АВТОМАТНАЯ ПАРАДИГМА ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ

В.С. Любченко

1. О парадигме параллелизма

На текущий момент наблюдается "бум" параллельного программирования. На аппаратном уровне идея проста, - есть одно ядро, то почему бы не быть нескольким? На программном уровне подобные мысли роились уже давно. С тех времен, когда, рассуждая, если уж есть один процесс, решили, почему бы не быть их множеству? Так или почти так появилось многопоточное программирование.

Сейчас программный параллелизм удобно устраивается на аппаратном уровне. Особенно ему комфортно, когда количество [активных] потоков равно числу [доступных] ядер. Однако, разрабатывая принципы параллелизма, как-то забыли не то, чтобы о науке, но - о реальном мире и его устройстве. А в нем есть многое, что предстоит еще познать и создать аналог. И если у нас начало "бума", то в природе уж давно царит параллелизм и иерархия. Уж тут, действительно, кто-то когда-то уже все создал...

Посмотрим на человека, как на идеал процессора/ядра. Его нейроны полный аналог того, к чему мы стремимся. Природа уже создала "многоядерную систему", в которой ядер гораздо больше, чем можно даже представить. А потому, может, логичнее присмотреться к устройству природных систем, чем создавать новые принципы и механизмы, осваивая, как кому-то представляется, чистое поле?

Но дело даже не в числе ядер. Как можно предположить, - в их качестве, т.е. в законах, которым подчинен "природный параллелизм". Их мы пока просто не учитываем. Да и, если уж говорить прямо, не знаем. А когда сталкиваемся с противоречиями, то думаем, что проблемы в количестве, в скорости работы ядер, в ошибках модели и т.д. и т.п. Короче во всем том, что обобщенно называется парадигмой вычислений [1].

Казалось бы, проблемы параллелизма должны подводить к мысли, что мы делаем что-то не так. Например, явно игнорируется достаточно простой принцип: если нет понимания законов, то их можно постичь, копируя и анализируя поведение аналогов. Только при достижении определенного, а в идеале - полного, соответствия [аналогу], можно говорить о том или ином понимании. В том же известном тесте Тьюринга искусственный интеллект признается таковым, если внешний наблюдатель, задавая вопросы, не может отличить, кто ему отвечает.

Если сравнивать современные параллельные системы с их аналогами, то мы явно идеализируем ситуацию, восхищаясь многоядерными процессорами и многопоточным программированием. Любой "природный аналог" даст сто очков вперед самому мощному суперкомпьютеру. Хотя бы по числу элементов и количеству связей между ними. Но - опять же! Собрать огромный суперкомпьютер - не проблема! Проблема - в законах работы составляющих его элементов!

2. Автоматный параллелизм

Ни что не создается на пустом месте. Определенную предысторию имеет и параллельное программирование. Например, известно и востребовано нейронное программирование. Другой пример - клеточные автоматы. Кстати, нейрон по определению простейший автомат. Только клеточные автоматы, обладая большими "мыслительными возможностями", ограничены топологией связей (каждая "клетка" общается в основном только со своими соседями).

Таким образом, есть нечто, что можно, сняв те или иные ограничения, положить в основу параллельных систем. Например, известны попытки аппаратной реализации клеточных автоматов в виде машин клеточных автоматов. Только последние сохранили ограниченность исходной модели, а потому их применение не стало таким уж широким. Хотя уже только попытка аппаратной реализации на какое-то обобщение рассчитывает.

Начала теории автоматов - а именно об автоматной модели пойдет далее речь - заложены работами упомянутого Тьюринга и, наверное, еще более известного фон Неймана. Ключевыми, по крайней мере, у нас, можно считать работы академика В.М.Глушкова (особенно по синтезу цифровых автоматов). Очень интересны, но почему-то мало принимаемы во внимание, работы А.Н.Мелихова по алгебре автоматов. Просто великолепны книги Д.А.Поспелова и В.И.Варшавского и т.д. и т.п.

Можно видеть, что серьезная основа для создания автоматных систем универсального типа есть. Однако есть вопросы, требующие явного решения. Например, основные результаты в алгебре автоматов получены в основном для абстрактных автоматов, хотя с практической точки зрения удобнее оперировать автоматами структурными.

