УДК 519.68

ДОБАВЛЕНИЕ СТАТИЧЕСКОЙ ТИПИЗАЦИИ В ЯЗЫК ФУНКЦИОНАЛЬНО-ПОТОКОВОГО ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ

А. И. Легалов, И. А. Легалов, И. В. Матковский

Сибирский федеральный университет, г. Красноярск

legalov@mail.ru, igor@legalov.ru, alpha900i@mail.ru

Аннотация

Предложено добавить статическую систему типов в функционально-потоковую модель параллельных вычислений и разработанный на ее основе язык функционально-потокового параллельного программирования. Использование статической типизации повышает возможность трансформации функционально-потоковых параллельных программ в программы, выполняемые на современных параллельных вычислительных системах. Предложены языковые конструкции. Описаны их синтаксис и семантика. Отмечена необходимость использования принципа единственного присваивания при формировании хранилищ данных конкретного типа. Рассмотрены особенности инструментальной поддержки предлагаемого подхода.

Ключевые слова*: парадигмы программирования, параллельное программирование, функционально-потоковое параллельное программирование, статическая типизация, модели параллельных вычислений*

ВВЕДЕНИЕ

Современные методы разработки параллельных программ сильно зависят от особенностей архитектур параллельных вычислительных систем (ПВС), что находит соответствующее отражение в языках программирования. Практически любые изменения архитектуры ПВС ведут к переписыванию и модификации уже разработанного и отлаженного кода. Попыткой преодолеть эту ситуацию является применение концепции архитектурно-независимого параллельного программирования (АНПП), ориентированного на разработку программ с использованием языковых и инструментальных средств, предназначенных для абстрактных

(виртуальных) параллельных систем с неограниченными вычислительными ресурсами и стратегиями управления вычислениями по готовности данных. Такие подходы развиваются в разных направлениях. Можно отметить язык программирования COLAMO, разработанный для систем на кристалле [1, 2]. Создание универсальных языков, напрямую не связанных с архитектурными ограничениями, можно проследить на примере функциональных языков параллельного программирования Sisal [3] и Пифагор [4].

Наиболее последовательно концепция АНПП нашла свое отражение в языке Пифагор. Она непосредственно учитывается в его модели функционально-потоковых параллельных вычислений. Модель программы описана как ресурсно-неограниченный ациклический безусловный граф, в котором управление осуществляется по готовности данных. Помимо этого в ней реализован принцип единственного использования вычислительных ресурсов [4]. На уровне модели предполагается, что для выполнения любых операций выделяются свои уникальные ресурсы, реальное распределение которых осуществляется после того, как разработана и отлажена логическая структура программы. Для апробации возможностей языка разработаны инструментальные средства, поддерживающие процесс создания, преобразования и выполнения функционально-потоковых параллельных программ [5].

Однако следует отметить невысокую эффективность выполнения программ, что обуславливается использованием интерпретатора. Последнее связано с тем, что в языке применяется динамическая типизация данных, а операторы, представленные в модели вычислений, обладают динамическим поведением, позволяя формировать списки произвольной размерности во время вычислений. В связи с этим практически невозможна эффективная трансформация написанных программ в современные статически типизированные языки, используемые при реальном параллельном программировании.

Вместе с тем эксперименты, проводимые с применением разработанных инструментальных средств, показали возможность эффективного применения данной парадигмы для оптимизации [6], формальной верификации [7] и отладки [8] программ еще до того момента, как начнется их преобразование к конкретной архитектуре. Это позволяет иметь программу, перенос которой на реальные ПВС мог бы осуществляться более формально за счет наложения ресурс-

ных ограничений, учитывающих особенности конкретной архитектуры, с сохранением уже отлаженной общей логики функционирования.

В связи с этим перспективной видится модификация функционально-потоковой модели параллельных вычислений (ФПМПВ), направленная на учет особенностей организации данных в современных языках программирования, что позволило бы упростить процесс трансформации функционально-потоковых параллельных (ФПП) программ. В основном эта модификация связана с применением статической типизации и фиксацией размерности списковых и контейнерных структур данных, что ведет к пересмотру ряда концепций ФПМПВ. Вполне естественно, что в соответствии с этими изменениями должен измениться и язык ФПП программирования.

В результате проведенных исследований сформирована статически типизированная модель функционально-потоковых параллельных вычислений (СТМФППВ). Как и предшествующая ей ФПМПВ, она определяет программу как информационный граф с управлением по готовности данных. Однако операторы, описывающие алгоритм программы, разработаны с учетом возможных преобразований в статически типизированные языки программирования, что ведет к изменению ряда аксиом и алгебры преобразований. На основе предложенной модели разработан статически типизированный язык функционально-потокового параллельного программирования (СТЯФППП) Smile.

