К ОПРЕДЕЛЕНИЮ ПАРАДИГМЫ ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ

Н.В. Шилов, Л.В. Городняя, А.Г. Марчук

1. **Введение. Что такое "парадигмы программирования"?**

Одна из "культовых" книг по философии и методологии науки – это диссертация Томаса Куна "Структура научных революций", защищенная около 40 лет назад [8]. Согласно Т. Куну парадигма – это метод, подход к формулировке задач (проблем) и путей их решения. Само слово "парадигма" греческого происхождения и означает "пример", "образец", а в общефилософском смысле обозначает категорию, состоящую из сущностей с общими характеристиками. Первым, кто явно ввел в употребление понятие "парадигма программирования" был Роберт Флойд в лекции в 1978 г. по случаю присуждения ему премии им. Тьюринга [10]. В своей лекции Р. Флойд ссылается на книгу Т. Куна и трактует "парадигмы программирования" именно как разные способы постановки и решения программистских задач, но подчеркивает, что концептуально эти "разные способы" фиксируются обычно на уровне языков программирования.

Многообразие языков программирования является распределенным хранилищем научного знания о парадигмах, опыте и истории применения информационных технологий. Для оперирования этим знанием необходимо иметь разумную систему классификации языков и парадигм, с современной технической и образовательной поддержкой. Подход к созданию такой классификации, намечен в работе [1]. Поэтому что бы разобраться с парадигмами параллельного программирования придется в некоторой степени разобраться с многообразием языков программирования.

Издательство O'REILY, специализирующееся на публикации литературы по языкам программирования, подготовило плакат History of Programming Languages [12]. Его полная версия содержит сведения о хронологии и влиянии друг на друга 2500 языков программирования. Заметим, однако, что этот плакат не содержит сведений о парадигмах языков программирования, и его можно принять за основу "путеводителя" только лишь на начальном этапе становления программирования. Дело в том, что за полувековую историю развития компьютерных наук и информационных технологий эволюционировало само понятие алгоритма. От предписанной последовательности действий конечного автомата произошел переход к произвольному графу интероперабельных систем, функционирующих над семейством взаимодействующих автоматов. Именно этот переход влечет и появление парадигмы параллельного программирования.

Сложность такого перехода можно оценить геометрической аналогией. Точка не имеет измерений, линия имеет длину, поверхность – длину и ширину и т. д. Аналогично, действие взятое отдельно не имеет измерения. Программа имеет одно изменение: это последовательности действий, образующих процесс ее выполнения. А параллельная программа имеет два измерения: её «ширина» равна количеству параллельных ветвей, к которым может быть сведено ее выполнение. Поэтому становится актуальным создание удобных языков параллельного программирования [3, 5, 6].

2. **Мир языков программирования и параллелизм**

В середине 1970-х годов активное исследование методов параллельного программирования рассматривалось как ведущее направление преодоления кризиса технологии программирования. Теперь рост интереса к параллельному программированию связан с переходом к массовому производству многоядерных архитектур.

Проблемы создания языков параллельного программирования связаны с большой дистанцией между высоким уровнем абстрактных понятий, в которых описываются решения сложных задач, и низким уровнем конкретных аппаратных средств и архитектурных принципов управления параллельными вычислениями (потактовая синхронизация, совмещение действий во VLIW-архитектурах, сигналы, семафоры критических участков, рандеву и т. п.) Еще одна проблема связана с неоднозначностью семантической декомпозиции сложных определений и трудоемкостью определения реализационных особенностей практичных систем программирования. Для удобной классификации наиболее важно выделение минимальных учебного концентра и реализационного ядра в виде взаимодействия основных семантических систем, таких как обработка данных, их хранение и структурирование, управление обработкой данных. Нередко реализационное ядро языка содержит ряд понятий, не имеющих прямого представления в анализируемом языке.

На практике параллельные процессы встречаются при работе на уровне ОС, при организации вычислений на базе суперкомпьютеров, при сетевой обработке данных, при компиляции и оптимизации программ и т. п. Проблемы подготовки параллельных программ для всех столь разных работ обладают общностью, но есть и существенная специфика, требующая понимания разницы в критериях оценки программ и информационных систем для различных применений.

Чтобы представить эту прагматическую разницу при отработке методики определения парадигмы языка программирования, выделены три уровня сложности, отражающие расширение языковой поддержки жизненного цикла программ и рост реализационной сложности определения языка:

- кодирование на языках низкого уровня (ЯНУ);
- программирование на языках высокого уровня (ЯВУ);

- разработка программ на языках сверхвысокого уровня (ЯСВУ).

