ИПМ им.М.В.Келдыша РАН

Абрау-2017 • **Труды конференции**

А.И. Легалов, В.С. Васильев,
И.В. Матковский

Изменение стратегий управления вычислениями при архитектурно-независимом параллельном программировании ...

Рекомендуемая форма библиографической ссылки

Легалов А.И., Васильев В.С., Матковский И.В. Изменение стратегий управления вычислениями при архитектурно- независимом параллельном программировании ... // Научный сервис в сети Интернет: труды XIX Всероссийской научной конференции (18-23 сентября 2017 г., г. Новороссийск). — М.: ИПМ им. М.В.Келдыша, 2017. — С. 341-350. — URL: **http://keldysh.ru/abrau/2017/17.pdf** doi:**10.20948/abrau-2017-17**

Размещена также презентация к докладу

Изменение стратегий управления вычислениями при архитектурно-независимом параллельном программировании

А.И. Легалов[1], В.С. Васильев[1], И.В. Матковский[1]

1 Сибирский федеральный университет

Аннотация. Рассматриваются особенности формирования различных стратегий управления вычислениями, при использовании функционально-потоковой парадигмы, поддерживающей архитектурно-независимое параллельное программирование. В процессе трансляции, на основе классификации стратегий управления в вычислительных системах и языках программирования, предлагается разбиение функционально-потоковой параллельной программы на информационный и управляющий графы. Последующие варианты трансформации управляющего графа позволяют обеспечить различные подходы управления вычислениями, связанные с использованием особенностей архитектур вычислительных систем.

Ключевые слова: архитектурно-независимое параллельное программирование, функционально-потоковое параллельное программирование, стратегия управления вычислениями, преобразование программ, модель вычислений

1. Введение

Разнообразие архитектур параллельных вычислительных систем (ПВС) порождает методы написания параллельных программ, достаточно сильно отличающиеся друг от друга. Во многом это обусловлено тем, что языки программирования разрабатываются с учетом особенностей ПВС, что ведет к трудностям, связанным с переносимостью программного обеспечения. Появление универсальных языков параллельного программирования (ЯПП) мало способствует улучшению ситуации, так их концепции зачастую не учитывают особенности, влияющие на переносимость.

Каждый язык параллельного программирования базируется на определенной модели вычислений (МВ), задающей особенности параллелизма, определяющие способы порождения и взаимодействия процессов, выделение требуемых ресурсов. Для исследования методов организации параллельных вычислений в работе [1] были предложены классификации стратегий управления в ПВС и ЯПП. Она базируется на общности информационного графа, отражающего алгоритм решаемой задачи и отличиях, возникающих в способах запуска исполняемых операций. Выделены 27 чистых стратегий

управления в вычислительных системах и 8 стратегий управления, доступных при использовании языков программирования.

Отношения между данными и операциями определяются тем, что любая программа разрабатывается для выполнения конкретных вычислений. Они описываются информационным графом. Момент выполнения операции задается управляющим графом и механизмом продвижения разметки по данному графу. Корректное выполнение любой операции обработки данных возможно только в том случае, если в ходе предварительных вычислений будут сформированы значения всех ее операндов перед тем, как разметка управляющего графа передвинется в узел выполнения операции.

Зависимость от ресурсов вычислительных систем связана с тем, что они не являются безграничными. Поэтому, даже когда программа ориентирована на использование неограниченных ресурсов, приходится, тем или иным способом, накладывать соответствующие ограничения, обуславливаемые архитектурой вычислительной системы, что снижает параллелизм задачи и требует ввода дополнительных механизмов управления вычислениями.

Зависимость от субъективных управляющих воздействий показывает, что выполнение отдельных операций определяется моментом их запуска, который выбирается не только по объективным причинам (готовность данных и наличие вычислительных ресурсов). Зачастую программист руководствуется внутренним восприятием, обуславливаемым его пониманием разрабатываемого алгоритма, что, в свою очередь, изменяет параллелизм программы.

