i)$ должен «просачиваться» вниз через слой $L(i-1)$, пока не достигнет слоя $L(2)$, который способен выполнить запрос. Эти трудности, связанные с проблемой эффективности, могут склонить к принятию структуры, представленной на рис. 2, в которой каждый слой $L(i)$ (где $2 < i < n$, n – количество слоев) может непосредственно обращаться к слою $L(2)$.

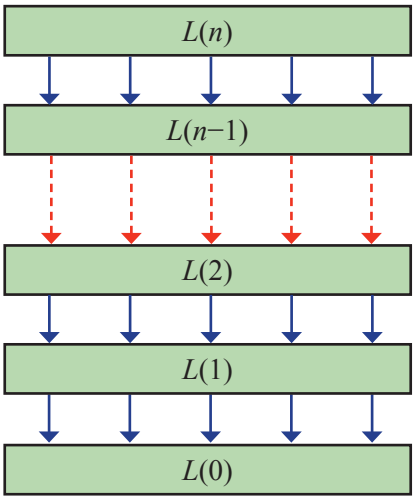

Рис. 1. Вариант классической структуры

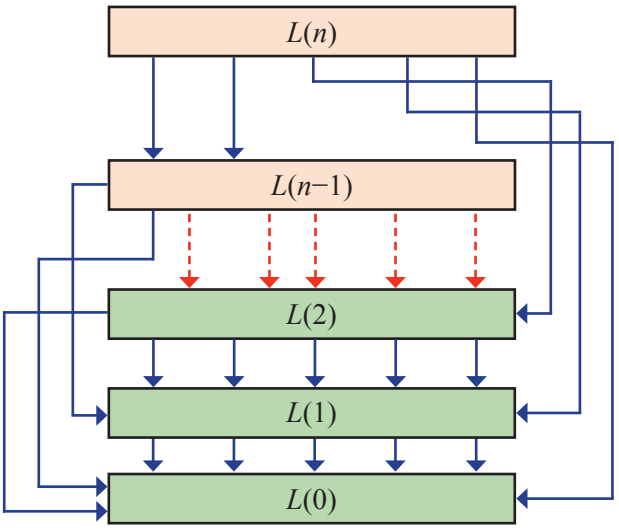

Рис. 2. Вариант структуры с полными связями

Модель многослойной архитектуры произвольной структуры удобно описывать множеством булевых переменных

$$X = \{x_{ij} \in \{0,1\} \mid i = n, n-1, ..., 2; j = n-1, n-2, ..., 1; i > j\}, \quad (1)$$

где x_{ij} =1, если существует связь между слоями i и j, или x_{ij} =0, если такой связи нет. Так как между смежными слоями всегда имеется связь, то

$$(\forall i \mid i = j+1)(x_{ij}=1),\ i = n, n-1, ..., 2. \quad (2)$$

Если в многослойной структуре программы, представленной выражением (1), принимают единичные значения только переменные, описываемые условием (2), то эта программа имеет структуру, соответствующую варианту на рис. 1. Если справедливо условие

$$(\forall i \mid i = n, n-1, ..., 2)(\forall j \mid (i-j \geq 1))(x_{ij}=1),$$

то эта программа имеет структуру, соответствующую варианту, показанному на рис. 2.

Если справедливо условие

$$(\exists i \mid (i = n, n-1, ..., 2))\&(\exists j \mid (i-j \geq 2))(x_{ij}=1),$$

то программа имеет структуру, показанную на рис. 3, которая соответствует промежуточному варианту между вариантами структур, представленными на рис. 1 и 2.

Альтернативой многослойной архитектуре является многоуровневая архитектура, которая предполагает разделение ПС на уровни по принципу взаимодействия клиент–сервер [10]. Архитектура может иметь один, два и более уровней, разделяющих ответственности между сервером-поставщиком данных и клиентом-потребителем. В отличие от многослойной архитектуры, многоуровневая архитектура предлагает масштабируемость, которая может быть, как горизон-

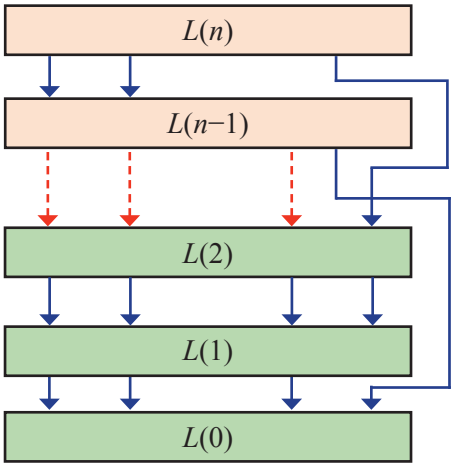

Рис. 3. Вариант структуры с произвольными связями

тальной (масштабирование сети с помощью высокопроизводительных узлов), так и вертикальной (масштабирование каждого узла путем повышения его производительности).

Реализация многоуровневой системы, как правило, обходится дороже, но такая система может обеспечить более высокую производительность. Этот подход можно сочетать с современной сервис-ориентированной архитектурой, чтобы создавать сложнейшие модели. Таким образом, наиболее целесообразной архитектурой БПС в рамках данной статьи будем считать многослойную архитектуру.

Изучение структуры программы и ее проектирование (или рефакторинг) связаны с разбиением программы на слои в соответствии с некоторыми принципами расслоения. Наиболее распространенная идея расслоения программной системы по принципу равной доступности модулей слоев была сформулирована Рабочей группой по методологии программирования Международной федерации по обработке информации (в группу входили Н. Вирт, У. Дал, Э. Дейкстра и др.). Изложение этой идеи было дано в начале 1980-х годов в монографии [11], однако, оно не потеряло актуальности и в настоящее время.

Методы коррекции архитектуры программной системы. Отвлекаясь от метода получения начальной архитектуры ПС, сосредоточимся на коррекции и оптимизации этой архитектуры. Вопросы оптимизации архитектуры остаются актуальными на всех этапах создания ПС от разработки технического задания до рефакторинга системы. На всех этих этапах процесс изменения внутренней структуры программы, который имеет целью облегчить понимание ее работы, не должен затрагивать внешнее поведение программы. В основе оптимизации лежит последовательность эквивалентных (т.е. сохраняющих поведение программы) преобразований, не меняющих функциональную семантику базового кода.

По ходу трансформаций часто встает задача выявления смысловой нагрузки модулей. Для решения подобных задач зачастую приходится исследовать реальный программный код, анализировать сигнатуры функций и комментарии, а при отсутствии последних – и сам код функций. Задача специалиста, вовлеченного в процесс архитектурных преобразований, состоит в том, чтобы по возможности минимизировать объем семантического анализа (например, путем удаления вспомогательных блоков) и сделать его последовательным и направленным.

