УДК 004.43 БК 22.18

ПЕРСПЕКТИВЫ ФУНКЦИОНАЛЬНОГО ПРОГРАММИРОВАНИЯ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ

Л. В. Городняя[0000-0002-4639-9032]

Институт систем информатики им. А.П. Ершова Сибирского отделения Российской академии наук, Новосибирский государственный университет, г. Новосибирск

lidvas@gmail.com

Аннотация

Статья посвящена результатам анализа современных тенденций функционального программирования, рассматриваемого как метапарадигма решения проблем организации параллельных вычислений и многопоточных программ для многопроцессорных комплексов и распределённых систем. С учетом мультипарадигмальности параллельного программирования использован парадигмальный анализ языков и систем функционального программирования. Такой анализ позволяет снижать сложность решаемых задач методами декомпозиции программ на автономно развиваемые компоненты, оценивать их сходство и различия. Учёт парадигмальных особенностей необходим при прогнозировании хода процессов применения программ, а также при планировании их изучения и разработки. Есть основания рассчитывать, что функциональное программирование помогает повышать производительность программ. Показано разнообразие парадигмальных характеристик, присущих подготовке и отладке долгоживущих программ параллельных вычислений.

Ключевые слова: *функциональное программирование, парадигмальная декомпозиция, параллельные вычисления, система программирования, мультипарадигмальность.*

ВВЕДЕНИЕ

Рассматривая идеи функционального программирования (ФП) и практику их применения при анализе проблем, средств и методов организации параллельных вычислений, можно обратить внимание, что уже создано значительное количество языков и систем параллельного программирования, решающих многие проблемы подготовки многопоточных программ для многопроцессорных комплексов и распределённых систем. Новые работы по ФП существенно нацелены на поиск более производительных решений проблем параллельного программирования [1].

Изложение начинается с обсуждения особенностей термина «функциональное программирование», основных принципов и ограничений, обнаруживающихся при переносе техники ФП на параллельные вычисления (ПВ). Затем рассматривается ряд парадигм ПВ, поддержанных в известных языках программирования (ЯП). Анализируется проблема учёта уровня новизны в задачах ПВ и перехода к производственному ФП.

1. О ТЕРМИНЕ «ФУНКЦИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ»

Понятие «функциональное программирование» в настоящее время допускает два смысла. Исторически в начале 1960-х годов Дж. Маккарти провозгласил, что все понятия программирования могут трактоваться как функции или результат их применения. В середине 1970-х годов Дж. Бекус привлек внимание к ФП, призывая преодолеть узость так называемого «бутылочного горлышка» оператора присваивания. Собственно определение термина не было дано, оно использовалось интуитивно без особых разночтений и постепенно стало восприниматься как отдельная парадигма, противостоящая императивному программированию, в которой осознано, что *правильность важнее эффективности*. ФП даёт приоритет организации вычислений и сложных структур данных, а вопросы эффективной обработки памяти и управления процессами уходят на второй план.

Примерно с середины 1990-х годов кристаллизовалась идея «чистого функционального программирования» как раздела дискретной математики, исследующего прямые однозначные функции в дискретном пространстве над значени-

ями типа целых чисел, символьных строк или графов. Чистое ФП можно рассматривать как математическую основу более общего производственного ФП. Постепенно это «чистое ФП» стали называть просто «ФП». Такое разделение смыслов примерно соответствует различию в стандартах на академические и производственные ЯП. Теперь общее определение ФП начинают с утверждения, что характеристическая его особенность заключается в том, что *одинаковые формулы в одинаковом контексте имеют одинаковое значение.* Из этого вытекает исключение присваиваний, глобальных переменных, побочных эффектов, работы с внешними устройствами, передач управления. Эта изящная формула уязвима из-за отсутствия единого, точного определения термина «контекст».

Понятие «контекст» в программистской литературе обладает некоторой двойственностью. Это одновременно фрагмент программы, иногда называемый областью существования или видимости, и контекстная таблица соответствия символов и их значений, используемых в этом фрагменте. Семантика ЯП может по-разному ограничивать правила воздействия на контекстную таблицу. В одном ЯП может одновременно существовать две и более контекстных таблицы для одного фрагмента – статика и динамика, а кроме того, глобальный и локальный контексты. Системы программирования (СП) для одного ЯП могут по-разному решать вопрос о порядке перебора контекстных таблиц для определения значения формулы, более того, такой порядок может быть изменён по опциям из программы или задания на её компиляцию.

Можно обратить внимание, что при оптимизирующей компиляции нередко контекст – это линейный участок между двумя соседними присваиваниями. В таком случае можно считать, что на внутреннем языке компилятора происходит приведение программы к функциональной форме, удобной для обнаружения невычисляемых, константных или дублирующихся формул. Компилятор заменяет константную формулу на константу, вычисленную при компиляции, а дублирующиеся формулы — на переменную, значение которой будет вычислено при выполнении программы. Не исключено, что возможность при решении сложных задач безопасно выполнять подобные оптимизирующие преобразования является сильным мотивом использования чистого ФП. Эта же причина позволяет ФП быть полезным дополнением для любой парадигмы параллельных вычислений.

Если контекстом считать линейный участок между двумя соседними присваиваниями, то приведённая выше характеристика не отличает парадигму ФП от других парадигм. В процессе компиляции программ практикуется именно такая декомпозиция программ на линейные участки с целью решения проблем распределения памяти. При переходе к ФП каждый такой интервал можно представить как отдельный контекст, в котором каждое значение связано со своей локальной переменной.

2. СЕМАНТИЧЕСКИЕ ПРИНЦИПЫ

Обычно ФП подразумевает поддержку ряда семантических и прагматических принципов, удобных для создания функциональных моделей на этапе исследовательского компьютерного эксперимента, естественного при решении новых и сложных задач**.** Программы обработки любых данных могут быть заданы символьными формами, в которых выделен один элемент – представление функции, остальные – ее аргументы. Границы применимости таких форм определены интерпретатором или компилятором, позволяющим вычислять значения символьных форм. Функциональное программирование поддерживает *внешние* семантические принципы представления алгоритмов, такие как универсальность, параметризация, самоприменимость, и системно-реализационные *внутренние* прагматические принципы поддержки информационной обработки, такие как интегральность ограничений, неизменяемость данных, строгость результата. Семантическим принципам программист следует при подготовке программы. Прагматические принципы обеспечивает система программирования, освобождая программиста от непринципиальных решений, не зависящих от природы задачи.

