УДК 004.42(07)

ОТЛАДКА ПРОГРАММ НА ФУНКЦИОНАЛЬНО-ПОТОКОВОМ ПАРАЛЛЕЛЬНОМ ЯЗЫКЕ С ПОДСТАНОВКОЙ ИНТЕРВАЛЬНЫХ ЗНАЧЕНИЙ И ПОЛЬЗОВАТЕЛЬСКИХ ФОРМУЛ

Удалова Ю.В.
научный руководитель д-р техн. наук Легалов А.И.
Институт Космических и Информационных Технологий
Сибирский Федеральный Университет

1. 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 (ФПП) парадигма 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 ФПП программ [3,4] гораздо менее развиты, чем отладчики императивных параллельных программ.

В статье представлен один из разработанных режимов [5] отладки ФПП программ, а именно режим проверки формул, использующий и 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.

2. 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 увидеть 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 поддерживает-

ся 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 число A;

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

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

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.

3. Пример отладки ФПП программы в режиме проверки формул с подстановкой интервальных значений

Функция Add (рисунок 1) должна находить сумму положительных элементов списка произвольной длины.

```
Add << funcdef v
{       //сравнение длины списка с единицей
        [((v:|,1):[<=,>]):?]^
        //если длина списка <= 1
        ( {(0,v:1):       //ноль или элемент списка v:1
        [((v:1,0):<=,(v:1,0):>):?]}, //в зависимости от знака v:1
        //если длина списка > 1
        /*сумма первых двух элементов списка и список v после удаления двух
    его первых элементов или только список v с изъятой первой парой элементов
    подаются на вход рекурсивной функции Add*/
        { ( ((v:1,v:2):+, v:-1:-1:[]):Add:. , (v:-1:-1:[]):Add:. ):
         [((v:1,0):>,(v:2,0):>):?]} //в зависимости от знаков v:1, v:2
        ) :. >>return;
}
```

Функция записана не правильно, ошибка заключается в условии [((v:1,0):>,(v:2,0):>):?]. Вместо того чтобы суммировать положительные элементы, будет производиться команда ((v:1,v:2):+, v:-1:-1:[]):Add:., если первый элемент положительный (v:1,0):> и команда (v:-1:-1:[]):Add:., если второй элемент списка положительный (v:2,0):>. Таким образом, если второй слагаемый элемент отрицательный, то все равно находится сумма элементов и дополнительно, если оба элемента положительные, то выполнятся обе команды, произойдут два рекурсивных вызова.

Для отслеживания корректности вычислений программы в режиме проверки формул добавлены пользовательские условия к вершинам-операторам:

№24 – (NODE,0):>= - результат оператора больше либо равен нулю;

№20 – ((NODE:1,0):>,(NODE6,0):>,(NODE7,0):>):* – результат оператора список, первый элемент которого больше нуля и первый элемент списка v (NODE6 – это результат оператора с автоматически присвоенным номером 6, то есть v:1 больше нуля и второй элемент списка v больше 0.

Пусть спецификация начальных данных имеет вид (~ge0,~lt0). Тогда первая итерация рекурсии даст следующие результаты (рисунок 2).

