Об отладке и верификации функционально-потоковых параллельных программ

Ю.В. Удалова, А.И. Легалов, Н.Ю. Сиротинина, М.С. Кропачева

Исследуются методы отладки и верификации программ, написанных на функционально-потоковом параллельном языке. Использование данного языка позволяет не учитывать распределение ресурсов. Это, на наш взгляд, позволяет первоначально акцентировать внимание на логике функционирования, обеспечивая в дальнейшем перенос на целевые архитектуры правильно работающих программ. Применение функционально-потоковых языков вносит свои особенности в отладку и верификацию, что ведет к разработке новых методов и инструментальных средств, рассматриваемых в работе.

1. Введение

Разработку программного обеспечения практически невозможно проводить без отладки. В настоящее время этот этап хорошо проработан для последовательного программирования. Существуют разнообразные отладчики, обеспечивающие пошаговый анализ кода и отображающие состояние переменных [1]. Они позволяют использовать разнообразные приемы, например: пропуск уже отлаженных фрагментов программ, отладку только требуемых функций или процедур, удобный просмотр уже отлаженного кода.

При переходе к параллельным вычислениям процесс отладки сопровождается дополнительными трудностями, обуславливаемыми спецификой взаимодействия процессов. В частности, необходимы дополнительные функции, позволяющие анализировать конфликтные и тупиковые ситуации. В настоящее время методы отладки параллельных программ развиваются. Достигнуты определенные успехи в разработке соответствующих средств, для систем, использующих потоки или процессы, расширяющие технику последовательного выполнения программ [1].

Верификация позволяет проводить проверку правильности программы на более формальном уровне, чем отладка. Отладка программы предполагает исследование некоторого ограниченного множества путей вычислений, тогда как задача верификации заключается в анализе свойств самой программы по ее тексту. Методы верификации программы делятся на методы доказательства теорем и методы проверки моделей [2-4]. Программа считается корректной, относительно некоторой спецификации. Методы доказательства теорем используют последовательность операторов программы, некоторое множество аксиом и определенную систему вывода для формирования дополнительных утверждений. Программа считается корректной, если из полученных утверждений, следуют утверждения спецификации.

Методы проверки моделей строят для программы модель с конечным числом состояний, проверка истинности формул осуществляется в основном посредством переборных алгоритмов. Методы проверки моделей широко используются на практике, в отличие от методов доказательства теорем, что обусловлено сложной реализацией последних. Примерами реализаций методов проверки моделей, в том числе и для многопроцессных параллельных программ, могут служить верификаторы Spin и Bogor.

Существуют разнообразные способы построения параллельных программ, часть которых существенно отличается от широко используемых в настоящее время. Исследование нетрадиционных методов параллельного программирования может привести к появлению перспективных технологий, повышающих эффективность процессов разработки параллельных программ. К таким способам построения следует отнести и функционально-потоковое параллельное программирование, обеспечивающее описание ресурсно-независимого параллелизма [5-6]. На основе этой модели разработан язык программирования Пифагор [7], позволяющий писать архитектурно-независимые параллельные программы. Методы отладки и верификации функционально-потоковых параллельных программ позволяют использовать ряд специфических приемов, определяемых особенностью модели вычислений, в которой отсутствует понятие пере-

менной. Выполняемые функции связаны напрямую, а существующие данных имеют динамическую типизацию.

Целью работы является исследование методов отладки и верификации функционально-потоковых параллельных программ. Предлагаются три режима отладки, использующие различные способы отображения информационного графа параллельной программы. Рассматривается применение методов доказательства теорем при верификации функционально-потоковых параллельных программ.

2. Режимы проведения отладки

Модель вычислений определяет функционально-потоковую параллельную программу как информационный граф, в котором вершины являются операторами, а дуги определяют связи между ними. Параллельное выполнение основано на том, что одновременно выполняться могут только те операторы, между которыми нет информационных связей. Чтобы оператор мог быть вычислен, необходимо выполнение всех тех предшественников, которые связаны с ним. Отладка функционально-потоковой параллельной программы может проходить в одном из трех режимов: пошаговой отладки, отладки слоев и отладки ветвей.

