РАЗВИТИЕ МОДЕЛЕЙ ПАРАЛЛЕЛИЗМА В ЯЗЫКАХ ВЫСОКОГО УРОВНЯ

А.Г. Марчук, Л.В. Городняя

В докладе рассмотрены средства и методы представления программ организации параллельных процессов, исторически сложившиеся в современных языках программирования, и резервы уточнения семантики языков высокого уровня с целью повышения эффективности и надежности использования новых возможностей аппаратуры и информационных технологий.

Введение

Прогресс производства аппаратуры намного опережает развитие технологий программирования. Параллельные архитектуры позволяют принципиально повышать производительность программ решения многих задач. Методы автоматического распараллеливания программ способны обеспечить значительное ускорение вычислений, явно сводимых к комплексу независимых процессов обработки элементов векторов, но отступают перед программами более широкого класса. Перспектива выражения языковыми средствами параллелизма на уровне постановки новой задачи связана проблемами оптимизирующей верифицирующей компиляции, обеспечивающей аккуратный выбор эффективной и надежной схемы параллелизма. Кроме того трудоёмкость повторного программирования и отладки параллельных алгоритмов обуславливает рассмотрение готовых последовательных программ в качестве эталонов при оценке результатов параллельных вычислений, что сдерживает развитие языков программирования и методов их реализации.

Fortran

Средства представления параллельных вычислений доступны, начиная с первых языков высокого уровня. Языки Fortran II и Fortran IV [1] были достаточно универсальны для представления программ организации параллельных процессов. Механизм сопрограмм, допускающий многовходовые (Entry) процедуры, позволял представлять программы взаимодействующих процессов и декомпозировать программу на управляющую и вычисляющую части, выглядящие как независимые компоненты программы, но это не привело к практике параллельного программирования и постепенно сопрограммы превратились модули, обеспечивающие представление иерархии функций подобно иерархии классов в ООП. Синтаксически средства параллельного программирования выглядят в современных Fortran-программах как разметка текста ключевыми словами и вызовы библиотечных функций.

APL

Более удачной оказалась идея поэлементной обработки однородных структур данных в языке APL, особенно с появлением векторных архитектур Cray. В 1962 году был предложен интересный механизм реализации многомерных векторов, приспособленный к расширению и распараллеливанию обработки данных. Сложные данные представляются как пара из последовательности скаляров и паспорта, согласно которому эта последовательность структурируется. Такое решение позволяет любое определение функции над скалярами автоматически распространять на произвольные структуры данных из однотипных скаляров. И в настоящее время для большинства специализированных языков параллельного программирования типично, что сложные построения факторизуются с учетом особенностей структуры данных так, что выделяются несложные отображающие функции, "просачиваемые" по структуре данных с помощью функций более высокого порядка - функционалов. В результате можно независимо варьировать структуры данных, функционалы, методы сборки полного результата и набор отображаемых множеств. Целенаправленно выделяются конвейерные процессы, приспособленные к минимизации хранения промежуточных результатов. С 1980-ых годов эта идея наследуется многими языками, поддерживающими параллелизм и в наше время фактически является стандартной.

Algol-68

К концу 1960-х годов сложилось значительное разнообразие теоретических моделей параллелизма, при исследовании которых проявилась проблема надежности параллельных вычислений, выразившаяся в неожиданном различии поведения последовательности действий в зависимости от включаемых в нее фрагментов «независимых» процессов. Для профилактики таких эффектов в семантику языка Algol-68 включается идея непрерывно исполняемых критических участков и представления их защиты в терминах семафоров.

Setl

Другой подход к надежности программирования предложен в языке теоретико-множественного программирования Setl, ориентированном на активизацию интуиции грамотных математиков при разработке спецификаций программ в терминах преобразования множеств, естественно подразумевающих возможность

параллельной обработки элементов множества, причем в реализационно независимом стиле. Наследование решений из универсальных языков сверх высокого уровня, таких как Setl, абстрагирование данных и процессов в которых приспособлено к гибкому и строгому структурированию, удобно для культивирования доказательных построений в практике параллельного программирования. В этом плане представляет особый интерес эксперимент по развитию теоретико-множественной семантики языка Setl, в котором весьма общее построение формул с кванторами над множествами погружено в обычную схему последовательного управления процессами. Реализация языка Setl характеризуется богатым полиморфизмом. Для представления множеств используется около двадцати разных структур данных, выбор которых осуществляется системой программирования в зависимости от динамики операций над множествами. В результате программируемые функции слабо зависят от реализационной структуры данных. В практике управления процессами используется понимание команд как позиций независимого порождения процессов. Такое понимание естественно согласуется с идеями теории множеств о независимости элементов множеств и может служить основой архитектуро независимой семантики языка программирования.