Необходимость в архитектурных преобразованиях системы может быть связана со следующими причинами:

1. Развитие программы, обусловленное необходимостью изменений. При этом из-

менения часто вносят программисты, которые не до конца понимают архитектуру ПС в целом, и в результате постепенно код становится менее структурированным, а разбираться в нем все труднее. Архитектурные преобразования улучшают композицию ПС.

2. Повышение производительности ПС. Изменения, уточнения и усложнения программного модуля, несомненно, приводят к замедлению выполнения программы, но при этом делают ее более понятной и податливой для настройки производительности.
3. Потребность в функциональных изменениях ПС. Внедрение новой функциональности не должно затронуть логику системы. Изменение существующей архитектуры может быть хорошим шагом на пути внедрения новой функциональности, облегчающим дальнейшую эволюцию системы.
4. Смена аппаратной платформы ПС (например, переход на программируемые логические матрицы). Смена платформы ПС должна минимально затрагивать существующий код. Желательно ограничиться изменениями только в узкой платформенно-зависимой прослойке системы. Выделение такой прослойки всегда сопряжено с необходимостью изменения архитектуры.
5. Возможность организации распараллеленного (многопоточного) вычислительного процесса при наличии в вычислителе нескольких ядер.
6. Обновление технологии разработки программного продукта, связанное, например, с переходом на более совершенную технологию программирования.

Прежде чем говорить о коррекции архитектуры, следует задаться вопросом: как оценить качество структуры ПС? Из практики проектирования известно, что лучшее решение обеспечивается иерархической структурой в виде дерева. Степень отличия реальной проектной структуры от дерева характеризуется невязкой структуры. Известно, что у полного графа с n вершинами число ребер составляет $e_c = n(n-1)/2$, а у дерева n вершинами ребер существенно меньше – $e_t = n–1$.

Формулу невязки можно построить, сравнивая количество ребер полного графа, реального графа и дерева. Для проектной структуры с n вершинами и e ребрами невязка определяется выражением

$$Nev = \frac{e - e_t}{e_c - e_t} = \frac{2(e-n+1)}{n(n-1)-2(n-1)} = \frac{2(e-n+1)}{(n-1)(n-2)}. \quad (3)$$

Значение невязки лежит в диапазоне от 0 до 1. Если $Nev = 0$, то проектная структура является деревом, если $Nev = 1$, то проектная структура – полный граф. Ясно, что невязка дает грубую оценку структуры. Для увеличения точности оценки следует применить характеристики связности и сцепления. Использование таких характеристик для оценки качества программы показано в монографии [10].

Вообще заметим, что формализовать процесс коррекции архитектуры ПС или тем более построить алгоритм этой коррекции довольно затруднительно. Однако в ряде случаев, выделив отдельные фрагменты структуры ПС, можно их преобразовать, стремясь к получению наилучшей структуры, например, к дереву. Чаще всего это удается сделать путем изменения числа слоев, перемещения модулей между слоями, объединения (поглощения) или разделения модулей. Некоторые примеры такой коррекции даны в статье [12]. Остановимся на основных приемах коррекции архитектуры, с помощью которых можно преобразовать архитектуру ПС с целью наилучшего использования вычислительных возможностей БВС, обратив внимание на возможный эффект такого преобразования.

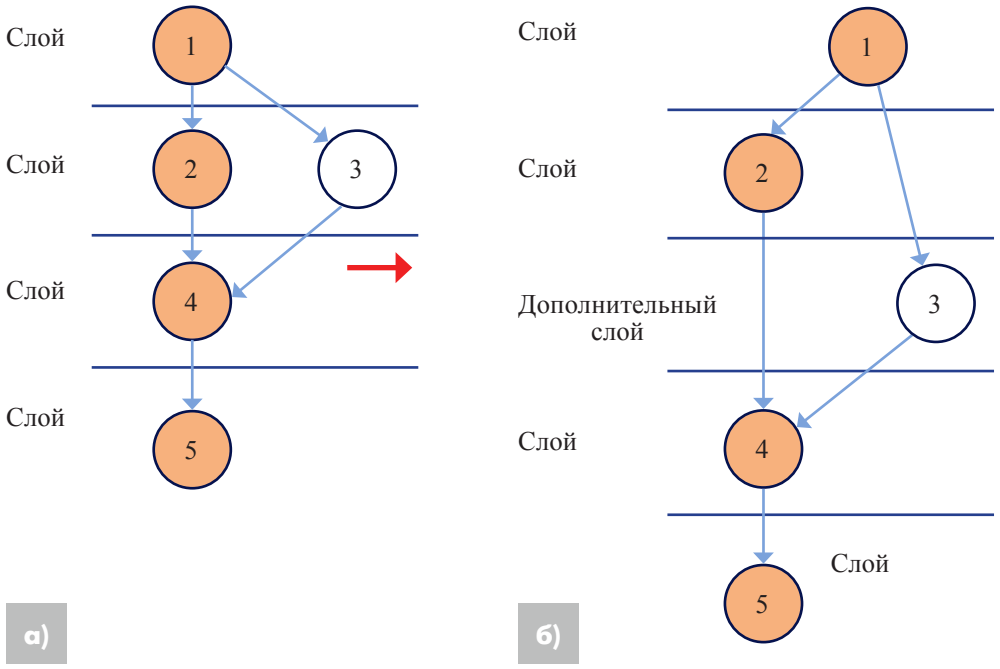

Рис. 4. Коррекция перемещением за счет введения дополнительного слоя

На рис. 4 представлен вариант коррекции архитектуры ПС путем введения в систему дополнительного слоя. На рис. 4, *а* показана возможность параллельного выполнения модулей 2 и 3. Если в БВС имеется только один вычислитель, такая возможность не реализуется. В этом случае возможно только последовательное выполнение этих модулей. Коррекция архитектуры системы в этом случае – введение дополнительного слоя, как показано на рис. 4, *б*. Это, конечно, приведет к увеличению времени выполнения ПС, но для однопроцессорной (одноядерной) БВС иного решения в данном случае нет.

Вариант увеличения скорости вычислительного процесса показан на рис. 5. В данном случае коррекция идет по двум направлениям: уменьшается количество слоев ПС и увеличивается параллельность вычислительного процесса. Из рис. 5, *а* ясно, что можно удалить слой, где размещается модуль 3, и передвинуть его в вышележащий слой. При этом можно выполнять параллельно модули 2 и 3 (если в БВС есть свободный вычислитель), как это показано на рис. 5, *б*.

