Разработка языка программирования для dataflow вычислений

С.М. Салибекян

Национальный исследовательский университет «Высшая школа экономики»

Фокус внимания статьи сосредоточен на описании разработанного автором языка программирования (ОА язык), «природно» ориентированного на описание алгоритма в dataflow парадигме вычислительного процесса: описание обмена данными в виде токенов между устройствами вычислительной системы. Язык обладает рядом преимуществ: предельная простота синтаксиса, что ускоряет процесс программирования; возможность описывать сложных структур данных, например, семантические сети, и обрабатывать их; управление как аппаратной dataflow системой, так и виртуальной, которую можно запустить на ЭВМ классической архитектуры; является мультипарадигменным.

Ключевые слова: dataflow, параллельные вычисления, параллельный язык программирования, высокопроизводительные вычисления, распределенные вычислительные системы.

1. Введение

Настоящая статья посвящена описанию языка программирования (объектно-атрибутный (ОА) язык), созданного для описания алгоритма в dataflow вычислительной системе (ВС) [1]. Отличительной особенностью языка является то, что его синтаксис максимально адаптирован для описания именно в dataflow парадигме вычислений, тогда как большинство языков для подобных целей относятся к функциональной парадигме (Sisal [2], VAL, ID, ПИФАГОР [3]). Язык позволяет управлять dataflow системами, реализованными как на аппаратном, так и на программном уровнях. Его отличительными особенностями являются: возможность настройки устройств, входящих в вычислительную систему; описание сложно структурированных информационных конструкций; трансформация информационных конструкций; описание алгоритма для распределенной ВС. В настоящее время имеется реализация транслятора данного языка. Транслятор создается методом раскрутки, когда первая версия транслятора реализует самую примитивную функциональность языка, затем на созданном языке пишется следующая версия транслятора языка с бОльшей функциональностью и т.д. В настоящее время происходит реализация уже третьей версии ОА-языка.

Следует отметить, что ОА-язык предназначен для управления вычислениями в dataflow ВС, относящейся к классу одноуровневых машин Венну [4]. Т.е. токены (операнды, снабженные служебной информацией) поступают непосредственно на исполнительные устройства. Исполнительные устройства производят накопление в своих внутренних регистрах полного комплекта данных для осуществления определенной операции над ним. ОА ВС, реализованная программным способом, т.е. в ней исполнительные устройства представляются в виде программных блоков, несколько напоминает акторную систему программирования [5, 6]. Актор – это программная функция, которая, однако, получает свои операнды не одновременно, как в классическом программировании, а последовательно в виде токенов. У каждого актора существует интерфейс, каждый слот которого описывает входной операнд или выходное данное. В ОА-языке аналогом актора является функциональное устройство (ФУ). Однако, существует одно отличие – это универсальный интерфейс ФУ. Он принимает только по одному операнду в виде токена, а распознание операнда осуществляется по атрибуту токена. Таким образом значительно упрощается программирование ОА системы, т.к. избавляет программиста от рутинного описания интерфейса функции актора.

В области создания dataflow языков программирование проделана достаточно серьезная работа. В качестве примера привести статью [7]. Описанный в статье язык предназначен для решения ограниченного класса задач, однако относится, именно, к парадигме dataflow. В статье [8] представлен графический язык для dataflow системы. К потоковой парадигме также можно отнести практически все графические языки программирования (например, LabView и Simulink), т.к. они строятся на базе потокового графа [9], который описывает процесс пересылки токенов между различными вычислительными устройствами ВС.

ОА язык изначально разрабатывался для описания вычислительного процесса в ВС объектно-атрибутной (ОА) архитектуры [10], относящейся к классу dataflow. Однако он может применяться и для других dataflow ВС, относящихся к классам одноуровневых и двухуровневых машин по классификации Венна [4].

Вторая глава статьи посвящена описанию синтаксиса ОА-языка, в том числе и его последний, разрабатываемой в настоящее время, версии. Внимание третьей главы фокусируется на среде ОА программирования и моделирования, предназначенной для создания и запуска программ на ОА языке. В четвертой главе приведены примеры решения различных прикладных вычислительных задач с применением ОА языка.

2. ОА язык: основы

2.1 Синтаксис ОА языка

Основы ОА языка были представлены на конференции ПаВТ 2013 [11]. Философия языка следующая. Весь вычислительный процесс реализуется с помощью набора функциональных устройств (ФУ); по названию ясно, что данное устройства реализует какую-то вычислительную функцию (например, арифметико-логические операции, ввод-вывод на консоль или передача данных по вычислительной сети). Они принимают токены с операндами, накапливают в своих внутренних регистрах комплект данных, необходимый для осуществления операции, производят операцию и пересылают результат своих вычислений на другие ФУ для дальнейшей обработки. Токены в ОА языке именуются милликомандами (МК). Такое название пошло от того, что для осуществления классической команды необходим приход на ФУ нескольких токенов с исходными данными, т.е. одна МК является частью классической команды.