Режим пошаговой отладки потоковой параллельной программы подобен последовательному режиму отладки. Возможность его использования обусловлена спецификой языка программирования. Хотя при непосредственном вычислении программы может быть использован другой путь обхода графа, чем при пошаговой отладки, эта ситуация не приведет к возникновению конфликтов, таких, как появление невычисленных вершин, бесконечное ожидание вычисления предыдущего результата, возникновение некорректных данных. Например, если выполняется множество независимых операторов, то неважно, в каком порядке они были вычислены. Процесс отладки все равно показывает верный результат для каждого оператора и позволяет выявить ошибку. Если последовательность выполнения операторов идет в глубину графа, каждая отдельная ветвь может быть выполнена отладчиком только в определенном порядке, и различные ветви, не обладающие общим путем, не могут содержать взаимосвязанные операторы. Благодаря этому, переход пошагового отладчика от выполнения одной ветви к другой не приводит к противоречиям с непосредственным выполнением программы, на каком бы уровне вложенности функций он не произошел.

Режим отладки слоев (ярусов) предполагает, что на первом шаге отладки выполняется первый слой, т.е. все те операторы, которым требуются только входные параметры и константы (рис. 1). После того, как эти операторы окажутся вычисленными, в графе программы соответственные им вершины помечаются как выполненные, и на следующем шаге для вычисления выбираются все те операторы-вершины, которые имеют на своем входе вычисленные данные, таким образом формируется следующий слой.

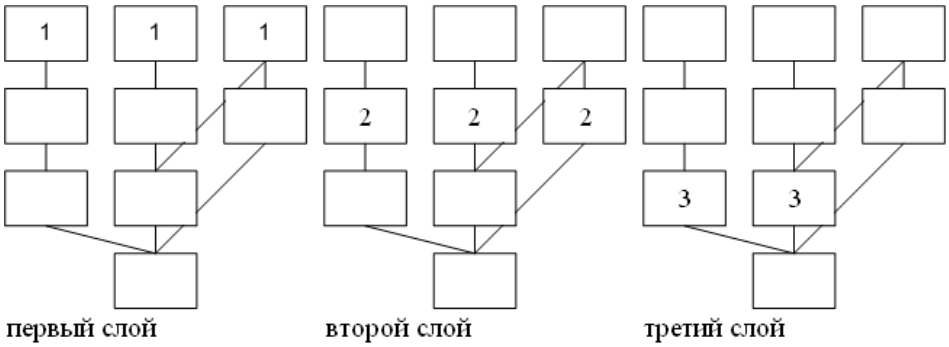

Рис. 1. Пример послойного выполнения операторов функционально-потоковой параллельной программы

В режиме отладки слоев за один шаг отладки выполняются несколько операторов, и здесь возникает вопрос о поведении системы при возникновении ошибки на одном или нескольких операторах. Такая ситуация не является критической вследствие того, что операторы одного слоя независимы между собой. Но группа операторов в следующем слое получает ошибку как

входной параметр. Если в функционально-потоковой программе некоторый программный оператор получает на входе ошибку, то он все равно производит над ней необходимые действия. Чаще всего в этом случае формируются выходные данные, также содержащие ошибку, и такие выходные данные обрабатываются последующими операторами. В процессе отладки можно рассмотреть все последствия распространения возникшей ошибки.

Отладка по слоям предоставляет пользователю возможность наглядно увидеть количество слоев программы и количество операторов в каждом слое, т.е. фактически оценить уровень параллелизма написанной программы. Кроме того, такая схема выполнения программы при отладке, наиболее точно показывает работу модели вычислений функционально-параллельных программ.

Описанный режим отладки обладает следующей спецификой. Если среди совокупности операторов существуют вызовы функций, то такие вызовы выполнятся сразу в полном объеме, и лишь по завершении вычисления всего слоя будет произведена отладка вызванной функции. Такая схема работы, с одной стороны, не разрушает целостности восприятия слоев, но, с другой стороны, затрудняет, поиск ошибки при наличии бесконечного рекурсивного вызова функции.