1. СТАТИЧЕСКАЯ ТИПИЗАЦИЯ НА УРОВНЕ ОПЕРАТОРОВ

Как и в предшествующей ФПМПВ [4], операторы задают узлы информационного графа, в котором вычисления выполняются по готовности данных. Однако существует ряд особенностей, связанных с изменением требований. Необходимо обеспечить поддержку следующих свойств, характерных для статически типизированных языков программирования:

— эффективную трансформацию статически типизированных функционально-потоковых параллельных программ в другие модели вычислений вместо их интерпретации;

— повышение контроля за счет использования сильной типизации;

— сохранить принцип управления по готовности данных и общую концепцию функционально потоковой модели параллельных вычислений;

— каждый из программоформирующих операторов должен опираться на типизированные данные, контролируемые на этапе компиляции;

— контейнерные (списковые) данные должны иметь фиксированный размер, определяемый либо во время компиляции, либо во время выполнения;

— аксиоматика языка должна быть упрощена, чтобы уменьшить число динамических проверок и преобразований во время выполнения;

— упрощение алгебры эквивалентных преобразований.

Приведенные требования ведут к изменению практически всех операторов ФПМПВ, в результате чего сформирована модель вычислений, обладающая иными свойствами. Эти свойства определяются через особенности функционирования программоформирующих операторов СТМФППВ.

Оператор интерпретации описывает функциональные преобразования аргумента. Он имеет два входа, на которые через информационные дуги поступают функция **F** и аргумент **X**. Как аргумент, так и функция могут являться результатами предшествующих вычислений. Основными особенностями новой версии данного оператора являются:

— типы аргументов на входах оператора должны быть известны во время компиляции;

— тип результата на выходе также вычисляется во время компиляции;

— на входе и выходе оператора допускаются именованные типы данных, неименованные структуры и кортежи;

— для именованных типов допустима только именованная эквивалентность;

— для кортежей допускается структурная эквивалентность;

— предопределенность базовых операций, для которых на уровне языка прописаны все возможные типы данных аргументов и результатов.

Исходя из этого, для базовых функций языка изначально определяются сигнатуры, задающие типы аргументов и результатов. Для функций, определяемых пользователем, типы аргументов и результатов явно задаются во время определения функций. Допускается дуализм некоторых базовых данных, которые в зависимости от использования в операторе интерпретации могут выступать как в качестве аргумента, так и функции. В этом случае для них возможно определение двойного типа (тип данных и сигнатура функции).

Оператор интерпретации запускается по готовности данных, что фиксируется появлением разметки на входных дугах. Получение результата задается разметкой выходной дуги.

Вместо группировки в список данных в СТМФППВ используется **группировка в кортеж**. Можно выделить следующие основные свойства данного оператора:

— размер кортежа определяется во время компиляции (что обусловливается необходимостью знать типы группируемых данных и их размер);

— элементы кортежа являются данными именованных типов;

— обеспечивается сравнение на структурную эквивалентность с другими кортежами;

— готовность кортежа к выполнению определяется по готовности всех его данных;

— отсутствуют внутренние эквивалентные преобразования, изменяющие размер кортежа во время выполнения (сигнал, удаляемый из списка в ФПМПВ, является типом данных без значения и сохраняется в явном виде).

Изменены также аксиомы, определяющие преобразование кортежей во время вычислений, что также обуславливается введением дополнительного контроля во время компиляции.

Группировка в параллельные списки заменяется на **группировку в рой**. Используется для объединения данных, над которыми выполняется одна массовая операция. К свойствам роя относятся:

— размер роя определяется во время компиляции;

— элементами роя являются данные одного именованного типа или все элемента роя структурно эквивалентны;

— готовность роя к выполнению определяется по готовности хотя бы одного элемента (асинхронность в обработке отдельных его элементов);

— отсутствуют внутренние эквивалентные преобразования, изменяющие размер роя во время выполнения;

— внутри кортежей рой не вырождается в последовательность элементов кортежа, а является единым элементом.

— алгебра эквивалентных преобразований роев реализуется только во время компиляции.

Приведенные характеристики позволяют рассматривать рой в качестве набора независимых данных, запускаемых по мере их поступления. На выходе оператора интерпретации также формируется рой, состоящий из элементов одного типа.

Группировка в задержанный список заменяется на **оператор задержки вычислений**, который отличается от ранее предлагаемой возвратом только одного значения, тип которого определяется во время компиляции и может быть любым. В языке с динамической типизацией результатом являлся параллельный список. В новой модели выдача вместо параллельного списка роя тоже возможна, но только при явном его задании в качестве результата задержки. Раскрытие задержки осуществляется сразу же после того, как она становится аргументом оператора интерпретации. Это позволяет в ряде случаев использовать данный оператор в качестве скобочного выражения, изменяющего приоритет операций.

