УДК 519.712.1

К ПРОБЛЕМЕ СОЗДАНИЯ МОДЕЛИ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ

В.С. Любченко
ООО «Креол ЛТД"
Россия, 601650, г. Александров, Свердлова ул., 2
E-mail: sllubch@mail.ru

Ключевые слова: технология программирования, параллельное программирование, автоматное программирование, конечный автомат, дискретное время, сети Петри, SWITCH-технология, моделирование систем, UML
Key words: software engineering, parallel programming, automaton-oriented programming, finite state automaton, discrete time, Petri nets, SWITCH-technology, UML

В статье рассмотрены вопросы создания формальной модели и технологии проектирования параллельных систем на базе модели конечного автомата. Дано сравнение с другими моделями и программными технологиями.
TO THE PROBLEM OF CREATION OF MODEL OF PARALLEL CALCULATIONS / V.S. Lyubchenko (OOO «Kreol Ltd», 2 Sverdlova, Alexsandrov 601650, Russia, E-mail: sllubch@mail.ru). In the paper, there are considered issues of creation of formal a model and technologies of designing of parallel systems on the basis of the finite state automaton model. A comparison with other models and program technologies is given.

1. Введение

Одна из основных проблем параллельных вычислений это создание формальной универсальной модели. Под последней понимается некая идеализированная машина, работающая в рамках *концепции неограниченного параллелизма*, «на которой можно было бы реализовать для конкретного алгоритма *любой режим*, как параллельный, так и последовательный» (далее, если нет явной ссылки, цитаты и термины взяты из [1]). Пока такой модели нет. Из-за этой проблемы, а также по ряду других причин, сейчас более чем когда-либо возникла необходимость «критически осмыслить научный фундамент параллельных вычислений». Это будет нашим ориентиром, когда мы поведем разговор о проблемах создания модели параллельных процессов вообще и их физических аспектах (физики параллелизма) в частности.

Новая формальная модель это, как правило, и новая архитектура вычислителя. Первоначально она может быть представлена виртуальной машиной. Но когда ее возможности достигают нужной степени универсализма, допускают эффективную аппаратную реализацию, а опыт ее применения достаточен, то следующий этап развития модели – аппаратная реализация.

Новая формальная модель это чаще всего и новая технология программирования. Параллельная технология это не только и не столько более высокие скорости, но еще и возможность создавать более простые и ясные решения, с которыми в силу их сложности не справляется последовательный подход. Все дело в

том, что даже очень сложная по логике работы система может быть представлена суммой весьма простых и понятных параллельных компонентов.

Все большее число не выявленных ошибок, все меньшая надежность программ настоятельно требуют поиска новых подходов к проектированию программного обеспечения. Параллельный подход во многом решает эту весьма острую по нынешним временам проблему. И это опять же потому, что отдельный компонент параллельной системы более прост, а потому и более надежен, чем спроектированная система в целом.

2. Физические аспекты параллелизма

Решить проблему легче, если удается формализовать ее. Занимаясь абстрактными вычислениями, мы создаем формальный мир, который материализует реальная машина/процессор. В последовательных вычислениях все, что связано с этой темой, проработано в деталях. Но можно ли для параллельных вычислений создать абстрактную машину, которая по своей роли и функциям была бы аналогична последовательным машинам Тьюринга, Поста и аналогичным машинам?

Ответ на этот вопрос положителен, хотя бы по той простой причине, что некий кибернетический «черный ящик» не обязан быть только последовательным. Подробное и строго объяснение этого – отдельная тема, но все дальнейшее, по существу, и есть ответ на заданный вопрос, но только в общей форме. Кроме того, большой интерес к абстрактной параллельной машине вызван и тем, что ее создание означает переход от стихийного развития к развитию, базирующемуся на строгой математической основе [1].

В чем особенность предлагаемого «критического взгляда» на параллельные вычисления? В том, что, создавая вычислительную модель, необходимо, как в реальной физике, выявлять аналогии с реальными параллельными процессами. Существующий же взгляд фактически абстрагирован от подобных процессов. С ним можно познакомиться по высказываниям М.Минского (см. главу «Физические машины и их абстрактные модели» в книге «Вычисления и автоматы» [2]). Но, может, именно такой, слишком отвлеченный от реалий подход не позволяет сделать качественный рывок в развитии параллельных вычислений, где наблюдается определенный застой. Здесь большинство существующих и перспективных решений представляют собой по сути «кальки» с давно известных результатов из области «больших» компьютеров.