Для ЯНУ характерно, что все действия в программе выражены явно. На уровне операционной системы (ОС) информационная обработка выглядит как семейство взаимодействующих процессов.

Программе на ЯВУ обычно соответствует семейство допустимых процессов, определение которого представлено формальной семантикой языка. Система программирования (СП), поддерживающая ЯВУ, как правило, порождает один из процессов этого семейства. Такое сужение диктуется необходимостью воспроизведения процессов при отладке программ. Текст программы на ЯВУ обычно обретает билинейность – линия представления процесса обработки данных в нем совмещена с описанием типов данных. Возникает нечто вроде пространственной аппроксимации процесса, используемой при проверке корректности программ − статический или динамический контроль типов данных. Главное, чего не хватает в ЯВУ для представления взаимодействующих процессов – это типизация схем управления.

Подготовка программ на базе языков сверхвысокого уровня (ЯСВУ) нацелена на длительный срок жизни запрограммированных решений особо важных и сложных задач. Удлинение жизненного цикла достигается представлением обобщенных решений с определенной степенью свободы по отношению к полным пространствам допустимых смежных компонент, реализованных ранее или планируемых на будущее. Реализационное сужение семейства процессов, допускаемых семантикой ЯСВУ, противоречит его целям или концепциям по следующим прагматическим мотивам:

- высокий уровень абстрагирования программируемых решений;
- решаются задачи, зависящие от непредсказуемых внешних факторов;
- базовые средства и алгоритмы вычислений используют разные модели параллелизм;
- актуальны прагматические требования к темпу и производительности вычислений;
- эксплуатируются динамически реконфигурируемые многопроцессорные комплексы.

Основные парадигмы последовательного программирования на ЯВУ – это императивно-процедурное, функциональное и логическое программирование [11].

Для стандартного, императивно-процедурного программирования характерно четкое разделение понятий "программа" и "данные" с учётом статических методов контроля типов данных и оптимизации программ при компиляции. Общий механизм интерпретации стандартной программы естественно представить как автомат с отдельными таблицами имен для переменных, значения которых подвержены изменениям, и для меток и процедур, определения которых неизменны.

В функциональном программировании постановка задачи и решение задачи – определения функции, а вычисление – это цепочка равенств, где каждое следующее получается из предыдущего в результате однократного применения функции к аргументу (или аргументам). Функциональное программирование рассматривает процесс обработки данных как композицию их отображений с помощью универсальных функций. Программа при таком подходе не более чем одна из разновидностей данных. Реализация процесса – функция, определяющая ход процесса. Функциональная модель представления процессов позволяет легко описывать взаимодействие процессов в виде функционалов, т. е. функций над процессами, представленными как функциональные переменные.

Функциональный подход к параллельному программированию [4] ценен возможностью унификации понятий, различие которых мало существенно с точки зрения их реализации. При необходимости можно одинаково работать с процессами, объектами, системами и контекстом. Все это определенные структуры данных с заданной схемой взаимодействия и разными предписанными ролями в более общем процессе информационной обработки. При необходимости взаимодействие процессов может быть пошагово синхронизовано. Взаимодействия процессов естественно могут быть заданы как взаимосвязанные функции или потоки.

Логическое программирование (ЛП) сводит обработку данных к выбору произвольной композиции определений (уравнений, предикатных форм), дающей успешное получение результата. Именно обработка формул является основой – вычисления рассматриваются как поиск доказательства формул. При неуспехе происходит перебор вариантов определений. В языках логического программирования считают возможным прямой перебор вариантов, сопоставляемых с образцами, и организацию возвратов при неудачном выборе. Перебор вариантов выглядит как обход графа в глубину. Имеются средства управления перебором с целью исключения заведомо бесперспективного поиска.

3. **Модели параллелизма в языках высокого уровня**

В параллельном программировании на языках высокого уровня постановка задачи и ее решение предполагают, что несколько вычислительных потоков или/и процессов могут работать одновременно (как говорят "параллельно") с целью получить ускорение (во сколько раз быстрее?) за счет использования имеющихся аппаратных ресурсов [2]. Так как существует определенный терминологический разнобой в использовании понятий "поток" и "процесс", то сначала следует обсудить и зафиксировать наше понимание этих терминов.