2. СТАТИЧЕСКАЯ ТИПИЗАЦИЯ НА УРОВНЕ ДАННЫХ

В отличие от языка ФПП программирования Пифагор, в котором представлены только базовые типы данных, язык программирования Smile имеет развитую систему типов, обуславливаемую необходимостью повышения контроля на этапе компиляции. Вводимые базовые типы данных во многом повторяют типы, используемые в современных статически типизированных языках. Однако помимо этого предлагаются типы, обеспечивающие возможность манипуляции с параллельными списками, что ведет к их определенному влиянию на СТМФППВ.

Выделяются следующие базовые типы: целый, булевский, сигнальный, функциональный, ошибки. Эти типы являются основообразующими и используются не только при обработке произвольных данных, но и в ключевых операторах языка. Дополнительные типы, такие, как действительные числа, символы и другие, рассматриваются в качестве расширений, определяемых проблемной ориентацией, и могут включаться в различные предметно-ориентированные версии языка. В целом можно отметить, что вопросы, связанные с расширением базовых типов, не являются принципиальными на уровне модели вычислений.

К составным типам относятся: массив, структура, кортеж, обобщение, рой, поток, функциональный, ссылочный. Эти типы используются для формирования производных абстракций, определяемых программистом, и состоят как из базо-

вых, так и производных типов. Они в основном подменяют ранее используемые понятия списка данных и параллельного списка, однако при этом являются описаниями, а не операторами, что позволяет на их основе формировать соответствующие хранилища данных, используемые по принципу единственного присваивания. Массив, структура и кортеж являются специализированными разновидностями списка данных ФПМПВ.

Тип «массив» предназначен для описания данных одного типа. Во многом он аналогичен использованию многомерных массивов традиционных императивных языков программирования. Массив имеет фиксированные размерность и длины по каждому измерению. Описание данного типа на уровне языка программирования задается с использованием следующего синтаксиса:

Массив ::= ИмяТипа «(» Размерность «)»

Размерность ::= Целое { «,» Целое }

Примеры массивов:

```
A << type int(100)
B << type bool(30, 40)
```

Тип «структура» обеспечивает группировку разнотипных данных по аналогии со структурными типами различных языков программирования. Структура состоит из полей, каждое из которых имеет имя и тип. Описание структуры имеет следующий синтаксис:

Структура ::= «(» ПолеСтруктуры { «,» ПолеСтруктуры } «)»

ПолеСтруктуры ::= ИмяПоля «@» ИмяТипа

| «[» ИмяПоля { «,» ИмяПоля } «]» «@» ИмяТипа

Примеры структурных типов:

```
Triangle << type (a@int, b @ int, c @int)
Rectangle << type ([x,y]@int)
```

Тип «кортеж» отличается от структуры отсутствием именованных полей. Он аналогичен вектору, но может содержать разнотипные элементы. Обращение к элементам кортежа осуществляется по номеру поля. Для задания кортежей используется следующий синтаксис:

Кортеж ::= «(» ИмяТипа { «,» ИмяТипа } «)»

Примеры задания типов кортежей:

```
C << type (int)
```

```
B << type (int, bool, signal)
```

Тип «обобщение» во многом аналогичен по организации и использованию обобщениям, используемым в императивных языках. Основной его задачей является описание вариантных данных. Существуют различные подходы к организации обобщений, включая методы, поддерживающие полиморфизм. В языке используются обобщения, поддерживающие процедурно-параметрическую парадигму программирования, которая обеспечивает более гибкую поддержку эволюционного расширения программ по сравнению с другими подходами [9]. Правила, определяющие синтаксис обобщений, имеют следующий вид:

Обобщение ::= «{» ПолеОбобщения { «,» ПолеОбобщения } «}»

ПолеОбобщения ::= ИмяТипа { «,» ИмяТипа }

| ИмяПризнака «@» ИмяТипа

| «[» ИмяПризнака { «,» ИмяПризнака } «]» «@» ИмяТипа

Примеры описания обобщений:

```
Figure1 << type {Triangle, Rectangle}
Figure2 << type {trian@Triangle,
                 rect@Rectangle,
                 rhomb@Rectangle}
WeekDay << type{[Sun,Mon,Tue,Wen,Thu,Fri,Sat]@signal}
```

Тип «рой» используется для описания независимых данных, над которыми возможно выполнение массовых параллельных операций. Все элементы роя имеют один тип, а функция, осуществляющая их обработку, может одновременно выполняться над каждым элементом. Результатом является также рой, размерность которого равна размерности роя аргументов. Синтаксические правила, определяющие данный тип, имеют следующий вид:

Рой ::= ИмяТипа «[» Целое «]»

Пример описания типа

```
R << type int[100]
```

Тип «поток» является альтернативой асинхронного списка [10]. Он используется для обработки данных поступающих последовательно и асинхронно в произвольные промежутки времени. Размерность поступающих данных при этом неизвестна, поэтому завершение обработки возможно только по признаку конца потока. Поток готов к обработке при наличии в нем хотя бы одного эле-

