О парадигме параллельного программирования

Л.В. Городняя

ИСИ СО РАН им. А.П.Ершова, НГУ

Доклад посвящен проблеме формирования самостоятельной парадигмы параллельного программирования, вызванной расширением и развитием системы базовых понятий, необходимых для рациональной разработки информационных систем управления процессами на современной аппаратуре.

1. Введение

Парадигмы в программировании характеризуются стилем мышления при решении задачи, системой используемых понятий и особенностями их реализации. Стиль мышления и система понятий для параллельного программирования уже сложились в процессе эволюции языков программирования, но типовая их поддержка в системах программирования ещё не сформировалась [1]. Переход к параллельным алгоритмам связан с пересмотром содержания многих понятий и введением новых терминов, отражающих разного рода явления и эффекты, не имевшие особого значения для обычных последовательных алгоритмов. Ведущую роль в результативности такого перехода бесспорно играет содержание образовательных программ подготовки СКТ-специалистов.

2. Долгоживущие языки программирования

Новые и долгоживущие языки программирования как правило имеют мультипарадигмальный характер. Параллельное программирование использует средства, характерные для разных парадигм [3]. Это определяет возможность трансформационного подхода к накоплению правильности программных решений при разработке и модернизации параллельных программ на разных языках в рамках общей системы программирования. Следует особо отметить не столько сложность собственно параллельного программирования, но и трудоёмкость отладки программ для разных многопроцессорных конфигураций, необходимость разработки методов верифицирующей компиляции и оптимизации программных компонентов, средств масштабируемой макрогенерации кода и автоматизируемых трансформаций программ с удостоверением сохранения свойств при их комплексации из ранее отлаженных компонентов, приспособленных к многократному применению в разных условиях.

Развитие языков и систем программирования в настоящее время ориентировано на решение задач на основе общих библиотечных модулей, обеспечивающих эффективную организацию процессов, или подъязыков, допускающих многопоточное программирование. Существуют сотни функциональных языков программирования, ориентированных на разные классы задач параллельного программирования. Это не исключает реальную практику ручного распараллеливания ранее отлаженных обычных программ, приведения их к виду, удобному для применения производственных систем поддержки параллельных вычислений. Значительная часть таких работ носит технический характер и заключается в систематической реорганизации структур данных, изменении статуса переменных и включении в программу аннотаций, сообщающих компилятору об информационно-логических взаимосвязях. Существенным ограничением результата ручного распараллеливания является не только опасность повторной отладки алгоритма, но и его избыточная зависимость от характеристик целевой архитектуры.

3. Интеллектуальный вызов

Разнообразие моделей параллельных вычислений и расширение спектра доступной архитектуры следует рассматривать как вызов разработчикам языков и систем

программирования, способным решать проблему создания методов компиляции многопоточных программ для многопроцессорных конфигураций [2]. Язык должен допускать представление любых моделей параллелизма, проявляемого на уровне решаемой задачи или реализуемого с помощью реальной архитектуры, причем такое представление требует лаконичных форм и конструктивных построений, гарантирующих сохранение свойств программ при их реорганизации. Не менее важна расширяемость языка по мере развития средств и методов параллельных вычислений, темп которого превышает скорость осознания специалистами их возможностей.

Переход к параллельным алгоритмам влечёт пересмотр содержания многих понятий и введение новых терминов, отражающих разного рода явления и эффекты, не имевшие особого значения для обычных последовательных алгоритмов. Резкое расширение пространства решений задач меняет подходы к реализации решений, использующих параллелизм, и в некоторой мере сказывается собственно на постановке задач и планировании жизненного цикла программ решения задач, ориентированных на использование параллелизма.

Рассматривая задачу формализации языков параллельного программирования как путь к решению проблемы адаптации программ к различным особенностям используемых многопроцессорных комплексов и многоядерных процессоров, приходится признать, что решение этой проблемы требует разработки новых методов компиляции программ, и развития средств и методов ясного описания семантики языков параллельного программирования.

Для практичного решения проблемы разработки языков параллельного программирования и подходов к их реализации попадают в центр внимания вопросы системной поддержки парадигмы параллельного программирования как базовой, отражающей прагматические различия в условиях реализации и применения изобразительных средств, используемых в жизненном цикле программ.

Подготовка программ на базе языков сверхвысокого уровня позволяет представлять регулярные, эффективно реализуемые структуры данных, гарантирующие высокую производительность вычислений и надежность процесса разработки программ, включая подготовку программ для многопроцессорных конфигураций.

Парадигма параллельного программирования занимает нишу, связанную с реализацией программ выполнения вычислений на многопроцессорных системах для организации высокопроизводительных вычислений. Эта ниша обременена резким повышением трудоемкости отладки программ, вызванной комбинаторикой выполнения фрагментов асинхронных процессов. Полноценное решение проблем параллельного программирования требует создания более специализированного инструментария, некоторые механизмы реализации которого могут быть изучены в форме экспериментальной разработки учебного языка параллельного программирования [3].

