УДК 004.42

Событийная модель вычислений, поддерживающая выполнение функционально-потоковых параллельных программ

А.И. Легалов[1*], Г.В. Савченко[1], В.С. Васильев[1]

[1]Сибирский федеральный университет, пр. Свободный, 79, Красноярск, Россия
Статья поступила 12.11 2011, принята 20.02.2012

В работе рассматривается модель вычислений, описывающая выполнение программ на функционально-потоковом языке параллельного программирования Пифагор. Программы записываются в виде зависимостей по данным между операторами в концепции неограниченного параллелизма, что приводит к отсутствию побочных эффектов при вычислениях, а также приносит возможность эффективного распараллеливания. Модель включает внутреннее представление программ, состоящее из информационного и управляющего графов. Информационный граф эквивалентен исходному коду программы, для повышения эффективности вычислений к графу могут быть применены оптимизации для сокращения размера, например, удаление повторяющихся констант и поддеревьев. Управляющий граф формируется по заданному информационному графу и определяет пути продвижения управляющих сигналов между операторами программы. Варьированием управляющего графа достигаются различные стратегии управления вычислениями. Помимо этого, в модель включен слой автоматов, описывающих семантику выполнения операторов, определяющих специфику функционально-потоковой модели вычислений. Автоматы определяют состояния, в которых находятся программно-образующие операторы, а также определяют действия, происходящие при поступлении управляющих сигналов. Приводятся примеры, демонстрирующие особенности формирования элементов внутреннего представления и функционирования автоматов, реализующих операции функционально-потоковой модели параллельных вычислений.

Ключевые слова: управление вычислениями, параллельное программирование, модель вычислений, функционально-потоковая парадигма параллельного программирования, трансляторы, автоматные модели.

Computation event model backing the execution of functional data flow concurrent programs

A.I. Legalov [1*], G.V. Savchenko [1], V.S.Vasiliev [1]

[1]Siberian Federal University, 79, Svobodny av., Krasnoyarsk, Russia
Received 12.11.2011; Accepted 20.02.2012

The article considers the computation model describing the execution of the programs in the functional data flow language of concurrent programing PYTHAGOR. The programs are written down in the form of data dependencies between the statements in the idea of unbounded concurrency that results in nonoccurrence of computation side effects and adds the efficient paralleling possibility. The model includes internal program representation consisting of information and control graphs. The information graph is equivalent to the program source code. To enhance computation efficiency, certain optimizations can be applied to the graph to reduce its size, for instance, the recurrent constants and subtrees deletion. The control graph is formed according to the given information graph and determines the actuating signals movement path between the program statements. By control graph variation, diverse strategies for computation control are achieved. Besides, the automatons layer describing the statements execution semantics determining the specific character of functional data flow computation model has been included into the model. The automatons determine the state of program-constituent statements and determine actions taking place on the control signals entry. The examples demonstrating the specifics of the internal representation elements generation and automatons functioning that implement the functional data flow concurrent computation model operations are given.

Keywords: computation control, concurrent programing, computation model, functional data flow concurrent programing paradigm, translators, automaton models.

Введение. Разработка параллельных программ сопряжена с дополнительными трудностями, во многом обусловленными необходимостью достижения высокой производительности при решении прикладных задач на архитектурах существующих параллельных вычислительных систем (ПВС). Наряду с проблемами понимания решаемой задачи необходимо погружаться в особенности конкретного вычислителя, чтобы обеспечить его эффективную загрузку и балансировку. Эти задачи по-разному решаются для ПВС различного типа (*SMP*, *MPP*, *GPU*, *FPGA* и др.). Помимо этого, ориентация разработки программ на конкретные вычислительные ресурсы связана с необходимостью учета специфических ресурсных ограничений, а также преодолением различных ресурсных конфликтов между взаимодействующими параллельно выполняемыми процессами.

* E-mail address: alexander.legalov@gmail.com

Совместный учет особенностей решаемой задачи, вычислительных ресурсов, эффективного выполнения и балансировки процессов затрудняет отладку, верификацию, тестирование и сопровождение, что усложняет создание параллельных программ. Поэтому актуальны подходы, ориентированные на архитектурно-независимую (АН) разработку параллельного программного обеспечения.