Поток – это последовательная программа, при потоковом параллелизме все потоки работают над общей памятью (и поэтому не нуждаются в специальных средствах передачи данных друг-другу), зато нуждаются в средствах синхронизации (которые обычно задаются посредством управляющей конструкции "fork-join". Например, двухпотоковый алгоритм вычисления факториала числа N в одном потоке вычисляет произведение всех чётных чисел из диапазона [1..N], в другом – произведение всех нечетных чисел из этого же диапазона;

произведения всех четных и нечетных чисел вычисляются и хранятся в разных переменных общей памяти, а по завершении этих потоков значения этих переменных перемножаются. Обычно потоковый параллелизм реализуется на многоядерных процессорах.

Процесс – это поименованный программный шаблон, с которого можно сделать сколько угодно программ-копий, которые называются экземплярами процесса, каждый из которых имеет свою собственную память. Обмен данными и синхронизация между экземплярами процессов происходит посредством обмен сообщениями через общий буфер определенной ёмкости (в том числе – буфер нулевой емкости для организации рандеву). Обмен сообщениями через буфер обмена обычно задаётся командами "send-receive" Экземпляры процессов могут порождать новые экземпляры процессов по именам процессов. Обычно процессный параллелизм реализуется в "многопроцессорной вычислительной среде" – совокупности процессоров, соединенных каналами для передачи сообщений, каждый из которых имеет свою собственную (локальную) память, причём один или несколько процессоров – маршрутизаторы сообщений (какое сообщение кому передать). Например, можно представить себе следующую двухпроцессный алгоритм вычисления факториала числа N: единственный экземпляр "главного" процесса (создающийся при запуске алгоритма) порождает (создаёт) два экземпляра "вычислительного" процесса, посылает первому из низ сообщение "вычислить произведение всех чисел в диапазоне [1..(N/2)]", а второму – "вычислить произведение всех чисел в диапазоне [(N/2 + 1)..N]", ждёт ответных сообщений со значениями от них, и завершается вычислив произведение полученных значений.

Теперь можно уточнить, что такое парадигма "параллельного программирования". Так как парадигма – это способ постановки и решения задач, то надо определиться, что такое "параллельная постановка" и "параллельное решение" программистских задач. Параллельная постановка – это описание желаемого результата, имеющихся аппаратных возможностей, и, возможно, базовый "непараллельный" алгоритм, позволяющий получить этот самый желаемый результат. Параллельное решение – это пара алгоритмов: собственно параллельный алгоритм и алгоритм-планировщик, которые используют имеющуюся аппаратуру, не нарушают ни каких ограничений, вычисляют желаемый результат, возможно достигающий ускорения по сравнению с базовым алгоритмом. Описание желаемого результата в простейшем случае – это описание наблюдаемого поведения в терминах входных сообщений (получаемых алгоритмом) – выходных сообщений (посылаемых алгоритмом), возможно дополненное некоторым критерием к его качеству, надежности, безопасности и т. п. Аппаратные возможности и ограничения – это число и тип вычислительных устройств с индивидуальной памятью и как они связаны между собой каналами связи.

4. **Языки параллельного программирования**

Теперь скажем немного о языках параллельного программирования. Исторически первое предложение по организации языка высокого уровня для параллельных вычислений было сделано в 1962 году в виде языка APL, однако в параллельном программировании трудно привести примеры "классических" языков. Зато сейчас очень популярны и востребованы параллельные расширения FORTRAN и C библиотеками MPI (Message Passing Interface) и OpenMP (Open Multi-Processing): MPI – это библиотека поддержки процессного параллелизма, а OpenMP – потокового.

Довольно известна проблема второго языка программирования – это интеллектуальные, психологические и технологические затруднения, которые испытывает программист при необходимости перейти с привычного ему "первого" языка программирования на новый для него "второй" язык. Само существование этой проблемы объясняет, почему достаточно распространен подход к параллельному программированию через добавление средств параллелизма в популярные языки программирования, такие как Fortran, Pascal, Ada, C и др. Существуют версии ряда стандартных языков императивного программирования, приспособленные к выражению взаимодействия последовательных процессов в предположении, что в каждый момент времени существует лишь один процесс. При таком подходе в программе выделяются критические интервалы, учет которых полезен при распараллеливании программ.

Применение параллельных архитектур повышает производительность при решении задач, явно сводимых к обработке векторов. Но автоматические методы распараллеливания редко способны обеспечить значительное ускорение вычислений. Более успешным может быть выражение языковыми средствами параллелизма на уровне постановки задачи. В таком случае при оптимизирующей компиляции возможен аккуратный выбор эффективной схемы параллелизма.

