ПАРАДИГМА ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ: УЧИТЬ ИЛИ НЕ УЧИТЬ (ВОТ В ЧЁМ ВОПРОС)

Н.В. Шилов, Л.В. Городняя, Е.В. Бодин

1. Препятствие на пути к экзафлопному будущему

За последние двадцать лет мир информатики претерпел значительные изменения. Необходимость изучения, преподавания и понимания параллельного программирования возрастает. Если в середине 70-х годов XX века параллельное программирование считались главной перспективой для преодоления аппаратных ограничений, то теперь рост интереса к параллельному программированию обусловлен прогрессом в области аппаратуры: массовому производству устройств с многоядерной архитектурой, быстрое развитие суперкомпьютеров и распределённых систем обработки и хранения данных, широкому распространению графических процессоров и т. д.

Чтобы воспитать специалистов и экспертов в параллельном программировании, необходимы правильный подход к постановке образования по параллельному программированию, а не только к суперкомпьютерному образованию [28]. Мы думаем, что

- постановка образования в области параллельного программирования на данном этапе находится в зачаточном состоянии и поэтому пока является скорее препятствием на пути к овладению новыми аппаратными возможностями;
- преодолеть это препятствие в высшей школе возможно только на пути комплексного изучения параллельного программирования как отдельной парадигма программирования [11, 12], заслуживающей изучения в полном объёме, а не на уровне конкретных пакетов и архитектур;
- раннее (школьное) преподавание параллельного программирования также существенно, как и знакомство с функциональным и логическим программированием: это требует разработки и реализации понятных учебных языков программирования, поддерживающих различные парадигмы программирования, в том числе - параллельное программирование [4].

При подготовке настоящей статьи был использован широкий список публикаций, поскольку нам хотелось сохранить объективность в анализе образовательных и парадигмальных проблем параллельного программирования. Но это не просто обзор литературы по данному вопросу, но самостоятельное исследование философских и методологических оснований преподавания параллельного программирования, базирующееся как на нашем образовательном опыте, так и на публикациях других исследователей и преподавателей.

Остальная часть статьи организован следующим образом. В следующем разделе 2 мы конспективно изложим наше понимание парадигм программирования. В разделе 3 обсуждаются особенности параллельных постановок и решений, а в разделе 4 – теории и языки параллельного программирования. Раздел 5 посвящён «учебному плану параллельного программирования», представляет подход «рано и часто» в преподавании параллельного программирования и базовую концепцию языка параллельного программирования КУБИК для преподавания в школе.

2. Парадигмы программирования: думай по-разному!

Парадигма - это подход к формулированию задач и способам их решения. Термин происходит от греческого слова со значением *пример*, *модель*, *образец*. Современное значение этот термин получил в известной книге Томаса Куна «Структура научных революций», впервые опубликованной в США в 1960 г., выдержавшей 3 переиздания и переведённой на многие языки (в том числе на русский в 2003 г.). Роберт Флойд впервые явно использовал его в контексте информатики. В частности, он посвятил свою лекцию по случаю присуждения ему премии им. А. Тюринга в 1968 г. *парадигмам программирования*, но, к сожалению, он не определил явно это понятие. (Подробнее об этом см. в [12].)

Недавно Петер Ван Рой опубликовал таксономию *Основные парадигмы программирования* (http://www.info.ucl.ac.be/~pvr/paradigms.html) с 27 различными парадигмами и уточнил свою позицию в статье [24]. Как ни удивительно, ни эта упомянутая статья, ни монографическая учебная литература (например [23, 10]) не дают убедительного и точного определения понятия *парадигма программирования*. Можно привести только следующую цитату из [24]: парадигма программирования - это «подход к программированию компьютера, основанный на математической теории или согласованном множестве принципов. Каждая парадигма поддерживает множество понятий, которое делает её лучшей для некоторого рода задач». (Тем не менее, это концептуальное определение и «парадигмизация» - краеугольные камни подхода «Понятие - Метод - Модель», используемого при обучении информатике и программированию в Лёвенском католическом университете [25]).