Специфической особенностью архитектурно-независимого процесса разработки программного обеспечения (ПО) является описание вычислений без учета специфики вычислительных ресурсов. Это возможно в том случае, если параллелизм будет рассматриваться на уровне базовых операций, вычислительные ресурсы будут считаться неограниченными, а процесс привязки разработанной программы к требуемым вычислительным ресурсам будет осуществляться после того, как эта программа будет отлажена, верифицирована и протестирована. Подобное условие предъявляет специфические требования к языкам программирования, которые, для уменьшения сущностей, должны поддерживать неявное описание параллельных вычислений (необходимо отказаться от фон-неймановской архитектуры).

Одним из подходов, связанных с разработкой архитектурно-независимых параллельных программ, является использование функциональных [1] и функционально-потоковых [2] языков параллельного программирования. Описывая каждую функцию программы как отношение между входящими в нее функциями, такие языки обеспечивают неявное описание параллелизма решаемой задачи. Выполнение функций по готовности данных позволяет не описывать явное управление вычислениями, присущее императивным языкам.

Применение данных языков считается перспективным. Однако в настоящее время оно сдерживается по разным причинам. Несмотря на значительное отличие параллельного программирования от последовательного, широко распространено мнение, что использование функциональных языков непривычно для разработчиков. Считается, что проще сочетать уже изученный последовательный стиль с организацией параллельных потоков и процессов. Это мнение подкрепляется также существованием ПВС, архитектура которых никоим образом не связана с архитектурно-независимыми параллельными языками.

Другим, более важным фактором является то, что для эффективного использования методов и языков архитектурно независимого параллельного программирования необходимо наличие методов эффективного преобразования написанных программ в архитектуры реальных ПВС. Существуют исследования, в которых делается попытка решить эту проблему. Выражается оптимизм в скором появлении инструментальных средств, обеспечивающих эффективное распараллеливание практически любых программ [3]. Однако в настоящее время такие методы не обладают должной эффективностью.

Вместе с тем, отсутствие решения проблемы в ряде областей никоим образом не ведет к отказу от поиска методов ее решения, так как использование архитектурно-независимого параллельного программирования уже сейчас позволяет ускорить разработку, отладку и верификацию параллельных программ. Поэтому актуальной является задача не только организации эффективных вычислений, но и решения всего комплекса проблем, связанных с разработкой параллельного программного обеспечения. Для разностороннего исследования различных вопросов необходимо формирование системы, ориентированной на эффективную поддержку параллельного программирования. Для достижения этого необходимо проведение соответствующего анализа внутренней организации подобных систем и создание такой структуры, которая бы позволила в комплексе решать встающие задачи по разработке, отладке, верификации, оптимизации и выполнению параллельных программ.

В рамках работ по системе функционально-потокового параллельного программирования на языке Пифагор предлагаются инструментальные средства, направленные на эффективную разработку архитектурно-независимых параллельных программ. Пифагор является языком, который удовлетворяет вышеперечисленным требованиям архитектурной независимости, ориентирован на создание программ в концепции неограниченного параллелизма, поскольку программы на нем записываются в виде зависимостей операторов по данным без указания порядка их вычислений [2].

Для трансляции и выполнения функционально-потоковых программ, написанных на языке программирования Пифагор, разработана инструментальная система, состоящая из транслятора исходных текстов программы в промежуточное представление, и интерпретатора, использующего построенное промежуточное представление для выполнения программы. Помимо этого, в ходе проводимых преобразований исходной программы допускаются различные оптимизационные преобразования. Для обеспечения эффективного выполнения этих функций промежуточное представление разделено на следующие слои:

- слой реверсивного информационного графа (РИГ);
- слой данных;
- слой управляющего графа;
- слой управляющих автоматов, отслеживающих готовность данных.

Формирование слоев происходит во время трансляции функции и состоит из следующих этапов:

- построение реверсивного информационного графа транслируемой функции и частичное заполнение слоя данных используемыми константами;
- создание управляющего и автоматного слоев функции на основе РИГ.