Различия между миром абстрактным и миром реальным можно выявить, сравнив абстрактную модель некоторого процесса/процессов со структурой и поведением соответствующего реального физического объекта. И если реальный объект состоит из какого-то числа параллельно работающих частей/процессов и определенного числа связей между ними, то подобную структурную организацию должна иметь и его модель. Также важно, чтобы результаты, полученные для параллельной модели, могли распространяться и на последовательную модель. При этом последняя, с одной стороны, может рассматриваться лишь как частный случай более общей параллельной модели, а, с другой стороны, она может быть обобщением параллельной модели, выступая в качестве ее эквивалентной модели.

Есть задачи, которые могут наглядно показать недочеты формальной модели в описании/моделировании поведения реальных объектов. Одина из них -

моделирование RS-триггера [3, 4]. С одной стороны, имеются явные пробелы в описании RS-триггера на формальном уровне, с другой стороны, мало кто понимает, как работает этот, казалось бы, простой со структурной точки зрения объект (два процесса и две связи между ними). А все потому, что, несмотря на структурную простоту, это хитрая параллельная система, имеющая перекрестные информационные связи между процессами. Все подобные противоречия во взглядах на триггер наглядно отражают материалы по RS-триггеру и дебаты по его поводу его функционирования на сайте SoftCraft [4].

Итак, если мы хотим создать модель вычислений, отвечающую физике параллелизма, то она должна порождать абстрактные модели, адекватные реальным объектам не только по результатам работы, но и по своему поведению, и по структурному построению.

2.1. О времени процессов

Как ни странно, но большинство вычислительных моделей не учитывают время. А это, ведь, давно известное и важное свойство реального мира. Пока же о времени мы вспоминаем лишь тогда, когда возникает задержка с получением результата. Решение данной проблемы в последовательных вычислениях часто сводится к выбору более быстродействующего процессора. Из этого можно сделать вывод, что отдельному процессу время его работы не важно. Оно больше интересует тех, кто ждет результаты.

Совсем иная ситуация в случае параллельных процессов. На них влияние времени имеет качественно иную природу. Поскольку параллельные процессы взаимодействуют между собой, то их работа может существенно зависеть от моментов взаимодействия: простое изменение скорости работы отдельного процесса/процессов может привести к фатальным ошибкам или к разным результатам работы параллельного алгоритма. Чтобы этого не произошло, формулируют протоколы, которые учитывают разнообразные нюансы взаимного влияния процессов друг на друга.

Современное параллельное программирование – это история изобретения способов корректного взаимодействия процессов. Но и раньше и сейчас процессы изменяют и принимают информацию ровно тогда, когда они готовы к этому, и/или в соответствии с протоколами взаимодействия. Но сплошь и рядом возникают ситуации, когда в силу тех или иных причин (часто из-за логических ошибок в протоколах) пропускаются и/или получаются разными какие-то промежуточные/конечные результаты, приложение ведет себя непредсказуемым образом и т.д. и т.п. Подобные ситуации просто провоцирует многопоточное программирование, в котором моменты взаимных «встреч» процессов/потоков непредсказуемы.

Может, если процессам придать свойства времени, то таких проблем будет меньше?

3. Модель процессов, учитывающая время

Если процессы не взаимодействуют, то для них время не существенно, т.к. выполнять их в этом случае можно просто последовательно и/или в любом порядке. Все это на результатах их работы никак не скажется. Разве только на времени получения результата от того или иного процесса.

И хотя отдельному процессу время не нужно, оно, как мы убедимся далее, будет остро необходимо множеству взаимодействующих процессов. Следовательно, перед переходом к модели множества процессов необходимо рассмотреть модель процесса, обладающего свойством времени.