4. Пространство парадигм программирования

В настоящее время по существу различимо более двух десятков парадигм программирования (ПП). Многие языки программирования (ЯП) относят к пяти-восьми парадигмам. Часть упоминаемых в разных источниках парадигм можно характеризовать как стили или методики, отражающие поиск путей снижения трудоёмкости программирования и повышения надёжности программ на базе доступных систем программирования (СП). Например, аспектно-ориентированное программирование поддерживается как макрорасширение ООП. Структурное программирование фактически сводится к ряду рекомендаций по стилю представления императивно-процедурных программ. Мета-программирование представляет собой технику компиляции программ в комплекте с типовыми элементами данных. Недетерминированное программирование иногда рассматривают как частный случай параллелизма.

При определении ПП обычно противопоставляются следующие характеристики ЯП:

- программируемые решения представляются в императивно-процедурной или в декларативной форме;
- обрабатываемые элементы данных позиционируются как адресуемые блоки памяти или независимо размещаемые значения;

- программа может быть защищена от изменений в процессе её выполнения или допускать программируемые модификации по ходу получения результатов вычисления;
- программа может быть целостной или собираться из типовых компонент и шаблонов;
- представленные в программе функции могут быть частичными, типизированными, обрабатывающими значения заданного домена или универсальными, дающими разумную реакцию на любой элемент данных;
- управление вычислениями выполняется последовательно или параллельно;
- порядок действий может быть определённым или недетерминированным;
- вычисления могут быть энергичными или ленивыми;
- области видимости имён могут быть глобальными или локализованными по иерархии конструкций с возможностью восстановления контекста;
- распределение и повторное использование памяти может быть действием в программе или выполняться автоматически СП;
- инициирование памяти первоначально размещаемыми значениями может требовать программируемых действий или выполняться в СП по умолчанию;
- домены значений могут быть независимыми или допускать пересечения;
- результат выполнения программы может быть рассредоточен по ряду переменных или сконцентрирован в специальном регистре;
- контроль правильности может выполняться статически – при анализе текста программы или динамически – при выполнении кода программы.

В практике признают основными парадигмы императивно-процедурного, функционального, логического и объектно-ориентированного программирования, суммарно покрывающие это поляризованное пространство, поддерживающее практичные механизмы снижения трудоёмкости полного жизненного цикла программ, с тенденцией продвижения к параллельному программированию, встраиваемому в контекст привычных ПП. Привычные Си и Фортран предлагают следующий выбор предпочтений:

- программируемые решения представляются в императивно-процедурной, дополненной описаниями типов данных, включая функции/процедуры;
- обрабатываемые элементы данных позиционируются как непрерывно адресуемые блоки памяти, а независимо размещаемые значения допускаются в основном при передаче параметров;
- программа обычно защищена от изменений в процессе её выполнения, но полезность программируемых модификаций получила право на жизнь в языке C#;
- программа целостна, хотя на уровне компиляции собирается из типовых шаблонов;
- представленные в программе функции обычно являются частичными, типизированными, поскольку универсальность часто приводит к противоречиям с системой типового контроля, упрощающей работу компилятора;
- управление вычислениями выполняется последовательно, а параллельное управление проникает в программу через специальные библиотечные модули и доработку программы «напильником»;
- порядок действий заранее определён, если нужна недетерминированность, то она моделируется;
- энергичные вычисления понятнее, хотя ленивые признаются более результативными;
- области видимости имён преимущественно являются глобальными или отчасти локализованными по иерархии модулей, вызовов процедур или классов объектов с возможностью восстановления контекста;
- распределение и повторное использование памяти выполняется действием в программе, но в языках Java и C# появилась возможность автоматической «сборки мусора»;
- инициирование памяти первоначально размещаемыми значениями может требует программируемых действий, хотя инструментальные оболочки делают это по умолчанию;
- домены значений обычно независимы, роль пересечений выполняют преобразования типов данных;

- результат выполнения программы рассредоточен по ряду переменных;
- контроль правильности выполняться статически при анализе компилируемой программы, что обосновывается экономией памяти и времени исполнения для динамического контроля.

5. Параллельные алгоритмы

Прежде всего, следует прояснить следующие вопросы, связанные с постановками практических задачи:

A) Насколько изменится трудоёмкость жизненного цикла программы решения задачи с помощью параллельного алгоритма?
B) В какой мере при постановке задачи следует учитывать модель параллелизма?
C) Как обосновать и измерить выигрыш от разработки параллельного алгоритма?
D) Насколько изменяется постановка задачи при переходе к параллелизму? Что даёт парадигма параллельного программирования на уровне разработки параллельного алгоритма?
E) Какими средствами представляются разрабатываемые параллельные алгоритмы решения задачи на этапе, предшествующем разработке программы?