Полученные структуры данных определяют промежуточные представления разработанных функций. Перед выполнением осуществляется их компоновка в единый исполняемый модуль, который и используется событийным процессором [4], обеспечивающим выполнение функционально-потоковых параллельных программ. Предлагаемые средства отладки и верификации позволяют анализировать корректность, обеспечивая разнообразный обход информационного графа программы, представленного как в текстовом, так и в графическом режимах [5].

Построение реверсивного информационного графа и частичное заполнение слоя данных. Исходными данными для построения реверсивного информационного графа (РИГ) является функция, написанная на языке программирования Пифагор. Поэтому ее трансляция связана с формированием внутреннего представления в аналогичной форме, удобной для дальнейших манипуляций и преобразований.

Отличительные особенности РИГ проявляются в следующем:

1. Ребра направлены от вершин, принимающих данные, к вершинам, являющимся их источниками. Впоследствии эти сваязи используются для чтения данных, формируемых в ходе вычислений.

2. Задержанные списки, представляющие собой вычисления, отложенные до момента, когда их значение попадает на вход операции интерпретации, представляются в виде констант, всегда имеющих готовое значение. Операторы, входящие в задержанные списки, маркируются номером задержанного списка, который впоследствии используется для анализа того, раскрыт данный список или нет при обработке возникающих событий.

3. Конкретные значения данных вынесены в отдельный слой. В РИГ остается вершина – константный оператор, – ссылающаяся на значение константы в слое данных.

Существуют следующие типы вершин РИГ:

1. Константа (*const*). Представляет собой значение, явно заданное в коде программы. Это может быть строка, символ или число. Константа не имеет зависимостей по данным от других вершин, и, соответственно, не имеет исходящих ребер. Ее атрибутом является значение, находящееся в слое данных. Значением может быть «*hello world*», «*a*», 42 и так далее. Константой при трансляции также представляются задержанные списки, в этом случае значением константы является пара (номер задержанного списка, номер вершины-корня задержанного списка).

2. Аргумент функции (*arg*). Представляет аргумент функции. При трансляции функции аргумент, как правило, неизвестен, однако при поступлении данных он ведет себя как фиксированное константное значение. Не имеет зависимостей по данным от других вершин и, соответственно, исходящих ребер.

3. Список данных (*list*, в исходном коде «(---)»). Служит для группировки данных в список. Количество исходящих ребер равно количеству элементов списка. Количество входящих ребер – одно.

4. Параллельный список (*parlist*, в исходном коде «[---]»). Служит для группировки данных в параллельный список, количество входных дуг равно количеству элементов списка. Количество выходных дуг равно количеству элементов списка.

5. Операция интерпретации (*int*, в исходном коде «:»). Операция интерпретации имеет зависимости по данным от параллельного списка аргументов и функций. Данные зависимости обозначаются исходящими ребрами. Количество исходящих ребер зависимостей по аргументам и функциям может быть равно одному в частном случае, когда на вход операции интерпретации подаются одна функция и один аргумент.

6. Возвратная (*return*). Используется для выполнения операций, связанных с возвратом значения из функции. Имеет одно исходящее ребро, определяющее зависимость от вершины, обеспечивающей возвращаемое значение.

В качестве примера построения реверсивного информационного графа рассмотрим преобразование функции нахождения абсолютной величины числа. На рис. 1 приводится текст функции на языке Пифагор, представленный в соответствии с описанием модели вычислений [3].

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

Рис. 1. Исходный код функции вычисления модуля числа.

После трансляции формируются реверсивный информационный граф и слой данных, обобщенное внутреннее представление которых приведено в таблице 1.

Таблица 1

Внутреннее представление РИГ и слоя данных функции вычисления модуля числа

№ вершины РИГ	Слой реверсивного информационного графа			Слой данных
	№ задержанного списка	*Операция*	*Связи*	*Значение*
0	0	*arg*		
1	0	*const*		–
2	1	:	0, 1	*None*
3	0	*const*		{1}2
4	0	(---)	3, 0	*None*
5	0	*const*		0
6	0	(---)	0, 5	*None*
7	0	*const*		<
8	0	*const*		>=
9	0	(---)	7, 8	*None*
10	0	*const*		?
11	0	:	6, 9	*None*
12	0	:	11, 10	*None*
13	0	[---]	12	*None*
14	0	*const*		.
15	0	:	4, 13	*None*
16	0	:	15, 14	*None*
17	0	*return*	16	*None*

Исходящие ребра связей зависимостей по данным записываются в колонке «Связи» как номера вершин, от которых зависит текущая вершина.