Мы предлагаем наше собственное определение, более детальное, но, как мы надеемся, не противоречащее общепринятому представлению.

1. Парадигмы программирования - альтернативные подходы к задачам представления и обработки данных. Парадигмы фиксируются в виде формальных теорий и аккумулируется в языках программирования.
2. Научное и индустриальное значение парадигмы программирования характеризуется множеством задач или областей приложения, для которых эта парадигма подходит лучше других.
3. Образовательное значение парадигм программирования состоит в том, чтобы научить думать по-разному о задачах программирования и выбирать наиболее подходящую парадигму для решаемой задачи.

Предварительные формулировки первого пункта этого определения была опубликована нами ранее (см., например, [11, 12]). Второй пункт предложен во многом под влиянием вышеприведённого определения Петера Ван Роя. Последний пункт был предложен и обоснован в [1]. Впервые это определение было сформулировано в недавней статье [26] в более общем виде (для *компьютерных языков*).

3. Параллельные постановки и решения

К сожалению, многие исследователи полагают, что параллельное программирование не является самостоятельной парадигмой в программировании, а можно только вести речь о параллельной исполнении программ, разработанных в «основных» парадигмах программирования. Так, например, в уже знакомой нам таксономии «Основные парадигмы программирования» [23] *параллельного программирования* нет, а есть *concurrency*, «размазанная» по нескольким (6-8) классам таксономии. (Здесь мы хотим заметить, что русский перевод английских терминов *concurrency* и *parallelism* одним русским термином *параллелизм* нам представляется неправильным, как это будет объяснено ниже. Правда, есть вариант перевода *concurrency* как *многопоточность*, но этот вариант теперь занят для перевода термина *multithreading*. Поэтому мы в этой статье будем пользоваться английским термином *concurrency* и русским термином *параллелизм*).

Не только нам такая ситуация представляется странной.

Начнём с цитаты из доклада Дэна Гроссмана, сделанного на прошлогоднем семинаре *Curricula for Concurrency and Parallelism* (Невада, 17 октября 2010 года): «Под *параллелизмом* я подразумеваю использование дополнительных вычислительных ресурсов для ускорения решения задачи. Под *concurrency* я понимаю правильно и эффективно организованный доступ к разделяемым ресурсам. Хотя такое использование этих терминов и не является общепризнанным, различие существенно» [17].

Хотелось бы уточнить и расширить это видение в соответствии с нашим общим понятием парадигмы программирования.

Прежде всего все парадигмы программирования должны характеризоваться особенностями постановки задач и обработки данных. В частности, постановка задачи в параллельном программировании обычно включает

1. некоторую известную формализацию задачи и (возможно) одно или несколько *базовых* (условно говоря «непараллельных») решений этой задачи,
2. описание (спецификацию) целевой *параллельной архитектуры* (ресурсы и вычислительные устройства, топология и свойства коммуникационного оборудования),
3. целевую функцию в виде *показателей производительности* (performance metrics) (например, ускорение по сравнение с базовыми решениями, эффективность в расчёте на одно вычислительное устройство, степень загрузки вычислительных устройств и т. д.).

Параллельная программа (параллельный алгоритм) - это чаще всего *пара* программ (алгоритмов):

1. первая из них - это *выполнимая спецификация*, определяющая шаги вычисления, их зависимости и условия для одновременного выполнения;
2. вторая - это concurrency-менеджер (или *планировщик*), распределяющий нагрузку (по шагам вычисления) между вычислительными устройствами и доступ к ресурсам и коммуникационному оборудованию.

Образно говоря, для написания выполнимой спецификации надо уметь *думать параллельно*, а для написания планировщика – уметь *думать конкурентно*. Заметим, что явное *думать параллельно* не то же самое, что неявный параллелизм в функциональном и логическом программировании.