мента. Тип всех элементов потока одинаков. Синтаксические правила, определяющие поток:

Поток ::= ИмяТипа «{» «}»

Пример описания потокового типа:

```
A << type int{}
```

Тип «функция» (или функциональный тип) позволяет задать сигнатуру функции, определяя имя типа, тип аргумента, а также тип результата. В целом определение функционального типа отличается от общепринятых только тем, что любая функция имеет только один аргумент и возвращает только один результат. Синтаксические правила, определяющие описание функционального типа:

ФункциональныйТип ::= func Аргумент «->» Результат

Аргумент ::= ИмяТипа | Кортеж

Результат ::= ИмяТипа | Кортеж

Примеры описаний:

```
F << type func int -> int
F2 << type func (bool, int, int) -> (int, bool)
```

Тип «ссылка» (или ссылочный тип) обеспечивает поддержку указателей на различные хранилища определенного типа, что позволяет передавать значения между функциями без их копирования. Основное назначение заключается в дополнительном контроле типов в ходе передач. Синтаксические правила, определяющие описание ссылочного типа:

Ссылка ::= «&» ИмяТипа

ОткрытыйМассив ::= ИмяТипа «(» «» { «,» «*» } «)»*

3. СТАТИЧЕСКАЯ ТИПИЗАЦИЯ ПРИ ОПИСАНИИ ФУНКЦИЙ

В отличие от языка ФПП программирования Пифагор, при описании функций используется явное задание типов аргумента и результата, что обеспечивает дополнительный контроль во время компиляции. Эти изменения затрагивают заголовок функции, что определяется следующим синтаксическим описанием:

Функция ::= func Аргумент «->» Результат ТелоФункции

Аргумент ::= ИмяАргумента «@» (ИмяТипа | Кортеж) | Структура

Результат ::= ИмяТипа | Кортеж | Структура

Примеры:

```
Factorial << func n@int -> int {...}
TrianPerimeter << func ([a,b,c]@int) -> int {...}
Sum << func t@(int, int) -> int {t:+ >> return}
```

4. ДОПОЛНИТЕЛЬНЫЕ РАСШИРЕНИЯ

Наличие статической типизации ведет к появлению дополнительных возможностей по разработке функционально-потоковых параллельных программ. К ним следует отнести описание хранилищ данных предопределенного типа. С ними можно взаимодействовать как с использованием принципа единственного присваивания, так и с применением многократного доступа, аналогичного с доступом к переменным в императивных языках программирования. Последнее ведет к потере надежности программ, но может иногда использоваться после формального доказательства непротиворечивости, например, для ускорения вычислений. Хранилища описываются следующими синтаксическими правилами:

Хранилище ::= (safe | var) Тип

ИменованноеБезопасноеХранилище = ИмяХранилища «@» Тип

Первое правило явно определяет безопасное (safe) хранилище, заполняемое по принципу единственного присваивания (с контролем во время заполнения) или небезопасный вариант (var), при котором не контролируется возможность повторной записи. Второе правило является «синтаксическим сахаром» для описания безопасных хранилищ. Примеры:

```
t1 << safe Triangle ≡ t1@Triangle
t << var Triangle
```

Для взаимодействия с хранилищами необходимо ввести дополнительный набор операций, ведущих к изменению семантики модели вычислений и влияющих на синтаксис языка программирования. Эти операции обеспечивают доступ к хранилищам по чтению и записи. Наличие хранилищ разного типа предопределяет и разнообразие описаний, отражаемых в соответствующих синтаксических правилах:

ЧтениеВсегоХранилища ::= ИмяХранилища «:» Функция

ЧтениеЭлементаМассива ::=

ИмяХранилища «(» индексы «)» «:» Функция

ЧтениеЭлементаКортежаИлиОдномерногоМассива ::=

ИмяХранилища «:» индекс «:» Функция

ЧтениеЭлементаСтруктуры ::=

ИмяХранилища «.» ИмяПоля «:» Функция

ЧтениеЭлементаПотока ::= ИмяХранилища «:» get «:» Функция

Запись_в_хранилище ::= Значение «:» ИмяХранилища

Запись_в_массив ::= Значение «:» ИмяХранилища «(» индексы «)»

Запись_в_кортеж ::= Значение «:» ИмяХранилища «(» индексы «)»

Запись_в_структуру ::= Значение «:» ИмяХранилища «.» ИмяПоля

Запись_в_поток ::= Значение «:» ИмяХранилища

Применение данных операций сопровождается выполнением в надежных хранилищах принципа единственного присваивания и управления по готовности данных.