Графическое представление реверсивного информационного графа, формируемого соответствующими утилитами, приведено на рис. 2. На РИГ константы размещаются в прямоугольниках, операция интерпретации в окружности, а операции над списками – в прямоугольниках со скругленными углами. Аргумент и результат отмечены пятиугольниками.

Вершины информационного графа нумеруются (колонка «№ вершины РИГ») и, помимо этого, им ставится в соответствие номер задержанного списка (колонка «№ задержанного списка»), в который они входят. Сами задержанные списки преобразуются в константы, значением которых (колонка «Значение») являются номер задержанного списка и ссылка на узел реверсивного информационного графа, порождающего результат, определяемый этим задержанным списком. Нулевой номер задержанного списка означает его отсутствие.

Слой данных (колонка «Значение») предназначен для хранения результатов вычислений, формируемых в каждом из узлов информационного графа. Поэтому число элементов в нем соответствует количеству узлов РИГ. Первоначально на этом слое (таблица 1) хранятся только значения констант, используемых в программе; к константам относятся используемые внешние функции, символьные и числовые значения, встроенные операции. Там, где значения отсутствуют, формируется ссылка на неопределенную величину (*None*). Она будет окончательно определена во время выполнения вычислений.

По определению, функция нахождения абсолютной величины возвращает либо аргумент *Param*, если *Param* – неотрицательное число, либо *–Param*, если оно отрицательное. Для этого формируется список данных, представляемый вершиной 4 РИГ, первое значение которого представляет собой унарное вычитание из *Param*, находящееся в задержанном списке {*Param*:-}. Это означает, что значение *–Param* может и не вычисляться в случае, если *Param* – неотрицательное число. Значение задержанного списка представляется константной вершиной 3, имеющей значение «{1}2», что означает принадлежность первому задержанному списку, определяемому вершиной 2. Второе значение списка – вершина 0, т. е. аргумент функции *Param*. Исходный код списка ({*Param*:-}, *Param*).

Из списка, формируемого вершиной 4, необходимо выбрать либо первый, либо второй элемент. Для этого используется вершина операции интерпретации 15, второй зависимостью по данным которой является индекс элемента списка для выбора. Индекс представляется параллельным списком 13. Операция интерпретации 12, вычисляющая значение (*Param*,0):(<,=>):?, возвращает 1, если *Param* < 0, и 2 в противном случае.

После выбора первого или второго элемента списка 4 выполняется операция интерпретации 16, назначение которой – раскрыть задержанный список, если был выбран первый элемент списка 4, либо ничего не делать, если был выбран второй элемент – аргумент 0, не находящийся в задержанном списке.

Над построенным реверсивным информационным графом возможно проведение различных оптимизационных преобразований, связанных с удалением повторяющихся поддеревьев, избыточных операторов, повторяющихся констант. В дальнейшем можно использовать как исходный, так и оптимизированный РИГ.