На рис. 6, *а* показана последовательная цепочка модулей 2 и 3, которые используются только модулем 1. Возможно, это стало следствием желания распараллелить работу по программированию этих модулей. Улучшить эту структуру ПС можно путем объединения модулей 2 и 3, как показано на рис. 6, *б*. Заметим, что если в этом случае модуль 1 обращается только к модулям 2-3 и 4, то сокращается число слоев ПС. Полезный эффект такого преобразования – сокращение числа операций обмена.

На рис. 7, *а* показан случай, когда результаты работы модулей 1 и 2 используются только модулем 3. Такую структуру ПС можно улучшить путем объединения модулей 1 и 2, как показано на рис. 7, *б*. Это может привести к сокращению числа вычислителей. Такая коррекция объединением модулей, принадлежащих одному слою, полезна в случае недостаточного количества вычислителей БВС.

Другой случай объединения модулей 2 и 3 приведен на рис. 8, *а*. Он возможен, если к мо-

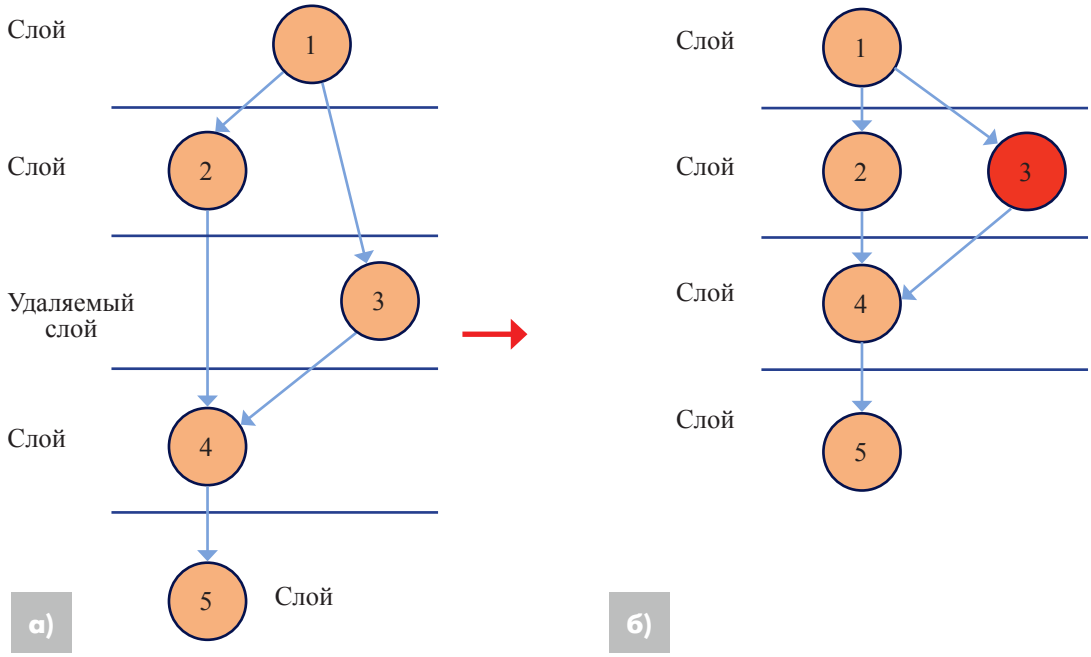

Рис. 5. Коррекция перемещением за счет удаления слоя

дулям 2 и 3 обращается только модуль 1, а сами модули 2 и 3 обращаются только к модулю 4. В результате объединения модулей 2 и 3, как показано на рис. 8, *б* упрощается структура ПС. Этот подход может быть использован, например, для уменьшения числа вычислителей и сокращения операций обмена результатами текущих вычислений.

При коррекции архитектуры ПС, кроме объединения модулей и их перемещения между слоями, возможны ситуации разделения модулей, как показано на рис. 9, *а* и *б*. Это может быть полезно, например, для организации параллельного вычислительного процесса и снижения сложности отдельных модулей. В данном случае (см. рис. 9, *а*) модуль 4 разделен функционально, поскольку

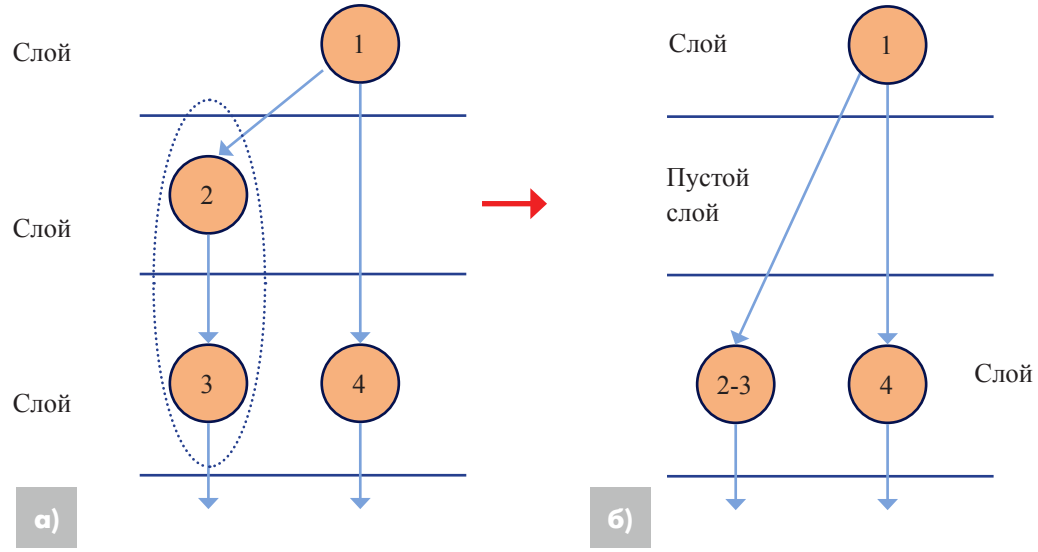

Рис. 6. Коррекция поглощением нижележащим слоем

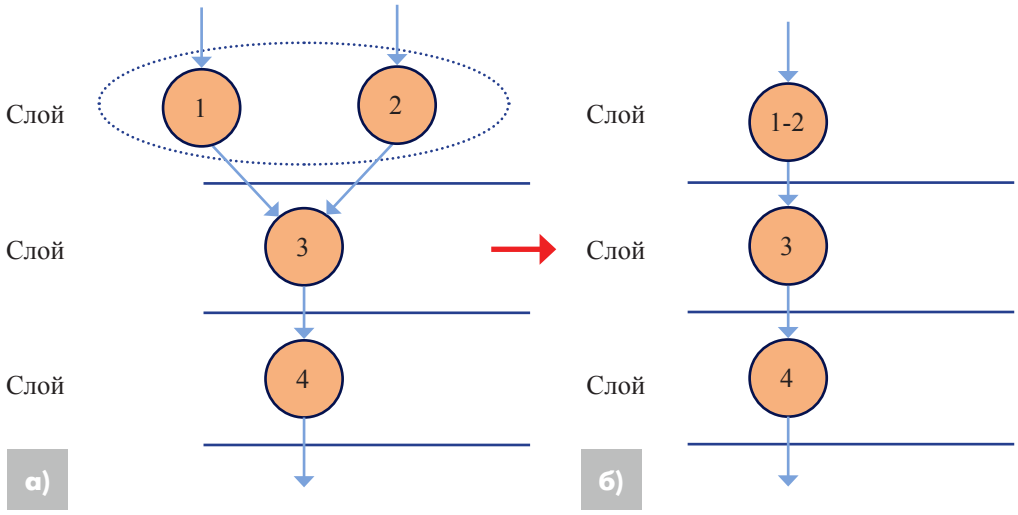

Рис. 7. Коррекция объединением (вариант 1)