3.1. Автоматная модель процесса

К моделям последовательных процессов, в которые время введено в явной форме, относятся модели автоматного типа (далее нас будут интересовать только конечные автоматы (КА)). Поэтому на уровне отдельного процесса, что-то придумывать даже не нужно, по крайней мере, с точки зрения модели времени: любой автомат по определению работает в дискретном времени, где дискретное время является моделью реального времени. Автомату можно указать, через какое число тактов нужно выдать информацию и/или на каком такте можно ее принять. У моделей, не имеющих встроенного дискретного времени, такой возможности нет просто по определению.

Заметим, что с введением времени модель приобретает еще одно качество реальных процессов – инерционность. В реальных процессах нет мгновенных действий. Задержке реакции реального объекта на входное воздействие в автоматной модели соответствует один или несколько дискретных тактов.

В связи с темой задержек необходимо ввести в модель понятие *транспортной* и *инерционной* задержек. Выделение двух типов задержек весьма важно для моделирования реальных объектов. И если последние обладают только инерционной задержкой, то абстрактные объекты могут иметь задержку и транспортного типа (подробнее о типах задержек см. [5]).

Итак, если немного заглянуть вперед, то, оперируя даже просто дискретными тактами, мы можем легко согласовать обмен информацией между параллельными автоматными процессами, организовав в определенное время (дискретное) «встречу» между ними.

3.1.1. Практическое приложение автоматных моделей. В программировании применение автоматов не является экзотикой. Наиболее известная область приложения автоматных моделей – проектирование компиляторов. В области проектирования систем весьма популярна автоматная модель языка UML [6].

Просто огромный пласт применения автоматных моделей приходится на проектирование управляющих систем. Здесь, как на аппаратном, так и на программном уровне применение автоматов – такая же классика, как автоматы в проектировании компиляторов. Так, приобретшая в последнее время достаточную известность и рассмотренная ниже SWITCH-технология, которая ориентирована на проектирование систем именно данного класса, базируется именно на модели автоматного типа.

И тут необходимо отметить, что если в области проектирования компиляторов время, действительно, интересно лишь с точки зрения получения конечного результата, то в UML, как в языке, позиционируемом в качестве языка моделирования, или в SWITCH-технологии, как технологии разработки управляющих систем, не использование возможностей дискретного времени представляется одним из их упущений. Этому есть свое объяснение, связанное уже с переходом к модели качественно иного уровня – параллельной. Там, где нет модели параллельных процессов или она не проработана, о едином времени говорить сложно.

3.2. Синхронные и асинхронные модели

Говоря о времени процессов, нельзя не обойти вниманием понятия синхронной и асинхронной их работы. В программировании понятие асинхронный процесс почти синоним понятия более быстрый. Но можно показать, что, вопреки бытующему мнению, это далеко не всегда так. А поскольку эффективность работы модели и принцип ее реализации – асинхронный или синхронный – важные вопросы, то выяснение, к какому типу отнести формальную модель, имеет большое значение.

В теории конечных автоматов существует несколько определений синхронных и асинхронных процессов. Мы будет придерживаться формулировки, когда синхронными считаются автоматы, у которых реальная длительность дискретных тактов одинакова, асинхронными тех, у которых она разная (см. также [6, 7]). Другое определение асинхронности представлено в [8].

Выбранные формулировки аналогичны определению асинхронности сетей Петри. «В реальной жизни различные события укладываются в различные интервалы времени», что и находит отражение в определении закона функционирования СП [9]. Сравнение автоматов с СП с точки зрения асинхронности поможет нам в дальнейшем при более детальном сравнении данных моделей.

4. О едином времени параллельных процессов

Выше мы рассмотрели модель отдельного процесса, обладающую свойством времени. Для этого мы взяли уже известную модель. Следующий логичный шаг – рассмотреть модель множества процессов, где каждый процесс представлен моделью конечного автомата. В результате мы приходим к модели параллельных процессов в форме автоматной сети.

Но для процессов важно не просто наличие «часов», а их точный единый синхронный ход. Мало проку от тех часов, которые идут с непредсказуемой скоростью. Таким образом, мы должны иметь не только структурно параллельную модель, но и модель, структурные блоки которой обладают временем, текущим для них одинаково. Сложно все же договориться о встрече, если потеряны часы. Субъекты, имеющие их, назначив время, встретятся без труда.