Для исключения субъективных управляющих воздействий, что позволяет уменьшить влияние программиста на процесс управления вычислениями, предложена концепция архитектурно-независимого параллельного программирования (АНПП), базирующаяся на функционально-потоковой модели параллельных вычислений [2]. На ее основе разработан язык функционально-потокового параллельного программирования Пифагор, в котором использовано только неявное управление по готовности данных. Программа на данном языке является информационным графом, отражающим только зависимости между функциями и обрабатываемыми ими данными. Заложенная в основу языка модель вычислений определяет выполнение операций по готовности данных.

Вместе с тем, следует отметить, что непосредственная реализация вычислителя с управлением по готовности данных в настоящее время вряд ли является целесообразной, так как не соответствует заявленной архитектурной независимости программ. Для поддержки этой независимости разработка программ осуществляется по следующей технологии. После создания функционально-потоковой параллельной программы, осуществляется ее трансляция, в ходе которой формируется промежуточное представление, являющееся реверсивным информационным графом (РИГ), отличающимся от изначального информационного графа обратной направленностью дуг, соединяющих операторы. Этот граф впоследствии используется при

выполнении программы для доступа к операндам, полученным в ходе вычислений. На основе РИГ создается дополнительное промежуточное представление, определяющее управляющий граф (УГ), который используется для запуска вершин РИГ, осуществляющих обработку данных [3, 4]. Изначально сформированный УГ реализует стратегию управления по готовности данных. Однако впоследствии эта стратегия может модифицироваться, что, тем самым, обеспечивает учет особенностей управления вычислениями в различных архитектурах уже на уровне формальных моделей. В работе рассматриваются подходы, демонстрирующие изменение стратегий управления вычислениями при неизменном информационном графе программы. Эти изменения осуществляются после трансляции программы в РИГ и проводятся за счет трансформации УГ.

2. Основные подходы к формированию управляющего графа

Основная идея в представлении функциональных процессов разрабатываемой программы заключается в описании отдельных функций [1] (рис. 1). Их корректное выполнение должно обеспечиваться предварительным поступлением информации, используемой для обработки данных ($\mathbf{I_{in}}$). Помимо этого, для реального выполнения любой функции, необходимо предоставление соответствующих вычислительных ресурсов ($\mathbf{R_j}$). Это должны быть как ресурсы процессора, используемые для выполнения функции, так и ресурсы памяти, предназначенные для хранения исходных данных и формируемых результатов ($\mathbf{I_{out}}$).

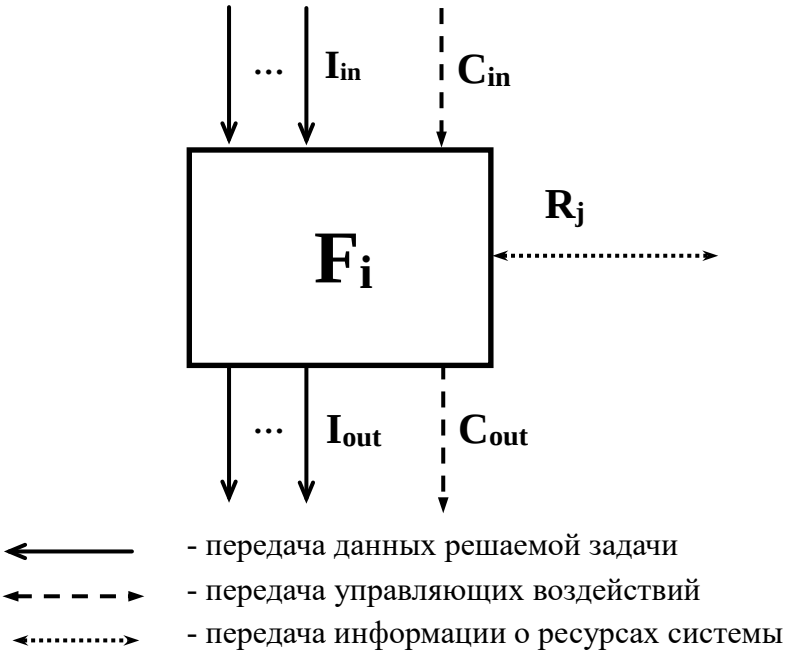

Рис. 1. Элементарный (функциональный) процесс

Взгляд на такую функцию с позиции процессов обеспечивает представление как данных **I**, так и предоставляемых вычислительных ресурсов **R** в виде соответствующих информационных потоков, поступающих в процесс