Транзакционная память

Независимо идеи явного порождения процессов и организации их взаимодействия через каналы возникают в языках управления заданиями (JCL) и процессами в операционных системах (Unix). Много более заметная проблема параллельной обработки данных независимыми операторами обнаружилась в практике применения общих баз данных, приведшей к выделению языка запросов (SQL) и концепции транзакционной памяти, теперь рассматриваемой как перспективная основа семантики языков параллельного программирования.

Occam

Середина 1970-х годов характеризуется кризисом технологии программирования, выход из которого тогда виделся в массовом переходе к параллельному программированию. Активные исследования разрешимых классов параллельных схем программ показали ряд неудобных, снижающих эффективность распараллеливания, конструкций, таких как ветвления. Э.Дейкстра опубликовал решение этой проблемы в форме защищенных команд [2], которая нашла свое место в определении языка Occam,предоставляющем для транспьютерного программирования модель взаимодействия CSP процесс-канал [3]. В эти же годы популяризируются идеи структурного программирования, нацеленные на снижение сложности отладки программ, близкие идеям функционального программирования, которое теперь рассматривается как один из универсальных методов представления удобно распараллеливаемых программ.

Ada

В проект языка Ada предпочли включить механизм «рандеву», сводящий представление взаимодействия процессов к рассредоточенному обмену сообщениями, подобному сигналам в оборудовании и модели CCS, что можно рассматривать как приаппаратное низкоуровневое средство, несколько диссонирующее с высоким уровнем языка.

Sisal

Произошедшее в конце 1970-х годов отвлечение внимания от кризиса технологии программирования на задачу освоения микропроцессоров не остановило поиск языковых решений для представления программ, обладающих параллелизмом. Появился функциональный язык параллельного программирования Sisal, позволяющий формировать пространства итераций для эффективного распараллеливания циклов компилятором.Название языка функционального программирования Sisal расшифровывается как “Streams and Iterations in a Single Assignment Language”. Система вычислений в языке Sisal использует понятие «мультизначение», позволяющее подобно языку APL распространять скалярные действия на данные любой структуры, а их обработку осуществлять на многопроцессорных конфигурациях. Отображение мультизначения рассматривается как обработка его элементов на независимых процессорах. Результаты отображения могут повергнуться свертке или фильтрации. Sisal-программа представляет собой набор функций, допускающих частичное применение, т.е. вычисление при неполном наборе аргументов. В таком случае по исходному определению функции строятся его проекции, зависящие от остальных аргументов, что позволяет оперативно использовать эффекты смешанных вычислений и определять специальные оптимизации программ, связанные с разнообразием используемых конструкций и реализационных вариантов параллельных вычислений. Основное продвижение по технике программирования в языке Sisal – развитие структуры циклов для их реализации на параллельных процессорах. Введено понятие «пространство итераций» и предложена специальная конструкция для фильтрации мультизначений, получаемых при совмещенном исполнении итераций и участков повторяемости. Формирование мультизначения управляется представлением пространства итераций и учетом зависимостей между одноуровневыми итерациями. Работа с именованной памятью (Name-oriented) освобождена от проблемы побочных эффектов методом локализации участков с однократными

присваиваниями – SSA-форм, что делает программы удобными для преобразований, оптимизации и компиляции, включая распараллеливание и масштабирование.