МК в ОА языке обозначается с помощью знака «=», с правой стороны от которого находится описание атрибута МК, слева – нагрузки (т.е. данных, прикрепленных к токену). К атрибуту данных может так же добавляться и имя ФУ, которому направляется МК. Название ФУ отделяется от атрибута данных знаком «.». Например, выполнение операции сложения на ОА языке будет выглядеть следующим образом: ALU.Sum1=2 ALU.Sum2=2 ALU.Out=y, где y – мнемоника переменной для записи результата вычисления, ;ALU – мнемоника ФУ арифметико-логическое устройство (АЛУ). МК Sum1 служит для индикации первого слагаемого, Sum2 – для индикации второго, по МК Out осуществляется вывод результата вычислений, полученный на ФУ АЛУ. Следует отметить, что последовательность прихода МК Sum1 и Sum2 может быть любой, однако Out должна появиться только после вычисления результата. Для того, чтобы выдача результата производилась автоматически по получению результата можно воспользоваться МК OutMkSet – установить МК для результата. Данная МК записывает атрибут, который прикрепляется к результату вычислений во время его выдачи. Например, мы можем выполнить ALU.OutMkSet=Console.Print, где Console.Print – МК вывода на консоль. Теперь программка преобразуется в ALU.OutMkSet=Console.Out ALU.Sum1=2 ALU.Sum2=2. Тогда при выполнении данной последовательности МК ФУ, получив два необходимых операнда, осуществит выдачу МК с результатом автоматически. Результат будет оформлен в виде МК с атрибутом, установленным ранее (в нашем случае Console.Print), которая будет иметь вид: Console.Print=Rez, где Rez – результат вычисления, полученный АЛУ. Сгенерированная МК передана выредается на ФУ-консоль, которая осуществит вывод данных.

Последовательность МК может объединяться в так называемую информационную капсулу (ИК). ИК хранит к себе упорядоченную последовательность МК. Например, ИК может хранить программу обработки ошибки, которую может зафиксировать ФУ. Указатель на ИК с програм-

мой может быть записан во внутренние регистру ФУ с помощью МК ErrProgSet. Например, фрагмент программы может выглядеть так: ALU.ErrProgSet={Console.Print="Arithmetic error" ALU.Stop}. Данная программа будет запушена при возникновении ошибки при вычислениях, выполняемых ФУ АЛУ. В ОА языке под выполнением программы подразумевается последовательная выдача МК из ИК. В результате запуска на консоль будет выведено сообщение об ошибке, а таже произведена остановка работы ФУ АЛУ.

В ОА ВС имеется набор типов ФУ. Например, в разработанной среде ОА программирования и моделирования имеется порядка 25 типов ФУ. ФУ каждого типа может обрабатывать токены с определенным набором МК. Создание ФУ в ОА языке происходит по МК NewFU={Mnemo="ALU1" FUType=FUStreamFloatALU}, где Mnemo – атрибут мнемоники ФУ, FUType – атрибут типа ФУ. Так по вышеприведенной МК будет создано ФУ с именем «ALU1» и типом FUStreamFloatALU (дробное потоковое АЛУ). Вышеприведенный синтаксис составляет ядро ОА языка, на котором можно создавать программы.

ОА язык, например, позволяет легко описать вычисления, заданные посредством потокового графа [12] (операции в узлах потокового графа могут быть как мелкозернистыми (т.е. элементарные операции), так и крупнозернистыми (крупнозернистая операция может быть эквивалентна программе с нескольким сотнями или тысячами строк). Пусть, например, нужно вычислить арифметическое выражение следующего вида: $Y=A*x^2 + B*x + C$. Потоковый граф для данного выражения приведен на рис. 1. Реализация данного графа основывается на применении ФУ типа потоковое АЛУ. Его особенность состоит в том, что оно работает в потоковом режиме: принимает операнды в любой последовательности прихода, выполняет только одну операцию за время вычислительного процесса, автоматически выдает результат вычислений.

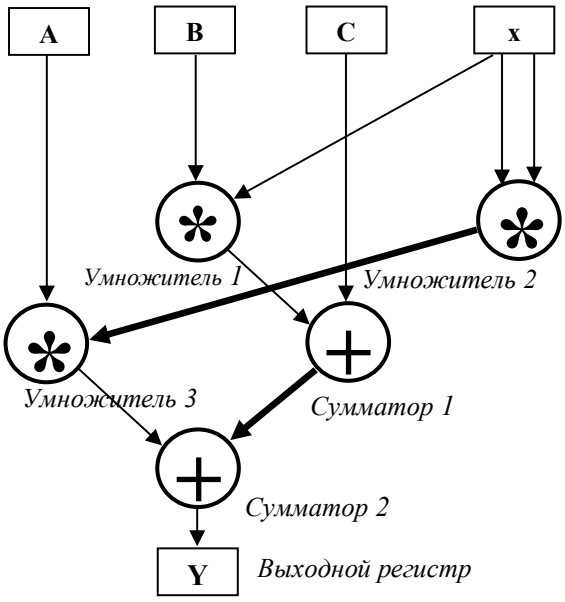

Рис. 1. Пример потокового графа

В ОА языке для реализации потокового графа на рис. 1 необходимо задать одну переменную, три константы, 6 ФУ. Программа с комментариями к коду приведена на рис. 2.