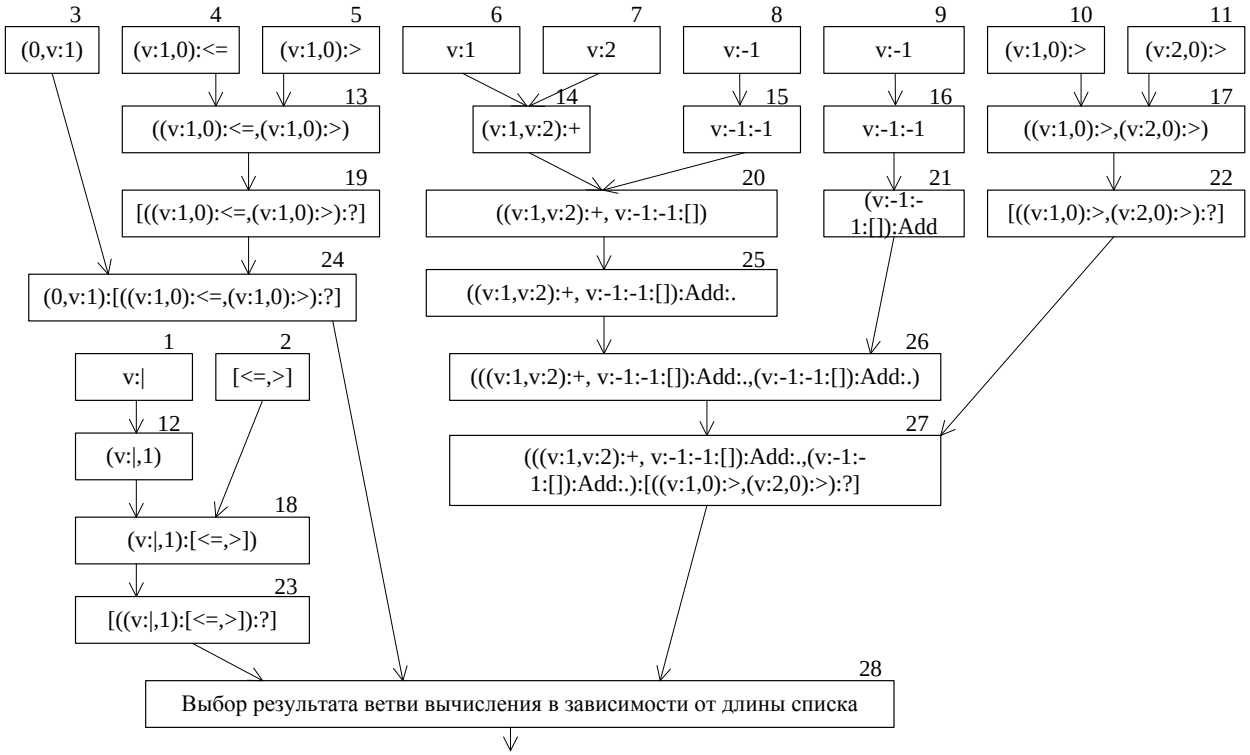

Рисунок 1. Сокращенный граф функции Add

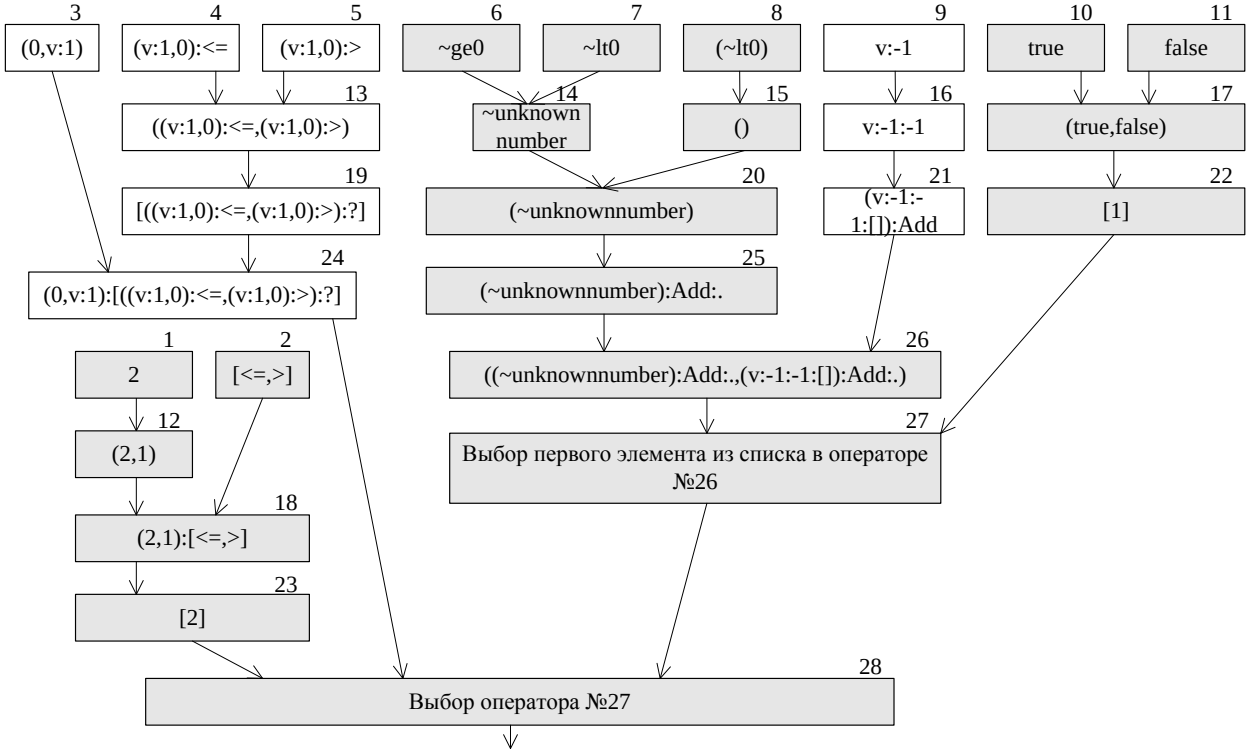

Рисунок 2. Первая итерация функции Add над данными (~ge0,~lt0)

На рис. 2 цветом выделены те вершины-операторы, которые были вычислены при заданном виде входных данных. Пользовательские условия на графе при входных данных (~ge0,~lt0), примут следующие значения:

№24 – (NODE,0):>= - ~unknown, так как оператор #24 не был вычислен при указанных входных данных;

№20 – ((NODE:1,0):>, (NODE6,0):>, (NODE7,0):>):* – (true, true, false):* даст значение false, и оператор №20 будет помечен на графе как не корректный, то есть не соответствующий требованию пользователя.

Таким образом, результат, полученный на первом шаге отладки в режиме проверки формул, показал, что вычисления итерации рекурсии не соответствуют спецификации и программа содержит логические ошибки. Однако необходимо заметить, что ошибочные команды могут содержаться не в самом операторе, к которому приписана формула, а в других вычисленных вершинах-операторах. Более того, при анализе результатов отладки в режиме проверки формул, надо учитывать, что ошибки могут быть не в программе, а в самой спецификации.

4. Заключение

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

5. Список литературы

1. Хьюз Джон. Сильные стороны функционального программирования. [Электронный ресурс] / Д. Хьюз // Режим доступа: http://softcraft.ru/paradigm/fp/whyfp.shtml - Загл. с экрана.

2. Легалов А.И. Функциональная модель параллельных вычислений и язык программирования "Пифагор". [Электронный ресурс] / А.И. Легалов, Ф.А. Казаков, Д.А. Кузьмин, Д.В. Привалихин // Режим доступа: http://softcraft.ru/parallel/fpp/fppcontent.shtml - Загл. с экрана.

3. Гордеев Д.С. Визуализация внутреннего представления программ в системе функционального программирования SFP. [Электронный ресурс] / Д.С. Гордеев // Режим доступа: http://www.iis.nsk.su/files/articles/sbor_kas_16.pdf - Загл. с экрана.

4. Bernard Pope. Buddha - A declarative debugger for Haskell. [Электронный ресурс] - Режим доступа: http://www.berniepope.id.au/docs/BerniePope.Hons.Thesis.pdf - Загл. с экрана.

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 (2011 4) – 2011. – С. 213-224. ISSN 1999-494X