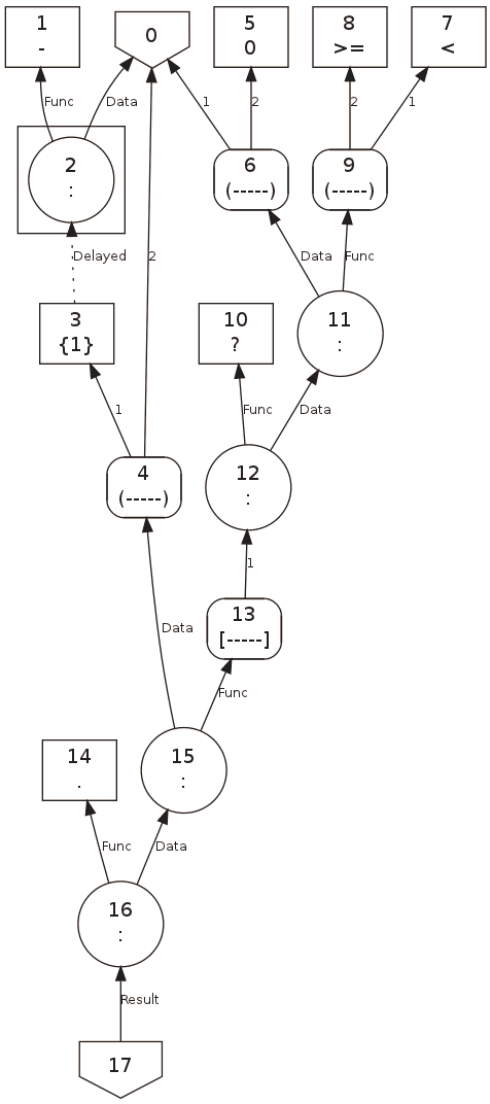

Рис. 2. Графическое представление реверсивного информационного графа.

Построение управляющего и автоматного слоев. Управляющий граф (УГ), задающий соответствующий слой, формируется по реверсивному информационному графу. Он состоит из вершин, которые связаны между собой в сеть, обеспечивающую передачу управляющих сигналов, несущих информацию о готовности данных, формируемых в соответствии с операторами, описанными в узлах информационного графа. Управляющие связи, в отличие от связей РИГ, направлены от источников сигналов к приемникам. Предполагается, что каждый сигнал является мгновенным событием, передаваемым из одной вершины в другую и сохраняемым в очереди полученных сигналов вершины-приемника. При описании сигналов задаются номер вершины-приемника и номер входа вершины, на который поступает сигнал.

При первоначальном построении УГ число его вершин совпадает с их количеством в РИГ. В ходе последующих оптимизационных преобразований оно может изменяться. Изменяться могут и управляющие связи между вершинами. Изменение связано с нумерацией вершин, которая после этого не будет совпадать с исходной их нумерацией.

Каждая из вершин УГ связана с автоматом, последовательно реагирующим на поступающие из очереди управляющие сигналы. Сигнал передает автомату информацию о номере входа, на который он поступил. Для автомата сигнал срабатывает как внешнее событие, вызывающее изменение состояния. Тип автомата зависит от типа вершины информационного графа, по которой строится вершина управляющего графа. В соответствии с набором операторов функционально-потоковой модели вычислений [3], выделяются следующие типы автоматов:

- автомат обработки аргумента функции (*arg*);
- константный автомат, предназначенный для управления при наличии констант (*const*);
- автомат операции интерпретации (*int*);
- автоматы обработки сигналов, поступающих в список данных (*list*), параллельный список (*parlist*);
- автомат для обработки операции возврата результата функции (*return*).

Номенклатура автоматов может расширяться в ходе дальнейшей разработки, при добавлении новых стратегий управления вычислениями и проведении оптимизационных преобразований управляющего графа. Вновь вводимые автоматы могут подменять уже встроенные. Каждый автомат может иметь несколько состояний и дополнительную рабочую память.

Наряду с типом, в автоматном слое задается начальное состояние автомата, а также связанная с данным автоматом вершина информационного графа, определяющая впоследствии требуемую вычислительную операцию и местоположение результата вычислений.

При непустой очереди сигналов вершины происходит последовательная обработка имеющихся сигналов и переход автомата в соответствующее состояние.

Пример внутреннего представления управляющего графа и автоматного слоя функции нахождения абсолютного значения числа приведен в таблице 2.

Таблица содержит данные, определяемые следующим образом. Колонка «Управляющие связи» для каждой вершины УГ содержит описание исходящих управляющих ребер связей в виде списка пар «вершина – номер входа». Колонка «Начальные сигналы» определяет набор первоначальных управляющих сигналов, которые помещены в очереди событий соответствующих автоматов. Колонка «Тип автомата» определяет тип используемого автомата, но не его точную реализацию, поскольку она не фиксирована и может меняться. Колонка «Текущее состояние» определяет начальное состояние всех автоматов согласно диаграмме состояний каждого из них. Колонка «Вершина РИГ» определяет связанную вершину РИГ. Описание управляющих сигналов (событий) автоматов, а также их состояний будет приведено ниже, при описании специфики функционирования каждого автомата.