Программа на листинге выше начинается с команд создания ФУ. Пять из них имеют тип FUStreamFloatALU (потоковое АЛУ). Шестое – тип FUConsole (консоль ввода-вывода), которое служит для вывода результата вычислений. Далее идет секция инициализации переменных и описания именованных констант. Затем расположена секция описания пересылок данных между ФУ. МК ReceiverMkSet устанавливает для ФУ код атрибута МК с результатом выполнения арифметико-логической операции. Например, Mul1.ReceiverMkSet=Sum1.Add программирует ФУ Mul1 на то, чтобы оно выдало результат своих вычислений с атрибутом Sum1.Add – в итоге данное число будет передано на ФУ Sum, а по атрибуту Add ФУ Sum1 поймет, что данный операнд является слагаемым. Аналогичным образом происходит настройка пересылок и между другими ФУ. А для ФУ, которое получает окончательный результат, приемщиком является консоль, выводящая этот результат на экран (МК Console.Print).

```
NewFU={Mnemo="Sum1" FUType=FUStreamFloatALU}// Создание виртуального ФУ
NewFU={Mnemo="Sum2" FUType=FUStreamFloatALU}
NewFU={Mnemo="Mul1" FUType=FUStreamFloatALU}
NewFU={Mnemo="Mul2" FUType=FUStreamFloatALU}
NewFU={Mnemo="Mul3" FUType=FUStreamFloatALU}
NewFU={Mnemo="Console" FUType=FUConsole}

// Инициализация переменных и констант
x=0.3 // Инициализация переменной
A#5   // Константа
B#-2  // Константа
C#28  // Константа

// Описание пересылок токенов
Mul1.ReceiverMkSet=Sum1.Add // Описание пересылки операнда из Умн1 на Сум1
Mul2.ReceiverMkSet=Mul3.Mul // из Умн2 на Умн3
Mul3.ReceiverMkSet=Sum2.Add // из Умн3 на Сум2
Sum1.ReceiverMkSet=Sum2.Add // из Сум1 на Сум2
Sum2.ReceiverMkSet=Console.Print // Выдача результата с Сум2 на консоль

// Выдача исходных данных
Mul1.Mul=B
Mul1.Mul=x
Mul2.Sqr=x
Mul3.Mul=A
Sum2.Add=C
```

Рис. 2. Реализация ОА программы для потокового графа на рис. 1

Только после настройки пересылок данных между ФУ в ОА ВС можно подать исходные данные. Данные снабжаются соответствующими атрибутами, и сформированные таким образом МК выдаются на ФУ для обработки. Например, когда оба операнда поступают на ФУ Mul1, оно вычисляет их произведение, оформляет его в виде токена с атрибутом Sum1.Add и выдает МК, которая передается на ФУ Sum1 и порождает дальнейшие вычисления. Если во время выполнения вычислительного процесса данные придут сразу на несколько ФУ, то ФУ будут работать параллельно.

Как видно из приведенного примера кода ОА программы, синтаксис ОА языка весьма прост, однако он представляет собой весьма мощный инструмент параллельного программирования, т.к. позволяет описывать программы с самораспараллеливанием вычислений. Дело в том, что в dataflow парадигме, синхронизация вычислений происходит по данным: как данные поступили на исполнительное устройство и набрался полный комплект для выполнения операции, операция начинает выполняться, а результат выполнения пересылается на другие ФУ. Заметим, что программист не заботится о распараллеливании вычислительного процесса, процесс как бы сам распараллеливается в зависимости от приходящих данных.

2.2 Дальнейшее развитие ОА языка

В листинге на рис. 2 приведена программа на второй версии ОА языка (компилятор реализуется с 2008 года). А в настоящее время проводится работа по реализации третьей версии, обладающей более широким набором синтаксических конструкций. Расширение функциональности ОА языка позволит значительно снизить трудоемкость создания ОА программы. Отметим, что компилятор ОА языка сам пишется на ОА языке, т.к. ОА язык, помимо прочего, оказался весьма удобным в том числе и для создания трансляторов языков высокого уровня [13]. Перечислим нововведения, которые ожидаются в третьей версии.

Во-первых, арифметико-логические выражения. Заметим, что во второй версии была возможность присваивать переменным только константные значения, а также записывать в них результаты вычислений с ФУ. Если же была необходимость вычислить какое-то значение, то это

приходилось делать с помощью ФУ АЛУ и вручную прописывать алгоритм вычисления выражения. Во-вторых, введение возможности работы с векторами. В-третьих, добавление высокоуровневых императивных конструкций типа if-then-else, while, for. Отметим, что императивные конструкции в основном предполагают последовательное выполнение вычислений, что противоречит основной задачи dataflow – максимальному распараллеливанию. Однако в некоторых случаях последовательные вычисления приемлемы, например, в случае, если алгоритм по своей природе последовательный и его невозможно распараллелить, или, если вычислительный процесс удобно разбить на несколько параллельных вычислительных нитей, каждая из которых выполняется последовательно. К тому же, такие высокоуровневые синтаксические конструкции значительно упрощают код программы и делают его более читабельным. В-четвертых, в синтаксис добавляется много разных приятных мелочей, упрощающих создание кода.

В качестве примера таких мелочей можно привести возможность создания вектора ФУ. Каждое ФУ в этом векторе имеет свой индекс, по которому к нему можно обращаться. Например, ALU[3].Add=10, т.е. мы пересылаем константу 10 в качестве слагаемого для АЛУ с индексом 3. Если же возникает необходимость передать МК сразу нескольким ФУ, то можно указать сразу несколько ФУ или диапазон индексов ФУ, например, ALU[2,6,9].Set=0 (установить в нуль аккумуляторы ФУ с индексами 2, 6, 9) или ALU[3..7].Set=0 (установить в нуль аккумуляторы ФУ с индексами от 3 до 7). Также можно создать и матрицу ФУ, тогда каждое ФУ будет адресовать сражу двумя индексами, 3-мерный массив АЛУ и т.д. Такая возможность значительно облегчит описание вычислительных сеток, состоящих из ФУ (подробнее о вычислительных сетках будет рассказано в главе 4). Также полезным нововведением в синтаксис является применение табуляции в качестве программных скобок (аналогично языку Python).