За полвека традиционного последовательного программирования отлажено колоссальное количество программ, аккумулированных в системы программирования и стандартные библиотеки. Изменение постановок задач, уже имеющих готовые отлаженные программные решения, ради учёта допустимого параллелизма чревато повторным программированием и, что гораздо более трудоёмко, повторной отладкой.

Основной аргумент за разработку параллельных алгоритмов – целесообразность учета естественного параллелизма на уровне постановки задачи, утрачиваемого при решении задачи посредством обычных алгоритмов. Число языков параллельного программирования, удобных для реализации параллельных алгоритмов год от года растёт, хотя и их применение решает не все проблемы организации параллельных вычислений. Так, отмечая простоту записи параллельной композиции, можно сложность её отладки даже для несложных программ оставить в тени [5].

Итак, параллельный алгоритм может быть реализован по частям на множестве различных устройств с последующим объединением полученных результатов и получением целевого результата. Возникают чисто практические вопросы:

1. Каким образом в определении алгоритма выделены части, выполняемые отдельными устройствами?
2. Обязана ли реализация алгоритма использовать в точности представленный в его определении набор устройств?
3. Можно ли последовательный алгоритм рассматривать как параллельный, исполняемый на одном устройстве?

Следующая обойма вопросов касается категории «время» и связана с проблемами синхронизации:

4. Могут ли части параллельного алгоритма обладать своим независимым или централизованным отсчетом времени?
5. Может ли синхронизация частей алгоритма противоречить его информационным связям и логике управления?
6. Можно ли синхронизацию частей алгоритма рассматривать как частный случай асинхронности?

Особые сложности параллелизма вызывают вопросы доступа к памяти:

7. Каким образом взаимодействующие части параллельного алгоритма обмениваются данными?

8. Могут ли части параллельного алгоритма изменять состояние общей памяти и памяти других частей?
9. Может ли часть параллельного алгоритма воспрепятствовать использованию своей памяти другими частями?

Кроме того, части алгоритма могут быть определены в разных моделях вычислений и над разными структурами данных. Оценка результата разработки параллельного алгоритма, кроме оценки сложности вычислений и объёма данных, осложнена целесообразностью оценивать выполнение разного рода трудно формализуемых критериев, часть которых, однако, поддаётся современным средствам верификации на моделях.

6. Системы программирования

При измерении производительности суперкомпьютеров и в экспериментах с распараллеливанием программ активно используются задачи научных расчётов, преимущественно реализующих алгоритмы векторной обработки данных. Практические задачи современного параллельного программирования обычно выглядят как приведение больших ранее отлаженных программ на самых живучих системах программирования для языков C или Fortran к форме, дающей выигрыш от распараллеливания с помощью штатных средств, включаемых в доступные системы программирования. В целом такая работа сводится к следующим видам работы:

1. Разметка участков программы, допускающих автоматическое распараллеливание.
2. Анализ участков и причин, препятствующих распараллеливанию программы.
3. Выбор участков программы, допускающих их техническое приведение к форме, пригодной для распараллеливания.
4. Изобретение рецептов полуручного преобразования текста программы с целью расширения возможностей распараллеливания.
5. Приведение текста программы к предельно распараллеливаемой форме.
6. Прогон распараллеленной версии программы для оценки выигрыша от параллелизма.
7. Частичное перепрограммирование и отладка фрагментов программы для исключения или смягчения эффектов, препятствующих достижению нужных характеристик производительности.
8. Установление частичной функциональной эквивалентности исходной программы и результирующей её версии.

Обычно компилятор поддерживает оптимизации, обеспечивающие устранение неиспользуемого кода, чистку циклов, слияние общих подвыражений, перенос участков повторяемости для обеспечения однородности распараллеливаемых ветвей, раскрутку или разбиение цикла, втягивание константных вычислений, уменьшение силы операций, удаление копий агрегатных конструкций и др.

Рассматривается зависимость ускорения вычислений от числа процессоров и объема общей и распределенной памяти. Выполняется систематическая замена рекурсии на циклы. Предпочитается однородное пространство процессоров, общая память, быстрые обмены, соседство, гарантирующие улучшение производительности систем для высокопроизводительных вычислений. Заметно влияние дисциплины работы с памятью на характеристики параллельных процессов. Используется защищенная и размазанная память. Различны решения, принятые в разных языках программирования, по работе с многоуровневой и разнородной памятью (доступ, побочный эффект, реплики, дубли и копии). Обработка транзакций становится одной из типовых семантик работы с памятью в языках программирования.

Особый круг проблем связан с навыками учёта особенностей многоуровневой памяти в многопроцессорных системах. Обычное последовательное программирование такие проблемы может не замечать, полагаясь на решения компилятора, располагающего статической

информацией о типах используемых данных и способного при необходимости выполнить оптимизирующие преобразования программы.