Режим отладки ветвей (рис. 2) предполагает, что для выполнения на каждом шаге отладки выбираются все операторы независимой ветви информационного графа. Такие операторы всегда выполняются в строгой последовательности, заданной графом программы. Этот режим позволяет четко выделять в параллельной программе цепочку операторов, которые могут быть выполнены только последовательно один за другим, но при этом не нуждаются в получении каких либо других входных значений из других вершин. При этом цепочка операторов–вершин, выполняемая на отдельном шаге отладки, является логически связанной, все операторы, стоящие в цепи служат одной и той же цели вычисления. Подобная схема отладки не слишком хорошо согласуется с моделью вычислений потоковых параллельных программ, однако здесь не возникает несоответствия между данными, полученными на шагах отладки, и данными, вычисленными при выполнении программы. Режим отладки ветвей выполняет односвязную последовательность команд за один шаг, что в определенных случаях может сократить время проверки программы.

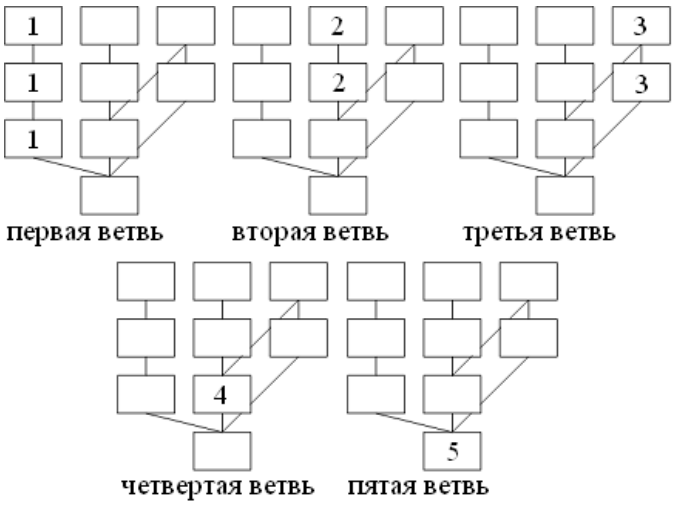

Рис. 2. Использование ветвей при отладке операторов функционально-потоковой параллельной программы

3. Инструментальная поддержка методов отладки

Описанные режимы отладки реализованы в среде разработки программ на функционально-потоковом параллельном языке Пифагор. Каждый из режимов обеспечивает использование графического и текстового представления программы.

Графическое представление программы задается в виде информационных графов всех ее функций. Перед началом вычисления любой из функций строиться ее информационный граф. При выполнении оператора внутри функции осуществляется разметка соответствующей ему

вершины графа. Каждая вершина содержит доступную пользователю информацию об операторе и результате его вычисления.

Текстовое представление процесса отладки оперирует списком функций или сверткой функций. Список функций показывает историю шагов отладки, сохраняя текст выполняемой функции на каждом шаге. Свертка функции отображает этот же процесс более компактно, запоминая только предыдущее и вычисленное состояние функции.

Перед отладкой программы, вызывается диалог выбора режима отладки (рис. 3), в котором разработчик определяет режим отладки, способ отображения процесса отладки и ряд других опций. «Функция для запуска» - это функция, которая будет запущена для отладки, она выбирается из списка, который формируется автоматически. «Аргумент» - входной параметр для запускаемой функции, он задается пользователем. Функции, не имеющие входного параметра, помечаются в списке функций восклицательными знаками.