«Часы» – универсальный, простой и весьма эффективный механизм организации взаимного согласования (синхронизации) реальных процессов. Если бы был более простой механизм, то мы бы использовали их только в качестве украшения. Из сказанного логично предположить, что у вычислительных процессов тоже должно быть какое-то подобие «часов» и «часов» единых.

Таким образом, формальная модель параллельных процессов с присущим ей свойством времени, которое течет к тому же для них всех одинаково, исходя из физики параллелизма, да и просто практических соображений, наиболее адекватно соответствует реальным параллельным процессам. Можно достаточно просто на конкретных примерах убедиться, что единое время во многом только облегчает жизнь процессам.

4.1. Автоматная модель параллельных процессов

Автоматы хороши тем, что для них очень просто организовать единое время. Для этого достаточно заставить автоматную сеть работать в общем дискретном времени. В этом случае компонентные автоматы сети могут «встречаться»

друг с другом, не рискуя потерять информацию или сорвать моменты обмена ею, а назначив лишь время, которое будет определяться числом тактов дискретного времени. Хорошее обоснование необходимости введения единого времени для параллельно работающих автоматов можно найти у В.М. Глушкова и, пожалуй, только у него [6].

Единое дискретное время позволяет резко упростить процедуру обмена информацией между вычислительными процессами. Синхронные протоколы, использующие единое время, могут в разы увеличить скорость работы приложений. К примеру, параллельная сортировка, использующая протокол на базе единого времени, в несколько раз быстрее, чем параллельная сортировка с более сложным асинхронным протоколом [10].

В принципе в любую параллельную систему не составляет большого труда ввести единое время, включив в число ее процессов тактовый генератор. Он возьмет на себя роль «часов» и по нему будут синхронизировать работу все остальные процессов. Здесь будет, правда, одна проблема, сделать генератор асинхронным, чтобы повысить эффективность работы системы. В модели с единым временем генератор и соответствующая синхронизация процессов в пределах одно такта дискретного времени встроены по определению. И этот генератор асинхронный.

Мы не будем рассматривать детально системные потери процессов, функционирующих в едином времени. А такие потери, особенно, если параллелизм моделируется, будут, т.к. в общем случае реальная длительность отдельного дискретного такта системы будет определяться длительностью самого медленного действия, выполняющегося на текущем такте работы реальной автоматной сети. Можно лишь сказать, что если отдельный автомат и проиграет в скорости работы (особенно, если он автономен), то, работая в рамках сети и взаимодействуя с другими автоматами, он от такой организации только выиграет. Т.к., с одной стороны, взаимодействующему процессу в любом случае нужно ждать момента обмена информацией, а, с другой стороны, автоматная модель позволяет организовать само взаимодействие с минимальными системными потерями.

Для полноты картины осталось только отметить, что действия, выполняемые автоматами на том или ином такте работе системы, на формальном уровне считаются мгновенными и непрерываемыми. Исходя из этого, нет проблем с тем, как «уложить» то или иное действие в текущий такт работы системы. Если действие мгновенно, то оно всегда завершится в пределах такта, длительность которого может быть сколь угодно малой, но не равной нулю. И, по-видимому, здесь сложно придумать что-то иное. Например, именно такое соглашение в отношении действий существует в языке UML [11].

Безусловно, наряду с дискретным временем в автоматной модели могут применяться и хорошо известные способы взаимодействия/синхронизации между процессами. Например, когда оговаривается процедура согласованного обмена информацией между процессами. Но в дискретном времени подобный протокол, как правило, будет проще, а потому и быстрее. Если есть такая возможность – единое время, то для повышения быстродействия параллельного алгоритма ее необходимо использовать (см. сравнение работы параллельных сортировок с синхронным и асинхронным протоколом [10]).

4.2. Об асинхронности автоматной модели параллельных процессов

Один из аргументов критиков модели с единым временем – синхронность действий, выполняемых в пределах одно дискретного такта. Действительно, если на некотором такте действие хотя бы одного процесса по длительности будет превышать длительность действий остальных процессов, то возникнут явные потери, связанные с выравниванием времен действий процессов. На каком-то такте такие задержки может уже создать другой процесс и т.д. Т.е. то, что на уровне формальной модели не вызывает опасений, т.к. любые действия по соглашению мгновенны, в реализации может привести к системным потерям.