БАРС

В нашей стране разработаны языки БАРС и Поляр с разными концепциями сетевого управления процессами и представления дисциплины доступа к памяти. Программирование на уникальном по уровню средств управления процессами языке БАРС нацелено на обеспечение высокопроизводительных вычислений и организацию асинхронных параллельных процессов. При создании языка БАРС в качестве базового ЯВУ был привлечен популярный язык Pascal, в 1970-е годы перераставший из учебного в производственный язык системного программирования. При сохранении основных принципов семантики вычислений были существенно обобщены средства структуризации данных на основе понятия «мультимножество», приспособленного к именованию элементов структур данных и учета кратности их использования. Работа с именованной памятью (Name-oriented) дополнена возможностью задавать дисциплину доступа к элементам памяти. Идеи более ранних языков параллельного программирования были развиты и обогащены в языке БАРС в трех направлениях:

1. в качестве базовой структуры данных были выбраны мультимножества (размеченные множества с кратностью элементов);
2. описание элементов памяти сопровождается предписанием дисциплины доступа к памяти;
3. средства управления асинхронными процессами включали механизм сетей Петри, координирующих работу независимо созданных функциональных фрагментов.

Процедуры в таком языке приспособлены к варьированию дисциплины доступа к данным и схемы управления процессами обработки данных. Сети Петри позволяют независимые описания процессов связывать в терминах разметки. Узлы с одинаковой разметкой срабатывают одновременно. 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. Вычисления, как и в языках APL и Sisal, распространяются со скаляров на сложные структуры. Радикальное продвижение в повышении уровня программирования, предложенное в языке БАРС, заключается в переносе механизма типизации данных на проблему типизации схем управления.

Oberon

1980-е годы знаменует переход к сетевой обработке данных и признанию потенциала ООП при организации информационных бизнес-процессов. Появляются Oberon, Eiffel, SmallTalk-80, C++, Erlang, Perl и другие языки, отчасти компенсирующие недостаток базовых средств и методов реализации массово используемых императивных языков программирования в новых условиях. [4]

Python и Ruby

Общий прогресс в эксплуатационных характеристиках оборудования с 1990-х годов резко расширил возможности сборки информационных систем из готовых компонентов и сделал доступными свободно распространяемые программные инструменты конструирования систем программирования, как правило, поддерживающие организацию параллельных процессов, если не собственно на уровне языка, то на уровне библиотечных компонент. Мультипарадигматические языки Python и Ruby показывают хорошие результаты в программировании сетевых процессов для многопроцессорных комплексов и привлекают большое число сторонников. [5]

Haskell

Кроме того существуют сотни функциональных языков программирования, ориентированных на разные классы задач параллельного программирования. Языки функционального программирования обогатились типовыми средствами практически всех известных подходов к представлению программ и организации вычислительного эксперимента и информационных процессов. Обеспечена организация параллельных процессов. Возможна визуализация данных и программ. Имеются средства стандартного и объектно-ориентированного программирования. Поддержано управление компиляцией и конструирование компиляторов. Методы функционального проектирования и программирования обеспечивают технику представления и отладки функциональных моделей, спецификации и верификации программ, исследования их свойств и экспериментального сравнения моделируемых параллельных процессов с моделями и прототипами. Функциональный подход исторически является основой для исследования средств и методов программирования, прототипирования и декомпозиции программируемых систем и развития современных методов параллельного и многоязыкового программирования. Разработан чисто функциональный язык Haskell, предлагающий эффективную модель «ленивых» вычислений с мемоизацией промежеточных результатов по принципу полузабытых методов «математического динамического программирования».

MPI и Open MP

Появляются популярные системы MPI и Open MP, обеспечивающие эффективность параллельного программирования в рамках языков Fortran и C. В MPI взаимодействие процессов обеспечивается через посылку сообщений между процессорами. Open MP предоставляет процессам возможность использовать разные виды памяти, включая быструю общую память, что при удачном ее распределении позволяет достигать высокой эффективности.

mpC

Практически в мире параллелизма все базовые понятия программирования претерпели изменение или расширение (программа, ветвление, цикл, событие, память, результат). Появился ряд специфических для параллельного программирования понятий (процессор, поток, ожидание, длительность, фильтр, барьер). Программы стали многопоточными, циклы – параллельными, память обретает копии и реплики, события происходят одновременно в разных синхронизуемых процессах, вычисление результата может не означать завершение процесса. Такая ревизия понятий влияет не только на стиль программирования, но и изменяет характер компиляции на этапе генерации кода программы. Возрастает роль техники использования многократно используемых компонент схемного уровня, соответствующего средствам типизации управления процессами. Так например, язык mpC предлагает более детальный учет механизмов взаимодействия параллельных процессов в терминах барьеров и специальных категорий переменных, обладающих особым, аппаратно реализуемым поведением.