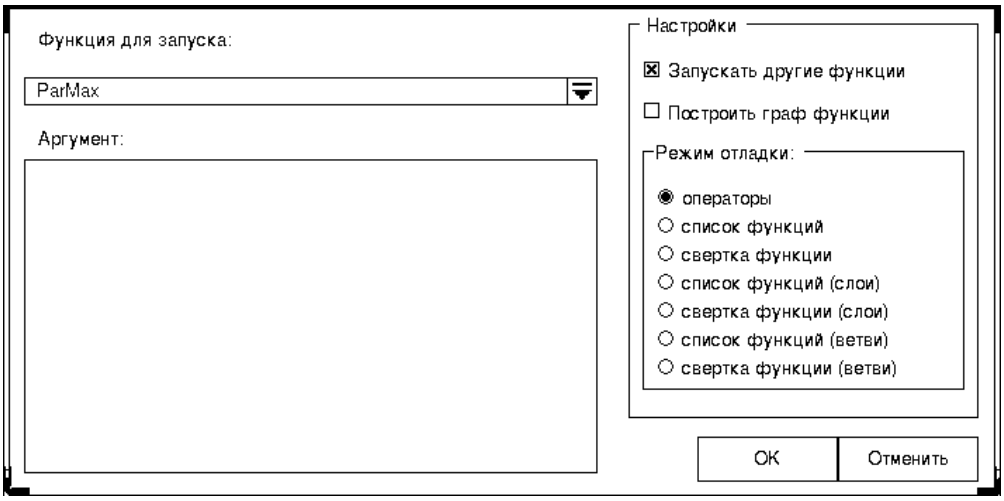

Рис. 3. Диалог выбора режима отладки

Если флаг «Запускать другие функции» отключен, то другие функции не будут запускаться для отладки, это дает возможность сосредоточиться на выполнении выбранной функции, не переключая внимание на просмотр ненужных вычислений. Если флаг включен, отладка проходит полный путь, начиная с запуска, выбранной пользователем функции. Флаг «Построить граф функции» позволяет по желанию пользователя построить граф функции. Тогда во время отладки программы будет доступен как этот граф, так и текстовое отображение процесса отладки.

Режимы «операторы», «список функции», «свертка функции» - это различные способы отображения пошаговой отладки, «список функции (слои)», «свертка функции (слои)» - различные способы отображения отладки слоев. И соответственно «список функции (ветви)», «свертка функции (ветви)» - это различные способы отображения отладки ветвей.

Режим пошаговой отладки имеет один дополнительный способ текстового отображения – «операторы» по сравнению с режимами отладки ветвей и отладки слоев. Данный способ подобен большинству стандартных способов представления процесса отладки для последовательных программ. Существует текстовое поле, содержащее всю программу, по которому перемещается цветовой указатель, показывающий, какой оператор выполняется на данном шаге. Также в главном окне располагается дополнительное текстовое поле, отображающее список всех вычисленных операторов, в виде:

1. Исходной строки оператора;
2. Строки оператора после подстановки в нее всех значений, вычисленных на предыдущих шагах;
3. Вычисленного значения оператора.

Представленный способ отображения результатов отладки отличается от методов визуализации, применяемых в последовательных программах. Если для последовательной программы операторы выполняются строго в том порядке, в каком они записаны в тексте программы, то

для параллельной программы такая ситуация, скорее всего, является исключением. В функционально-потоковых языках последовательность вычисления операторов определяется не их расположением в тексте программы, а информационными зависимостями между функциями, определяемыми при построении информационного графа. Таким образом, оператор, расположенный в некоторой строке исходного текста программы, может быть вычислен раньше операторов, стоящих в предшествующих строках. Подобная ситуация не противоречит модели вычислений потоковых параллельных программ, но является не лучшей для восприятия процесса отладки пользователем. В связи с этим были разработаны такие способы текстового представления отладки как создание списка функций и свертка функций, применимые ко всем трем описанным режимам отладки.

Создание списка функций предполагает, что на первом шаге отладки текстовое поле дополняется телом функции, запущенной для отладки, в котором все операторы, вычисляемые на данном шаге, выделяются некоторым цветом. После получения вычисленных значений в поле текста снова дублируется тело функции, но уже с подставленными в него полученными на этом шаге отладки значениями, которые выделены другим цветом. При переходе к следующему шагу отладки поле текста снова дополняется функцией, в которую подставляются вычисленные значения, причем функция содержит и все предыдущие полученные значения. Таким образом, текстовое поле содержит список функций, где каждая последующая функция содержит те операторы, которые не были выполнены на предыдущих шагах, и также содержит все вычисленные данные.

Свертка функции отличается от списка функций только формой отображения информации, получаемой при выполнении шагов отладки. Если была выбрана свертка функции, то на экране отображается исходное тело функции, которое не меняется при отладке, а используется для выделения тех операторов, которые выполняются на данном этапе отладки. Кроме этого, отображается и текущее состояние функции, в котором записаны все предыдущие вычисления, и также добавлены данные, полученные на текущем шаге, последние выделяются цветом. Таким образом, начиная с некоторого шага отладки, исследуемая функция начинает «свертываться». На момент возврата результата вычислений функции, тело функции содержит только записи вида: значение >> идентификатор. Свертка функции отображает процесс отладки более компактно и наглядно, чем список функций, но при этом не позволяет просмотреть историю всех изменений и переходов.

