РАЗДЕЛ 2. ОБРАБОТКА СИГНАЛОВ И ДАННЫХ

УДК: 681.3

ОТЛАДКА ПРОГРАММ НА ФУНКЦИОНАЛЬНО-ПОТОКОВОМ ПАРАЛЛЕЛЬНОМ ЯЗЫКЕ ПИФАГОР С ПОДСТАНОВКОЙ ИНТЕРВАЛЬНЫХ ЗНАЧЕНИЙ

Ю.В. Удалова, А.И. Легалов, Н.Ю. Сиротинина

В статье описан разработанный режим отладки программ на функционально-потоковом параллельном языке Пифагор, способный оперировать интервальными значениями и проверять пользовательские формулы, тем самым выявляя корректность программы относительно спецификации пользователя.

Ключевые слова: функционально-потоковое программирование, отладка, верификация, спецификация, интервалы

Введение

В теории и практике параллельного программирования задачи повышения надежности, отладки и верификации параллельных программ занимают одно из доминирующих положений из-за значительного числа вариантов взаимодействий параллельных участков программы, в которых легко скрываются логические ошибки, риски появления ошибок, связанных с механизмами синхронизации и передачи сообщений, риски тупиковых ситуаций и ресурсных конфликтов.

В настоящее время, кроме императивного программирования, считается одним из перспективных направлений для параллельных вычислений функционально-потоковая парадигма параллельного программирования [1,2], ориентированная на разработку программ, напрямую не связанных с архитектурой конкретных вычислительных систем, поддерживающих параллелизм на уровне базовых операций языка и управление вычислениями по готовности данных.

Существующие отладчики функционально-потоковых параллельных (ФПП) программ [3,4] менее развиты, чем отладчики императивных параллельных программ. Отладчик ФПП языка Пифагор до настоящего времени поддерживал только базовые опции отладки.

Модель ФПП языка Пифагор исключает ряд ошибок, присущих императивным параллельным программам, таких как тупики и критические секции. Более того, при различных запусках ФПП программы не могут быть получены различные вычисленные значения. Такие особенности позволили при разработке отладчика сосредоточиться на логической части и учете требований разработчика.

В статье [5] представлен один из разработанных режимов отладки ФПП программ, а именно режим проверки формул, использующий текст программы и ее графическое представление (графы функций), способный оперировать как точными вычисленными значениями, так и их интервальными оценками, проверяющий соответствие вычислений программы спецификации разработчика. Дополнительно применение интервальных значений при отладке позволяет оценивать границы вычисляемых значений и возможности их выхода за границы типов данных.

Режим проверки формул

Режим проверки формул позволяет пользователю закрепить за произвольными узлами-операторами графа программы собственные утверждения, являющиеся выражениями на языке программирования, использующие в качестве значений известные величины, аргументы функции или значения, вычисленные любым узлом-оператором графа. Если отлаживаемая функция не является рекурсивной, отладка выполняется за один шаг, на котором вычисляются все узлы графа и все дополнительные утверждения. Если отлаживаемая функция является рекурсивной, число шагов отладки совпадает с числом итераций рекурсивной функции, то есть пользовательские формулы повторно проверяются на каждой итерации.

Вершины графа, содержащие утверждения, помечаются как истинные, если выражения, введенные пользователем, возвращают истину, или как ложные, если хотя бы одно из выражений не равно истине. Для каждой такой вершины вместе со значением соответствующего оператора программы можно уви-

деть вычисленные значения утверждений. Пользовательская формула может иметь не только логическое значение, но и любое другое: целочисленное, вещественное, строковое, но в этом случае узел графа будет помечен как ложный.

Значение формулы или нескольких входящих в нее операндов может быть и интервальным, обработка таких данных возможна только при отладке и не поддерживается при непосредственном выполнении программы. Интервальные значения пользователь может указать среди входных данных отлаживаемой функции, для чего используются следующие конструкции:

~unknownnumber – неизвестное число;

~unknownbool – неизвестное логическое значение (ложь или истина);

~gt A – число большее, чем указанное число A;

~lt A – число меньшее, чем указанное число A;

~ge A – число больше, либо равное указанному числу A;

~le A – число меньше, либо равное указанному числу A;

~A interval B – число лежащее в указанном интервале [A, B];

~unknown – неизвестное значение, которое может быть числом, текстом, логическим значением, списком, строкой или любым другим данным программы. Использование этой константы для спецификации имеет малую пользу, константа *~unknown* часто является результатом выполнения программного оператора над данными, не определенными во множестве правил для отладки ФПП программы с интервальными значениями.