5. ОСОБЕННОСТИ ИНСТРУМЕНТАЛЬНОЙ ПОДДЕРЖКИ

Добавление в язык статической системы типов ведет к модификации инструментальных средств, обеспечивающих поддержку функционально потокового параллельного программирования [5]. Разработанный язык обеспечивает представление параллелизма на уровне элементарных операций, при котором каждая функция описывает только информационный граф алгоритма без каких-либо управляющих связей. Транслятор преобразует исходный текст функции в промежуточное представление, которое используется для оптимизации существующих зависимостей по различным критериям, а также для построения на его основе управляющего графа, задающего порядок выполнения в соответствии с выбранной стратегией управления вычислениями [11]. Трансформация управляющего графа и его оптимизация позволяют получать стратегии, отличающиеся от управления по готовности данных и учитывающие различные ограничения, свойственные, например, реальным вычислительным системам.

Общая схема, отображающая различные варианты использования предлагаемых инструментальных средств, приведена на рис. 1. В рамках создаваемой среды выделяются следующие подсистемы:

Рис. 1. Состав инструментальных средств, поддерживающих функционально-потоковое параллельное программирование

— транслятор с языка функционально-потокового параллельного программирования в промежуточное представление, называемое реверсивным информационным графом (РИГ);

— генератор управляющего графа (УГ), формирующий граф управления вычислениями;

— событийная машина, обеспечивающая выполнение функционально-потоковых параллельных программ в автоматическом и отладочном режимах, использующая в качестве программы РИГ и УГ;

— средства оптимизации реверсивного информационного графа;

— средства оптимизации управляющего графа;

— средства формальной верификации ФПП программ;

— средства преобразования ФПП программ в программы для других архитектур ПВС.

Транслятор ориентирован на обработку текстовых файлов, каждый из которых содержит один из артефактов языка. Для каждой функции в памяти компьютера порождается реверсивный информационный граф (РИГ), который сохраняется в репозитории функций в текстовой форме. Выбор текстового пред-

ставления для описания РИГ обусловлен тем, что формирование на его основе внутреннего представления в памяти компьютерной системы может быть легко выполнено с помощью простых транслирующих программ. Помимо этого разработчик может легко читать и анализировать оттранслированные функции, рассматривая данную форму как аналог языка ассемблера. В отличие от РИГ языка с динамической типизацией, данный граф содержит дополнительную информацию по типам для каждого узла.

Реверсивный информационный граф, порождённый транслятором, позволяет построить управляющий граф, определяющий выполнение функции. Для этого предназначена специальная утилита, которая формирует УГ, задающий управление вершинами РИГ по готовности данных. УГ сохраняется в репозитории в текстовом виде.

Тестирование и отладка функционально-потоковых параллельных программ на текущем этапе осуществляется специальным интерпретатором (событийной машиной), состоящим из множества событийных процессоров (СП), управляемых менеджером событийной машины. Каждый из этих процессоров (рис. 2) осуществляет обработку только одной функции, запускаемой в отдельном потоке. Выполнение операций внутри функции в настоящий момент осуществляется последовательно за счёт изменения состояния вершин УГ, которые инициируют вычисления в вершинах РИГ.

Функционирование СП осуществляется следующим образом. Исходные сигналы, фиксирующие протекание в системе различных событий и определяемые начальной разметкой УГ, загружаются в очередь, из которой передаются обработчику в соответствии с дисциплиной обслуживания. В простейшем случае это может быть дисциплина FIFO. Обработчик управляющих сигналов анализирует поступившее событие и выбирает указанный в нем узел управляющего графа. На основе анализа состояния узла УГ он может обратиться к связанной с ним вершине информационного графа за кодом выполняемой операции. В случае, когда операция обработки данных должна быть выполнена, происходит обращение к обработчику узлов РИГ, который осуществляет требуемые функциональные преобразования и сохраняет промежуточные результаты. После обработки данных управляющий узел переходит в новое состояние и при необходи-

мости формирует сигнал, передаваемый следующему узлу, который поступает в очередь управляющих сигналов.