Универсальность. Понятия «функция» и «значение» представляются теми же средствами, как и любые данные для компьютерной обработки. Исторически близкое понятие – «принцип хранимой программы».

Этот принцип позволяет строить представления функций из их частей и вычислять части по мере поступления и обработки данных. Нет принципиальных ограничений на манипулирование средствами языка, функциями из определения семантики языка, конструкциями реализации языка в СП и выражениями в программе. Всё, что понадобилось при реализации ЯП, может пригодиться при его

применении. Равноправие разноуровневых средств обуславливает открытый характер систем функционального программирования. Строго говоря, программирование, в отличие от математики, вообще не имеет дела ни со значениями, ни с функциями. Программирование работает с данными, которые *могут представлять* значения или функции.

Идея хранимой программы впервые сформулирована в описании аналитической машины Чарльза Беббиджа, через сто лет она реализована в компьютерах Конрада Цусе и определении машины Алана Тьюринга, позднее провозглашена в архитектуре Джона фон Неймана [2]. Исторически подтверждена возможность такого символьного представления информации, при котором нет принципиального различия в природе данных для изображения значений и функций. Следовательно, нет и препятствий для обработки представлений функций теми же средствами, какими обрабатываются данные. Поэтому представления функций можно строить из их частей – символов. Их даже можно формировать по ходу процесса вычислений, поступления и обработки информации о них. Именно так компиляторы конструируют программы.

При компиляции программ выполняется распределение памяти для функций, переменных и констант. Эффективность такого распределения зависит от учёта особенностей базовых средств обработки машинных кодов, что обычно формулируется как тип данных, удобных для обработки компьютером, но несколько противоречит принципу универсальности. Вообще, тип данных – это множество объектов с соответствующим ему набором допустимых операций. Обычно тип данных задают в тексте программы или выводят его при статическом анализе, чтобы экономно распределять память и обнаруживать некоторые ошибки несоответствия выбора операций при обработке кодов данных. Динамическое управление вычислениями и конструированием программ может по существу требовать представления и более детального анализа типов данных в процессе вычислений.

Параметризация. Представление любой выделенной формулы можно рассматривать как параметр некоторой функции. Это значит, что части представления функций можно вычислять в зависимости от промежуточных результатов и конструировать функции, учитывающие условия их применения, в частности, расположение их определений и вызовов на разных уровнях иерархии представления

программы. Так работают интерпретаторы и отладчики программ. Реализация языков ФП обычно содержит механизмы, приспособленные к дополнению определений отдельных элементов программируемой системы. Любая символьная форма в определении функции может быть выделена из него как параметр и, наоборот, подставлена в него. Функции-переменные допустимы равноправно с обычными функциями-константами и могут быть значениями аргументов или выработаны в качестве результатов других функций.

Процесс вычисления результатов по заданным аргументам методов выполнения определённого алгоритма часто рассматривается как исполнение неизменяемой программы, заранее определённой конструкции – константы. В большинстве ЯП реализация функций подразумевает процесс вычисления результатов по заданным аргументам методов выполнения определённого алгоритма. Функция – это соответствия между аргументами и результатами; и то, и другое, и сама функция могут быть значениями переменных. Отсутствие навыков работы с функциональными переменными говорит лишь о том, что надо осваивать такую возможность, потенциал которой может превзойти ожидания теперь, когда программирование становится все более компонентно-ориентированным.

Обеспечение многократного использования данных выполняется с помощью именования. Переменная – именованная часть памяти, предназначенная для многократного доступа к изменяющимся данным, а константа – к неизменным данным. Переменные отличаются от именованных констант частотой изменения связи между именем и соответствующим ему данным. Поэтому можно их реализацию делать одним и тем же способом. Именования данных обычно используются при их описании, предназначенном для многократного использования описанных конструкций – значений или функций.

Самоприменимость. Представления рекурсивных функций прямо или косвенно используют сами себя, что позволяет строить ясные лаконичные символьные формы.

Примеры самоприменимости дают многие математические функции, особенно рекурсивные, такие как факториал, числа Фибоначчи, суммирование рядов и многие другие, определение которых использует математическую индукцию. В технологии программирования некоторым сходством обладает метод раскрутки

программ. Этот метод сводит организацию процесса программирования к ряду шагов, каждый из которых даёт или работоспособную часть программы, или инструмент для выполнения очередных шагов раскрутки. Первые реализации языка Lisp были выполнены методом раскрутки, причем в составе системы сразу были предусмотрены и интерпретатор, и компилятор функций. Оба эти инструмента были весьма точно описаны на самом языке Lisp, причем основной объем описаний не превосходил пару страниц. Похожая история произошла и при разработке языка Си, ядро которого по трудоёмкости оценивалось как один человеко-месяц.

3. ПРАГМАТИЧЕСКИЕ ПРИНЦИПЫ

Прагматические принципы поддерживают система программирования, точнее, её разработчики.

Интегральность ограничений. Оперативный пересмотр распределения памяти или её освобождения поддержан для профилактики необоснованных простоев памяти.

Интегральность ограничений на пространственные характеристики позволяет исключать необоснованные простои памяти. Бывает, что не хватает памяти принципиально не на всю задачу, а лишь на отдельные блоки данных, возможно мало существенные для ее решения. Такая проблема в системах ФП решается принципом интегральности ограничений на пространственные характеристики. Многие СП поддерживают чётко фиксированное распределение памяти на части для хранения собственно программы, используемых в ней переменных, констант, стека вызовов, динамически размещаемых данных – «куча». Возникают ситуации, когда одни из таких частей исчерпаны, в то время как в других остаётся недоиспользованное пространство. В системах ФП эту проблему решает специальная функция – «мусорщик» (garbage collector), пытающаяся при недостатке любой области памяти автоматизировать перераспределение или освобождение памяти.