Таблица 2

Внутреннее представление УГ и слоя автоматов функции вычисления модуля числа

№ узла управляющего графа	Слой управляющего графа			Слой автоматов		
	№ задержанного списка	*Управляющие связи*	*Начальные сигналы*	*Тип автомата*	*Текущее состояние*	*Вершина РИГ*
0	0	(2,1), (4,2), (6,1)		*arg*	*NAw*	0
1	0	(2,2)	*e_NC_r*	*const*	*NCn*	1
2	1	(3,1)		*int*	*NIi*	2
3	0	(4,1)	*e_NC_r*	*const*	*NCn*	3
4	0	(15,1)		*list*	*NLi*	4
5	0	(6,2)	*e_NC_r*	*const*	*NCn*	5
6	0	(11,1)		*list*	*NLi*	6
7	0	(9,1)	*e_NC_r*	*const*	*NCn*	7
8	0	(9,2)	*e_NC_r*	*const*	*NCn*	8
9	0	(11,2)		*list*	*NLi*	9
10	0	(12,2)	*e_NC_r*	*const*	*NCn*	10
11	0	(12,1)		*int*	*NIi*	11
12	0	(13,1)		*int*	*NIi*	12
13	0	(15,2)		*parlist*	*NPLi*	13
14	0	(16,2)	*e_NC_r*	*const*	*NCn*	14
15	0	(16,1)		*int*	*NIi*	15
16	0	(17,1)		*int*	*NIi*	16
17	0			*return*	*NRTi*	17

Динамика исполнения определяется алгоритмами срабатывания автоматов, каждый из которых имеет свои особенности. Например, при поступлении сигнала *e_NC_r* на вход автомата обработки константной вершины из начального состояния выполняется переход в конечное. При этом в вершину управляющего графа выдается сигнал, подтверждающий наличие константы в слое данных (она записывается туда на этапе формирования реверсивного информационного графа). В связи с возвратом истинного значения вершина управляющего графа сразу же рассылает сигналы во все вершины-приемники. Подобная специфика в срабатывании при наличии констант позволяет оптимизировать управляющий граф до начала выполнения, убрав из него все вершины с отработавшими автоматами и перенаправив полученные управляющие сигналы из этих вершин в вершины-приемники.

Аналогичные оптимизационные преобразования могут осуществляться и в других вершинах в соответствии с типами используемых в них автоматов. При этом автоматы могут переходить в промежуточные состояния еще до начала вычислений.

Особенности автоматов.

Константный автомат. Как было сказано выше, очередь автомата (рис. 3) может содержать один сигнал *e_NC_r*. Изначально автомат находится в состоянии *NCn*, затем переходит в состояние *NCr* после обработки данного сигнала. При обработке в вершину управляющего графа выдается сигнал, подтверждающий наличие константы в слое данных.

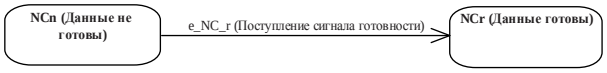

Рис. 3. Диаграмма состояний константного автомата.

Автомат обработки аргумента функции. Очередь автомата обработки аргумента функции *Aarg* (рис. 4) может состоять из одного сигнала *e_NA_r*. Изначально автомат находится в состоянии *NAw*; при получении аргумента, что определяется сигналом *e_NA_r*, выполняется переход в состояние *NAr*. При этом в вершину управляющего графа выдается сигнал, подтверждающий получение аргумента.

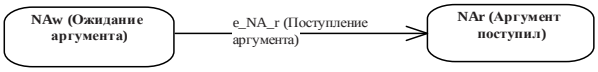

Рис. 4. Диаграмма состояний автомата обработки аргумента функции.

Автомат операции интерпретации. Автомат операции интерпретации принимает сигналы готовности аргумента и сигналы готовности функции. Количество данных сигналов произвольно, конечно и зависит от РИГ программы. Например, если аргументом операции интерпретации является параллельный список, то функция будет выполнена над каждым его элементом независимо, и количество управляющих сигналов, которое будет помещено в очередь, составит количество элементов параллельного списка плюс один сигнал, сигнализирующий о готовности функции. Данная схема обобщается на произвольную длину параллельного списка аргументов и списка функций.