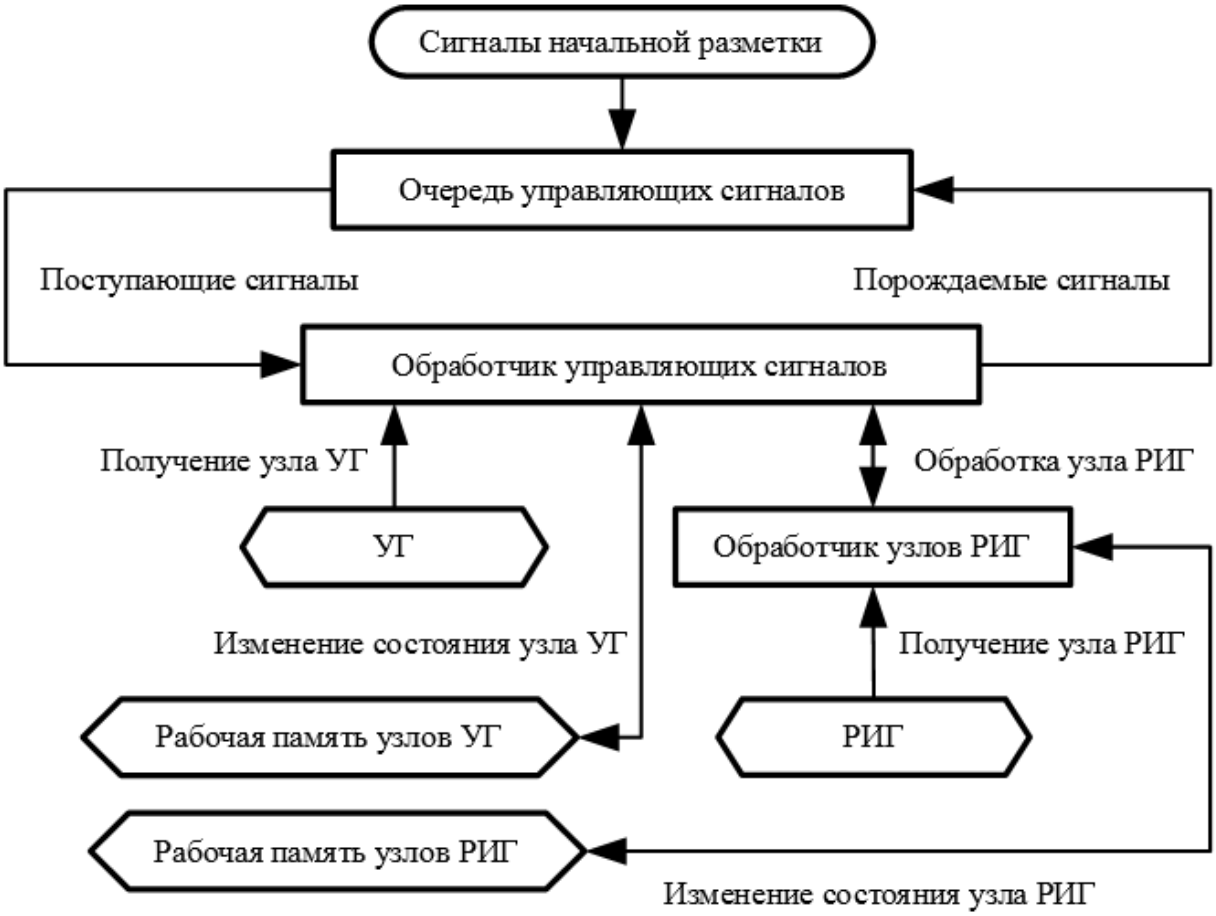

Рис. 2. Обобщённая структура событийного процессора

Основные методы оптимизации, разработанные в настоящее время, затрагивают преобразование промежуточных представлений функционально-потоковых параллельных программ. Они направлены на изменение информационного и управляющего графов. Проводимые преобразования во многом аналогичны методам, используемым при оптимизации исходных текстов программ и их промежуточных представлений в других языках программирования, и предназначены для решения схожих задач. Специфика функционально-потоковой модели параллельных вычислений накладывает свои особенности на реализацию этих методов. Она обусловлена алгеброй эквивалентных преобразований модели, реализованной в языке: информационный и управляющий графы могут изменяться независимо друг от друга. В ходе оптимизации необходимо обеспечивать согласованность РИГ и УГ, однако для многих задач доста-

точно обработки РИГ. В таких случаях оптимизация управляющего графа должна осуществляться после трансформации информационного графа и построении на его основе нового УГ. Следует отметить, что разрабатываемые в настоящее время утилиты не затрагивают распределение реальных вычислительных ресурсов.

Наличие в программе только информационных зависимостей и отсутствие ресурсных ограничений позволяют облегчить формальную верификацию. Основными задачами в данном направлении работ являются: исследование специфики применения формальных методов доказательства корректности; разработка инструментальных средств, упрощающих верификацию. Акцент сделан на доказательство корректности программы с использованием дедуктивного анализа на основе исчисления Хоара [12]. Тройка Хоара представляется как информационный граф программы, к входной и выходной дугам которого привязаны формулы на языке спецификации (предусловие и постусловие). Процесс доказательства корректности программы заключается в разметке дуг информационного графа формулами на языке спецификации, модификации графа и его свёртке. В результате получается несколько информационных графов, у которых все дуги размечены. Каждый из полностью размеченных графов может быть преобразован в формулу на языке логики. Тождественная истинность всех полученных формул свидетельствует о корректности программы. Методы, разработанные для языка Пифагор [13], применимы и к языку со статической типизацией.