Графическое отображение процесса отладки выполнено посредством визуализации программного графа отлаживаемой функции. По желанию разработчика оно может использоваться совместно с текстовым представлением. В программном графе еще не вычисленные вершины выделяются другим цветом, чем вычисленные. По нажатию на любую вершину графа появляется информация об исходной строке соответствующего этой вершине оператора и вычисленном значении оператора, если оно существует. Каждая вершина отображает тип соответственного ей оператора, посредством специального символа. Граф функции всегда располагает операторы-вершины одного слоя на одном уровне, независимо от того, какой режим отладки был выбран.

4. Верификация функционально-потоковых параллельных программ

Функционально-потоковые параллельные программы (ФПП программы) более легко верифицируемы, чем многопроцессные параллельные программы. Это обусловлено моделью ФПП языка. Вычисления по готовности данных, возможность выполнения операторов, имеющих на входе пустой список или список ошибок, позволяют при верификации ФПП программ обращать гораздо меньше внимания на такие качества программы как завершимость и наличие тупиков. Отсутствие явного выделения процессов или потоков исключает ошибки в синхронизации параллельных участков программы, а также возможности бесконечного ожидания получения данных одного процесса другим или остановки определенной части процессов, до завершения их выполнения. Но благодаря специфичной модели вычислений, нельзя говорить о том, что верификация ФПП программы полностью аналогична верификации последовательной программы.

Верификация методами доказательства теорем ФПП программы базируется на методе индуктивных утверждений. Метод индуктивных утверждений [2-3] проводит доказательство правильности программы, относительно ее спецификации. При этом спецификация должна содержать входное и выходное утверждения: входное утверждение описывает все необходимые входные условия для программы и играет роль ограничителя входных данных, для которых проверяется правильность программы, выходное утверждение описывает ожидаемый результат вычислений, для заданного множества входных данных. Входное утверждение предполагается истинным. Затем строятся дополнительные утверждения, которые выводятся на основании семантики последовательности операторов, расположенных между входным и выходным утверждениями. Программа соответствует цели, если полученные утверждения удовлетворяют выходному утверждению.

Метод индуктивных утверждений – базовый метод для верификации как последовательных, так и многих параллельных программ. Для использования этого метода необходимо знать последовательность выполнения программы. При верификации последовательных программ структура программы может быть построена прямо по тексту программы, для описания циклов и рекурсий подключаются дополнительные методы, например, метод счетчиков или добавление инварианта [5, 6]. При верификации параллельных программ, опирающихся на работу процессов, структура каждого отдельного процесса также может быть построена прямо по его программному коду. Для анализа взаимодействия процессов используются дополнительные методы, например метод невзаимодействующих доказательств или метод выделения коммуникационно-замкнутых слоев [7].

В качестве структуры функционально-потоковой параллельной программы для метода индуктивных утверждений можно использовать граф ФПП программы. Такой граф является ациклическим и однозначно определяет операторы-вершины и связи между ними, он хорошо подходит для разбора программных операторов и вывода утверждений. Так как параллельно могут выполняться только те операторы-вершины, между которыми нет пути, граф может быть преобразован в последовательность операторов, т.е. функционально-потоковая параллельная программа может быть представлена как последовательная программа без ущерба для цели и логики вычислений. Но, используя метод индуктивных утверждений для верификации функциональной параллельной программы, выгодно пользоваться именно графом программы, а не выводимой из него последовательность действий. Это обусловлено тем, что метод индуктивных утверждений оперирует большим множеством предикатов, как заданных пользователем, так и выводимых из программных операторов. А граф позволяет сужать множество утверждений необходимых для рассмотрения при проверке конкретного оператора, выделяя только связанные между собой вершины.