F и вытекающих из него. При этом для представления сведений об используемых ресурсах применяется двунаправленная связь, так как после проведения вычислений и становления неактуальными ранее полученных данных выделенные ресурсы возвращаются обратно вычислительной системе и могут быть использованы для выполнения других процессов.

Для запуска функции необходимо подать соответствующий управляющий сигнал $\mathbf{C_{in}}$, который активизирует обработку: $\mathbf{F_i(I_{in}) \rightarrow I_{out}}$ в ресурсах $\mathbf{R_j}$. Завершение вычислений (и, следовательно, появление результатов на выходе $\mathbf{I_{out}}$) определяется выдачей управляющего сигнала $\mathbf{C_{out}}$. Данный сигнал направляется либо напрямую к процессам, выполняющим обработку других функций, либо к специальным управляющим операторам, обеспечивающим согласованную поддержку взаимодействия множества процессов и управляющих операторов.

Взаимодействие множества элементарных процессов можно представить в виде сети с заданными отношениями между информацией, управляющими сигналами и ресурсами (**ICR**-сеть). Она может рассматриваться как совокупность трех подграфов определяющих: информационный граф, граф управления и граф ресурсов (рис. 2). Информационный граф задает связи между данными, обрабатываемыми в ходе вычислений. Он полностью определяет алгоритм решаемой задачи при заданном наборе операций обработки и не изменяется при любых способах организации управления вычислениями.

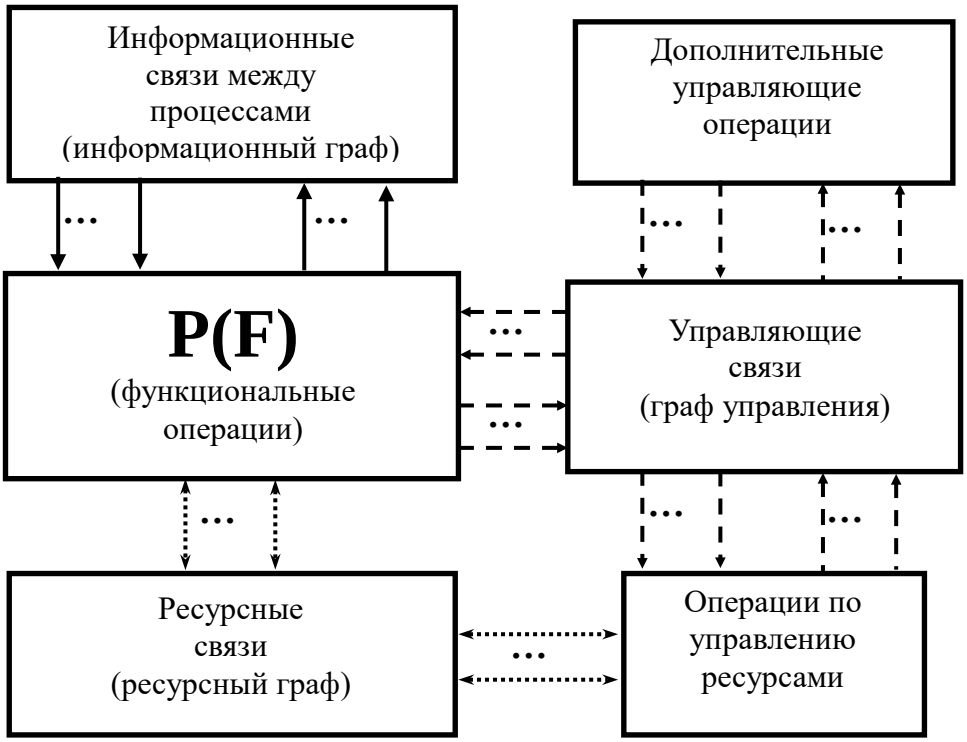

Рис.2. ICR-сеть, описывающая поведение процессов в ресурсах вычислительной системы