Тут сразу напрашивается сравнение с другими параллельными моделями. Как обстоят с этим дела у них? Насколько справедливы упреки, высказанные в адрес автоматной параллельной модели с единым временем? Так, если мы возьмем те же сети Петри, то там такая же ситуация: системные потери на переходе от одной разметки к другой ничем не отличаются от системных потерь модели СМКА.

А теперь посмотрим, что будет, если мы уберем единое время. Во-первых, «рухнет» математика, т.к. не будет работать алгебра автоматов, и мы не сможем делать доказательные утверждения о свойствах модели. Кому-то это может показаться не столь страшным, но на самом деле это сразу повлечет огромные проблемы в надежности системы. А поскольку надежность упадет, то необходимо будет прилагать дополнительные усилия по ее обеспечению. А это может свести на нет выгоду, полученную от отказа от единого времени.

Покажем справедливость сказанного на простом примере. Пусть есть два абсолютно одинаковых процесса, моделирующих перемещение с одинаковой скоростью. Программно их моделируют два циклических счетчика, где значение счетчика – аналог длины пройденного пути. Если скорости процессоров, на которых они исполняются, одинаковы, то за одно время счетчики достигнут одного значения. А если скорости процессоров разные? А если процессы моделируются многопоточностью?

Эксперимент показал, что многопоточность не гарантирует одинаковую скорость течения процессов. В случае нашего элементарного примера процессы, прерванные в некоторый момент, покажут разное значение счетчиков. В модели, обладающей концепцией единого времени такое просто невозможно. Будь то аппаратная ее реализация, многопоточная или любая другая, абсолютно одинаковые процессы должны быть абсолютно одинаковы по всем свои параметрам в любой своей реализации.

Таким образом, приведенный пример показывает, что без единого времени даже в простейшей ситуации может потребоваться синхронизация, которая сама по себе требует накладных расходов. А они могут быть очень большими. Эксперименты на примере приведенной задачи показали, что ее реализация на виртуальной машине, реализующей СМКА модель, на порядок быстрее аналогичной реализации на потоках. И это притом, что на потоках не выполнялась синхронизация с целью корректного моделирования поставленной задачи.

Результаты сравнения синхронного и асинхронного алгоритмов, созданных в рамках одной модели – автоматной, убеждают в наличии больших потерь, вызванных необходимостью синхронизации асинхронных процессов. На примере философов Дейкстры, выполняющих вместо еды спагетти полезную работу – сортировку вилок, они работают в пять раз медленнее параллельной сортировки на базе простого синхронного протокола (подробнее см. [10]). Кстати, сами фи-

лософы примерно в три раза работают эффективнее, чем последовательная быстрая сортировка Хоара.

В целом у нас нет серьезных оснований, кроме чисто интуитивного предположения об эффективности асинхронной модели, для отказа от концепции единого времени. Проведенные тесты сравнения, например, с многопоточностью, результаты использования автоматной модели в системах реального времени (см. например, [12]) убеждают в высокой эффективности работы концепции единого времени в автоматной интерпретации. И это уже на уровне виртуальной машины. Ее можно реализовать аппаратно, причем проще, чем любую другую из известных параллельных машин (тех же сетей Петри).

Эффективность реализации виртуальной автоматной модели также связана с асинхронной реализацией ее дискретного времени. А его именно так можно назвать, рассматривая синхронную или асинхронную реализацию дискретных тактов (см. также п. 3.2.). Дело в том, что каждый такт может выполняется ровно столько, сколько нужно на реализацию действий, выполняемых на этом такте (в этом же, кстати, выражается и асинхронность СП).

Но если бы мы выбрали синхронную реализацию дискретного времени, то пришлось бы выбрать длительность такта заведомо больше, чем длительность вероятных единичных действий в системе. В этом, видимо, заключена и причина неэффективности многопоточной реализации, где переключения между потоками происходят с заданным интервалом времени. Причем, когда нужно и когда не нужно. В автоматной модели это делается только в нужные моменты времени.