Отметим, что функциональные языки способствуют разработке корректных параллельных программ. Одна из причин заключается в том, что функциональные программы свободны от побочных эффектов и ошибок, зависящих от реального времени. Это существенно упрощает отладку. Функциональные программы переносимы на разные архитектуры, операционные системы или инструментальные окружения. В отличие от императивных языков, функциональные языки ближе к языкам спецификаций (Рефал и ML, например, изначально задумывались как языки спецификаций, а не языки программирования), поэтому использование функциональных языков в проектах упрощает кодирование, статический анализ информационных потоков и схем управления.

Функциональную программу, которая является параллельной, легче написать и отладить, освободившись от большинства сложностей, связанных с выражением частичных отношений порядка между отдельными операциями уровня аппаратуры. Это позволяет разработчику сконцентрироваться на конструировании алгоритмов и разработке программ в терминах крупноблочных и регулярно организованных

построений, опираясь на естественный параллелизм уровня постановки задачи. Реализация процесса - функция, определяющая ход процесса. Функциональная модель представления процессов позволяет легко описывать взаимодействие процессов в виде функционалов, т. е. функций над процессами, представленными как функциональные переменные.

Основные идеи языков параллельного программирования были развиты и обогащены в языках БАРС и Поляр [7, 9] в направлении использования сетевых представления при организации асинхронных процессов. Высокопроизводительное программирование на языке БАРС нацелено на обеспечение высокопроизводительных вычислений и организацию асинхронных параллельных процессов. Радикальное продвижение в повышении уровня программирования, предложенное в языке БАРС, заключается в переносе идеи типизации данных на проблему типизации схем управления. Процесс обработки данных рассматривается как распределенная система, находящаяся под сетевым управлением. Узлы такой системы могут сработать в зависимости от условий готовности разной природы: доступность ресурсов, сигналы монитора, внутри сетевые отношения, иерархия сетей, правила функционирования разносортных подсетей.

ЛИТЕРАТУРА:

1. Т.А. Андреева, И.С. Ануреев, Е.В. Бодин, Л.В. Городняя, А.Г. Марчук, Ф.А. Мурзин, Н.В. Шилов Н.В. "Образовательное значение классификации компьютерных языков" // Прикладная информатика, 2009, №6 (24), стр. 18-28.
2. В.В. Воеводин, Вл.В. Воеводин "Параллельные вычисления". СПб.: БХВ-Петербург, 2002.
3. Л.В. Городняя "Парадигмы параллельного программирования в университетских образовательных программах и специализации" // Всероссийская научная конференция "Научный сервис в сети Интернет: решение больших задач" Новороссийск-Москва, 2008, стр. 180-184.
4. Л.В. Городняя "Функциональный подход к описанию парадигм программирования" // Российская академия наук, Сибирское отделение, Институт систем информатики им. А.П. Ершова, припрете 152, 2009.
5. Л.В. Городняя "О проблеме начального обучения параллельному программированию" // Пятая Сибирская конференция по параллельным и высокопроизводительным вычислениям. Томск. 2009. стр. 18-22.
6. Л.В. Городняя, Н.В. Шилов "Модели параллелизма в языках и преподавании программирования" // Труды XIV Байкальской Всеросийской конференции "Информационные и математические технологии в науке и управлении". Часть II. Институт систем энергетики им. Л.А. Малентьева СО РАН, Иркутск, 2009, стр. 257-266.
7. В.Е. Котов "МАРС: архитектуры и языки для реализации параллелизма" // В сб. "Системная информатика", вып 1. Новосибирск: Наука. Сиб. Отд-ние, 1991, стр. 174-194.
8. Т. Кун "Структура научных революций", издательство АСТ, 2003.
9. Т.И. Лельчук, А.Г. Марчук "Язык программирования Поляр: описание, использование, реализация"// Академия наук СССР, Сибирское отделение, Вчислительный центр, Новосибирск, 1986. 94 стр.
10. Р. Флойд "О парадигмах программирования" // В сб. "Лекции лауреатов премии Тьюринга". М: Мир, 1993.
11. Н.В. Шилов "Заметки о преподавании парадигм программирования" // IV Международная научно-практическая конференция "Современные информационные технологии и ИТ-образование". М.:ИНТУИТ.РУ. 2009. Стр.318-325.
12. "History of Programming Languages" // Доступен на http://www.oreilly.com/news/graphics/prog_lang_poster.pdf (Проверено 28 мая 2010.)