Верификация методами проверки моделей прежде всего решает задачу построения модели по тексту программы. Полученная модель должна иметь конечное число состояний и соответствовать самой программе, т.е. она должна правильно отражать программные вычисления, и не отображать несуществующих в программе вычислений. Если модель построена неправильно, то и результат метода проверки моделей недействителен. Граф ФПП программы может служить ее моделью в методах проверки моделей, граф представляет модель функционально-потоковых параллельных вычислений и поэтому, не может возникнуть противоречий между ФПП программой и ее моделью. Но при этом не каждая ФПП программа, может быть представлена конечным графом.

5. Спецификация функционально-потоковых параллельных программ

Спецификация ФПП программы должна содержать цель вычислений и может содержать начальные и промежуточные утверждения. Цель и входные утверждения удобнее всего приписывать к некоторой отдельной функции. Для записи промежуточных утверждений следует использовать граф ФПП программы, таким образом, чтобы каждое промежуточное утверждение было приписано к определенной дуге или вершине графа (Рис. 4).

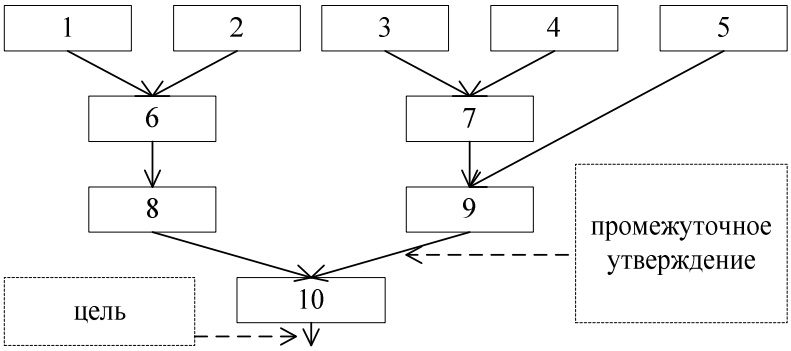

Рис. 4. Спецификация на графе функционально-потоковой параллельной программы

Для определения цели и начального условия функции применение графа не требуется. Цель – это утверждения, задающие набор требований к возвращаемому функцией значению. Если функция является рекурсивной, то цель должна выполняться для значения, получаемого после завершающей итерации рекурсии. Начальное или входное утверждение описывает свойства аргумента функции, для обозначения аргумента в спецификации можно использовать его имя в программе. Начальное утверждение считается истинным и используется при проверке любого оператора функции, использующего аргумент. К функции может быть добавлено произвольное количество промежуточных утверждений. Промежуточное утверждение должно быть истинным после выполнения всех операторов, ведущих к той дуге графа, к которой это утверждение приписано. Для рекурсивной функции промежуточные утверждения, в отличие от цели, проверяются на каждой итерации.

Для записи спецификации можно применять как специальный язык, так и сам ФПП язык программирования. Отдельный язык спецификации позволяет сокращать запись многих свойств, также используемый язык можно реализовать наиболее близко к правилам записи логических выражений. При проверке программы на соответствие спецификации, заданной на специальном языке, необходимо преобразовывать результаты работы программных операторов в выражения используемого языка. Если спецификация программы записана в терминах языка программирования, возникает обратная задача – преобразовать совокупность программных операторов в исследуемые свойства алгоритма.

Использование самого языка программирования для спецификации обладает существенным плюсом: появляется возможность использовать спецификацию при выполнении программы. При спецификации программы, начальное и конечное условия для функции, можно формировать таким образом, чтобы при подстановке конкретных данных, возвращать либо истину, либо ложь. Это упорядочит запись требований и позволит, по желанию пользователя, автоматически создавать копию программы, учитывающую заданные условия – программу, включающую в себя аннотации пользователя. Если есть функция с заданными начальными и конечными условиями, средствами функционально-потокового параллельного языка можно создать дубликат этой функции, вычисляющий условия спецификации и на основе этих вычислений вызывающий или не вызывающий первоначальную функцию (Рис. 5).