Параллельная программа вместе со списком *реальных* замеров эффективности образует параллельное решение задачи.

Из-за ограничений объема здесь нет места для подробного (с использованием статистики) обоснования корректности описанной выше трактовки «шаблона» постановки и решения задач в параллельном программировании, поэтому мы ограничимся только одним примером, в качестве которого сошлёмся на статью нашей уважаемой коллеги Ольги Леонидовны Бандман «Discrete Models of Physicochemical Processes and Their Parallel Implementation» [14]. Эта статья предлагает параллельный метод для *симуляции дискретных моделей физико-химических кинетических процессов* (это известная формализованная задача). Метод состоит в представлении моделей в виде *расширенного клеточного автомата* (КА) Фон-Неймана (эти модели представляют архитектуру и определяют средства задания выполнимых спецификаций) вместе с

преобразованиями асинхронных КА в блочно-синхронные КА (это планировщик). Указанный метод показал *ускорение* по сравнению с традиционным способом моделирования.

4. Параллельные теории и языки

Наше концептуальное определение парадигм программирования предполагает существование формальной (математической) теории и языков программирования, ассоциированных с этой парадигмой, а также набора задач или областей, для которых данная парадигма подходит лучше других.

Есть множество различных формальных математических моделей параллелизма и concurrency. Всем (а всем ли?) известны, например, *сети Петри* [7, 8] как *семантическая модель* для concurrency. В России достаточно популярны исследование и преподавание *семантической теории процессов* [2, 3, 9]. За сетями Петри по популярности в учебных программах высших учебных заведений следует теория *взаимодействующих последовательных процессах* CSP Хоара [18, 8], которая является *языком с фиксированным синтаксисом и денотационной (алгебраической) семантикой*. (Многие ли знает, что такое *денотационная семантика*?) Заграницей исследуют и преподают несколько *синтаксических исчислений*, формализующих различные аспекты concurrency и параллелизма: *исчисление взаимодействующих систем* (CCS) [21] и *пи-исчисление* [22] Милнера для взаимодействующих мобильных систем, *исчисление окружений* и его вариации для мобильных агентов и безопасности Кардели [15]. (Но где и кто преподаёт эти темы у нас?) И наконец следует назвать виды *динамических*, *процессных* и (особенно) *временных (темпоральных) логик* [19, 20, 16, 27, 6], которые используются для *спецификации* и *верификации* concurrency и параллелизма в семантических моделях и завоевали свою популярность за рубежом и у нас благодаря автоматической верификации протоколов распределённых систем и оборудования.

Хотя «математику уж затем учить надо, что она ум в порядок приводит» (М.В. Ломоносов), но, к сожалению, перечисленные математические теории пока не вошли в программы **Интернет-Университета Суперкомпьютерных Технологий** (http://hpcu.ru/). Остаётся довольствоваться факультативным уровнем преподавания математической теории concurrency и параллелизма. (См., например, программу такого курса для студентов Новосибирского Государственного университета [13].)

В области языкотворчества для параллельного программирования наблюдаются две тенденции: разработка новых языков программирования с явной или неявной параллельностью или concurrency, и разработка специальных «параллельных» библиотек или программных интерфейсов (API) для «обычных» языков программирования. Так как второй подход широк представлен как в традиционном образовании, так в дистанционном (см., например, программы Интернет-Университета Суперкомпьютерных Технологий), поэтому мы не будем здесь обсуждать эту тему. Точно также мы ограничимся только кратким замечанием, что задачи, наиболее подходящие для параллельного программирования, это относятся к областям, где вычислительная мощность действительно имеет значение.

5. Решения для будущего

Если познакомиться с докладами, сделанными на семинаре *Curricula for Concurrency and Parallelism* (Невада, 17 октября 2010 года), то складывается впечатление, что основы параллельного программирования следует преподавать рано (на первых курсах) и часто (в нескольких курсах) с преподаванием элементов формальных моделей и использованием учебного специализированного компьютерного инструментария. Но как ни удивительно, нам не известно работ, посвящённых преподаванию параллельного программирования на школьном уровне. Одна из целей нашей статьи - представить проект КУБИК, предназначенный восполнить этот дефицит.