Представление данных в памяти может быть скрыто специальными функциями, освобождающими программиста от непринципиальных проблем для решения более важных задач, отдавая приоритет алгоритму и структурам данных. Управление обработкой значений и организация доступа к памяти актуальны после выбора и отладки принципиальных решений, без преждевременного отвлечения на проблему повторного использования памяти при её дефиците. Новые

реализации механизма «мусорщик» рационально учитывают преимущества восходящих процессов на больших объемах памяти.

*Неизменяемость данных***.** Представление каждого результата применения функции размещается в новой части свободной памяти без искажения аргументов этой функции, которые могут быть полезны для других функций.

Таким образом существенно упрощается отладка программ и обеспечивается обратимость любых действий. Можно быть уверенным, что все промежуточные результаты сохранены, их можно анализировать и снова использовать в любой момент. Если определение функции – это статическая конструкция, то процесс можно рассматривать как композицию из функций, разворачиваемую по этой конструкции в динамике. Таким образом можно поддерживать уточнение или улучшение программируемых решений по ходу вычислений. Реализация языка ФП может содержать списки свойств символов, приспособленные к внешнему доопределению отдельных элементов поведения программируемой системы. Такие списки использовались в ранних работах по представлению знаний.

В этом плане вычисление – это процесс решения задачи, сводимой к обработке данных, – чисел, кодов или символов, рассматриваемых как модели реальных объектов, представленных с помощью обозначений, смысл которых может изменяться по ходу обработки данных и появления внешних данных с сохранением прежних смыслов. Отдельный аспект связан с переходом от целых чисел к вещественным, возможно допускающим изменение точности представления по ходу вычислений. Логически они остаются константами, а реализационно обрабатываются как переменные. Кроме того, возникают некоторые отклонения от принципа неизменяемости данных в пользу восстановимости данных, не так уж влияющие на подготовку и отладку программ, если это поддерживается на системном уровне или возникает на заключительных этапах отладки.

*Строгость результата***.** Любое число результатов функции может быть представлено как одна символьная форма, из которой при необходимости можно выбрать нужный результат. Такой принцип удобен для описания интерпретатора программ. Всегда ясна граница между аргументами и результатами, размещаемыми в стеке, – результат последней вычисленной функции расположен на вер-

шине стека. Нередко этот принцип трактуется как требование однозначности математических функций, что приводит к сомнениям в правомерности функций целочисленного деления, извлечения корня, обратных тригонометрических функций и многих других категорий математических функций.

Способы определения функций были достаточно различными ещё задолго до появления компьютеров. Отличаются и подходы к сохранению результатов функций, в частности, в виде таблиц или специальных приборов типа логарифмической линейки. Могут отличаться и методы решения одних и тех же задач. Например, существует более двадцати методов сортировки, результаты которых будут одинаковы. Различия при выборе метода зависят от условий применения программы, особенностей сортируемых данных и критериев эффективности. Выбор техники реализации функции обычно зависит и от способов определения правила и методов получения результата функции по заданному правилу. При одних и тех же аргументах способы могут быть различны, например:

- алгоритм (поиск наибольшего общего делителя);
- таблица (сложение или умножение для целых чисел);
- процесс (непосредственное измерение);
- устройство (вольтметр, термометр, часы);
- формализованный текст (процедура, подпрограмма, макрос и т. п.).

Использование чисел и кодов нередко обусловлено поддержкой эффективности, надёжности и безопасности, смягчением роли человеческого фактора. Тем не менее, во многих современных информационных сервисах можно видеть решения, существенно снижающие и надёжность, и безопасность. Работа с паролями теперь часто имеет кнопочку для показа его текста. Нередко диалог с «личным кабинетом» содержит простенькую процедуру смены пароля. Идентификация пользователя на сайтах, работающих с деньгами и документами, происходит по IP-адресу, без учёта того, что один компьютер может быть во владении разных пользователей. Многие банковские инструменты на неожиданные ситуации вместо диагностики практикуют отказ в обслуживании.

4. СЛЕДСТВИЯ ПРИНЦИПОВ

Представление алгоритмов в виде функциональных программ даёт практически значимые следствия: конструктивность, факторизация и доказуемость вытекают из семантических принципов, а из прагматических принципов следуют интуитивные скрытые модели продолжаемости процессов, обратимости действий, параллелизма, дающие основу для интуитивного выстраивания функциональных моделей, допускающих непосредственный компьютерный эксперимент.

Конструктивность является следствием принципа универсальности, позволяющего представления программ обрабатывать так же, как любые данные. Это даёт поддержку мета-компиляция, включая синтаксически управляемые методы генерации и анализа программ, а также однородно-гомогенные представления программ, внешне сохраняющих аналогию или подобие обрабатываемым данным или прототипам, в том числе смешанные и частичные вычисления, оптимизирующие преобразования, макрогенерацию и многое другое, необходимое для создания операционных систем и систем программирования.

Факторизация непосредственно вытекает из принципа параметризации с учетом принципа универсальности. Любой помеченный фрагмент программы может быть вынесен из её представления и ассоциирован с определённым именем, допуская возможность восстановления исходного представления. Можно обратить внимание, что параметры при вызове функции вычисляются на одном и том же уровне иерархии, в общем контексте согласно принципу неизменяемости данных. Поэтому порядок вычисления параметров не имеет значения, может быть произвольным. Это позволяет декомпозировать программу на автономно развиваемые модули и накапливать правильность, а также представлять параллельные потоки, ленивые (отложенные) или опережающие вычисления. Можно сказать, что программа приводится в факторизованную форму по тем или иным параметрам в зависимости от цели её преобразования. Благодаря обратимости действий, т. е. неизменяемости данных, процесс отладки обретает сходимость.

Доказуемость основана на связи принципа самоопределимости с методами рекурсии, математической индукции и логики. Появляется возможность логически выводить отдельные свойства программ и благодаря этому обнаруживать некоторые трудно уловимые ошибки, что повышает надёжность и безопасность

программ, хотя не позволяет решить проблему правильности в полном объёме. Подобным образом формируется подход к разработке программ по методике раскрутки с выделением минимального ядра, с последующими шагами его расширения до полного практичного решения задачи.