На рис. 4 показана диаграмма состояний автомата операции интерпретации, выполняющего функцию [*a1, a2*]: *f*, принимающую в качестве аргумента параллельный список из двух элементов, и одну функцию. Внешние управляющие события такого автомата – это события *e_NI_f* (поступление функции), *e_NI_a1* (поступление первого элемента списка), *e_NI_a2* (поступление второго элемента списка). Соответственно, состояние имеет вид *NI*{[*a1*][*a2*][*f*]}, где все три части – необязательны, и обозначает готовность первого, второго элементов списка аргумента и функции соответственно. Начальным состоянием является *NIi*, а конечным – *NI*{*a1, a2, f*}.

Запуск функции выполняется сразу же после готовности пары «аргумент-функция», что происходит на всех переходах, кроме исходящих непосредственно из состояния *NIi*. Это означает, что при обработке события, если новая пара «аргумент-функция» готова, то выполняется запуск данной функции над аргументом.

После выполнения функции над аргументом на переходе автомат в той же функции перехода выдает управляющий сигнал о готовности данных соответствующей управляющей вершине.

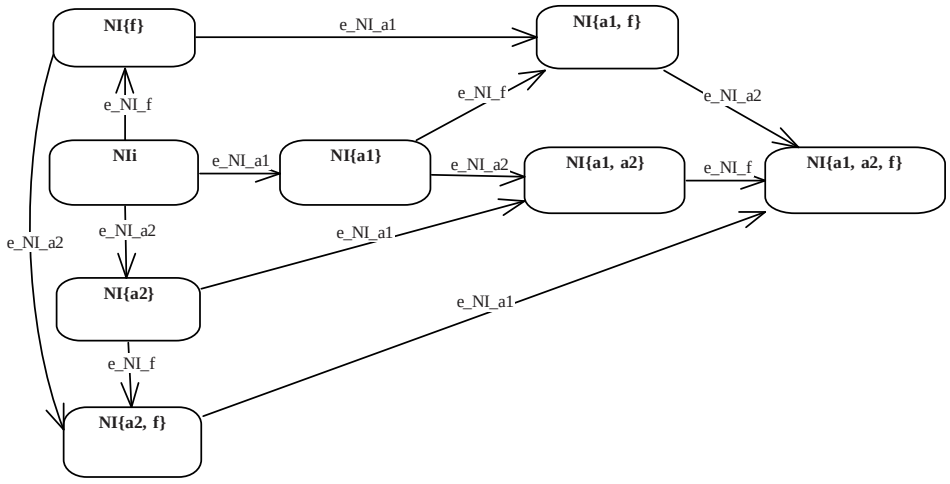

Рис. 5. Пример диаграммы состояний автомата операции интерпретации.

Автомат обработки сигналов, поступающих в список данных. Автомат обработки сигналов списка данных, состоящего из *N* элементов, имеет *N* входных событий. Для примера рассмотрим автомат списка, содержащего два элемента. Такой автомат принимает два события – *e_NL_a1*, *e_NL_a2*. Его диаграмма состояний показана на рис. 6.

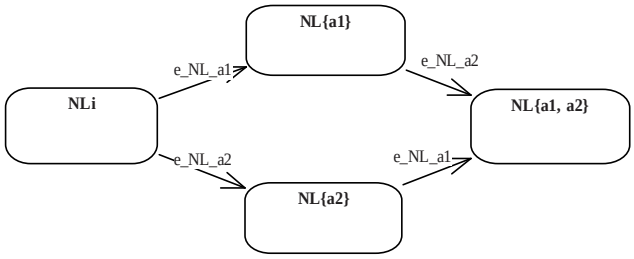

Рис. 6. Пример диаграммы состояний автомата обработки сигналов, поступающих в список данных.

Автомат на переходе в состояние *NL{a1, a2}* выдает связанной управляющей вершине сигнал, говорящий о готовности данных, поскольку список данных является готовым, когда поступил управляющий сигнал для всех его элементов.