Ресурсный граф определяет предоставление процессам вычислительных ресурсов, без наличия которых обработка данных была бы невозможна. Алгоритмы предоставления ресурсов во многом определяются архитектурой вычислительной системы, а не только тем, как будет написана программа. В распоряжение программиста могут быть предоставлены операции по управлению ресурсами, что ведет к расширению графа дополнительными ресурсными операциями. Управляющий граф задает передачу сигналов управления между процессами. Эти сигналы формируются по возникновению в сети различных событий, определяющих изменения в состоянии общего вычислительного процесса. Часть этих событий может быть связана с механизмом продвижения фишек между элементарными процессами. Кроме этого сигналы управления могут порождаться и использоваться подсистемой управления ресурсами. Для придания гибкости могут быть введены дополнительные операции управления, изменяющие порядок проведения вычислений, но сохраняющие при этом корректность информационных преобразований. Именно управляющий граф определяет порядок выполнения операций и корректность проводимых вычислений.

3. Стратегии управления вычислениями

Вычислительный процесс можно рассматривать как совокупность элементарных процессов, каждый из которых обеспечивает выполнение простой операции. Для корректного выполнения необходимо, чтобы к моменту запуска элементарного процесса соблюдались условия: готовности данных, выделения ресурсов, подтверждения использования результатов.

Условие готовности данных или **I-условие** (Information) сигнализирует, что перед запуском процесса на его информационных входах присутствуют все необходимые аргументы и функции, составляющие операционный пакет. Отсутствие каких-либо данных к моменту запуска процесса ведет к получению неправильного результата. То есть, если существуют процессы, результаты которых являются аргументами рассматриваемого процесса, то они должны завершиться к моменту запуска текущего процесса.

Условие выделения ресурсов или **R-условие** (Resource) показывает, что выполнение процесса протекает в соответствующих ресурсах. Поэтому, перед его запуском, эти ресурсы должны быть определены и предоставлены. Ресурсный вход должен содержать информацию, подтверждающую выделение ресурсов, необходимых для успешного выполнения процесса. В противном случае его запуск просто невозможен.

Условия подтверждения использования результатов или **A-условие** (Acknowledge) определяет, что ресурсы, занимаемые процессом, могут освобождаться или повторно использоваться только после того, как результаты вычислений будут использованы всеми процессами, являющихся их приемниками в качестве аргументов. В противном случае данные будут потеряны за счет наложения новых результатов, что приведет к неправильным

вычислениям. Необходимость задержки в освобождении ресурсов мотивируется тем, что память, используемая для хранения операндов, является разделяемым ресурсом. В нее записываются результаты выполнения элементарного процесса. Из нее же происходит чтение аргументов, необходимых другим процессам.

Информация о выполнении условий готовности подтверждается соответствующими управляющими сигналами:

(1) сигналами готовности данных, необходимых для обработки;
(2) сигналами готовности необходимых вычислительных ресурсов;
(3) сигналами, подтверждающими отсутствие конфликтов при использовании разделяемых ресурсов.

Эти входные сигналы формируются на основе выходных сигналов завершающихся процессов. Завершение элементарного вычислительного процесса одновременно подтверждает готовность результатов и использование им, в качестве своих аргументов, результатов, полученных ранее в процессах – «передатчиках». Естественно, что при более сложном представлении процессов, схема их взаимодействия и моменты формирования управляющих сигналов тоже будут сложнее.