Таким образом, спецификация начальных данных может иметь, например следующий вид: *(5, ~lt 0, ~unknownbool)* – список, первый элемент которого равен пяти, второй меньше нуля и третий является ложью или истиной.

Наличие хотя бы одного интервального параметра среди аргументов отлаживаемой функции, вероятно, приведет к получению других интервальных (а не точных) значений при вычислении операторов функции.

Кроме спецификации входных данных функции, можно приписать к любому оператору произвольное число пользовательских формул (условий) с использованием следующих конструкций:

- *ARG* – аргумент функции;
- *NODE* – значение той вершины - оператора графа функции, к которой добавлено пользовательское условие;
- *NODE<натуральное число>* - значение оператора с указанным номером, который назначается операторам автоматически перед началом отладки и отображается в интерфейсе разработанной среды;
- базовые операторы и функции ФПП языка Пифагор (функции, описанные разработчиком в программе, использовать нельзя);
- описанные выше интервальные значения.

Таким образом, пользовательское условие может быть, например, таким *((NODE, ARG):<, (NODE, ~ 0 interval 1):!=):**, то есть значение текущего оператора меньше чем аргумент функции и не равно интервалу (0,1).

При вычислении пользовательских условий могут встречаться интервальные значения, и, если во множестве правил обработки интервальных значений требуемое правило отсутствует, значение пользовательского условия будет либо выставлено в ~unknown, либо запрошено у пользователя. Фактически это будет означать, что условие разработчика о желательных свойствах вычислений программы не удалось ни подтвердить, ни опровергнуть.

Если пользовательское условие возвратило *true*, это подтверждает соответствие вычислений программы требуемым условиям, если *false*, опровергает. В случае, когда пользовательское условие отлично от *true, false* и *~unknown*, анализ полученного результата целиком ложится на разработчика.

Пример отладки с подстановкой интервальных значений

Функция Abs получает число P и вычисляет его модуль. Граф функции на рисунке 1.

```
Abs << funcdef P
{ ({P:-}, P): [(P,0):(<,>=) :?]:. >> return ; }
```

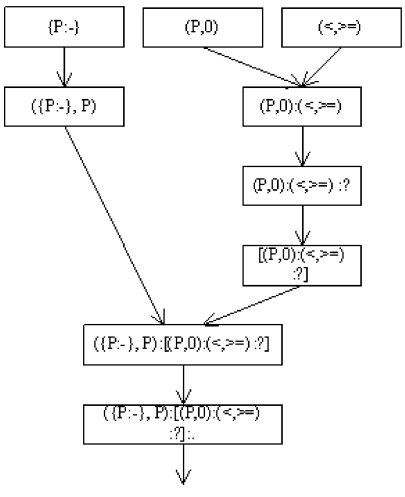

Рисунок 1 – Граф функции Abs

Пусть спецификацией начальных данных функции является *~lt 0*, то есть значение P меньше нуля. Результат отладки функции для указанной спецификации представлен на рисунке 2.

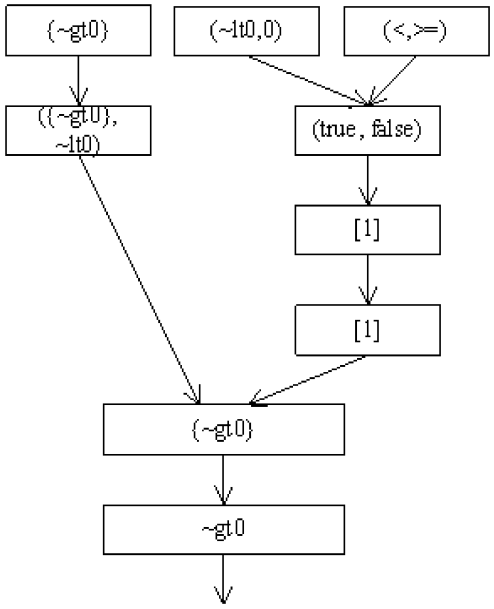

Рисунок 2 - Граф функции Abs с вычисленными значениями для аргумента ~lt 0

Оператор *{P:-}* при подстановке P станет *{~lt0:-}*. Поиск соответствия в наборе правил даст результат *{~gt0}*, то есть задержанный список с положительным числом.