Следует отметить, что использование языков параллельного программирования в качестве языка представления исходной программы не гарантирует её приспособленность к удачному распараллеливанию. При анализе пригодности программы к распараллеливанию анализируются потенциальные зоны риска, требующие дополнительных испытаний и отладки [2].

7. Учебные языки и системы программирования

Пришло время изучать информатику и программирование, начиная с мира параллелизма. Теперь трудно не заметить, что выполнение «одинаковых» действий обладает разной длительностью, что длительность реагирования информационной системы может зависеть от текущей ситуации, что собственно выполнимость действий зависит от не всегда заранее известных условий, что системы можно и нужно настраивать, что ряд систем могут работать одновременно и влиять на работу друг друга. На практике видно, что существуют кэши, протоколы, верификаторы, резервное копирование, транзакции, «гонки» данных, «смертельное объятие» и многое другое. Известно об успехах любительской астрономии, перспективах GRID-технологий и «облачных» вычислений. Активизация, вербализация и формализация таких знаний образует основу для быстрого изучения средств и методов параллельного программирования, начало которому пытается дать экспериментальный курс «Парадигма параллельного программирования».

Интересно отметить появление новых архитектур, обладающих полным набором команд с условным исполнением. В этом процессе расширяется пространство решений сложных задач, модернизируются методы развития информационных систем (ИС) на основе компьютерных сетей и многопроцессорных комплексов. Полезно рассмотреть перспективы развития парадигм программирования, обусловленных изменением условий эксплуатации современных информационных систем, особенно связанных с повсеместным распространением сетевых технологий, меняющих критерии оценки качества программ и методы обеспечения надежности и производительности программирования. Две основные линии такого развития – разработка распределенных информационных систем (РИС) и компонентное программирование (КП).

Варьирование правил функционирования сетей допускает как асинхронную, так и синхронную организацию срабатывания действий, включая дозирование нагрузки и специализацию процессоров и распределение действий по потокам выполнения. Использование иерархических, многоуровневых, структурированных и расширяемых сетей обеспечивает моделирование практически любых, накопленных в языках программирования, техник программирования и представления структур данных.

Сколь ни сложен мир параллелизма, программистам предстоит его понять, освоить и создать его полноценную поддержку языками и системами программирования!

При оценке образовательного значения ПП появилась тенденция выделять функциональное, параллельное и императивно-процедурное программирования в качестве принципиальных ПП, а логическое и объектно-ориентированное рассматривать как дополнительные ПП, изучаемые в виде расширения принципиальных парадигм. [6] Не исключено, что такое выделение обусловлено зависимостью преподавания логического программирования и ООП от владения развивающейся областью знаний или приложений, усложняющего учебный процесс.

8. Заключение

Мультипарадигмальность долго живущих ЯП и тенденция XXI-го века по созданию новых мультипарадигмальных ЯП говорят о созревании единой ПП, объединяющей выверенные в практике механизмы программирования. Тем более обосновано формирование общей парадигмы параллельного программирования и создание учебных языков и систем программирования, поддерживающих раннее обучение программированию как параллельному

программированию, что и предсказал Кеннет Айверсон, автор языка APL [4]. Тридцать лет назад проект академика А.П. Ершова по обучению школьников информатике слегка обидел профессионалов, убеждённых, что программирование – это занятие для людей с высшим образованием. Возможно, что опережающему освоению параллелизма не легко будет найти признание. Но реализация такой идеи возможна в контексте современных дистанционных технологий и массового распространения мобильных устройств.

Литература

1. Воеводин В. В. Параллельные вычисления. / В. В. Воеводин, Вл. В. Воеводин. — СПб.: БХВ-Петербург, 2002. — 608 с.
2. Городняя Л.В. О проблеме автоматизации параллельного программирования // В сборнике Международной суперкомпьютерной конференции «Научный сервис в сети Интернет: многообразие суперкомпьютерных миров» — URL: http://agora.guru.ru/abrau2014 (Проверено 8 июня 2015).
3. Городняя Л.В. Парадигмы программирования. Часть 4. Параллельное программирование //Новосибирск. Препринт ИСИ СО РАН. — URL: http://www.iis.nsk.su/files/preprints/gorodnyaya_175.pdf (Проверено 8 июня 2015).
4. Магариу Н.А. Язык программирования АПЛ. – М.: Радио и связь. – 96 с.
5. Хоар Ч. Взаимодействующие последовательные процессы. / Ч. Хоар. М.: Мир, 1989. 264 с.
6. Peter Van Roy. The principal programming paradigms (2008). — URL: https://www.info.ucl.ac.be/~pvr/paradigmsDIAGRAMeng108.pdf (Проверено 8 июня 2015).