результат его работы необходим различным модулям нижележащего слоя. Заметим, что такое разделение не всегда возможно. Например, если результат модуля 2 нужен для модуля 4-2, то появится дополнительная межмодульная связь: модуль 2 – модуль 4-2. Также если результат модуля 3 нужен для модуля 5, то появится дополнительная связь: модуль 3 – модуль 4-1. Таким образом, в этом случае вместо шести связей будет семь, что ставит под сомнение эффективность такого функционального разделения.

В других ситуациях разделение может понадобиться для увеличения скорости выполнения модуля. В этом случае необходимо переработать алгоритм модуля: найти возможность его распараллеливания для реализации на двух и более вычислителях (рис. 10). На рис. 10, *а* показан модуль 2, время реализации которого существенно замедляет вычислительный процесс. На рис. 10, *б* модуль перепрограммирован и содержит два параллельно выполняемых модуля 2-1 и 2-2, которые могут выполняться на отдельных вычислителях. Однако в каждом конкретном случае решение о той или иной коррекции структуры ПС должно приниматься после детального анали-

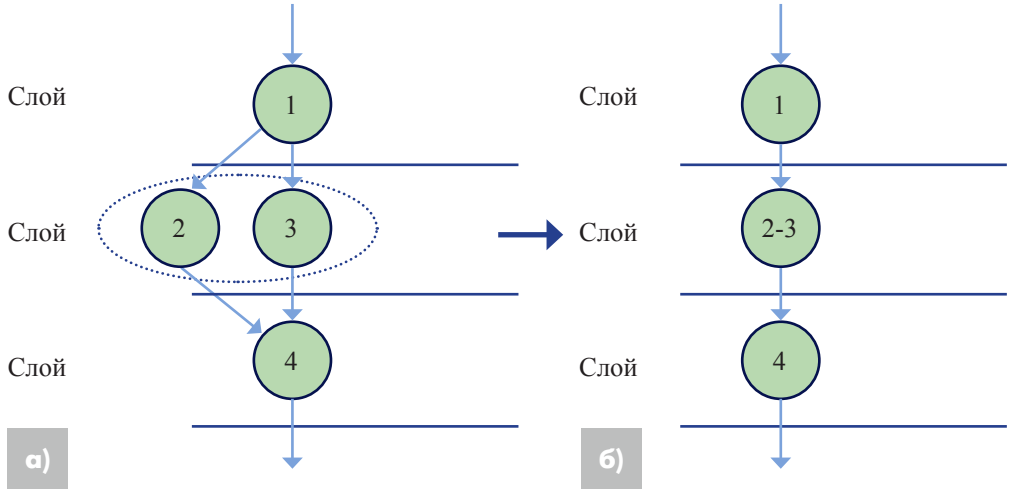

Рис. 8. Коррекция объединением (вариант 2)

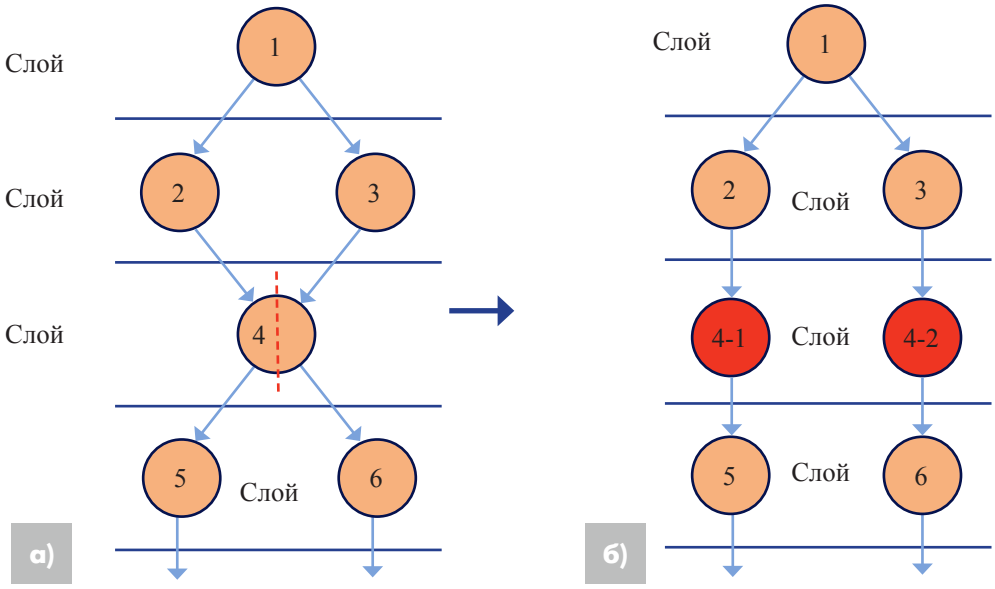

Рис. 9. Коррекция функциональным разделением

за и оценки ее целесообразности.

Результат коррекции архитектуры в каждом из рассмотренных случаев должен быть спроецирован на реальный программный код системы. При проецировании удаления модулей из модели необходимо определить множество строк и файлов, которое соответствует удаленному блоку в программном коде. После этого необходимо удалить из программного проекта выявленные строки и файлы. При проецировании на код переноса модуля в модели переносятся соответствующие строки и файлы в исходном коде программной системы и т.д. Производимые таким образом трансформации можно рассматривать как архитектурно-управляемый рефакторинг программного кода.