Следствия принципов системно-реализационной прагматической поддержки информационной обработки в системах ФП выражаются в использовании интуитивных моделей, таких как продолжаемость процессов (бесконечность), обратимость действий и параллелизм.

Продолжаемость процессов интуитивно вытекает из прагматической поддержки принципа интегральности ограничений, позволяющего значительную часть работы выполнять на основе модели неограниченной памяти без особой заботы о её границах и разнообразии характеристик скорости доступа к разным структурам данных. Прагматика таких решений нацелена на освобождение внимания программиста от не самых важных проблем в пользу существа решаемой задачи. Во многих языках ФП поддержана имитация работы с бесконечными структурами данных.

Обратимость действий базируется на иллюзии неизменяемости данных, механизмы которой скрыты в СП, их применение почти не требует заботы при подготовке программы и основного объёма отладки. Это позволяет поддерживать механизм мемоизации функций на ранее обработанных аргументах. Фактически необходимые изменения данных, такие как повторное использование памяти, просто автоматизированы, программист может позволить себе не вмешиваться в реализацию таких средств до тех пор, пока не возникнут проблемы с производительностью.

Параллелизм основан на принципе строгого результата, позволяющего при необходимости любое число представленных результатов рассматривать как общую структуру из них. Дополнением является принцип параметризации, гарантирующий одинаковость контекста при вычислении параметров функции одного уровня. Поскольку результаты часто являются аргументами объемлющих функций, логично возникает и сопутствующий *принцип единого аргумента*. Функция любого числа аргументов может быть преобразована в функцию одного аргумента. Появляется возможность представлять независимые потоки и объединять

их в многопоточные или многопроцессорные программы, в общий проблемно-ориентированный комплекс. Это делает ФП удобным для работы с программами, нацеленными на организацию параллельных процессов. Кроме того, возможность перехода от списка параметров или результатов к строгому результату или единому аргументу позволяет отойти от привычной схемы операций, отображающей два операнда в один результат, к операциям, отображающим один ряд операндов в другой ряд результатов, что может соответствовать структуре некоторых аппаратных узлов и тем самым допускать представление более эффективных решений.

Такой комплект следствий из принципов ФП позволяет уточнять и улучшать запрограммированные решения при отладке программ решения новых задач, допускает *множественность* определений функций при исследовании свойств решаемой задачи на уровне редактируемых константных символьных форм. Множественные определения символов в рамках настраиваемой интерпретации обеспечивают, кроме общеизвестного полиморфизма, более управляемые схемы построений, отладки и конструирования программ.

5. ПАРАДИГМАЛЬНАЯ ХАРАКТЕРИСТИКА

Используя систематизацию парадигм программирования на основе различий в приоритетах принятия программируемых решений, можно сделать вывод, что одной из причин сложности разработки программ параллельных вычислений является их скрытая мультипарадигмальность [3]. Для разработки параллельной программы многое требуется продумывать по-разному одновременно в различных парадигмах, удобных для решения отдельных подзадач без возможности решения полного комплекса подзадач в единой обстановке. Наиболее очевидно парадигмальные различия видны при решении задач масштабирования вычислений на различные многопроцессорные комплексы, синхронизации взаимодействий над локальной и общей памятью в многопоточных программах, представлении естественного асинхронного параллелизма уровня постановок задач и достижения высокой производительности программ с учётом критерия полноты и равномерности загрузки доступных многопроцессорных комплексов или распределённых систем. Создано заметное число языков и систем программирования

(ЯиСП), позволяющих решать отдельные из этих задач в рамках парадигм, поддержка которых представлена в разных языках программирования (Таблица 1).

Таблица 1.

№	*Проблема*	*Парадигма*	*ЯП и API*
1	Масштабирование	Многопроцессорное программирование	VHDL, XC, СИГМА, bash, Occam, mpC, Эль-76, Limbo, Kotlin, MPI
2	Синхронизация потоков	Синхронное (синхронизующее) программирование	APL, VAL, Sisal, Alef, E, X10, LuNA, Charm, Kotlin, Go, Java, Scala, Rust, Пифагор, OpenMP
3	Постановки задач	Асинхронное программирование	БАРС, Haskell, Erlang, JavaScript, Python, C#
4	Производительность программ	Высокопроизводительное программирование	Setl, HPF, G, Sparkel, mpC, Sanscript, D, Rest, F#

Таким образом, для каждой из трудно решаемых задач параллельных вычислений уже сформирована отдельная удобная парадигма её решения и создан ряд ЯП, поддерживающих такую парадигму. Любая из таких парадигм может быть дополнена моделями и методами ФП. Различие между парадигмами проявляется в упорядочении важности средств и методов, используемых при решении отдельных задач, другие задачи требуют иного упорядочения. В каждый момент разработки программы обычно используется одна парадигма. Соответственно и в разных ЯП выделяется одна ведущая парадигма. Ряд таких ЯП (Kotlin, APL, VAL, Sisal, Go, Haskell, Erlang, F#) изначально относят к функциональным, почти все содержат подъязыки, позволяющие представлять программы в функциональном стиле.

Требования к решению достаточно сложных задач ПВ связано с целым рядом затруднений, что влечёт необходимость использования разных парадигм на разных этапах их создания и фазах их жизни. При переходе к технологии параллельного программирования важна гарантия получения практичного результатов в заданные сроки, что требует поддержки полного спектра парадигм, используемых на разных этапах разработки программ. Трудоёмкость использования разных парадигм при решении одной задачи обычно минимизируется созданием многоязыковых систем, допускающих по мере необходимости возможность перехода

от одной парадигмы к другой без затрат на освоение разных интерфейсов. Это показывает целесообразность создания мультипарадигмального языка ПВ, поддерживающего одновременно все основные парадигмы параллелизма, дополненные ФП.

6. ПАРАДИГМАЛЬНАЯ ДЕКОМПОЗИЦИЯ

Как показывает опыт применения языков БАРС и Haskell, в таких случаях удобно выделять в определении ЯП отдельные подъязыки, поддерживающие основные парадигмы или монады, нацеленные на конкретные модели подготовки и представления программ так, чтобы на каждом этапе разработки программы локализовать использование одной парадигмы, характеризуемой сравнительно небольшим набором средств и методов в рамках одного способа мышления. Каждая парадигма имеет свои наполнение категорий семантических систем и упорядочение их роли в процессе программирования [3].