5. О сравнении вычислительных моделей

Очень мало моделей, в отношении которых можно сказать, что они учитывают физику параллелизма. В то же время, для большинства моделей существуют формальные процедуры перехода к эквивалентной конечно-автоматной модели. К примеру, для любой реальной программы, построив ее блок-схему, можно, используя процедуру разметки граф-схем алгоритмов (ГСА) (см. [8]), легко построить конечный автомат. Таким образом, дискретное время можно легко сопоставить любой модели, если существует или возможна процедура перехода от нее к эквивалентной модели автоматного типа.

Немного сложнее разбираться с единым временем. Но если удается привести отдельные процессы параллельного алгоритма к конечным автоматам, то тем самым можно легко получить ответ на вопрос, в каком времени работают эти процессы. Так, сети Петри следует отнести к параллельной модели с единым асинхронным временем. В них единое время определяется моментами перехода от одной разметки к другой, а асинхронный характер времени связан с плавающим временем перехода между соседними разметками сети.

А вот в UML единого времени нет. То, как время течет в рамках для множества автоматов, в рамках UML просто не рассматривается. Все откладывается на этап реализации, а реализация, если это не оговорено на уровне определения модели, может быть произвольной. Но от такой «свободы» только больше проблем, чем ясности.

Другая автоматная модель, в какой-то мере затрагивающая параллельные процессы, но в которой нет понятия единого времени – модель, положенная в основу SWITCH-технологии [12]. Ее мы подробно рассмотрим ниже.

Есть еще одно не столь очевидное, но очень важное свойство единого времени. В подобной модели на множестве процессов можно ввести математические операции. В данном случае речь идет об алгебре автоматов [7]. Без единого времени о каких-то математических операциях на множестве процессов говорить сложно, если вообще возможно. Сама же возможность оперировать процессами на формальном уровне перекрывает многократно все те возможные неудобства, которые несет с собой введение единого времени для процессов на практике.

5.1. Конечно-автоматная технология

Конечно-автоматная технология проектирования параллельных программ (КА-технология) и являющаяся ее ядром параллельная автоматная модель – Сеть Машин Конечных Автоматов (СМКА), учитывающие «физику параллелизма», представлены в [13-15]. В КА-технологии любая параллельная программа – это множество параллельно функционирующих и взаимодействующих между собой структурных блоков. На алгоритмическом уровне поведение блоков представлено конечным автоматном (КА). Но если, к примеру, в UML автоматы весьма специфичны, то в СМКА это классический автомат Мили. Таким образом, здесь можно без каких-либо изъятий применять теорию конечных автоматов, с чем в UML будут проблемы.

Все автоматы СМКА-модели составляют автоматную сеть, автоматы которой функционируют в едином дискретном времени. Это позволяет определить алгебру конечных автоматов [7]. Введенная в рамках КА-технологии алгебраическая операция умножения автоматов позволяет находить для любой СМКА-модели последовательный эквивалентный автомат, дающий однозначное и строгое представление о работе автоматной сети. Это позволило сделать подробный анализ решенной в форме СМКА-модели задачи о философах Э. Дейкстры, дав ее полное и корректное решение [16]. Другие решения данной задачи, как правило, такой доказательной базы не имеют (см., например, [9, 17, 18]).

Строго конечный автомат, да и сама СМКА модель, не могут быть в общем случае моделью программ. С целью определения модели программ в КА-технологии сформулировано понятие автоматных схем программ (подробнее о моделях программ см. [19]). Автоматные схемы программ (АСП) – это программная модель, в которой управление программы представлено моделью автоматного типа. В данном случае – СМКА моделью. Таким образом, автоматная модель – лишь часть общей программной модели, в которой кроме управления есть еще и операционная часть. Операционная часть АСП включает память (данные) программы, операции и функции, реализующие предикаты и действия автоматной модели (подробнее см. в [20]).

Взаимодействие между автоматами в СМКА может осуществляться любым способом - через общую память, путем обмена сообщениями и т.д. и т.п. Но есть и характерный только для модели автоматного типа способ синхронизации – синхронизация через состояния: любой автомат может узнать информацию о текущем состоянии любого автомата сети, чтобы затем руководствоваться в своих действиях этой информацией.