В синтаксис ОА языка добавляются следующие императивные конструкции:

– Тернарная конструкция [14]. Данная условная конструкция может быть вставлена в арифметико-логическое выражение.

– Оператор бинарного ветвления if. Синтаксис оператора следующий: if условие{прогр. блок 1} elif {прогр. блок 2} else {прогр. блок 2}.

– Операторы цикла for и after. Параметры цикла записываются в трех секциях, разделенных знаком «:». Синтаксис оператора for: for секция 1: секция 2 : секция 3 {тело цикла}. В секции 1 указывается начало диапазона генерируемой числовой последовательности, в секции 2 шаг последовательности, в секции 3 конец последовательности. Например, for 2:10:2{ генерирует последовательность чисел 2, 4, 6, 8 (правый диапазон, аналогично питоновскому range в диапазон не входит). В секциях также можно указывать условие продолжения цикла. Например, for i=0: i<x+y: i++{…}, тогда цикл становится аналогом циклу while языка С. Оператор after аналогичен for и отличается тем, что это – цикл с постусловием: первая итерация цикла выполняется безусловно и только со второй итерации начинается проверка условия продолжения цикла.

В дальнейшем предполагается расширение ОА языка возможностью программирования распределенных ВС. Так, появится функциональность описания секций программы, каждую из которых можно будет приписать к определенному вычислительному узлу распределенной ВС. Также предполагается создание различных средств автоматического распределения сегментов программы по различным вычислительным узлам.

3. ОА среда программирования и моделирования

Для программирования на ОА языке была создана среда ОА программирования и моделирования. Она представляет собой совокупность текстового редактора кода, различных инструментальных средств, компилятора ОА-языка. Также среда позволяет создавать виртуальную реализацию ОА ВС, на которой и происходит выполнение алгоритма, описанного на ОА языке (рис. 3).

Создание среды ОА программирования началось с 2008 года. Среда первой версии позволяла производить трансляцию программы на ОА языке во внутреннее представление программы для ОА ВС, и в таком виде она могла быть запущена на виртуальной реализации ОА ВС. Среда предоставляла программисту текстовый редактор, консоль вывода данных, различный полезный инструментарий для программирования: контекстное меню, всплывающие подсказки, предложение вариантов быстрого ввода синтаксических конструкций и т.д.

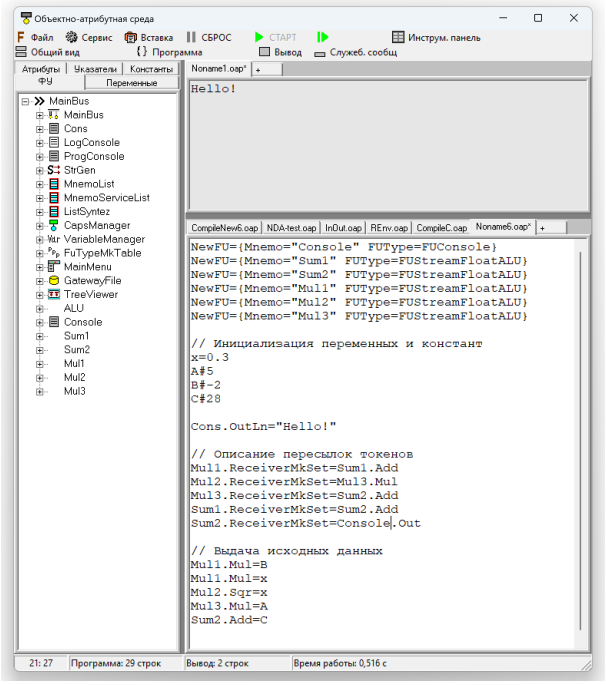

Рис. 3. Среда ОА программирования и моделирования

Вторая версия среды отличается более развитым компилятором ОА-языка, позволяющим создавать полноценные описания алгоритмов. В настоящее время рабочей является вторая версия ОА среды, хотя уже ведется реализация версии третьей на базе редактора VS Code. У третьей версии функциональность будет расширена следующими возможностями: подкраска текста программы, возможность создания многомодульных программ, подключение к git. К тому же, использование уже готового функционала редактора VS Code значительно улучшит качество работы в данном редакторе.

Среда ОА программирования и моделирования предоставляет пользователю обширные возможности по вводу-выводу и визуализации данных, имитационному моделированию вычислительного процесса. Так, для имитационного моделирования в состав виртуальной ОА-ВС входят специальные ФУ, с помощью которых можно задавать параметры параллельной ВС и параметры процесса моделирования: количество исполнительных устройств, конфигурация распределенной ВС, время выполнения вычислительных операций и пересылки данных. Имитационная модель ОА ВС строится на безе процессной сети Кана [15].

Среда второй версии была использована для моделирования выполнения многочисленных вычислительных задач на базе ОА ВС: перемножение матриц, алгоритм кластеризации данных Mean Shift [16], разностная схема решения дифференциальных уравнений, решение задачи «рюкзака» методом динамического программирования, семантический анализ естественного языка и т.д. Благодаря такому моделированию были оценены сложности решения такого типа задач на ОА ВС.