Моменты выдачи управляющих сигналов и сами принципы их формирования, определяющие наступление соответствующих событий, могут осуществляться одним из следующих способов:

– Явно, когда при описании взаимодействия процессов (в ходе написания программы) разработчик сам создает логику порождения и проверки управляющих сигналов, определяет пути и моменты их передачи. Назовем такое формирование управления субъективным (Subjective) или **S-управлением**.

– Неявно, когда, при описании процессов, механизм управления тем или иным условием готовности просто не задается из расчета, что процессы и так будут выполняться корректно.

Корректность неявного управления процессом может обуславливаться двумя причинами. Первая определяется тем, что ресурсы ВС организованы таким образом, что всегда поддерживают выполнение заданного условия. Назовем данный принцип формирования управления автоматическим (Automatic) или **A-управлением**. Вторая причина связана со спецификой описания взаимодействия процессов, которое может быть сделано таким образом, что, несмотря на отсутствие в ВС механизма автоматической поддержки, условие готовности будет всегда истинным, поэтому его не нужно проверять. Этот принцип управления назовем пустым (Empty) или **E-управлением**. Он опирается на дополнительные сведения о работе системы.

Каждое из трех условий готовности процесса к выполнению операции может быть реализовано с использованием одного из трех механизмов формирования момента выдачи управляющего сигнала. Это позволяет выделить двадцать семь основных стратегий управления вычислительными

системами, каждая из которых задается вектором **S** мощностью $|\mathbf{S}| = \mathbf{X}^{\mathbf{Y}}$, определяющим три различных способа задания условий готовности данных, ресурсов и подтверждений. Здесь:

– **S** – множество стратегий управления в вычислительных системах;

– **X** – трехэлементный вектор условий готовности:

$$\mathbf{X = (I, R, A)};$$

– **Y** – трехэлементное множество методов управления каждым из условий:

$$\mathbf{Y = \{S\text{-управление, A-управление, E-управление}\}}.$$

Пусть:

– $\mathbf{S_1}$**={Information×Y}** – множество методов управления данными;

– $\mathbf{S_2}$**={Resource×Y}** – множество методов управления ресурсами;

– $\mathbf{S_3}$**={Acknowledge×Y}** – множество методов управления подтверждениями.

Тогда множество стратегий управления определяется как множество троек, образуемых прямым произведением:

$$\mathbf{S = \{S_1 \times S_2 \times S_3\}}.$$

В реальных архитектурах, на уровне различных ресурсов одновременно могут быть использованы различные стратегии управления. Тогда их общее число в одной ПВС может превышать единицу.

Особенностью языков программирования является то, что неявное управление в них с помощью специальных средств не задается. Следовательно, при написании программ автоматическое и пустое управление не различаются. Тогда, для трех условий готовности существуют только два механизма управления: субъективный и неявный (Unevident) или **U-управление**. Поэтому, в языках программирования можно описать только восемь основных стратегий управления вычислениями:

$$|\mathbf{S_L}| = \mathbf{X}^{\mathbf{Z}},$$

где $\mathbf{S_L}$ – множество стратегий управления в языках программирования, **X** – трехэлементное множество условий готовности, **Z** – двухэлементное множество методов управления процессом:

$$\mathbf{Z = \{S\text{-управление, U-управление}\}}.$$

4. Инструментальная поддержка изменения стратегий управления вычислениями при функционально-потоковом параллельном программировании

Для реализации подхода, обеспечивающего гибкое изменение стратегий управления вычислениями при сохранении неизменным информационного графа программы, были разработаны соответствующие инструментальные средства [3, 5]:

– транслятор, обеспечивающий преобразование исходных текстов программ, написанных на языке Пифагор в реверсивный информационный граф;

– генератор управляющего графа, описывающий механизм передачи управления между своими узлами и запуск вершин РИГ, обеспечивающих непосредственное выполнение вычислений;

– компоновщик, собирающий исполняемую программу из множества оттранслированных функций, участвующих в организации вычислений;

– интерпретатор, обеспечивающий выполнение функционально-потоковых параллельных программ.