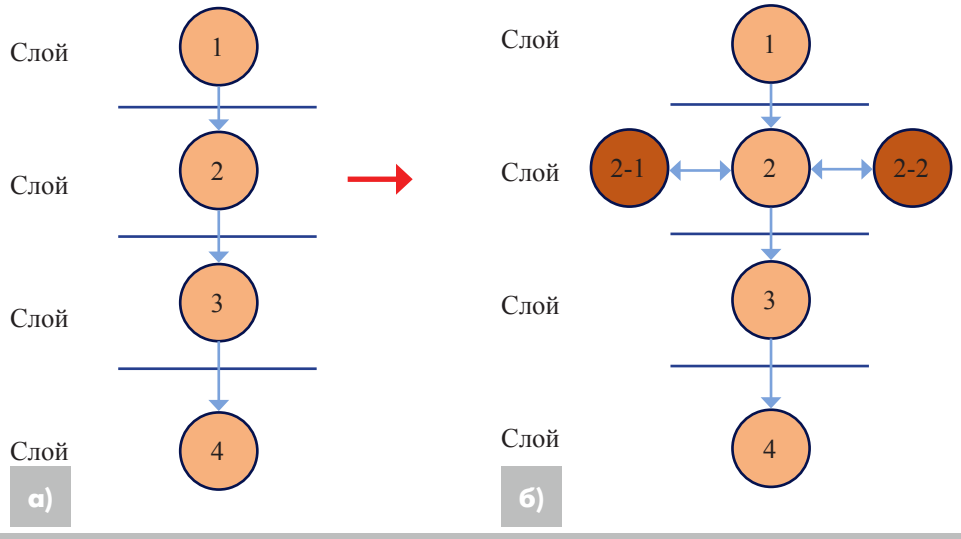

Рис. 10. Коррекция разделения распараллеливанием

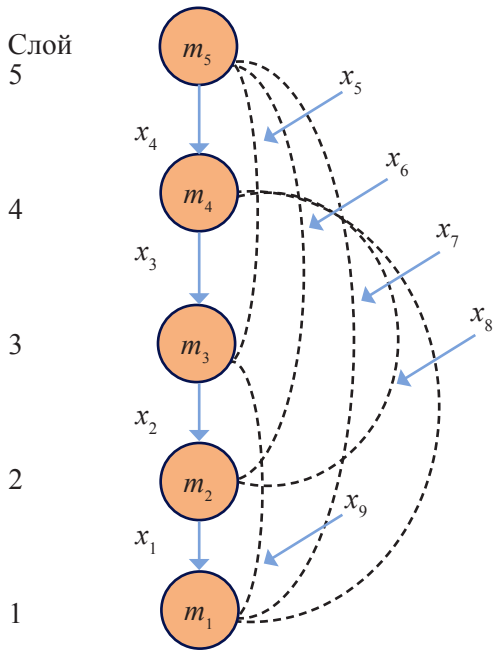

Рис. 11. Структура многослойной ПС в обобщенном виде

Пример оптимизации архитектуры для повышения производительности ПС. Рассмотрим использование изложенных выше методов коррекции архитектуры ПС с целью повышения производительности БВС [13]. Для простоты изложения структура многослойной ПС дана в обобщенном виде (рис. 11), каждый слой показан в виде одного модуля с возможностью организации связей с любым произвольным слоем системы. Такая модель позволяет рассмотреть любой вариант многослойной ПС, лежащий в диапазоне структур, приведенных на рис. 1–3.

Учитывая, что число слоев в большинстве ПС, как правило, не превышает трех-пяти, рассмотрим пятислойную программу, представленную на рис. 11, где группы модулей по слоям программы обозначены как $m_1, m_2, ..., m_5$, а возможные связи между слоями – $x_1, x_2, ..., x_9$. Пунктирными линиями показаны возможные дополнительные связи между слоями. Каждой линии поставлена в соответствие булева переменная, единичное значение которой означает наличие межслойной связи, а нулевое – отсутствие такой связи.

Будем считать, что ПС прошла полный этап тестирования и в процессе ее отладки были определены временные характеристики модулей и частоты обращения модулей любого слоя к модулям нижележащих слоев. Предположим, что передача (трансляция) запроса через слой i дополнительно загружает этот слой на некоторый промежуток времени t_i. Например, если x_5 =0, (т.е. отсутствует связь между слоями 5 и 3), то модуль m_4 дополнительно работает в течение промежутка времени t_5. Если x_5 =1, то дополнительное время модулю m_4 не потребуется. Однако в этом случае необходимо в программу добавить межмодульный интерфейс для взаимодействия модулей m_5 и m_3, который несколько увеличит время работы модулей m_5 и m_3 (на величину t_5') и увеличит программу на некоторую величину e_5. Аналогичные рассуждения справедливы и для других переменных, показанных на рис. 11.

Как правило, дополнительные связи между слоями программы сокращают время выполнения ее функций, но увеличивают размер программы. Необходимо также учесть тот факт, что дополнительно создаваемые связи между слоями могут работать с различной нагрузкой. Так, например, если создается связь, обозначаемая переменной x_5, то модуль m_4 освобождается от

трансляции только тех запросов, которые модуль m_5 адресует модулю m_3. Поэтому целесообразно каждой переменной x_i поставить в соответствие определенную интенсивность взаимодействия некоторой пары модулей λ_i. В этих условиях задача архитектурной оптимизации ПС сводится к определению такой структуры, которая обеспечивает наилучшую производительность программы при заданных ограничениях на размер дополнительных межмодульных интерфейсов.

В нашем случае структура многослойной ПС может быть представлена вектором $\mathbf{X}=\{x_i \mid i=5, 6, ..., 9\}$ (заметим, что всегда $x_1 = x_2 = x_3 = x_4 = 1$, так как эти переменные определяют связи между смежными слоями). Поэтому требуется найти такое значение $\mathbf{X}_{\text{opt}}$, при котором обеспечивается максимальный выигрыш во времени работы БПС

$$\text{Max}T = \sum_{i=5}^{i=9} \lambda_i (t_i - t_i') x_i \tag{4}$$

при выполнении ограничения на допустимое увеличение программы $E_д$ за счет дополнительных межмодульных интерфейсов

$$\sum_{i=5}^{i=9} e_i x_i \le E_4, \tag{5}$$

где e_i – дополнительные затраты памяти за счет межмодульного интерфейса i-ой пары модулей. Учитывая двоичный характер переменных, следует добавить ограничение

$$\forall i (x_i \in \{0,1\}). \tag{6}$$

Сформулированная задача (4)–(6) относится к классу задач о загрузке рюкзака. Малая размерность (в нашем примере) задачи позволяет легко решить ее полным перебором переменных, представляющих допустимые решения в условиях принятых ограничений. В реальных программных системах, содержащих по нескольку модулей в каждом слое, размерность задачи может существенно вырасти и потребуется применить боле сложные алгоритмы решения. Заметим также, что в связи с неточностью исходных данных в рассматриваемой задаче, не имеет смысла использовать сложные точные методы получения оптимального решения и можно ограничиться приближенными, быстро работающими алгоритмами. Результат решения должен быть спроецирован на реальный программный код системы.