4. Примеры описания решений вычислительных задач на ОА языке

4.1 Задача «рюкзака»

Задача «рюкзака» формулируется следующим образом. Имеется рюкзак ограниченного объема V и множество из N предметов. Каждый предмет имеет два атрибута: объем v_i и цена p_i. Пусть набор целых чисел $\omega = \{\omega 1,\omega 2,...,\omega N\}$ есть набор объемов предметов, $\rho=\{\rho 1,\rho 2,...,\rho N\}$ есть набор стоимостей предметов. Нужно найти набор бинарных величин $\beta=\{\beta 1,\beta 2,...,\beta N\}$, где $\beta i=1$, если предмет n_i включен в набор, $\beta_i=0$, если предмет n_i не включен, и такой, что:

1. $\beta_1\omega_1+...+\beta_N\omega_N \le V$
2. $\beta_1\rho_1+...+\beta_N\rho_N$ есть максимальна.

Т.е. суммарный объем предметов не должен превышать объем рюкзака, а суммарная стоимость всех предметов, поместившихся в рюкзак, должна быть максимальной. Существует несколько методов решения данной задачи, например, жадный и генетический алгоритмы, методы ветвей и границ. Однако наше внимание привлек метод решения с помощью динамического программирования [17], т.к. он дает точный результат, имеет небольшую вычислительную сложность, хорошо подходит для решения именно на базе ОА ВС. О других методах решения различных вычислительных задач посредством динамического программирования можно узнать в [18].

Методика решения задачи рюкзака с помощью динамического программирования следующая. Строится 2-мерное фазовое пространство решений задачи W размерностью N x V, где V – множество дискретных значений объема, занятого предметами, помещенными в рюкзак. В этом пространстве по столбцам откладывается возможные значения объем рюкзака, а строка обозначает номер добавляемого предмета. С каждой точкой фазового пространства W_{ij} связана величина, показывающая максимальную стоимость предметов от 1 до *i*, занимающих объем *j* единиц рис. 4. Наклон косых линий по оси x перед линией *i* равен объему *i*-го предмета, который на данном шаге добавляется в рюкзак.

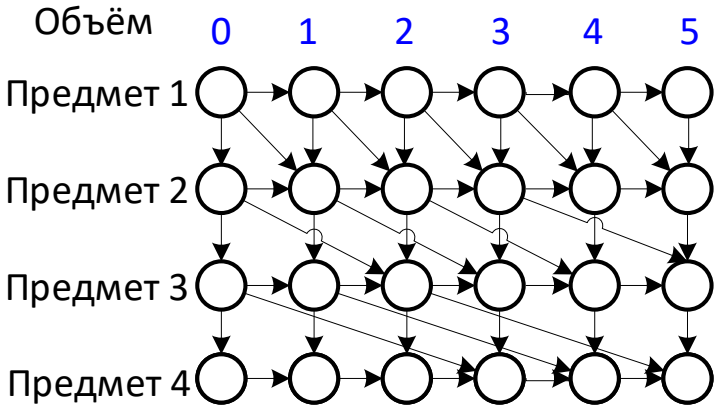

Рис. 4. Среда ОА программирования и моделирования

В начале выполнения алгоритма значения во всех точках фазового пространства равны 0. Заполнение пространства значениями идет сверху вниз. На каждой строке значение узла сетки вычисляется по формуле:

$$W[i,k]=\max(W[i-1, k-\omega[i]]+p[i], W[i-1, k]),$$

где W – множество точек фазового пространства;

i – номер добавляемого предмета;

k – объем рюкзака;

p[i] – цена i-го предмета;

ω [i] – объем i-го предмета.

Причем, значение i лежит в пределах от v_i до N, т.к. объем рюкзака меньший v_i предметом занят быть не может.

Решением задачи будет максимальное значение, находящееся на последней строке фазового пространства W[N]. Т.е. max(W[N, k]), n=1…NV есть решение задачи.

Где NV – количество разбиений объема рюкзака,

N – количество предметов.

Для решения данной задачи была сгенерирована вычислительная сетка, состоящая из устройств типа дробное потоковое АЛУ. Данный тип ФУ адаптирован к работе в составе вычислительной сетки. Для организации и управления сеткой применялось ФУ менеджер АЛУ.

Программа писалась на второй версии ОА языка, где нет описаний циклов, поэтому циклические действия организовывались с помощью ФУ менеджер, который обладает функциональностью по организации циклов.

На рис. 5 приведен фрагмент ОА программы, ответственный за формирование горизонтальных связей в вычислительной сетке. Связи между ФУ «налаживаются» посредством записи указателя на соседа во внутренний регистр ФУ. Данная ссылка используется ФУ для отправки информации соседнему ФУ или запроса информацию у соседнего ФУ.

```
    // Горизонтальные связи в сетке
IntAlu1.Set=V
IntAlu1.Dec // Количество итераций по одной строке V-1
Manager.ProgSet={ // Тело цикла, выполняемого менеджером
  FUCellularAutomatManager.Neitborder1To2Append // Утановить МК для данных
установить ссылку между соседями
  FUCellularAutomatManager.NeitborderMк1Append=FUCellularAutomat.In_0_Set
// Утановить МК для данных
}
IntAlu.For={ // Цикл по строкам
  IntAlu.OutMk=Manager.Ind1Set=Manager.Ind2Set//Задать индексы соседних ФУ
  Manager.Ind2Add=1 // Задать индексы соседних ФУ
  IntAlu1.OutMk=Manager.ForExec // Выполнить цикл по стобцам строки
  Manager.Neitborder1Append // Добавить нулевую ссылку для последнего ФУ
  Manager.NeitborderMк1Append=0 // Добавить нулевую ссылку для последнего ФУ
}
```