Представленный набор инструментов позволяет сформировать полный цикл по трансляции и выполнению функционально-потоковых параллельных программ. Вместе с тем, для решения более сложных задач, включая и преобразование данных программ в программы, выполняемые на архитектурах реальных параллельных вычислительных системах, необходимы дополнительные средства, обеспечивающие такие преобразования. В частности наряду с различными методами оптимизации информационного графа программы, соответствующие преобразования проводятся и над формируемым управляющим графом программ, что позволяет изменять стратегии управления вычислениями. Механизмы подобного преобразования можно рассмотреть на примере реализации функции «Исключающие ИЛИ» с использованием операций булевой логики «И», «ИЛИ», «НЕ». Простота примера в данном случае объясняется размером рисунков, демонстрирующих особенности подхода. На языке Пифагор реализация данной функции будет соответствовать следующему выражению над булевыми типами:

(x1:-, x2):*,(x1,x2:-)):+

Данное выражение определяет РИГ, представленный на рис. 3а.

Изначальное построение УГ осуществляется специальной утилитой с использованием принципа управления по готовности данных (рис. 3б). Управление вычислительными ресурсами не задается, так как интерпретатор реализует их автоматическое распределение. Обратная связь между вершинами РИГ позволяет при их запуске напрямую обращаться к результатам функций, являющихся источниками данных. После выполнения преобразований, выполненных в этих вершинах, управление от вершины УГ передается следующим его вершинам.

В ходе дальнейшего анализа и преобразований можно получать разнообразные стратегии вычислений путем трансформации управляющего графа. В частности, переход к последовательному явному управлению заключается в отказе от дополнительных управляющих вершин и соединении узлов информационного графа в последовательности, не нарушающей корректность вычислений. Пример наложения такого УГ представлен на рис. 3в.

Отсутствие управления возможно в тех случаях, когда для преобразований данных достаточно только их продвижения по узлам РИГ. В этом случае фиксация результата в выходной памяти может считаться правильной при их записи в любой момент времени и может определяться

внешними факторами, не зависящими от преобразований, выполняемых в текущей функции. Пример отсутствующего управления представлен на рис. 3г и может быть реализован, например, с применением комбинационных схем, преобразование в которые из исходного текста функционально-потоковой параллельной программы реализуется с применением соответствующих утилит.