Тем не менее, уже сейчас нет проблем представить параллельную систему, как коллектив автоматов (кстати, тут можно вспомнить множество работ по коллективам автоматам, по автоматам с перестраиваемой структурой и т.д. и т.п.). Таким образом, даже с учетом имеющихся пробелов в теории [алгебре структурных автоматов], ни что не мешает автоматные принципы реализовать, если не на аппаратном, то на программном уровне.

В свете сравнения вычислительных моделей нужно хотя бы упомянуть достаточно известную модель сетей Петри. Но они в чистом виде рассчитаны на моделирование. Кроме того, в практических целях их чаще всего модифицируют, а это порождает множество интерпретаций и разновидностей сетей, что вносит путаницу. В то же время существует доказательство эквивалентности сетей Петри автоматам. Или, другими словами, имея автоматную модель, проще реализовать на ее базе сеть Петри, чем доводить дело до обратного.

3. О реализации автоматной парадигмы параллелизма

Теорию автоматов нужно совершенствовать, но, как уже было отмечено, нет препятствий, чтобы по-новому работать даже с тем, что есть. Остановимся на существующих аппаратных и программных возможностях, которые позволяют эффективно реализовать автоматную модель параллельных вычислений.

3.1. Аппаратная часть.

О создании автоматной архитектуры пока можно только мечтать. Тут не на словах, а на деле должны произойти изменения в мышлении архитекторов современных процессоров. А это во многом процесс времени. Пока, если кратко, на аппаратном уровне идет активный и постоянный процесс "выжимания соков" из того, что есть.

Тем не менее, аппаратная часть универсальна и может быть базой для любой здравой идеи. Важно только, чтобы последняя не просто работала, а работала эффективно. Автоматная модель в этом смысле идеальна, т.к. ее реализация в сравнении с другими моделями может быть весьма эффективной. А если учесть, что программная реализация считается минимум на несколько порядков медленнее своего аппаратного аналога, то это резерв, позволяющий увеличить быстродействие вычислений не путем борьбы за "герцы", а переходом на иные архитектурные модели.

Итак, аппаратную часть - не трогаем. Просто потому, что изменить ее нет никакой возможности. Пока, конечно.

3.2. Программирование.

Языкам программирования несть числа, а с активным продвижением параллельного программирования они продолжают множиться даже активнее. Одна только беда - в рамках текущей модели вычислений. А потому, меняя, они, как это ни парадоксально, ничего не меняют.

Однако, среди множества языков своими возможностями, гибкостью, эффективностью и т.д. и т.п. выделяется язык C++. Он проверен временем. На нем можно реализовать любую идею, чтобы убедиться в ее абсурдности или, наоборот, полезности. Правда, и на его стандарт покушаются, вводя "параллельные фишки". Остается надеяться, что это ему не повредит.

Создав на C++ некую виртуальную "автоматную машину", можно быть уверенным, что на другом языке (ассемблеры оставляем в стороне) она не будет эффективнее. Кроме того, подобную реализацию пусть и с некоторыми проблемами можно перенести на множество аппаратных платформ и операционных сред, поддерживающих C++.

В языке C++ нет параллелизма. Но в нем есть объекты, в которые легко внедрить автоматы, наделяя их отсутствующим поведением. Так в высокоуровневый язык вводится не только новая модель вычисления, но совершенствуется то хорошее, что в нем есть. При этом каждый автомат отождествляется с объектом, модель поведения которого удобно представлять в форме автоматной модели - автоматного поведения.

Если есть модель параллелизма, то заставить параллельно работать объекты в C++ уже дело техники. При этом в связке процесс-множество процессов важна не столько модель отдельного процесса, сколько сама модель множества процессов. Если первую - автоматы мы можем реализовать на уровне почти любого языка, то вторую - модель множества автоматов - пока только на уровне так называемого микро-ядра или виртуальной машины. А для этого необходим язык, сравнимый по гибкости и мощности с C++.

Автоматы повышают уровень абстракции языков высокого уровня, наделяя их качеством непроцедурности. Известный пример подобного непроцедурного программирования - язык таблиц решений. Заметим, что таблица решений на формальном уровне эквивалентна всего лишь одному автомату с одним состоянием.