Заметим, что кроме решения задачи, представляет интерес анализ возможных вариантов структур с целью получения ответа на вопрос «что будет, если?». Другими словами, хорошо было бы иметь модель, которая помогла бы детально исследовать возможные варианты архитектур ПС. Такую модель легко построить, используя электронные таблицы, например, Excel.

Пример коррекции архитектуры ПС для заданного количества вычислителей. Представляет интерес использование предложенных выше методов коррекции архитектуры ПС для оптимизации ее структуры при организации вычислительного процесса с заданным количеством вычислителей. В качестве ограничивающих факторов в этом случае выступает количество вычислительных элементов БВС, которые могут быть использованы для выполнения ЗНЗ, и возможности операционной системы БВС по адаптивному перераспределению вычислителей по модулям ПС. Для дальнейшего анализа использования указанных выше возможных архитектурных преобразований (коррекций) рассмотрим некую архитектуру ПС, граф которой показан на рис. 12. В этом примере и далее будем использовать модульное расслоение программы, отражающее взгляд «сверху–вниз» на структуру программы. При этом подразумевается наличие базового слоя модулей ниже слоя 0. Пусть требуется провести коррекцию исходной архитектуры для случая двухпроцессорной БВС.

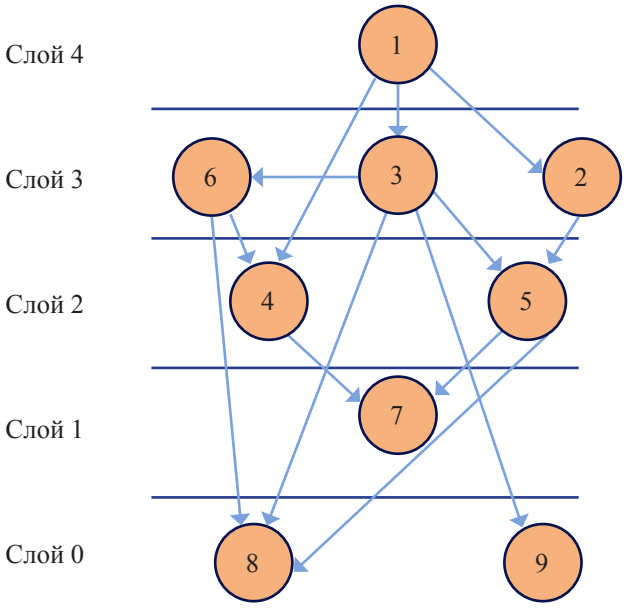

Рис. 12. Граф программной системы

Анализируя этот граф архитектуры, следует отметить, что он не отвечает каноническим правилам многослойной структуры, в частности, им не соответствует модуль 6. Известно, что выделение слоев – хорошая основа для улучшения системы. Найти строгие слои в произвольной ПС достаточно трудно, поскольку, как уже отмечалось, они могут содержать горизонтальные связи и сильносвязанные компоненты. Поэтому целесообразно расширить понятие слоя, позволив включать в произвольные слои сильносвязанные компоненты. Эти компоненты при таком подходе можно рассматривать, как атомарные модули. Заметим, что не всегда сильносвязанные компоненты на структурных диаграммах свидетельствуют о плохой архитектуре системы. Возможным дефектом архитектуры с поглощающими слоями может стать эффект «пропавшего слоя» – дефектная связь приводит к появлению модулей, которые по смыслу должны находиться на разных слоях.

Вернемся к структуре, приведенной на рис. 12. Оценим качество исходной архитектуры по характеристике, заданной формулой (3), при количестве вершин графа $n = 9$ и числе ребер $e = 14$:

$$Nev_{и} = \frac{2(e-n+1)}{(n-1)(n-2)} = \frac{2(14-9+1)}{(9-1)(9-2)} = 0,214.$$

Вторая важная характеристика структуры ПС – средняя ширина слоя. Это среднее количество модулей в одном слое программы (обозначим это значение через A). Средняя ширина слоя определяется выражением

$$A = \frac{1}{k}\sum_{i=1}^{k} w_i, \qquad (7)$$

где w_i – ширина слоя i (количество модулей в слое), k – число слоев. Прямоугольность графа ПС можно оценить средним отклонением ширины слоя от значения A. Обозначим это отклонение величиной D:

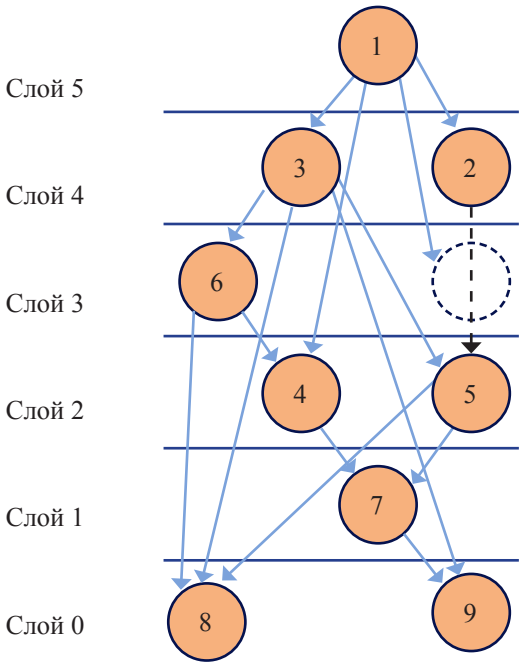

Рис. 13. Первый вариант коррекции архитектуры ПС

$$D = \frac{1}{k}\sum_{i=1}^{k} | w_i - A |. \quad (8)$$

Очевидно, что чем ближе значение D к нулю, тем лучше соответствие архитектуры ПС бортовой вычислительной системе. Для исходной архитектуры ПС в соответствии с (7) и (8):

$$A_{\text{и}} = \frac{(1+3+2+1+2)}{5} = 1,8;$$

$$D_{\text{и}} = \frac{|1-1,8|+|3-1,8|+|2-1,8|+|1-1,8|+|2-1,8|}{5} = 0,64.$$

Данные характеристики показывают приспособленность структуры ПС к реализации определенным количеством вычислителей (в нашем случае два вычислителя). Начинаем коррекцию исходной архитектуры (см. рис. 12). Ясно, что модуль 6 не может находиться с модулем 3 в одном слое. Так как модуль 3 для выполнения своих функций обращается к модулю 6, то последний должен быть перемещен в нижележащий слой. Возможный вариант новой структуры показан на рис. 13. Заметим, что в данном случае модуль 2 должен быть перемещен в нижележащий слой 3 или необходимо переместить модуль 5 в слой 3. Следует обратить внимание на увеличение количества слоев ПС после выполненной коррекции. Этот важный факт может привести к увеличению времени работы ПС. Отметим также, что после такой коррекции структуры не изменилась сложность ПС, определяемая по значению Nev, поскольку число вершин и ребер осталось прежним, т.е. $Nev_1 = Nev_{\text{и}}$.