Рис. 5. Фрагмент ОА программы для генерации вычислительной сетки для решения задачи рюкзака

Вкратце объясним работу этого фрагмента программы. Здесь генерируются горизонтальные линии вычислительной сетки. Для этого организуется вложенный цикл, где цикл внешний производит обход всех строк сетки, а внутренний «бежит» по строке и настраивает связь между соседними ФУ (ссылка от левого соседа правому). Для налаживания связи необходимо не только передать ссылку на соседа, но и установить МК, прикрепляемое к передаваемым данным, чтобы приемник мог эти данные идентифицировать.

После настройки всей вычислительной сетки начинается выдача исходных данных, исходные данные порождают вычислительную волну, которая проходит по сетке. По окончанию вычислений значения с последней строки сетки передаются на ФУ АЛУ, которое определяет максимум из них. Это значение является максимальной суммарной стоимостью предметов, которые можно поместить в рюкзак.

4.2 Кластеризация данных методом Wean Shift

Представим реализацию еще одной вычислительной задачи – кластеризация данных методом Mean Shift (сдвиг среднего значения). Алгоритм решения этой задачи на базе ОА ВС описан в [19]. Сейчас же речь пойдет о программной реализации данной задачи с помощью ОА языка.

Итак, алгоритм Mean Shift [20, 21] относится к классу кластеризации по плотности точек в фазовом пространстве. Пусть имеется множество точек T, положение которых в фазовом пространстве задается по N координатам. Каждая координата пространства, по сути, является мерой одной из характеристик объекта. В пространстве вводится определенная метрика например, евклидово расстояние $\rho(t,t')=\sum_{i=1}^{N}\sqrt{t_i^2+\left(t_i'\right)^2}$ или $\rho(t,t')=\sum_{i=1}^{N}\left|t_i^2+(t_i')^2\right|$, где $t,t'\in T$ и т.п. Для поиска кластера в фазовое пространство вводится шарообразная область C с центром m. Алгоритм подразумевает выполнение последовательности итераций, на каждой из которых вычисляется новый центр масс m' всех точек, попавших в область C, затем кластер передвигается в новую точку с центром масс m' (т.е. $m=m'$) и начинается следующая итерация. Математически доказано, что после конечного числа итераций центр масс смещаться не будет, т.е. $m=m'$; и, именно в этой точке и наблюдается максимальная концентрация точек.

Для реализации алгоритма нам пришлось разработать три новых специализированных типа ФУ: точка фазового пространства, кластер, и менеджер. Менеджер отвечает за управление всеми ФУ, участвующими в вычислениях по алгоритму Mean Shift. Данный пример иллюстрирует та-

кой подход к ОА программированию: если функциональности существующих типов ФУ не хватает для реализации вычислительной задачи, то можно создать новый тип ФУ, благо в виртуальной ОА ВС он создается программно. В нашем случае код, реализующий поведение ФУ, был написан на языке C++.

Также как и для примера из раздела 4.1, из ФУ, ассоциированных с точками фазового пространства, была сгенерирована вычислительная сетка. Т.к. точки из множества T расположены в пространстве нерегулярно, наилучшим решением было применение триангуляции Делоне, когда сетка строится в виде треугольников [22]. ФУ-кластер хранит в своем внутреннем регистре ссылку на ФУ-точку ближайшую к его центру масс, и через эту точку производит опрос других точек пространства, запуская опросные токены по вычислительной сетке. Структура вычислительной сетки приведена на рис. 6.

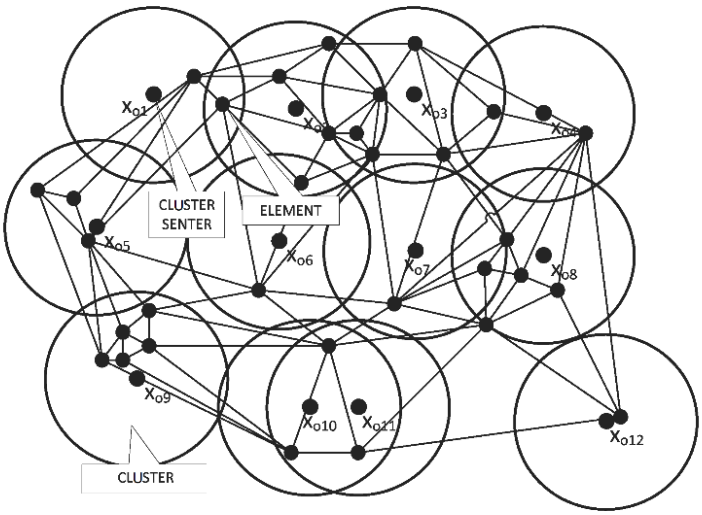

Рис. 6. Вычислительная сетка для реализации алгоритма Mean Shift

Для сбора данных от ФУ-точек ФУ-кластер связывается точкой, ближайшей к его центру масс и запускает вычислительную волну от нее для опроса близлежащих точек. Все точки, получившие опросный токен (прямой токен) и находящиеся внутри области кластера, посылают в начальную точку опроса обратный токен со сведениями о себе и передают опросный токен далее по вычислительной сетке. Если прямой токен приходит к точке за пределами кластера, то точка обратный токен не отправляет и прекращает дальнейшую передачу прямого токена, завершив тем самым продвижение опросной волны. Обратные токены собираются на ФУ-кластере. Исходя из собранных координат точек, ФУ-кластер может вычислить новый центр масс и перейти к новой итерации алгоритма (рис. 7)