Средства многопроцессорного программирования обычно опираются на данные и характеристики доступной архитектуры, включая основной многопроцессорный комплекс и взаимосвязи между его элементами. Оперирование комплексом позволяет инициировать процессы функционирования отдельных процессоров, их блокировку, возобновление и отмену процессов. По ходу процессов возможны обмены данными по определённым протоколам, результаты которых можно рассматривать как цель программы. Принятие решений начинается с определения пространства возможных многопроцессорных комплексов, что можно рассматривать как особую разновидность памяти со своей дисциплиной функционирования и взаимодействия элементов. Далее происходит выбор подходящих конфигураций и структурирования пространства итерирования процессов, предназначенных для выполнения на отдельных процессорах. Затем полученная схема управления процессами наполняется собственно действиями, выполняющими вычисления. Обычно предпочитаются процедуры без рекурсии, освобожденные

от сложностей управления вычислениями и побочных эффектов над общей памятью. Строится многопроцессорная программа, допускающая в динамике реконфигурацию многопроцессорного комплекса[1].

При синхронном (синхронизующем) многопоточном программировании выделяются достаточно чёткие схемы управления вычислениями, которые разделяются на регулярные участки и типичные модели программы, удобные для распараллеливания. Обычно выделяются фрагменты, свободные от побочных эффектов в памяти, и допускается неимперативное управление временем исполнения потоков программы с учётом иерархии схемы управления программой и некоторых временных отношений. Принятие решений начинается с выбора стандартных схем управления, используется понятие «пространство итерирования», которое можно структурировать в зависимости от распределения данных и методов их хранения, что может влиять на эффективность и дисциплину обработки многоуровневой памяти. Управление итерированием может использовать произвольные предикаты. Схема управления над пространством итерирования потоков наполняется фрагментами, сравнительно простыми для отладки, возможно отлаженными заранее или программируемыми автономно. Для вывода результатов программы выделяется специальные средства формирования результатов вычислений, полученных на равноправных потоках многопоточной программы. Таким образом получается многопоточная программа с динамически изменяемым пространством потоков над локальной памятью с возможностью эпизодической синхронизации их отдельных фрагментов[2].

Асинхронное программирование нацелено на предельное выражение независимых элементов программы, обусловленных природой решаемой задачи, что может быть базой для максимального распараллеливания при условии представления специальных схем организации вычислений с учётом специфики доступного оборудования. Начинается принятие решений с выбора схем управления действиями и представления условий их срабатывания. Действия могут использо-

1 Курс «Параллельное программирование с использованием OpenMP и MPI». URL: https://mooc.tsu.ru/mooc-openedu/mpi/

2 Курс «Основы MPI». URL: https://habr.com/ru/post/121925/

вать ту или иную дисциплину обработки памяти. Поддерживается неявное и программируемое разнообразие дисциплин доступа к памяти, включая иерархию неоднородный памяти, и схем вычислений – фрагментов, наполняющих общую схему программы процедурами или библиотечными модулями. Получается схема программируемых синхросетей, асинхронно управляющая выполнением действий в зависимости от условий их готовности к выполнению.

Для высокопроизводительного программирования необходим переход от отдельного прогона программы к учёту перспектив её многократного применения и улучшения. Появляется возможность использовать недогруженные мощности многопроцессорных комплексов выполнением фрагментов программы в расчёте на предстоящие в будущем прогоны при данных, возможно востребованных в предстоящих прецедентах её отладки и применения. Примерно так организуют программы, предназначенные для сверх быстрого реагирования на опасные события, например, при прогнозировании цунами. Принятие решений начинается со схем и моделей вычислений, возможно над общей памятью, но с приоритетом локальной памяти. Управление вычислениями учитывает особенности многократного выполнения программы при её отладке и применении, включая возможность наследования результатов между сеансами и сравнения измеримых характеристик производительности версий программы. Получается ряд улучшаемых версий программы, выбор одной из которых может учитывать особенности или изменение текущих условий применения, включая конфигурацию многопроцессорного комплекса и требования к измеримым характеристикам производительности программ.

Долгоживущие и учебные ЯП, как и новые ЯП нашего века, обычно мультипарадигмальны. Есть основания для заключения, что успешная практика параллельного программирования требует мультипарадигмальной поддержки полного спектра парадигм параллельных вычислений, допускающей их развитие, пополнение и применение по мере необходимости с возможностью перехода к очередной парадигме без изменения общеязыковой и системной обстановки.

7. ТРУДОЁМКОСТЬ

Жаргон современного практического программирования использует понятие «язык программирования» как «входной язык или расширенное подмножество ЯП типовой СП, функционирующей на базе определённой конфигурации оборудования». Различие заключается в том, что СП обычно сопровождает реализацию ЯП расширяемым комплектом библиотечных модулей и может не поддерживать отдельные сложности семантики ЯП. В результате происходит сглаживание видимых на практике различий между разными ЯиСП. Кроме того, прямые измерения трудоёмкости программирования и производительности программ почти не отражают зависимости результата от принятых программистом решений и выбора конструкций ЯП. Хотя программируемые решения представляются в терминах ЯП, их влияние растворяется в весьма сложном комплексе, наследующем производительность СП и оборудования с большим доминированием характеристик элементной базы и аппаратуры. Таким образом, существует проблема создания методики, позволяющей выявлять такие зависимости совмещением прямых измерений с результатами экспертных оценок особенностей ЯиСП, возможно отличающихся от оценок ЯП [4].