Определим для первого варианта коррекции ПС значения показателей по выражениям (7) и (8):

$$A_1 = \frac{(1+2+1+2+1+2)}{6} = 1,5;$$

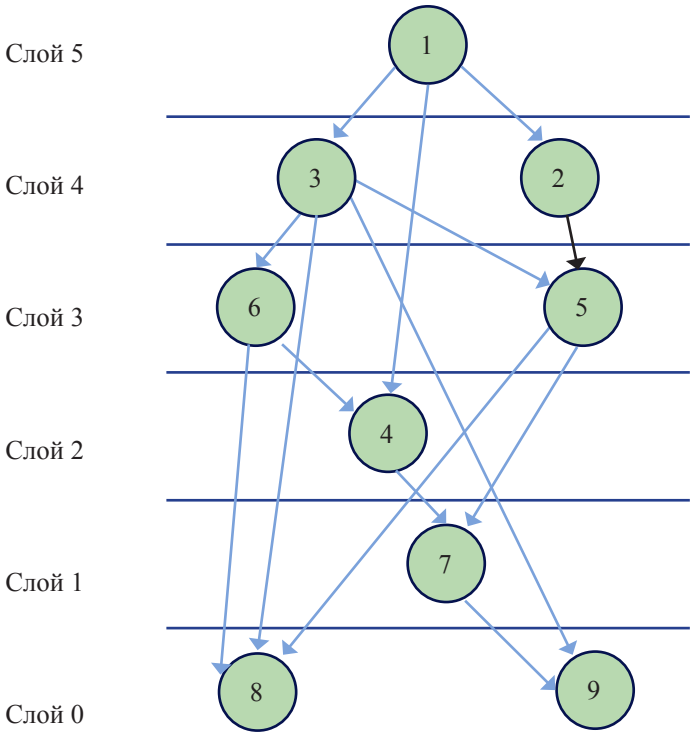

Рис. 14. Второй вариант коррекции архитектуры ПС

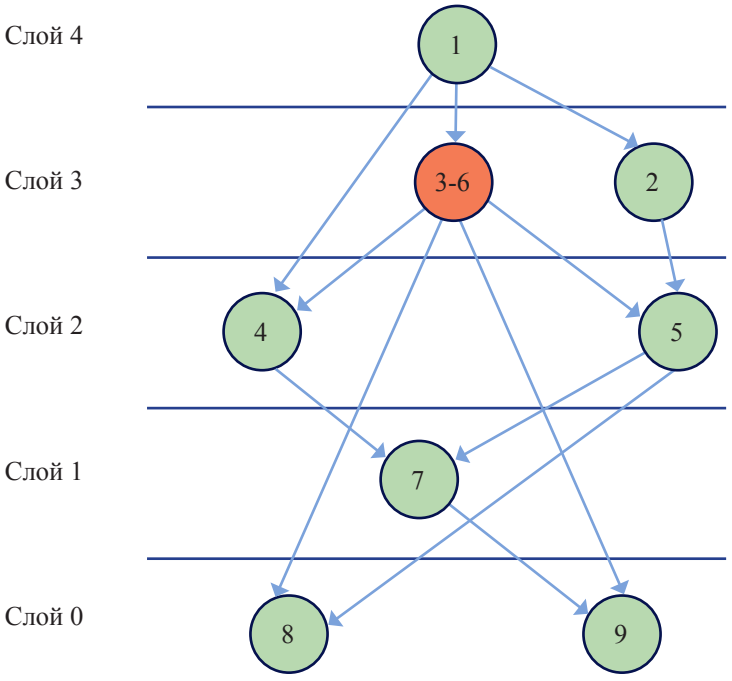

Рис. 15. Третий вариант коррекции архитектуры ПС

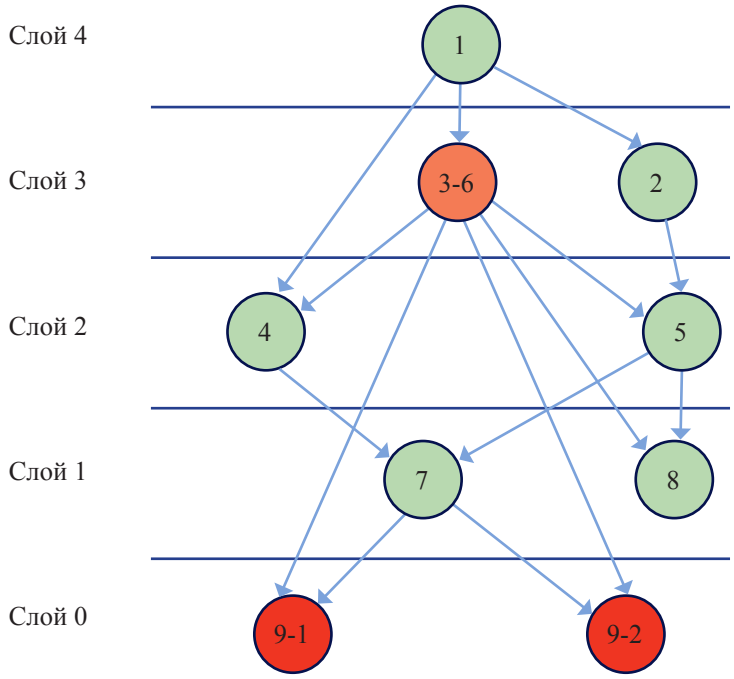

Рис. 16. Четвертый вариант коррекции архитектуры ПС

$$D_1 = \frac{|1-1,5|+|2-1,5|+|1-1,5|+|2-1,5|+|1-1,5|+|2-1,5|}{6} = 0,5.$$

Второй вариант (рис. 14) коррекции исходной архитектуры ПС отличается от первого варианта тем, что модуль 5 перемещается на вышележащий слой. Легко убедиться, что в этом случае значения показателей Nev_2, A_2 и D_2 равны предыдущим значениям Nev_1, A_1 и D_1.