3.3. Технология программирования

Если раньше понятие программной технологии больше связывали с самим процессом программирования - модульное программирование, структурное, объектно-ориентированное и т.п., то сейчас часто с его организацией - Agile и т.п. Автоматы возвращают исконный смысл понятию программная технология. Причем, пожалуй, тут и изобретать экстраординарно нового не надо. В рамках автоматной парадигмы процесс параллельного программирования во многом может походить на технологию проектирования цифровых схем, где автоматы используются уже давно.

На базе автоматов можно создать универсальную технологию, когда на этапе реализации можно решать, как воплощать систему - программно и/или аппаратно. Так, естественным образом используется ценный опыт, накопленный в теории и практике другой области деятельности, нежели программирование - проектировании цифровых схем (в том числе тех же процессоров).

Но, закладывая основы технологии, и в самом программировании есть, что заимствовать. Хорошие тому примеры - программные среды визуального программирования: VisSim, LabVIEW, CodeSys, HiAsm и т.п. Они весьма технологичны, т.к. просто и наглядно позволяют строить сложные объекты из набора "кубиков". А

если нужен нестандартный "кубик", то для его создания в визуальной среде есть, как правило, встроенный язык программирования.

Но современные визуальные среды, во-первых, специализированы, во-вторых, налицо явная ограниченность их встроенных языков программирования. Но если языки подтягиваются, то их параллелизм и параллелизм самих сред находится все еще в зачаточном состоянии. Отсюда множество проблем, с которыми приходится сталкиваться, когда что-то сделать нельзя и поправить невозможно. Легко привести примеры простейших систем, которые в такой ситуации или не реализуются или реализуются с большими проблемами (например, системы с циклическими связями). Особенно опасно, что подобные проблемы носят, как правило, неявный характер, а потому могут проявить себя самым неожиданным образом.

Таким образом, оперируя автоматной моделью параллельных процессов, представляя "автоматный кубик" подобно кубику визуального программирования, мы можем "скрестить" мощный универсальный язык программирования типа C++ с весьма привлекательными технологиями визуального программирования. Так, образно говоря, мы убиваем двух зайцев: C++ получает то, что у него есть весьма в условной форме (например, типа визуальной среды Visual Studio), а визуальное программирование - язык с возможностями, о котором можно лишь мечтать. Или, даже трех, если вспомнить о внедрении в высокоуровневый язык и в среду программирования параллелизма на базе параллельной автоматной модели вычислений.

Кроме теории, автоматы дают программистам то, о чем они, используя другие модели, могут даже не догадываться. Например, встроенную наблюдаемость за состояниями параллельных процессов (состояниями автоматов). А оперируя автоматным дискретным временем, можно реализовывать различные режимы работы. Среди них - пошаговый. Легко регулируется и индивидуальная скорость работы процессов путем изменения значения дискретного такта. При этом более медленным и/или более энергоемким процессам (например, процессы, выполняющие отображение информации) логично назначать больший дискретный такт, а процессам более ответственным (например, процессы, отвечающие за съем информации с объекта) явно необходима минимальная величина дискретного такта и т.д. и т.п.

4. Технология визуально-кубического (автоматного) программирования

Автоматная парадигма, представленная выше в общих чертах, во многом уже реальность, сформированная в технологию программирования, названную Визуально-Кубическим Программированием (автоматным). Сокращенно - технологию ВКП(а). Ей присуще:

1) Универсальность в силу универсальности базовой модели вычислений - конечного автомата.
2) Используется более высокоуровневая параллельная алгоритмическая модель. Это автоматы и множества автоматов (другое известное название подобной модели - коллективы автоматов), объединенных в сеть. Можно также оперировать множеством подобных сетей;
3) Автоматное программирование может добавить языку, например C++, свой "плюс", когда классы/объекты последнего расширяются до объектов с активным поведением (другое их название - агенты, "живые" объекты и т.п.). В существующей реализации разработан простой язык их описания, интерпретирующий текстовую форму представления таблиц переходов автоматов.
4) Автоматы служат основой модели "параллельного мира", где каждый "индивид" - автомат, а "коллектив" - автоматная сеть. Образно говоря, возможна организация различных параллельных миров без ограничения на вид связей между ними. Необходимо лишь учитывать, что алгебра автоматов будет выполнима только для автоматов, составляющих тот или иной "автоматный мир".
5) В ВКП(а) учтен важный аспект, связанный с моделью памяти. Без корректной модели, даже при наличии теоретически безупречной модели управления процессами, ошибок избежать невозможно.
6) Автоматы дают преимущества в разработке и отладке, которых нет у других моделей. Динамический контроль (протоколирование) возможен вплоть до выяснения сработавшего перехода/переходов.
7) Наличие явного дискретного времени у автоматной модели позволяет легко управлять динамикой работы программной системы: от назначения конкретного реального времени ее дискретному такту, до пошагового режима работы step-by-step.
8) Ясность, документируемость, хорошая сопровождаемость автоматных моделей легко дополняется огромным технологическим и организационным опытом из области проектирования цифровых автоматов, который почти без изъятий и самым естественным образом переносится на область проектирования программного обеспечения.
9) Программная реализация автоматов эффективна. На не самой "крутой" элементной базе в режиме виртуализации на ядре, написанном даже не на ассемблере (хотя и на C++), можно легко говорить о гарантии жесткого реального времени в районе одной миллисекунды и менее (занимающиеся системами реального времени, оценят данное качество).
10) Перспектива реализации автоматной модели на аппаратном уровне открывает новые границы для развития процессорных архитектур. Речь идет об увеличении их быстродействия на порядки. Поскольку мы достигли граничной частоты работы процессоров, то это представляется весьма важным и актуальным для будущих автоматных архитектур.
11) Но самое важное - у разработчика формируется новое мышление, при котором находят свое практическое применение теория множеств, теория автоматов, теория автоматического управления и

т.п., о которых современные программисты в своем большинстве имеют в лучшем случае достаточно общее представление.

Мы не останавливались подробно на использования автоматной модели, языка ее программирования и других эффектов использования в реальной работе. Не рассматривали детали применения автоматов на уровне проектирования отдельной программы, технологических приемов разработки, отладки, документирования и сопровождения параллельной системы в целом. Все это облегчает жизнь разработчику и влияет на его мышление, позволяя реализовать идеи, которые сложно, а, порой, и невозможно реализовать в рамках других парадигм.

Примером реального приложения технологии ВКП(а) может служить разработка системы управления представленная в [2]. Ее характеристики - по количеству параллельных процессов управления и необходимой для них величины дискретного времени такта (см. выше п.9), значения "зернистости" процессов (количество команд на такт дискретности), механизмы синхронизации процессов и т.д. и т.п. трудно достижимы в рамках обычной концепции многопоточного/многоядерного программирования.

Автоматная парадигма вычисления и соответственно технология ВКП(а) универсальны в своем приложении. Эффективность программной и аппаратной реализации автоматной модели, ее возможности по структурной организации и распараллеливанию вычислений особенно привлекательны для сферы суперкомпьютерных вычислений, где те же модели клеточных автоматов, нейронные сети и т.п. "автоматные формализмы" уже имеют свою историю применения[3].

Для понимания принципов параллельного автоматного программирования достаточно знания базовых основ теории конечных автоматов и теории множеств. На начальном уровне достаточно даже интуитивного представления об автоматной модели. Здесь нужно даже не столько подтягивать теорию, сколько привыкнуть к новому мышлению, базирующемуся на другой модели вычислений. Безусловно, процесс перехода от одной парадигмы программирования к другой мучительный. Тот, кто переходил с обычного программирования на объектно-ориентированное, испытал, как правило, подобное. Но другого пути нет. Более того, даже не предвидится...

ЛИТЕРАТУРА:

1. Н.В. Шилов, Л.В.Городня, Е.В.Бодин Парадигма параллельного программирования: учить или не учить (вот в чем вопрос). // Труды Международной суперкомпьютерной конференции "Научный сервис в сети Интернет: экзафлопсное будущее. - М.: Изд-во МГУ, 2011. - С.193-197 (доступна на http://agora.guru.ru/abrau2011/pdf/193.pdf)
2. Любченко В.С., Ломакин Р.Л., Перфилов С.А. Об опыте создания параллельной системы управления синтезом искусственных минералов на базе конечно-автоматной модели . Труды и пленарные доклады участников конференции УКИ'12 / Научное издание. Электрон. текстовые дан. - М.:ИПУ РАН, 2012 - 1 электрон. опт. диск (CD-ROM) - ISBN 978-5-91450-100-3 - С. 001354-001359
3. Бандман О.Л. Клеточно-нейронные модели пространственно-временной динамики. // Программирование. 1999, № 1, с. 4-17.