Процесс доказательства достаточно трудоёмок, так как требует рассмотрения большого количества различных вариантов графов и преобразований. Поэтому разработаны основные концепции архитектуры инструментального средства для поддержки формальной верификации программ на языке ФПП программирования [14]. Система получает на вход информационный граф программы и формулы предусловия и постусловия на языке спецификации. Она находит неразмеченные дуги графа и помогает с выбором аксиом и теорем, необходимых для их разметки. Весь процесс доказательства представляется в виде дерева, каждый узел которого является частично размеченным графом. Построение дерева завершается, когда все его листья содержат полностью размеченные информационные графы программы. После этого для каждого графа из листа генерируется формула на языке логики. Если все формулы тождественно истинны, то программа корректна.

ЗАКЛЮЧЕНИЕ

Наличие статической типизации в языке функционально-потокового параллельного программирования обеспечивает более строгий контроль данных, что повышает надежность разрабатываемых программ. Также повышается возможности по проведению более полной оптимизации и формальной верификации. Помимо этого трансформация функционально-потоковых параллельных программ в традиционные языки параллельного программирования становится более простой и эффективной, так как большинство типов данных используют практически однозначное отображение.

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

СПИСОК ЛИТЕРАТУРЫ

1. *Левин И.И., Дордопуло А.И., Гудков В.А.* Программирование реконфигурируемых вычислительных узлов на языке COLAMO. Учебное пособие. Таганрог: Изд-во ТТИ ЮФУ, 2011. 114 с.

2. *Дордопуло А.И., Левин И.И.* Ресурсонезависимое программирование гибридных реконфигурируемых вычислительных систем // Суперкомпьютерные дни в России: Труды международной конференции (25–26 сентября 2017 г., г. Москва). М.: Изд-во МГУ, 2017. С. 714–723.

3. *Kasyanov V.* Sisal 3.2: functional language for scientific parallel programming // Enterp. Inf. Syst. 2013. V. 7. No 2. P. 227–236.

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

5. *Legalov A.I., Vasilyev V.S., Matkovskii I.V., Ushakova M.S.* A Toolkit for the Development of Data-Driven Functional Parallel Programmes // Parallel Computational Technologies. PCT 2018. Communications in Computer and Information Science, vol 910. Springer, Cham. P. 16–30.

6. *Vasilev V.S., Legalov A.I.* Loop-invariant Optimization in the Pifagor Language // Automatic Control and Computer Sciences, 2018. V. 52. No 7. P. 843-849.

7. *Ushakova M.S., Legalov A.I.* Verification of Programs with Mutual Recursion in Pifagor Language // Automatic Control and Computer Sciences, 2018. V. 52. No 7. P. 850–866.

8. *Удалова Ю.В., Легалов А.И., Сиротинина Н.Ю.* Методы отладки и верификации функционально-потоковых параллельных программ // Журнал Сибирского федерального университета. Серия «Техника и технологии». Апрель 2011 (том 4, номер 2). С. 213–224.

9. *Legalov A.I., Legalov I.A., Matkovsky I.V.* Instrumental support of the evolutionary expansion of programs using a incremental development // 20th Conf. Scientific Services and Internet, SSI 2018; Novorossiysk-Abrau; Russian Federation; 17–22 September 2018. CEUR Workshop Proc. V. 2260. 2018. P. 346–359.

10. *Легалов А.И., Редькин А.В., Матковский И.В.* Функционально-потоковое параллельное программирование при асинхронно поступающих данных // Параллельные вычислительные технологии (ПаВТ'2009): Труды международной научной конференции, Нижний Новгород, 30 марта – 3 апреля 2009 г. Челябинск: Изд. ЮУрГУ, 2009. С. 573–578.

11. *Легалов А.И.* Об управлении вычислениями в параллельных системах и языках программирования // Научный вестник НГТУ. 2004. № 3 (18). С. 63–72.

12. *Hoare C.A.R.* An axiomatic basis for computer programming // Communications of the ACM. 1969. V. 10. No 12. P. 576–585.

13. *Kropacheva M., Legalov A.* Formal Verification of Programs in the Pifagor Language // Parallel Computing Technologies, 12th International Confernce PACT September-October, 2013. St. Petersburg, Russia. Lecture Notes in Computer Science 7979, Springer, 2013. P. 80–89.

14. *Ushakova M.S., Legalov A.I.* Automation of Formal Verification of Programs in the Pifagor Language // Modeling and Analysis of Information Systems. 2015. V. 22. No 4. P. 578–589.