Возможен также третий вариант коррекции архитектуры, показанный на рис. 15. Он связан с объединением модулей 3 и 6 (на рис. 15 это модуль 3-6). При этом меняется количество слоев ПС (их становится пять, что лучше, чем в первом и втором вариантах), и изменяется число вершин и ребер (8 вершин и 12 ребер графа). Это несколько снижает сложность ПС по значению невязки:

$$Nev_3 = \frac{2(e-n+1)}{(n-1)(n-2)} = \frac{2(12-8+1)}{(8-1)(8-2)} = 0,24.$$

Определим также как изменились показатели прямоугольности графа ПС:

$$A_3 = \frac{(1+2+2+1+2)}{5} = 1,6;$$

$$D_3 = \frac{|1-1,6|+|2-1,6|+|2-1,6|+|1-1,6|+|2-1,6|}{5} = 0,48.$$

Однако объединенный модуль 3-6 увеличивается в объеме, а его программирование усложняется. Дальнейшая коррекция ПС возможна перемещением модуля 8 на соседний вышележащий слой, этим улучшается прямоугольность графа структуры ПС. Если возможно разделение моду-

Характеристики различных вариантов структуры программной системы

Структура	Невязка *Nev*	Средняя ширина слоя *A*	Отклонение от средней ширины *D*	Число слоев
Исходная	0,214	1,8	0,64	5
Вариант 1	0,214	1,5	0,5	6
Вариант 2	0,214	1,5	0,5	6
Вариант 3	0,24	1,6	0,48	5
Вариант 4	0,214	1,8	0,32	5

ля 9 распараллеливанием его алгоритма, то получим окончательную (четвертую) структуру ПС, представленную на рис. 16. Определим основные характеристики этого варианта структуры ПС (9 вершин и 14 ребер):

$$Nev_4 = \frac{2(e-n+1)}{(n-1)(n-2)} = \frac{2(14-9+1)}{(9-1)(9-2)} = 0,214;$$

$$A_4 = \frac{(1+2+2+2+2)}{5} = 1,8;$$

$$D_4 = \frac{|1-1,8|+|2-1,8|+|2-1,8|+|2-1,8|+|2-1,8|}{5} = 0,32.$$

Для удобства сравнения полученных структур ПС результаты вычисления их характеристик представлены в таблице.

Анализируя характеристики полученного семейства архитектур программной системы, можно сделать вывод о наилучшем приспособлении варианта 4 для реализации ПС на двухпроцессорном вычислителе. Однако вопрос о действительной оптимальности выбранной архитектуры остается на данном этапе открытым. Это связано с асинхронным выполнением модулей в ходе вычислительного процесса. Потому вполне возможны низкие загрузки и даже простои вычислителей, что может поставить под сомнение целесообразность использования двухпроцессорной БВС. Принять окончательное решение можно, только построив диаграммы загрузки процессоров, используя временные характеристики модулей. Пример такого решения дан в статьях [14, 15].

Заключение. Бытует высказывание: «архитектура – это то, за что увольняют системного архитектора и руководителя проекта». Именно архитектор занимается проектированием архитектуры программной системы и разработкой архитектурного описания этой системы. Важнейшая обязанность архитектора заключается в разработке ключевых проектных решений о внутреннем устройстве программной системы. Это начальный этап создания программной системы, а, как известно, ошибки, допущенные на этом этапе, исправляются тем сложнее и дороже, чем позже они выявляются. В то же время не существует хорошо разработанных формальных методов проектирования архитектуры программных систем. И если само программирование до сих пор во многом считается искусством, то в значительной большей степени это относится к проектированию архитектуры программных систем.

Неоценима важность разработки архитектуры бортовой программной системы для класса летательных аппаратов, в которых ресурсы вычислительной системы существенно ограничены, особенно в части памяти. Иногда бывает так, что программная система разрабатывается без предварительного архитектурного описания. Но как бы она не разрабатывалась, в ней уже

заложена определенная архитектура. Далее начинается понимание важности и необходимости иметь ее архитектурное описание и начинается процесс «раскопки архитектуры» – представление созданной архитектуры, после чего начинается ее коррекция (рефакторинг). По мнению авторов, на этом этапе данная статья может оказать определенную помощь в процессах коррекции архитектуры создаваемых программных систем.

ЛИТЕРАТУРА

1. Басс, Л. Архитектура программного обеспечения на практике / Л. Басс, П. Клементс, Р. Кацман: пер. с англ. – Изд. 2-е. – СПб.: Питер, 2006. – 576 с.
2. ISO/IEC 42010:2007. Systems and software engineering – Recommended practice for architectural description of software-intensive systems. – https://www.iso.org/standard/45991.html (дата обращения 10.08.2021).
3. Software Engineering Institute. – https://www.sei.cmu.edu/ (дата обращения 10.08.2021).
4. The Concise Definition of The Zachman Framework by: John A. Zachman. – https://www.zachman.com/about-the-zachman-framework (дата обращения 12.08.2021).
5. Назаров, С.В. Оптимизация параллельных вычислений бортовых систем реального времени. Часть 1 / С.В. Назаров, А.Г. Барсуков // Электроника: Наука, Технология, Бизнес. – 2020. – № 10. – С. 110-116.
6. Хитт, Д. Стратегия Xilinx – быстрее двигаться к адаптируемому, интеллектуальному миру / Д. Хитт // Электроника: Наука, Технология, Бизнес. – 2018. – № 4. – С. 84-87.
7. Xilinx Ultrascale + Overview. – https://developer.ridgerun.com/wiki/index.php?title=Xilinx_Ultrascale%2B_Overview (дата обращения 12.08.2021).
8. The Reference Model of Open Distributed Processing (RM-ODP). – http://www.rm-odp.net/(дата обращения 12.08.2021).
9. IEEE 1471-2000 - IEEE Recommended Practice for Architectural Description for Software-Intensive Systems. – https://standards.ieee.org/standard/1471-2000.html (дата обращения 12.08.2021).
10. Назаров, С.В. Архитектура и проектирование программных систем: монография / С.В. Назаров. – М.: Инфра-М, 2020. – 413 с.
11. Дейкстра, Э. Дисциплина программирования / Э. Дейкстра. – М.: Мир, 1978. – 274 с.
12. Назаров, С.В. Структурный рефакторинг многослойных программных систем / С.В. Назаров, Н.Н. Вилкова // Информационные технологии и вычислительные системы. – 2016. – № 4. – С. 13-23.
13. Назаров, С.В. Эффективность и оптимизация компьютерных систем: монография / С.В. Назаров. – 2-е изд. – М.: РУСАЙНС, 2020. – 294 с.
14. Назаров, С.В. Оптимизация параллельных вычислений бортовых систем реального времени. Часть 2 / С.В. Назаров, А.Г. Барсуков // Электроника: Наука, Технология, Бизнес. – 2021. – № 1. – С. 130-137.
15. Назаров, С.В. Оптимизация параллельных вычислений в системах реального времени и пакетной обработки / С.В. Назаров // Norwegian Journal of the International Science. – 2020. – № 39. – P. 42-55.