Есть основания при прогнозировании трудоёмкости параллельного программирования учитывать не только степень изученности решаемых задач, но ещё и уровень квалификации и способностей разработчиков программы решения задачи, умеющих преодолевать понятийную сложность реализуемых и используемых программируемых и программных средств, особенности функционирования которых могут выходить за пределы привычных представлений [5]. Для параллельных вычислений степень изученности часто наследует результат ранее созданной последовательной программы решения задачи, обладающей математически точной постановкой. Возникает соблазн, тормозящий осознание или создание более эффективного параллельного алгоритма. Кроме того, уровень квалификации специалистов опирается на опыт императивно-процедурного программирования, препятствующего восприятию более сложных зависимостей в параллельных процессах. Понятийная сложность решаемых задач выходит за пределы обычного образования, и сами понятия обретают более широкое толкование, отчасти противоречащее привычному интуитивному пониманию. Представление

результатов оценки понятийной сложности, позволяющее структурировать пространство таких параметров, рассмотрено в статье [3].

8. СТЕПЕНЬ ИЗУЧЕННОСТИ

Практическое функциональное программирование существенно зависит от выбора постановок задач, решения которых представляются системами функций, сравнительно простых, не слишком трудоёмких и удобных при отладке. По степени изученности существенно различаются следующие категории постановок задач, влияющие на выбор методов решения задач и трудоёмкость их программирования:

– новые;
– исследовательские;
– практичные;
– точные.

Для новых постановок задач характерны отсутствие доступного прецедента практичного решения задачи, новизна используемых средств или недостаток опыта исполнителей. Задачи параллельных вычислений поставлены ещё в докомпьютерную эпоху, и постановки многих таких задач обладают математической точностью. Тем не менее, часть задач параллельного программирования их решений приходится рассматривать как новые из-за стремительного обновления ИТ, элементной базы и нерешённых образовательных проблем программирования в целом. Любая проблема, не получившая хорошего решения, остаётся в статусе новой задачи независимо от времени её постановки. Исследовательские постановки задач параллельного программирования в настоящее время следуют тенденции функционального подхода и изучения схем структурирования пространства итерирования процессов, позволяющего оптимизировать обработку памяти. Функциональный подход можно рассматривать как методику сведения решений сложных задач к композициям из планарных проекций, достаточно удобных для понимания, анализа и обработки. Практичные постановки математических задач ПВ, нацеленные на актуальность и удобство применения, преимущественно используют мощность доступных ИТ для воспроизведения математических моделей, ранее ограниченных низкой эффективностью оборудования, а

теперь получивших перспективу стать новой информационной революцией. Точные постановки большинства задач параллельного программирования сложились на базе последовательных алгоритмов и включают в себя испытание возможностей используемых средств, связанных с мерой организованности ранее созданной императивной программы. Некоторые сложности связаны с использованием однопроцессорных конфигураций как исходной модели параллельных многопоточных программ. Не исключено, что предельным случаем удобнее рассматривать двухпроцессорные конфигурации. Появление собственно параллельных алгоритмов встречается не так уж часто, хотя в середине 1990-х годов проводились конференции такого направления.

9. ОБРАЗОВАНИЕ И КВАЛИФИКАЦИЯ

Обучение параллельным вычислениям входит в образовательные программы многих университетов, что достаточно для понимания их сложности и постановки задач их исследования. Проблемой является переход к практике реализации высокопроизводительных программ, удовлетворяющих особо сложным, трудно удостоверяемым критериям надёжности и безопасности. Ещё ряд проблем связан с формированием интуитивной грамматики деятельности, достаточной для практичного программирования. Возможное решение этих проблем предлагается в проекте языка Синхро, предназначенного для учебного применения [6]

Характерной чертой системного или функционального подхода как ведущего метода программирования является переход к классам задач при содержательном анализе постановок задач. Границы этого класса устанавливаются выбором процесса решения задач. Переход к экспериментам на суперкомпьютерах показал, что именно системные решения могут дать весомый вклад в производительность ПВ, причём такой вклад может превышать теоретические прогнозы. Это можно рассматривать как обоснование необходимости более фундаментального подхода к программированию, особенно к системному программированию и его математическим основам, естественно представимым в чисто функциональном программировании [7].

При создании, формировании и исследовании математических моделей как фундаментального базиса для решения особо трудных проблем эффективности, надёжности и безопасности программного обеспечения важную роль играет развитие моделей, связанных со временем и ресурсами и слабо представленных в курсах по классической математике, тем не менее, доступных в рамках ФП.

Экстенсивное развитие ИТ заметно опережает возможности человека оперативно осваивать новые возможности аппаратуры и системных средств ИТ, выходящие за пределы пользовательского уровня, поддерживаемого поставщиками инструментария и программных продуктов. Миссия программирования – создание инструментария, позволяющего повышать качество информационных систем, включая поиск новых решений по обеспечению надежности и безопасности ИТ [7].

10. ПЕРЕНОС ПРИНЦИПОВ НА ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ

При переходе к многократно используемым программам и параллельным вычислениям *становятся более важными успешный опыт применения и производительность программ, чем их формальная правильность и эффективность*. Принцип универсальности имеет два аспекта — равноправие программ и данных и полная диагностичность определений функций. При решении задач ПВ универсальность сохраняет аспект равноправия программ и данных, традиционно востребованный в задачах операционных систем. Полнота диагностичности функций, удобная для сборки программ из отлаженных функций, может создавать проблемы из-за нарастания числа потоков в многопроцессорных программах. Иногда эта проблема преодолевается подбором выражений, не требующих ветвлений. Объём необходимой диагностики может быть отчасти сокращён средствами статического анализа и контроля типов данных.

Роль параметризации возрастает, она даёт решения проблем по реорганизации потоков при настройке на разные конфигурации многопроцессорных комплексов, требующей декомпозиции фрагментов программы. Факторизация программ на схемы и фрагменты позволяет разделять компоненты по уровню сложности отладки и наследовать правильность ранее отлаженных составляющих программы. Используются функции, не требующие предварительного вычисления параметров, подобно макротехнике.

Самоопределимость в виде рекурсивных функций обычно рассматривается как усложнение, влекущее опасное разрастание стека. Здесь следует обратить внимание, что многие системы ФП предлагают ряд решений, таких как отложенные действия, мемоизация, восходящая рекурсия, методы динамического программирования и оптимизация рекурсий сведением к циклам, во многих случаях позволяющие практически исключить чрезмерное разбухание стека. Да и поддержка работы со стеком в рамках принципа интегральности ограничений может быть поддержана более эффективно, чем в большинстве ЯиСП.