EVOLUTION OF VISUALIZATION METHODS FOR RESEARCH PUBLICATION COLLECTIONS

A. I. Legalov, I. A. Legalov, I. V. Matkovsky

Siberian Federal University, Krasnoyarsk

legalov@mail.ru

Abstract

It is proposed to add a static system of types to the dataflow functional model of parallel computing and the dataflow functional parallel programming language developed on its basis. The use of static typing increases the possibility of transforming dataflow functional parallel programs into programs running on modern parallel computing systems. Language constructions are proposed. Their syntax and semantics are described. It is noted that the need to use the single assignment principle in the formation of data storages of a particular type. The features of instrumental support of the proposed approach are considered.

Keywords*: visualization of document collections, text analysis, text and metadata visualization algorithms, LDA, NMF, word2vec*

REFERENCES

1. *Levin I.I., Dordopulo A.I., Gudkov V.A.* Programming reconfigurable computing nodes in the COLAMO language. Study guide. Taganrog: Publishing TTI of South Federal Univ., 2011. 114 p.

2. *Dordopulo A.I., Levin I.I.* Resource-independent programming of hybrid reconfigurable computing systems // Supercomputer days in Russia: Proceedings of the international conference (September 25–26, 2017, Moscow). M.: Publishing of Moscow State University. 2017. P. 714–723.

3. *Kasyanov V.* Sisal 3.2: functional language for scientific parallel programming // Enterp. Inf. Syst. 2013. V. 7. No 2. P. 227–236.

4. *Legalov A.I.* Functional language for architecture-independent programming // Computation technologies. 2005. № 1 (10). P. 71–89.

5. *Legalov A.I., Vasilyev V.S., Matkovskii I.V., Ushakova M.S.* A Toolkit for the Development of Data-Driven Functional Parallel Programmes // Parallel

Computational Technologies. PCT 2018. Communications in Computer and Information Science, vol 910. Springer, Cham. P. 16–30.

6. *Vasilev V.S., Legalov A.I.* Loop-invariant Optimization in the Pifagor Language // Automatic Control and Computer Sciences, 2018. V. 52. No 7. P. 843-849.

7. *Ushakova M.S., Legalov A.I.* Verification of Programs with Mutual Recursion in Pifagor Language // Automatic Control and Computer Sciences, 2018. V. 52. No. 7. P. 850–866.

8. *Udalova J., Legalov A., Sirotinina N.* Debug and verification of function-stream parallel programs // J. of SibFU. Engineering & Technologies. 2011. V. 4. No 2. P. 213–224.

9. *Legalov A.I., Legalov I.A., Matkovsky I.V.* Instrumental support of the evolutionary expansion of programs using a incremental development // 20th Conf. Scientific Services and Internet, SSI 2018; Novorossiysk-Abrau; Russian Federation; 17–22 September 2018. CEUR Workshop Proc. V. 2260. 2018. P. 346–359.

10. *Legalov A.I., Redkin A.V., Matkovsky I.V.* Dataflow Functional Parallel Programming using asynchronously incoming data // Parallel Computing Technologiws (PCT'2009): International Conf. 2009. P. 573–578.

11. *Legalov A.I.* About computation control in parallel system and programming languages // NGTU's Science Messenger. 2004. No 3 (18). P. 63–72.

12. *Hoare C.A.R.* An axiomatic basis for computer programming // Communications of the ACM. 1969. V. 10. No 12. P. 576–585.

13. *Kropacheva M., Legalov A.* Formal Verification of Programs in the Pifagor Language // Parallel Computing Technologies, 12th International Confernce PACT September-October, 2013. St. Petersburg, Russia. Lecture Notes in Computer Science 7979, Springer. 2013. P. 80–89.

14. *Ushakova M.S., Legalov A.I.* Automation of Formal Verification of Programs in the Pifagor Language // Modeling and Analysis of Information Systems. 2015. V. 22. No 4. P. 578–589.

СВЕДЕНИЯ ОБ АВТОРАХ

ЛЕГАЛОВ Александр Иванович – профессор Сибирского федерального университета. Сфера научных интересов – программная инженерия, системное программирование, параллельное программирование, языки и системы программирования, компиляторы

Alexander Ivanovich LEGALOV – Professor of Siberian Federal University. Research interests include software engineering, system programming, parallel programming, programming languages, compilers.

email: legalov@mail.ru

ЛЕГАЛОВ Игорь Александрович – доцент Сибирского федерального университета. Сфера научных интересов – языки программирования, компиляторы, web программирование.

Igor Alexandrovich LEGALOV – Associate Professor of Siberian Federal University. Research interests include programming languages, compilers, web programming.

email: igor@legalov.ru

МАТКОВСКИЙ Иван Васильевич – старший преподаватель Сибирского федерального университета. Сфера научных интересов – системное программирование, параллельное программирование, языки программирования, компиляторы

Ivan Vasilievich MATKOVSKY – Senior Lecturer of Siberian Federal University. Research interests include system programming, parallel programming, programming languages, compilers.

email: alpha900i@mail.ru

Материал поступил в редакцию 5 ноября 2019 года