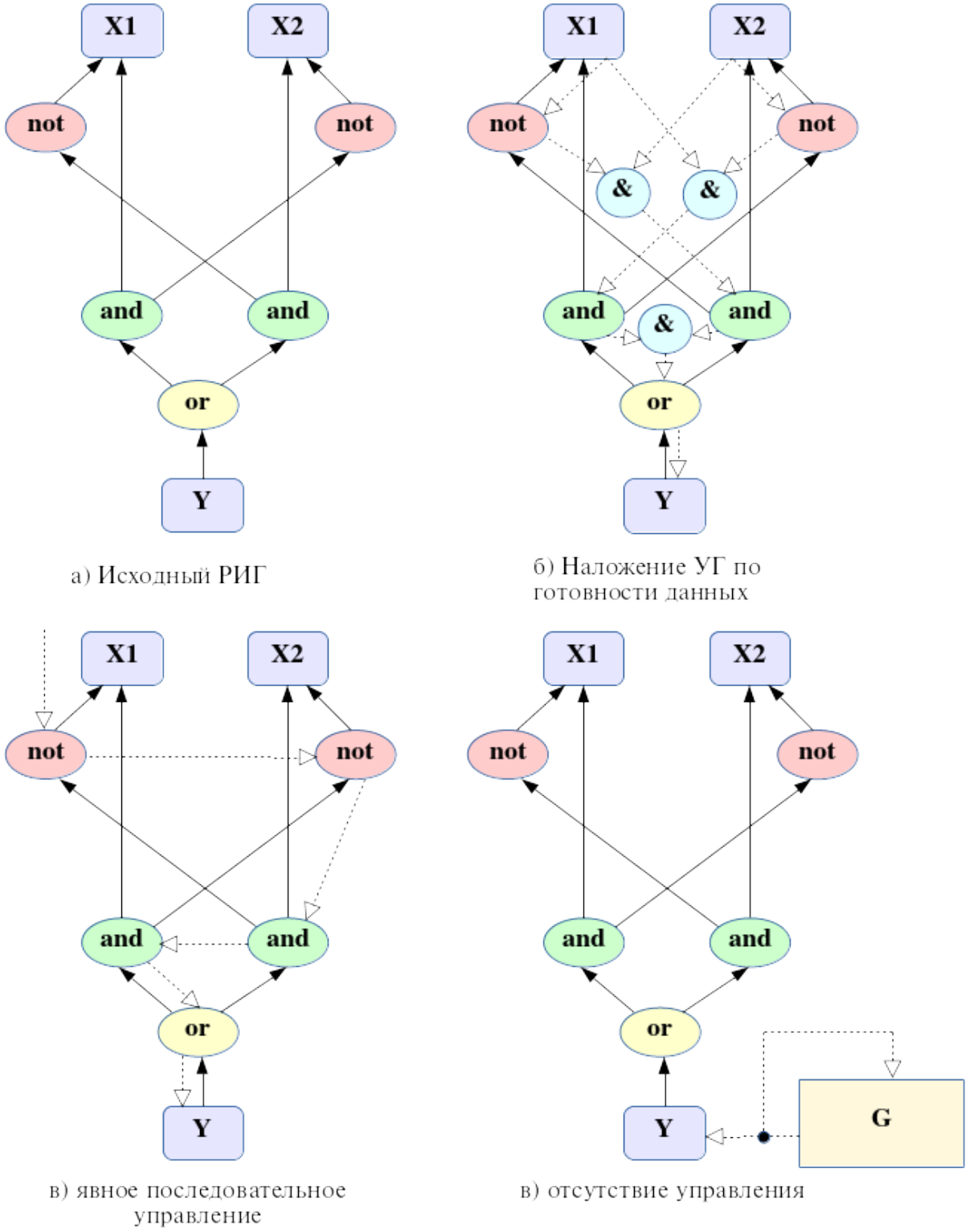

Рис. 3. Варианты управления вычислениями

4. Заключение

Представленная в работе концепция стратегий управления вычислениями позволяет отделить информационный граф программы от управления вычислениями. Возможно использование различных стратегий управления путем наложения соответствующих информационных графов. Разработанные для языка программирования Пифагор инструментальные средства позволяют на основе исходной программы, фактически являющейся информационным графом, выстраивать различные стратегии управления вычислениями, что обеспечивает поддержку архитектурно-независимого параллельного программирования.

Исследование выполняется при финансовой поддержке РФФИ в рамках научного проекта № 17-07-00288.

Литература

1. Легалов А.И. Об управлении вычислениями в параллельных системах и языках программирования // Научный вестник НГТУ. – 2004. – № 3 (18). – С. 63–72.
2. Легалов А.И. Функциональный язык для создания архитектурно-независимых параллельных программ // Вычислительные технологии. – 2005. – № 1 (10). – С. 71–89.
3. Матковский И.В., Легалов А.И. Инструментальная поддержка трансляции и выполнения функционально-потоковых параллельных программ // Ползуновский вестник. – 2013 – № 2. – С. 49–52.
4. Легалов А.И., Савченко Г.В., Васильев В.С. Событийная модель вычислений, поддерживающая выполнение функционально-потоковых параллельных программ // Системы. Методы. Технологии. – 2012. – № 1 (13). – С. 113–119.
5. Легалов А.И., Матковский И.В., Кропачева М.С., Удалова Ю.В., Васильев В.М. Технологические аспекты создания, преобразования и выполнения функционально–потоковых параллельных программ // Научный сервис в сети Интернет: труды Международной суперкомпьютерной конференции (23–28 сентября 2013 г., г. Новороссийск). – М.: Изд-во МГУ. – 2013. – С. 443–447.