**Статически типизированная версия языка   
функционально-потокового   
параллельного программирования**[[1]](#footnote-1)\*

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

Сибирский федеральный университет

Предлагается расширение языка функционально потокового параллельного программирования включением в него статической системы типов. Это ведет к изменению функционально-потоковой модели параллельных вычислений и, как следствие, методов трансформации, а также подходов к использованию функционально-потоковых параллельных программ. Проводится обзор модели вычислений и операторов языка программирования, сформированных в результате проведенных изменений. Отмечается, как эти изменения влияют на синтаксис и семантику. В частности, данные, формируемые ранее динамически во время выполнения программы, заменяются программными объектами, характерными для статически типизированных языков программирования, что позволяет обеспечить их идентификацию уже на этапе компиляции. Вместо списков данных предлагается использовать структуры, кортежи и многомерные массивы фиксированной размерности. Параллельный список, обеспечивающий выполнение массовых параллельных операций, трансформируется в рой фиксированного размера, все элементы которого имеют единый тип. Типы данных, поступающих на вход оператора интерпретации, также становятся известными во время компиляции. Обосновывается необходимость добавления принципа единственного присваивания, обусловленного формированием хранилищ конкретного типа. Упрощается алгебра эквивалентных преобразований, что также повышает эффективность трансформации архитектурно-независимых параллельных программ в программы, исполняемые на реальных вычислительных системах.

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

# 1. Введение

Современные методы разработки параллельных программ ориентированы на использование специфики конкретных архитектур параллельных вычислительных систем (ПВС), что находит соответствующее отражение в моделях параллельных вычислений и реализованных на их основе языках программирования, а также в библиотеках функций или классов. Практически любой перенос программы с одной архитектуры на другую ведет к переписыванию и модификации уже разработанного и отлаженного кода. Попыткой преодолеть это является применение концепции архитектурно-независимого параллельного программирования (АНПП), ориентированной на разработку программ с использованием языковых и инструментальных средств для абстрактных или виртуальных систем с неограниченными вычислительными ресурсами. Зачастую при этом используются стратегии управления вычислениями по готовности данных. Такие подходы развиваются в разных направлениях. Концепция ресурсно-независимого параллельного программирования реализована в языке COLAMO, ориентированного на разработку систем на кристалле [1, 2]. Создание универсальных языков, напрямую не связанными с архитектурными ограничениями, можно проследить на примере функциональных языков параллельного программирования Sisal [3] и Пифагор [4].

Попытка применить АНПП в наиболее общей форме привело к созданию языка функционально-потокового параллельного программирования Пифагор. Концепция непосредственно учитывается в его модели функционально-потоковых параллельных вычислений. Модель программы определяется как ресурсно-неограниченный ациклический безусловный граф, в котором управление осуществляется по готовности данных, а также используется принцип единственного использования вычислительных ресурсов [4]. Предполагается, что для выполнения любых операций выделяются свои уникальные ресурсы, реальное распределение которых осуществляется после того, как разработана, верифицирована 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 представленные в модели вычислений операторы обладают динамическим поведением, позволяя формировать и модифицировать 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 использования 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 ресурсных ограничений, учитывающих особенности конкретной архитектуры, с сохранением уже отлаженной общей логики функционирования. Зачастую эти преобразования осуществлялись уже после компиляции исходного текста программы в промежуточное представление и были связаны с ручными наложением конкретных типов данных и фиксацией размерности используемых списков.

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.

# 2. Описание типов 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.

Можно выделить 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 как расширения, определяемые 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 данных, определяемых программистом, и состоят как из базовых, так и производных типов. Они в основном подменяют ранее используемые понятия списка данных и параллельного списка, однако при этом являются описаниями данных, 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»** (array) предназначен для описания группы данных одного типа. Во многом он аналогичен использованию многомерных массивов традиционных императивных языков программирования. Массив имеет фиксированные 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. По сути он аналогичен одномерному массиву, 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:

**С << type (int)**

**В << type (int, bool, signal)**

Тип **«обобщение»** во многом аналогичен по организации и использованию обобщениям, используемым в императивных языках. Основной его задачей является описание вариантных структур. 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 описание 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 асинхронному списку [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 элемента. 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. Хранилища

Появление статической типизации ведет к дополнительным возможностям по разработке функционально-потоковых параллельных программ. К ним следует отнести описание хранилищ данных предопределенного типа, с которыми можно взаимодействовать как с использованием принципа единственного присваивания, так и с применением многократного доступа, аналогичного с доступом к переменным в императивных языках программирования. Использование многократного присваивания ведет к потере надежности программ, но может иногда использоваться, например, после формального доказательства непротиворечивости данной операции. Это позволяет в ряде ситуаций ускорить вычисления. 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**

# 4. Описание 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}**

# 5. Операторы статически типизированной модели вычислений

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

* эффективную трансформацию статически типизированных функционально-потоковых параллельных программ в другие модели вычислений вместо выполнения программ в режиме интерпретации;
* повышение контроля семантики операторов за счет использования сильной типизации;
* сохранение принципа управления по готовности данных и общей концепции функционально потоковой модели параллельных вычислений;
* каждый из операторов языка должен опираться на типизированные данные, контролируемые на этапе компиляции;
* контейнерные (списковые) данные, формируемые в ходе вычислений, 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 операторов СТМФППВ.

## 5.1 Оператор интерпретации

**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 неименованных структур и кортежей допускается структурная эквивалентность;
* допускается явное и взаимное приведение структур и кортежей, распознаваемое во время компиляции;
* 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.

## 5.2 Оператор группировки в кортеж

**Вместо группировки в список данных** в СТМФППВ используется **группировка в кортеж** (tuple).

Можно выделить следующие основные свойства оператора группировки в кортеж:

* 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;
* для неименованных структур и кортежей допускается неявное сравнение на структурную эквивалентность;
* допускается структурная эквивалентность с неименованными структурами;
* 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.

## 5.3 Оператор группировки в рой

**Группировка в параллельные списки** заменяется на группировку в рой (swarm). 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.

## 5.4 Оператор задержки

**Группировка в задержанный список** заменяется на оператор задержки вычислений (delay), который отличается от ранее используемой версии возвратом только одного значения, тип которого определяется во время компиляции и может быть любым. В языке с динамической типизацией результатом являлся параллельный список, обеспечивающий возврат множества независимых величин. 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.

## 5.5 Операции с хранилищами данных

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.

# 6. 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.

# Литература

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. C. 714–723.
3. Kasyanov V. Sisal 3.2: functional language for scientific parallel programming // Enterp. Inf. Syst. 2013. V ol. 7. No 2. P. 227–236. DOI: 10.1080/17517575.2012.744854
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. DOI:  10.1007/978-3-319-99673-8\_2.
6. Vasilev V.S., Legalov A.I. Loop-invariant Optimization in the Pifagor Language // Automatic Control and Computer Sciences, 2018. Vol. 52. No. 7. P. 843 849. DOI: 10.3103/S0146411618070295.
7. Ushakova M.S., Legalov A.I. Verification of Programs with Mutual Recursion in Pifagor Language // Automatic Control and Computer Sciences, 2018. Vol. 52. No. 7. P. 850–866. DOI: 10.3103/S0146411618070301
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 (Т. 4, № 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. Vol. 2260. 2018. P. 346–359. DOI: 10.20948/abrau-2018. URL: http://keldysh.ru/abrau/2018/proc.pdf.
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.

1. \* Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 17-07-00288. [↑](#footnote-ref-1)