Автомат обработки сигналов, поступающих в параллельный список**. Назначением параллель**ного списка является группировка данных, как правило, для их независимой обработки. Каждый элемент параллельного списка может обрабатываться независимо от других элементов, что определяет специфику автомата обработки сигналов параллельного списка. Наиболее тривиальным алгоритмом работы автомата может быть передача связанной управляющей вершине сигнала о готовности данных при каждом поступлении элемента списка. Таким образом, автомат играет роль повторителя сигналов. Более сложной стратегией управления может быть накопление сигналов готовности и выдача их при превышении некоторого количества сигналов. Такой подход позволяет организовать пакетную обработку данных.

Автомат для обработки возврата результата. Очередь событий данного автомата (рис. 7) может состоять из одного события *e_NRT_r*, означающего готовность данных к возврату. После обработки данного сигнала выполняется возврат значения и переход в состояние *NRTr*.

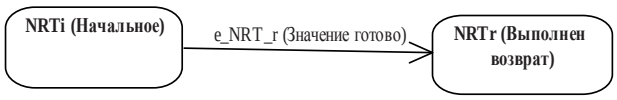

Рис. 7. Диаграмма состояний автомата возврата результата.

Заключение.

Предлагаемая организация внутреннего представления обеспечивает гибкость анализа функционально-потоковых программ, начиная с уровня кода и заканчивая процессом выполнения. Построение управляющего и автоматного слоев может производиться на основе оптимизированного информационного графа, в котором, например, удалены повторяющиеся поддеревья вычислений. Управляющий граф и автоматный слой могут быть далее оптимизированы для уменьшения количества обрабатываемых событий на этапе выполнения. Это достигается обработкой автоматами очередей событий на этапе трансляции программы и дальнейшим сохранением состояния слоев.

Литература

1.Касьянов В.Н., Бирюкова Ю.В., Евстигнеев В.А. Функциональный язык SISAL3.0 // Поддержка супервычислений в Интернет-ориентированные технологии: сб. ст. Новосибирск, 2001. С. 54-67.

2. Легалов А.И. Функциональный язык для создания архитектурно-независимых параллельных программ // Вычислительные технологии. 2005. №1(10). С. 71-89.

3. Крюков В.А. Автоматизация создания вычислительных прогрпмм для современных кластеров // Параллельные вычислительные технологии (ПаВТ 2011): тр. междунар. науч. конф. (Москва, 28 марта -1 апр. 2011 г.). Челябинск: изд. центр ЮурГУ, 2011. 730 с.

4. Редькин А.В., Легалов А.И. Событийное управление выполнением функционально-потоковых параллельных программ // Науч. вестн. Новосиб. гос. техн. ун-та. 2008. № 3(32). С. 11-117.

5. Удалов Ю.В., Легалов А.И., Сиротинина Н.Ю. Методы отладки и верификации функционально-потоковых параллельных программ // Журнал Сиб. федер. ун-та. Сер. Техника и технологии. 2011. Т. 4, № 2. С. 313-224.

References

1. Kas'yanov V.N., Biryukova Yu.V., Evstigneev V.A. Functional language SISAL 3.0 // Podderzhka supervychisleniy i Internet-orientirovannye tekhnologii. Novosibirsk. 2001. S. 54-67.

2. Legalov A.I. Functional language to develop architecturally independent concurrent programs. Vychislitel'nye tekhnologii. № 1 (10). 2005. S. 71-89.

3. Kryukov V.A. Development automation of the computational procedures for modern clusters / Parallel'nye vychislitel'nye tekhnologii (PaVT'2011): tr. mezhdunar. nauch. konf. (Moskva, 28 marta – 1 aprelya 2011 g.). Chelyabinsk: izd centr YuUrGU, 2011. 730 s.

4. Red'kin A.V., Legalov A.I. Activity-directed control of the functional data flow concurrent programs execution // Nauch. vestn. Novosib. gos. tekhn. un-ta. 2008. №3 (32). S. 111-117.

5. Udalova Yu.V., Legalov A.I., Sirotinina N.Yu. Debug and verification methods for functional data flow concurrent programs // Zhurnal Sib. feder. un-ta. Ser. Tekhnika i tekhnologii. 2011. T. 4. № 2. S. 213-224.