Учитывает КА-технология и объектный характер современных программных технологий. В реализацию КА-технологии на языке C++ введен базовый автоматный класс, от которого порождаются все остальные автоматные классы. Последние, став объектами, представляют отдельные автоматные процессы, а все вместе – некий параллельный процесс. На объектном уровне КА-технология

расширяет возможности объектной парадигмы программирования, в которой не оговорены вопросы параллельной работы объектов. Созданные в КА-технологии объекты – это по существу рассматриваемые в ООП в последнее время так называемые активные объекты, агенты и подобные им «живые объекты».

Для технологии программирования важно, что в СМКА автоматы могут быть вложенными. Вложение автоматов полностью аналогично вызову обычных программных процедур/функций. В результате одной и той же моделью покрывается как модульная организация программ, так и рекурсивная модель вычислений [21].

Фактически в рамках КА-технологии на уровне формальной модели нет различия, связанного со спецификой аппаратной и/или программной реализациями. Один из принципов разработки программ в КА-технологии – это наиболее полное использование опыта и наработок, имеющихся в области проектирования цифровых схем. Само разделение программы на управляющий и операционный блоки подобно разделению, принятому в построении моделей цифровых схем; автоматная модель Мили больше известна как модель цифрового автомата; из теории проектирования аппаратных средств заимствовано применении алгебры автоматов и т.д. и т.п.

Таким образом, вплоть до так называемого этапа синтеза вопрос о реализации модели открыт, т.к. то, что реализуется программно, столь же просто с применением типовых процедур синтеза цифровых схем может быть реализовано аппаратно.

5.2. SWITCH-технология

Рассмотрим подробнее одну из приобретших в последнее время известность программных автоматных технологий, сравнив ее с КА-технологией. Речь пойдет о SWITCH-технологии [12]. Пересекаясь в концепции – использовании конечных автоматов, как базовой модели для проектирования программного обеспечения – данные технологии весьма существенно отличаются друг от друга в деталях.

Сначала об используемых формальных моделях вычислений на уровне отдельного процесса. Если в КА-технологии на этом уровне используется только модель автомата Мили (остальные модели моделируются им), то в SWITCH-технологии отдают предпочтение автомату Мура, не исключая применение и остальных. Но изменения, внесенные в SWITCH-технологии в автоматную модель, столь значительны, что приводят к проблемам приложения классической теории автоматов к процессу формального проектирования.

Просто автомат – еще не формальная модель программы. И если с целью определения модели программы в КА-технологии введено понятие автоматных схем программ, то в SWITCH-технологии аналогичной модели нет. Таким образом, имеем достаточно парадоксальную ситуацию: есть автоматная технология, но нет формального определения автоматной программы.

Дальнейшее отличие качественного характера связано с формальной моделью уровня параллельных процессов. В SWITCH-технологии декларировано использование в качестве модели параллельных процессов системы взаимосвязанных графов (СВГ), в КА-технологии для этого разработана модель, представляющая собой множество конечных автоматов Мили, функционирующих в едином дискретном времени – СМКА. И если в КА-технологии для реализации введенной параллельной модели разработана виртуальная машина, то в

SWITCH-технологии подобной машины нет. В проектах, созданных с применением SWITCH-технологии, для реализации параллелизма используется обычно многопоточность, которая к модели СВГ, да и вообще к формальным моделям, имеет косвенное отношение.

Совершенно разные подходы используются в отношении программной реализации автоматов. В SWITCH-технологии для реализации КА используется переход к существующей программной модели с использованием языковой конструкции SWITCH. Поэтому, строго говоря, использование термина «автоматное программирование» применительно к данной технологии не верно. Программирование здесь обычное, т.е. на базе модели блок-схем (или граф-схем алгоритмов (ГСА) [9]). В КА-технологии используется прямая реализация автоматов. Уровень реализации автоматов в КА-технологии определяет новую архитектуру процессора, которая реализована на текущий момент в форме виртуальной машиной. Эта же виртуальная машина реализует и более сложную модель – параллельную.