Оператор *({P:-}, P)* при подстановке P и вычисленного для предыдущего оператора значения даст *({~gt0}, ~lt0)*.

Оператор *(P,0)* станет *(~lt0, 0)*. Команда *(P,0):(<,>=)* заменится на *(~lt0, 0) :(<,>=)*. Это означает проведение двух сравнений *~lt0<0* и *~lt0>=0*. Поиск в наборе правил выдаст результат *true* и *false*, что сформирует список *(true, false)*.

Оператор *(P,0):(<,>=):?* после подстановки предыдущих вычисленных значений станет равен *(true, false):?*. Оператор *(true, false):?* не содержит интервальных данных, поэтому его значение не ищется в наборе правил, а вычисляется интерпретатором. Оно равно *[1]*, это параллельный список из индексов истинных значений в списке.

Оператор *({P:-}, P): [(P,0):(<,>=) :?]* после подстановки вычисленных значений станет *({~gt0}, ~lt0) : [1]*, это выбор первого элемента из списка, что даст *{~gt0}*.

Оператор *({P:-}, P): [(P,0):(<,>=) :?]:.* при подстановке станет равен *{~gt0}:.*, что даст результат *~gt0*. Последний оператор *({P:-}, P): [(P,0):(<,>=) :?]:. >> return* завершит функцию с вычисленным значением *~gt0*, то есть с числом больше нуля.

Таким образом, отладка над указанными начальными данными *~lt 0* полностью автоматически (без запросов к пользователю) показала, что функция Abs, получающая на входе отрицательное число, вычисляет число положительное.

Если к заключительному оператору добавить пользовательское условие *(NODE, 0):*>, то есть ожидаемый результат больше нуля, условие трансформируется в *(~gt 0, 0):*> и возвратит истину, а заключительный оператор функции в разработанном отладчике будет отмечен цветом как корректный.

Заключение

Режим проверки формул и возможность выполнения программы в режиме отладки над интервальными оценками значений предоставляют инструментарий для исследования корректности ФПП программы относительно спецификации пользователя над обобщенным множеством начальных данных.

СПИСОК ЛИТЕРАТУРЫ

1. Легалов, А.И. Модель параллельных вычислений функционального языка / А.И. Легалов, Ф.А. Казаков, Д.А. Кузьмин, Д.А. Водяхо // Известия ГЭТУ, Сборник научных трудов. Выпуск 500. Структуры и математическое обеспечение специализированных средств. - С.-Петербург, 1996. - с. 56-63.
2. Легалов, А.И. Функциональный язык для создания архитектурно-независимых параллельных программ / А.И. Легалов // – Вычислительные технологии, № 1 (10) - Новосибирск, 2005. - с. 71-89.
3. Гордеев, Д.С. Визуализация внутреннего представления программ в системе функционального программирования SFP [Электронный ресурс] / Д.С. Гордеев. Режим доступа: www.iis.nsk.su/files/articles/sbor_kas_16.pdf
4. Pope, B. Buddha - A declarative debugger for Haskell [Электронный ресурс] / B. Pope. Режим доступа: http://www.berniepope.id.au /docs/BerniePope.Hons.Thesis.pdf
5. Удалова, Ю.В. Методы отладки и верификации функционально-потоковых параллельных программ / Ю.В. Удалова, А.И. Легалов, Н.Ю. Сиротинина // Journal of Siberian Federal University. Engineering & Technologies 2 – Красноярск, 2011. – с. 213-224.

Ст. преп. ***Ю.В. Удалова*** *- каф. высокопроизводительных вычислений; д.т.н., проф, зав. каф.* ***А.И. Легалов****, каф. вычислительной техники; к.т.н., доц.* ***Н.Ю. Сиротинина****, каф. вычислительной техники. Сибирский Федеральный Университет.*