Несколько сложнее с прагматическими принципами, требующими пересмотра системных решений на уровне разработки систем программирования. Интегральность ограничений обычно рассматривается по отношению к памяти. Естественно расширить её на временные границы, подобно тому, как в языке mpC происходит перераспределение объёма вычислений при обнаружении неравномерной загрузки процессоров [8].

Неизменяемость данных сохраняется на уровне локальных потоков, но вызывает проблемы при переходе к общей памяти. Не исключено, что механизмы общей памяти в большей мере требуют восстановимости данных, не считая математических аспектов работы с разноуровневой памятью, копиями, репликами и т. п., что похоже на динамическое редактирование сложных конструкций, пока проиллюстрированного на задачах работы с DSL-языками [1].

Строгость результатов убедительно поддержана в языке Sisal как концепция пространств итерирования, строящегося над перечислимыми множествами с помощью операций скалярного и декартового произведений [9].

11. ПРАКТИЧНЫЕ КОМПРОМИССЫ

Чистое ФП можно рассматривать как методику функционального моделирования при создании программ решения сложных задач. Более общая парадигма ФП позволяет переходить от таких функциональных моделей к производственным структурам данных и принимать при необходимости практичные решения по их обработке в зависимости от реальных условий. Кроме принципов и следствий из них в реальных СП производственная парадигма ФП допускает уравновешивающие механизмы, внешне в ЯП выглядящие как специальные функции.

Например, в языках Lisp 1.5, Clisp, Cmucl и других представителях лисповского семейства обычно предоставлены такие компромиссные функции:

- универсальность смягчается функциями статического и динамического контроля типов данных;
- параметризация дополняется функциями доступа к общей и внешней памяти;
- самоопределимость преодолевается функциями, имитирующими привычные схемы циклов;
- интегральности ограничений противодействуют функции для программируемого распределения памяти;
- неизменяемости данных противостоят деструктивные функции, позволяющие исключать избыточный расход памяти;
- строгость результатов расширяется функциями ввода-вывода данных, работы с файлами, протоколами и многими другими, дающими связь программы с окружающим миром.

ЗАКЛЮЧЕНИЕ

В феврале 2021 года состоялась 22-я конференция, посвященная современным тенденциям функционального программирования [1]. Представленные доклады убедительно показали нацеленность ФП на решение многих проблем организации ПВ.

В рамках ФП возможен подход, позволяющий учитывать особенности решаемых задач и методов программирования, необходимых для решения проблем ПВ, влияющих на их решения в зависимости от приоритетов в выборе языковых средств и реализационных конструкций. Можно наметить линию, позволяющую сравнивать языки, выделяя сопоставимые примеры программ, и анализировать результаты прямых измерений производительности программ, выделяя особенности базовых средств и реализационных решений в СП, нацеленных на улучшение создаваемых программных продуктов. Программирование давно уже стало массовой профессией, требующей объективных метрик для оценки качества программируемых решений. Парадигмальные ошибки, обнаруживаемые при эксплу-

атации привычных СП на современном многопроцессорном оборудовании, показывают, что часть из них были просто незаметны до появления сетей, мобильных устройств и суперкомпьютеров.

Многие вопросы пока не получили практичного ответа. Появление новых парадигм можно связать с проявлением круга новых задач, решение которых пока вызывает трудности. Не ясно, насколько целесообразно взаимодействие парадигм и каким должен быть механизм их взаимодействия. Остаются в стороне образовательные проблемы овладения новыми парадигмами. Возможно, в этом деле помогут методы визуализации программ [5]. Кроме того, особенности парадигм лишь отчасти выражаются на уровне представления программы, часть являются требованиями к прагматике системно-реализационной поддержки в ЯиСП, другие вообще имеют отношение к скрытой интуитивной грамматике деятельности. Граница между семантикой программируемых решений и прагматикой их системной поддержки у каждой парадигмы своя.

В целом следует отметить, что следствия из семантических и прагматических принципов ФП и высокая моделирующая сила аппарата функций, расширенные специальными функциями практичных компромиссов, позволяют полезно дополнять основные парадигмы ПВ и практиковать работы по повышению производительности программ. В этом плане существенный вклад дают мемоизация, отложенные и опережающие вычисления, возможность синтаксического конструирования и декомпозиции программ на автономно развиваемые модули, а на более общем уровне — мета-программирование, позволяющее строить специализированные проблемно ориентированные DSL-языки программирования. Мемоизация позволяет радикально снижать сложность многократно повторяемых вычислений. Отложенные и опережающие вычисления дают возможность перераспределения нагрузки. Синтаксическое конструирование – механизм надёжного использования прототипов и структур обрабатываемых данных. Декомпозиция программ приводит к выводу автономно развиваемых модулей, изменение которых не требует повторного программирования смежных модулей. Мета-программирование может быть применено для создания подъязыков, учитывающих индивидуальные особенности процессоров, а также для обустройства переходов от одного языка к другому.

СПИСОК ЛИТЕРАТУРЫ

1. *Koopman P., Michels S., Plasmeijer R.* Dynamic Editors for Well-Typed Expressions // Trends in Functional programming/ 22nd International Symposium, TFP 2021, February 17–19, 2021. Springer, LNCS 12834. P. 44–66.

2. *Городняя Л.В., Кирпотина И.А.* О проблеме достоверности доступной в Интернете исторической фактографии // Сборник трудов SoRuCom-2017. Четвертая Международная конференция «Развитие вычислительной техники в России и странах бывшего СССР: история и перспективы». Зеленоград, 3–5 октября 1917 г. Под редакцией д. ф.-м. н. А.Н. Томилина. М.: ФГБОУ ВО «РЭУ им. Г.В. Плеханова», 2017. С. 40–49.