В основе проекта лежит новый мультипарадигмальный язык программирования КУБИК, развивающий язык для раннего обучения программированию Робик [5]. Такой выбор обусловлен тем, что в языке Робик есть понятие *исполнителя*, причём несколько исполнителей могут существовать независимо, так что этот язык позволяет моделировать асинхронные процессы. Конечно, в новом языке должны быть соответствующие примитивы для синхронизации, передачи сообщений, часы и таймеры. Генерические (родовые) типы (массивы, списки, множества и т. п.) можно использовать для определения структурных типов данных, для которых можно использовать неявный функциональный параллелизм (операции типа map и reduce). Управляющие конструкции (такие как репликация, последовательная композиция, детерминированный и недетерминированный выбор, параллельная композиция) можно использовать для построения сложных процессов. Во время отладки должно быть возможно определить относительное время выполнения каждого действия. Кроме команд, изменяющих окружение, состояние и множество команд исполнителя, должны быть специальные команды для замены исполнителя/среды и для разрешения проблем взаимодействия (тупиков, голодания и т. д.).

Реализация языка КУБИК выполняется в рамках проекта 11-01-12087-офи-м-2011 и включает три уровня: элементарное ядро, прагматическое расширение и оболочку. Ядро будет поддерживать базовые инструменты уровня операционной системы: условия готовности и завершения процессов, очереди и пакетную обработку. Прагматическое расширение будет поддерживать множественные значения, приоритеты процессов, фильтры для данных, сетевую синхронизацию действий, оптимизацию и т. д.

ЛИТЕРАТУРА:

1. Т.А. Андреева, И.С. Ануреев, Е.В. Бодин, Л.В. Городняя, А.Г. Марчук, Ф.А. Мурзин, Н.В. Шилов Образовательное значение классификации компьютерных языков // Научно-практический журнал «Прикладная информатика». 2009. №6 (24). С.18-28.
2. И.Б. Вирбицкайте Семантические модели в теории параллелизма. Новосибирск: Изд-во ИСИ СО РАН, 2000.
3. И.Б. Вирбицкайте Сети Петри: модификации и расширения. Новосибирск: Изд-во НГУ, 2005.
4. Л.В. Городняя О постановке курса «Функциональное программирование параллельных вычислений» // В сб. Труды Международной суперкомпьютерной конференции «Научный сервис в сети Интернет: суперкомпьютерные центры и задачи» (электронное издание). / М: Изд-во Московского Университета. 2010. С.193-196.
5. Г.А. Звенигородский Первые уроки программирования. М. Наука, 1985. (Библиотечка «Квант», вып.41, доступна на http://www.math.ru/lib/files/djvu/bib-kvant/kvant41.djvu.)
6. Ю.Г. Карпов Model Checking. Верификация параллельных и распределённых систем. Спб.: БХВ-Петербург, 2009.
7. В.Е. Котов. Сети Петри. М.: Наука, 1984.
8. В.Э. Малышкин, В.Д. Корнеев Параллельное программирование мультикомпьютеров. Новосибирск: Изд-во НГТУ, 2006.
9. А.М. Миронов Теория процессов. 2008. (Доступна на http://intsys.msu.ru/staff/mironov/processes.pdf.)
10. Н.Н. Непейвода Стили и методы программирования. М.: Интернет-университет информационных технологий, 2005. (Доступна на http://www.intuit.ru/department/se/progstyles/.)
11. Н.В. Шилов, Л.В. Городняя, А.Г. Марчук К определению парадигмы параллельного программирования // В сб. Труды Международной суперкомпьютерной конференции «Научный сервис в сети Интернет:суперкомпьютерные центры и задачи» (электронное издание). / М: Изд-во Московского Университета. 2010. С. 130-139.
12. Н.В. Шилов, Л.В. Городняя, А.Г. Марчук Параллельное программирование среди других парадигм программирования // Научно-практический журнал «Прикладная информатика». 2011. №2 (30). С. 120-129.
13. Н.В. Шилов Формальные модели параллельных вычислений. Программа дисциплины. Новосибирский государственный университет. Факультет информационных технологий. Кафедра параллельных вычислений. (Доступна на http://fit.nsu.ru/data_/courses/m_form_model_pv.pdf.)
14. O.L. Bandman Discrete Models of Physicochemical Processes and Their Parallel Implementation // In Methods and Tools of Parallel Programming Multicomputers. Second Russia-Taiwan Symposium. Lecture Notes in Computer Science. 2010. V.6083. / Springer. P.20-29.
15. L. Cardelli{ Mobility and Security. // In Foundations of Secure Computation, Friedrich L. Bauer and Ralf Steinbrggen (Eds.), NATO Science Series, Proceedings of the NATO Advanced Study Institute on Foundations of Secure Computation, Marktoberdorf, Germany, 27 July - 8 August 1999. IOS Press, 2000. P. 3-37.
16. E.M.Jr. Clarke, O. Grumberg, D.A. Peled Model Checking. MIT Press, 1999. (Есть русский перевод: Э.М. Кларк, О. Грумберг, Д. Пелед Верификация моделей программ: Model Checking. М.: МЦНМО, 2002.)
17. D. Grossman Ready-For-Use: 3 Weeks of Parallelism and Concurrency in a Required Second-Year Data-Structures Course. SPLASH 2010 Workshop on Curricula for Concurrency and Parallelism (Reno, Nevada, USA, October 17, 2010). (Доступна на http://www.cs.pomona.edu/~kim/CCP2010Papers/grossman_cic2010.pdf.)
18. C.A.R. Hoare Communicating Sequential Processes. Prentice Hall International, 1985. This book has been updated by Jim Davies at the Oxford University Computing Laboratory and the new edition (2004) is available for download as a PDF file at the http://www.usingcsp.com/. (Русский перевод: А.Ч. Хоар Взаимодействующие последовательные процессы. М.:Мир, 1989.)
19. Z. Manna, A. Pnueli The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer, 1992.
20. Z. Manna, A. Pnueli The Temporal Verification of Reactive Systems: Safety. Springer, 1995.
21. R. Milner A Calculus of Communicating Systems. Springer, 1980. (Lecture Notes in Computer Science, v.92)
22. R. Milner Communicating and Mobile Systems: the Pi-Calculus. Cambridge University Press, 1999.
23. P. van Roy, S. Haridi. Concepts, Techniques, and Models of Computer Programming. The MIT Press, 2004.
24. P. van Roy Programming Paradigms for Dummies: What Every Programmer Should Know. // In G. Assayag and A. Gerzso (eds.) New Computational Paradigms for Computer Music / IRCAM/Delatour, France, 2009, p.9-38.

25. P. van Roy The CTM Approach for Teaching and Learning Programming Programming. // In: Horizons in Computer Science Research, v.2. / Nova Publishers, 2010, p.1-26.
26. N.V. Shilov, A.A. Akinin, A.V. Zubkov, R.I. Idrisov Development of the Computer Language Classification Knowledge Portal // In Proceedings of the 8 Int. Ershov Informatics Conference (Novosibirsk, Russia, June 27 - July 01, 2011). Novosibirsk: A.P. Ershov Institute of Informatics Systems, 2011. P.255-260.
27. C. Stirling Modal and Temporal Properties of Processes. Springer, 2001.
28. Vl.V. Voevodin, V.P. Gergel Supercomputing education: the third pillar of HPC // Internet-journal Numerical Methods and Programming. 2010. Vol.11. P. 117-122. (Доступна на http://num-meth.srcc.msu.ru/english/zhurnal/tom_2010/v11r212.html.)