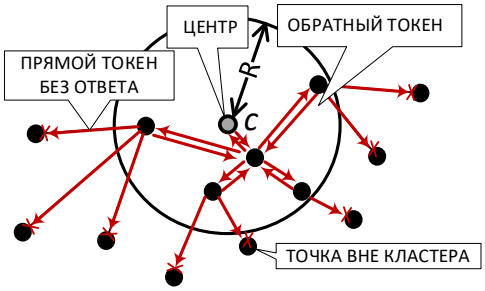

Рис. 7. Вычислительная сетка для реализации алгоритма Mean Shift

Созданные программные модули для реализации алгоритма Mean Shift были интегрированы в среду ОА программирования и моделирования, также была расширена функциональность компилятора ОА языка для обеспечения работы с новыми типами ФУ. После этого на ОА языке была реализована программа для имитационного моделирования вычислительного процесса. Результаты работы программы представлены на рис. 8, где иллюстрируется перемещение центров кластеров по 2-мерному фазовому пространству в направлении области с наибольшей концентрации

точек. В данном случае моделировалось движение сразу 16-ти кластеров, первоначально расположенных в виде прямоугольной сетки.

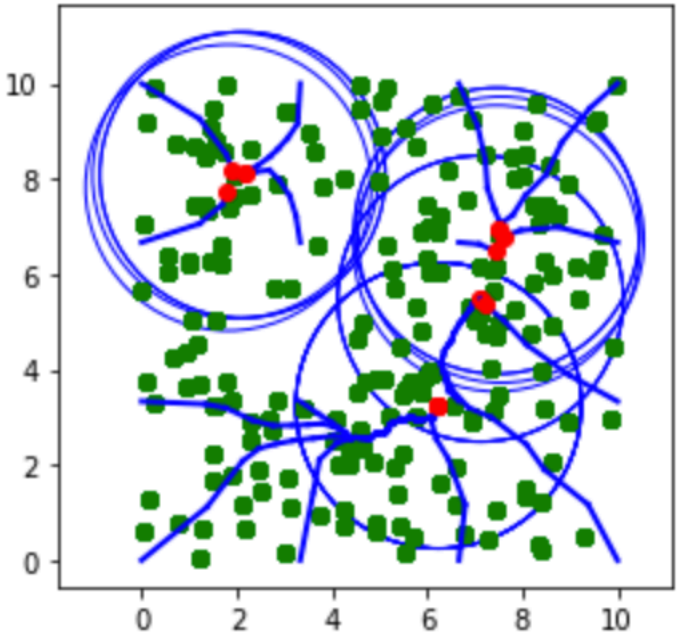

Рис. 8. Миграция 16-ти кластеров диаметром 5 на поле из 100 точек

5. Заключение

Описанный в статье ОА язык программирования представляется нам весьма перспективным и в том числе как средство описания алгоритмов для высокопараллельных и распределенных ВС. Это подтверждается тем, что он ориентирован на dataflow парадигму, которая обеспечивает самораспараллеливание вычислений. Как видно из приведенных фрагментов кода, ОА программа достаточно проста и понятна, что значительно упрощает процесс создания кода параллельной программы.

Язык, прежде всего, ориентирован на применение для ОА ВС, однако может быть использован и в качестве самостоятельного языка программирования. Язык дает большие преимущества: возможность организации сложных структур данных, удобное описание параллельных вычислений; кроме того, возможность создания и трансформации семантических сетей. Также ОА язык оказался весьма удобным для создания компиляторов языков высокого уровня и может быть применен подобно таким платформам, как LLVM [23]. Эффективность в области создания трансляторов языков подтверждается тем, что на ОА языке был написан его же компилятор, и в настоящее время создается компилятор 3-й версии.

ОА язык применяется для имитационного моделирования вычислительного процесса на базе ОА ВС, и он на практике показал свое удобство для создания параллельных программ. Так, программист не утруждает себя описанием вычислительных нитей и их синхронизацией – синхронизация происходит автоматически по данным непосредственно во время вычислительного процесса.

В настоящее время происходит создание компилятора новой версии, что повысит удобство программирования на ОА языке. Удобство будет обеспечено, прежде всего, за счет введение в ОА язык высокоуровневых конструкций: императивные конструкции, программные функции, арифметико-логические выражения. Для компилятора создается новая среда программирования на базе редактора VS Code, обеспечивающая подкраску синтаксиса программы, контекстные меню для выбора вариантов ввода, всплывающие подсказки, навигатор по информационным конструкциям, созданным в программе.

В дальнейшем предполагается применение ОА языка и среды программирования для него в научно-исследовательских целях, в частности, для имитационного моделирования вычислитель-

ного процесса на базе ОА ВС. Основной целью исследования будет выяснение емкостной и вычислительной сложностей вычислительного процесса для решения различных прикладных задач. Вычислительные задачи для моделирования будут выбираться из разных областей: высокопараллельные и распределенные вычисления, системы автоматизации, системы искусственного интеллекта, анализа естественного языка [24], сеточные вычисления и т.д.