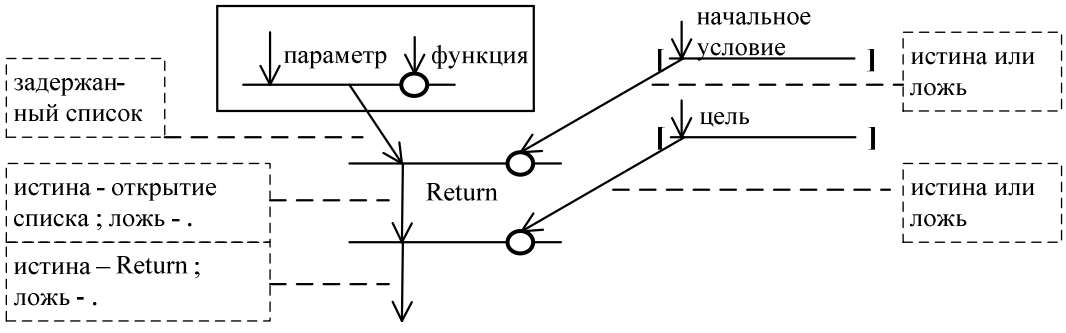

Рис. 5. Граф функции, совмещенной со спецификацией

Заключение

Описанные режимы отладки и способы ее отображения реализованы в рамках разработанной среды и могут быть использованы при разработке функционально-потоковых параллельных программ. Эффективность каждого отдельного режима зависит от свойств исполняемой программы. Так, простая функция может содержать только одну ветвь, и это значит, что при выборе отладки ветвей все вычисления будут выполнены за один шаг, тогда как отладка по слоям, в этом же случае, будет полностью аналогична пооператорной отладке. Плюсы и минусы описанных режимов зависят и от восприятия процесса отладки пользователем. Но, в общем случае, пооператорная отладка позволяет сосредоточиться на каждом программном операторе, а отладка ветвей выполняет прямые последовательности команд за один шаг, что может сократить время проверки программы, отладка слоев дает представление об уровне параллелизма рассматриваемой функции.

Различные варианты текстовой визуализации процесса отладки предоставляют такие возможности, как сохранение истории вычислений и вызовов функций, выделение текущих результатов отладки. Графическое представление отладки позволяет увидеть историю вычисления отдельной функции и эффективно проследить все последствия возникновения ошибки на графе, наглядно выделяет ветви и слои, показывает уровень параллелизма функции.

Верификация функционально-потоковых параллельных программ может проводиться с использованием ациклического графа ФПП программы. Наличие графа позволяет сужать множество утверждений, необходимых для осуществления логического вывода в методах доказательства теорем и предоставляет модель программы в методах проверки модели.

Благодаря модели вычислений ФПП языка, исключается часть проблем верификации многопроцессорных параллельных программ, связанная с синхронизацией и взаимодействием процессов, пересылкой данных от одного процесса к другому, использованием общих ресурсов. Исследование методов отладки и верификации функционально-потоковых параллельных программ позволило выделить ряд основных направлений развития инструментальных средств, повышающих эффективность процесса разработки программ. Реализация методов отладки в рамках экспериментальной среды позволяет повысить эффективность разработки функционально-потоковых параллельных программ.

Литература

1. **Калинов А.Я., Карганов К.А., Хоренко К.В.** Команда «шаг» в параллельных отладчиках // Труды Института системного программирования РАН Вып 5, 2004. - С. 89-101.
2. **Лисков Б., Гатэг Дж.** Использование абстракций и спецификаций при разработке программ. –М., 1989
3. **Непомнящий В.А. Рякин О.М**. Прикладные методы верификации программ. –М., 1988
4. **Кларк Э.М., Грамберг О., Пелед Д.** Верификация моделей программ. –М., 2002
5. **Легалов А.И., Казаков Ф.А., Кузьмин Д.А. Водяхо А.И.** Модель параллельных вычислений функционального языка // Известия ГЭТУ, Сборник научных трудов. Выпуск 500. Структуры и математическое обеспечение специализированных средств. – С.-Петербург, 1996. - С. 56-63.
6. **Легалов А.И.** Об управлении вычислениями в параллельных системах и языках программирования // Научный вестник НГТУ, № 3 (18), 2004. - С. 63-72.
7. **Легалов А.И.** Функциональный язык для создания архитектурно-независимых параллельных программ // Вычислительные технологии, № 1 (10), 2005. - С. 71-89.
8. **Грис Д.** Наука программирования. -М., 1984