Java, C#, Scala, F#

В новых системах программирования для языков Java, C#, Scala, F# и т.д. существенно повысилась результативность системных решений в области работы с памятью, компиляции, манипулирования комплектами функций и классами объектов, выделение которых по существу обусловлено результатами теоретических работ в области системного статического анализа, основу которых все в большей мере составляет функциональный подход. Получили значительное развитие методы декомпозиции программ в рамках объектно, субъектно и аспектно ориентированных подходов к определению систем программирования на базе многократно используемых компонент. Для современных областей программирования и проектирования характерна интеграция средств и методов из разных парадигм, что может привести к профессиональной консолидации программистского корпуса.

Promela и Spin

Интересно отметить отсутствие связи практики параллельного программирования с результатами исследований в области верифицирующей компиляции и типизации схем управления параллельными процессами, отражающих специфику требований, предъявляемых к высокопроизводительным программам. Между тем, методы и техника верификации выходят на уровень практичного применения: известны прецеденты успешного применения таких систем как Promela, Spin и др. в области проверки протоколов взаимодействия процессов. Основная трудность – практика ручного перепрограммирования параллельных алгоритмов. Вывод – еще не сложилась общая парадигма параллельного программирования, увязывающая в единую картину средства и методы создания параллельных программ и их усовершенствования по мере технического прогресса.

CUDA

Появление технологии CUDA, объединившей в графических ускорителях достоинства этих ранее сложившихся подходов, выводит параллельное программирование в ранг массово доступных методов создания программных систем благодаря преодолению стоимостного барьера. Следует ожидать, что развитие парадигмы параллельного программирования приведет к улучшению средств поддержки полного жизненного цикла программ, включая активное использование методов верификации взаимодействия процессов, автоматизацию приведения обычных программ к эффективно распараллеливаемой форме, обеспечение мобильности параллельных программ относительно параллельных архитектур, а следовательно и к повышению эффективности и надежности программ, приспособленных к многократному использованию типизированных решений особо важных наукоемких задач. На повестке дня – разработка методов архитектурно независимой кодогенерации масштабируемых параллельных программ, легко настраиваемых на особенности используемых вычислительных комплексов. Так, например, новый язык программирования OpenCL обеспечивает параллелизм на уровне инструкций и на уровне данных для различных графических и центральных процессоров. [6]

Заключение

Таким образом, в настоящее время осознана актуальность формирования парадигм параллельного программирования, вызванная расширением и развитием системы базовых понятий, необходимых для рациональной разработки систем управления процессами на современной аппаратуре. Парадигмы в программировании характеризуются стилем мышления при решении задачи, системой используемых понятий и

особенностями их реализации. Можно констатировать, что стиль мышления и система понятий частично опробованы в процессе эволюции языков программирования, но практика их эффективно масштабируемой реализации и применения ещё не сложилась.

ЛИТЕРАТУРА:

1. Сайты с материалами по языку Fortran. http://www.fh-jena.de/~kleine/history/languages/ansi-x3dot9-1966-Fortran66.pdf, http://wwwcdf.pd.infn.it/localdoc/f77_sun.pdf, http://archive.computerhistory.org/resources/text/Fortran/102653989.05.01.acc.pdf, http://www.fh-jena.de/~kleine/history/languages/GC28-6515-10-FORTRAN-IV-Language.pdf
2. Сайт с описание защищенных команд Э.Дейкстры. http://www.cs.virginia.edu/~weimer/615/reading/DijkstraGC.pdf
3. Описание языка Occam. http://www.wotug.org/occam/documentation/oc3refman.pdf
4. Описание языка Oberon http://www.inf.ethz.ch/personal/wirth/Articles/Oberon/Oberon.Report.pdf
5. Описание языка Ruby/ http://www.ipa.go.jp/osc/english/ruby/Ruby_final_draft_enu_20100825.pdf
6. Сведения об открытом языке OpenCL. http://en.wikipedia.org/wiki/OpenCL