ОА язык сможет найти применение и для создания ВС на базе ПЛИС (FPGA). Для этого будет необходимо реализовать конвертер из ОА языка на языки схемотехнического моделирования (например, VHDL или Verilog), для которых существует САПР для записи программы на ПЛИС.

Литература

1. Milutinovic V., Salom J., Veljovic D. et al. DataFlow Supercomputing Essentials, Research, Development and Education. Springer, 2017.
2. Kasyanov V. Sisal 3.2: Functional language for scientific parallel programming // Enterprise Information Systems. 2013. Vol. 7, no. 2. P. 227–236. DOI 10.1080/17517575.2012.744854.
3. Легалов А.И. Функциональный язык для создания архитектурно-независимых параллельных программ // Вычислительные технологии. 2005. Т. 10, № 1. С. 71–89.
4. Veen H. Dataflow Machine Architecture. Center for Mathematics and Computer Science, P.O. Box 4079, 1009 AB Amsterdam, The Netherlands. 1987.
5. Hewitt C., Bishop P., Steiger R. Universal Modular ACTOR Formalism for Articial Intelligence // IJCAI. 1973. P. 235–245.
6. Wipliez M., Roquier G., Nezan J.F. Software Code Generation for the RVCCAL Language // Journal of Signal Processing Systems. 2011. Vol. 63, no. 2. P. 203–213. DOI: 10.1007/s11265-009- 0390-zff.ffhal-00407950f.
7. Климов А.В., Левченко Н.Н., Окунев А.С. Преимущества потоковой модели вычислений в условиях неоднородных сетей // Информационные технологии и вычислительные системы. 2012. № 2. С. 36–45.
8. Климов А.В., Окунев А.С. Графический потоковый метаязык для асинхронного распределенного программирования // Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС). 2016. № 2. С. 151–158.
9. Hammad A. Applying Automated Program Repair to Dataflow Programming Languages // 2021 IEEE/ACM International Workshop on Genetic Improvement (GI), 2021.
10. Салибекян С.М., Панфилов П.Б. Вопросы автоматно-сетевого моделирования вычислительных систем с управлением потоком данных // Информационные технологии и вычислительные системы. 2015. № 1. С. 3–9.
11. Салибекян С.М., Панфилов П.Б., Гончарук Г.Н. Объектно-атрибутная модель программирования для параллельных вычислительных систем с управлением потоком данных // Параллельные вычислительные технологии 2013 (ПаВТ'2013): Труды международной научной конференции (Челябинск, 1-5 апреля 2013 г.). Челябинск: Издательский центр ЮУрГУ, 2013. С. 616.
12. Kavi K., Buckles B., Bhat U. A Formal Definition of Data Flow Graph Models // IEEE Transactions on Computers. 1986. Vol. C-35, no. 11. P. 940–948. DOI: 10.1109/TC.1986.1676696.
13. Салибекян С.М. Трансляция языков высокого уровня, управляемая потоком // Информационные технологии. 2024. Т. 30, № 5. С. 261–268. DOI: 10.17587/it.30.261-268.
14. Оператор ?: (C#) URL: https://msdn.microsoft.com/ru-ru/library/ty67wk28.aspx (дата обращения: 28.01.2025).

15. Ключев А.О., Кустарев П.В., Ковязина Д.Р., Петров Е.В. Программное обеспечение встроенных вычислительные систем. СПб.: СПбГУ ИТМО, 2009. 212 с.

16. Тюрин А.Г., Зуев И.О. Кластерный анализ, методы и алгоритмы кластеризации // Вестник МГТУ МИРЭА. 2014. № 2 (3). С. 86–97.

17. Art L., Mauch H. Dynamic Programming A Computational Tool. Springer, Berlin, 2007.

18. Cai Y., Judd K.L., Thain G. et al. Solving Dynamic Programming Problems on a Computational Grid // Comput Econ. 2025. Vol. 45. P. 261–284. DOI: 10.1007/s10614-014-9419-x.

19. Salibekyan S.M., Vishnekov A., Ivanova E. Methodology of Mean Shift Clustering Algorithm Implementation Based on Dataflow Computer // Proceedings of 2019 XVI International Symposium "Problems of Redundancy in Information and Control Systems" (REDUNDANCY). IEEE, 2019. P. 177–180. DOI: 10.1109/REDUNDANCY48165.2019.9003312.

20. Тюрин А.Г., Зуев И.О. Кластерный анализ, методы и алгоритмы кластеризации // Вестник МГТУ МИРЭА. 2014. № 2 (3). С. 86–97.

21. Comaniciu D., Meer P. Mean Shift: A Robust Approach Toward Feature Space Analysis // IEEE Transactions on Pattern Analysis and Machine Intelligence. 2002. P. 603–619.

22. Скворцов А.В. Триангуляция Делоне и ее применение. Томск: Изд-во Том. ун-та, 2002. 128 с.

23. Вьюкова Н.И., Галатенко В.А., Самборский С.В. LLVM как инфраструктура разработки компиляторов для встроенных систем // Программная инженерия. 2013. № 6. С. 2–10.

24. Салибекян С.М. Объектно-атрибутный подход для семантического анализа естественного языка // Информационные технологии. 2021. Т. 27, № 5. С. 267–274.