3. *Городняя Л.В.* О представлении результатов анализа языков и систем программирования // Научный сервис в сети Интернет: труды XX Всероссийской научной конференции (17–22 сентября 2018 г., г. Новороссийск). М.: ИПМ им. М.В. Келдыша, 2018. С. 262–277. https://doi.org/10.20948/abrau-2019-03

4. *Городняя Л.В.* Подход к оценке трудоёмкости программирования **//** Научный сервис в сети Интернет: труды XXII Всероссийской научной конференции (21–25 сентября 2020 г., онлайн). М.: ИПМ им. М.В. Келдыша, 2020. С. 192–209. https://doi.org/10.20948/abrau-2020-3
https://keldysh.ru/abrau/2020/theses/3.pdf

5. *Авербух В.Л.* Визуализация программного обеспечения. Екатеринбург: ИММ УрО РАН, 1995. 168 с.

6. *Городняя Л.В.* Учебный язык параллельного программирования СИНХРО // Языки программирования и компиляторы—2017. Труды конференции. Южный федеральный университет; под ред. Д.В. Дуброва. Ростов-на-Дону: Изд-во Южного федерального университета, 2017. С. 92–97.
URL: http://plc.sfedu.ru/files/PLC-2017-proceedings.pdf

7. *Городняя Л.В.* Перспективно стратегические парадигмы программирования Академика Андрея Петровича Ершова. 5-я международная конференция «Развитие вычислительной техники в России, странах бывшего СССР и СЭВ (SORUCOM 2020)». Москва, 6–8 октября 2020 г. С. 83–97.

8. mpC: A Multi-Paradigm Programming Language for Massively Parallel Computers // ACM SIGPLAN Notices. 1996. Vol. 31. No. 2. P. 13–20.

9. *Kasyanov V.N.* Sisal 3.2: functional language for scientific parallel programming. Enterprise Information // Systems. 2013. Vol. 7. No. 2. P. 227–236.

PERSPECTIVES OF FUNCTIONAL PROGRAMMING OF PARALLEL COMPUTATIONS

L. V. Gorodnyaya [0000-0002-4639-9032]

A.P. Ershov Institute of Informatics Systems (IIS)

Abstract

The article is devoted to the results of the analysis of modern trends in functional programming, considered as a metaparadigm for solving the problems of organizing parallel computations and multithreaded programs for multiprocessor complexes and distributed systems. Taking into account the multi-paradigm nature of parallel programming, the paradigm analysis of languages and functional programming systems is used. This makes it possible to reduce the complexity of the problems being solved by methods of decomposition of programs into autonomously developed components, to evaluate their similarities and differences. Consideration of such features is necessary when predicting the course of application processes, as well as when planning the study and organizing the development of programs. There is reason to believe that functional programming has the ability to improve programs performance. A variety of paradigmatic characteristics inherent in the preparation and debugging of long-lived parallel computing programs are shown.

Keywords*: functional programming, paradigm decomposition, parallel computing, multi-paradigm programming languages.*

.

REFERENCES

1. *Koopman P., Michels S., Plasmeijer R.* Dynamic Editors for Well-Typed Expressions // Trends in Functional programming/ 22nd International Symposium, TFP 2021, February 17–19, 2021. Springer, LNCS 12834. P. 44–66.

2. *Gorodnyaya L.V., Kirpotina I.A.* On the Problem of Reliability of Historical Factography Available on the Internet // Proceedings of the SoRuCom-2017. Forth International Conference «Computer Technology in Russia and in the Former Soviet Union». Zelenograd, the city of Moscow, October 3–5. Prof. A.N. Tomilin, Ed. Moscow, 2017. P. 40-49.

3. *Gorodnyaya L.V*. O predstavlenii rezul'tatov analiza yazykov i sistem programmirovaniya // Nauchnyj servis v seti Internet: trudy XX Vserossijskoj nauchnoj konferencii (17-22 sentyabrya 2018 g., g. Novorossijsk). M.: IPM im. M.V. Keldysha, 2018. S. 262–277. https://doi.org/10.20948/abrau-2019-03

4. *Gorodnyaya L.V.* Podhod k ocenke trudoyomkosti programmirovaniya // Nauchnyj servis v seti Internet: trudy XXII Vserossijskoj nauchnoj konferen-cii (21–25 sentyabrya 2020 g., onlain). M.: IPM im. M.V. Keldysha, 2020. S. 192–209.
https://doi.org/10.20948/abrau-2020-3
https://keldysh.ru/abrau/2020/theses/3.pdf

5. *Averbuh V.L.* Vizualizaciya programmnogo obespecheniya. Ekaterin-burg: IMM UrO RAN, 1995. 168 s.

6. *Gorodnyaya L.V.* Uchebnyj yazyk parallel'nogo programmirovaniya SINHRO // Yazyki programmirovaniya i kompilyatory—2017. Trudy konferen-cii. Yuzhnyj federal'nyj universitet; pod red. D.V. Dubrova. Rostov-na-Donu: Izd-vo Yuzhnogo federal'nogo universiteta, 2017. S. 92–97.
URL: http://plc.sfedu.ru/files/PLC-2017-proceedings.pdf

7. *Gorodnyaya L.V.* Perspektivno strategicheskie paradigmy programmirovaniya Akademika Andreya Petrovicha Ershova. 5-ya mezhdunarodnaya konferenciya «Razvitie vychislitel'noj tekhniki v Rossii, stranah byvshego SSSR i SEV (SORUCOM 2020)». Moskva, 6–8 oktyabrya 2020 g. S. 83–97.

8. mpC: A Multi-Paradigm Programming Language for Massively Parallel Computers // ACM SIGPLAN Notices. 1996. Vol. 31. No. 2. P. 13–20.

9. *Kasyanov V.N.* Sisal 3.2: functional language for scientific parallel programming. Enterprise Information // Systems. 2013. Vol. 7. No. 2. P. 227–236.

СВЕДЕНИЯ ОБ АВТОРЕ

ГОРОДНЯЯ Лидия Васильевна – старший научный сотрудник Института систем информатики имени акад. А.П. Ершова СО РАН, доцент Новосибирского государственного университета, специалист в области системного программирования и образовательной информатики.

Lidia Vasiljevna GORODNYAYA – Senior Researcher of A.P. Ershov Institute of Informatics Systems, Siberian Branch of the Russian Academy of Sciences, Associate Professor of Novosibirsk State University, a specialist in system programming and educational informatics.

email: lidvas@gmail.com
ORCID: 0000-0002-4639-9032

Материал поступил в редакцию 5 ноября 2021 года