Различны объектные возможности данных технологий. КА-технология уже давно использует парадигму ООП для описания и реализации автоматов. Делается это просто, прозрачно и ясно. В существующей реализации на C++ есть базовый автоматный объект, которому можно задать поведение в виде таблицы переходов (ТП) автомата в текстовой форме. Производные классы перегружают виртуальные автоматные методы (предикаты и действия) базового класса и любого другого, порожденного от него класса, передавая базовому классу адрес описания ТП. Созданные автоматные объекты имеют свойства обычных объектов, расширяя их автоматным поведение. Реализацию автоматного поведения подобных объектов берет на себя виртуальная машина, реализующая автоматную среду исполнения. В SWITCH-технологии нет четкой «объектной позиции» в отношении реализации автоматов. Существует несколько подходов, которые достаточно сложны и реализации, и в применении. А рассматриваемая шаблонная реализация автоматов вообще противоречит их динамической сути.

Отдельные слова необходимо сказать в отношении организации вложенной работы автоматов. В обеих технологиях для этого используется одинаковый термин – вложенные автоматы. Но это разное вложение. Только в КА-технологии оно полностью аналогично вызову процедуры в обычном программировании, что позволяет легко реализовать рекурсивное вложение автоматов. В SWICH-технологии в этой ситуации приходится избавляться от рекурсии, что достаточно трудоемко и как любое преобразование чревато внесением ошибок.

Но нагляднее, безусловно, сравнивать разные технологии на основе решения одинаковых примеров. Такие примеры есть. Это решение таких широко известных классических параллельных задач, как обедающие философы Дейкстры, стрелки Майхилла или, например, менее известная задача о преступниках (см. для сравнения [22] и [23]). Эти и подобные им решения находятся в свободном доступе на сайтах по SWITCH-технологии (см. разделы Проекты и Статьи сайта http://is.ifmo.ru) и КА-технологии (раздел Автоматы сайта SoftCraft http://www.softcraft.ru).

6. Заключение

С появлением многоядерных архитектур наступает эра универсальных параллельных вычислений. Но пока это параллелизм интуитивного представления

о реализации параллельных процессов, не базирующийся на какой-либо формальной модели, что плохо. И от того, какая в основу параллельной архитектуры будет положена модель, зависит эффективность работы не только и не столько процессора, а и программистов, использующих данную архитектуру. При этом формальная модель определяет технологию проектирования программ, от качества которой напрямую зависит качество программного продукта. К нему же ныне просто огромные претензии.

Нельзя не отметить вновь возникший интерес к автоматным моделям. Кроме UML, SWITCH-технологии и КА-технологии можно привести еще ряд примеров использования автоматных моделей в проектировании программ. Кроме этого есть целые разделы в программировании, где уже давно и постоянно ведется разработка программ с использованием автоматов.

Не менее веские доводы существуют в пользу введения единого времени, тем более что известны модели, имеющие его. К ним, кроме упомянутых выше сетей Петри, относятся клеточные автоматы и нейронные сети. В клеточном автомате в едином дискретном времени работают автоматы, расположенные в узлах клеток [25]. В нейронных сетях в едином времени работают нейроны [2]. От данных моделей модель СМКА отличается тем, что автоматы в узлах автоматной сети могут быть произвольными, как произвольными могут быть и связи между ними. У клеточных автоматов и у нейронных сетей автоматы просты и имеются, как правило, ограничения на число и/или вид связей между ними.

С учетом уже имеющихся теоретических и практических результатов имеется достаточная база, которая позволяет сравнивать между собой не только автоматные модели и технологии проектирования на их базе, но проводить сравнения с другими формальными моделями и технологиями. Например, с этой точки зрения очень интересно сравнение теоретических возможностей СП и СМКА в контексте решения задачи о разделении множеств Э. Дейкстры. Ее решение, полученное Ю.Г. Карповым, дано в пособии В.Э. Малышкина [26]. И если в своем решении Ю.Г. Карпов доказывает отсутствие тотальной корректности у данной задачи, то автоматное решение в рамках модели СМКА дает решение, тотальная корректность которого доказывается в рамках алгебры автоматов (см. подробнее в [27]).

В приложении дано параллельное автоматное решение задачи параллельного суммирования чисел на базе алгоритма сдваивания [1]. Его можно сравнить с аналогичным решением, полученным в рамках функционально-потоковой модели параллельных вычислений, которое приведено в [28].

Приложение. Реализация алгоритма сдваивания автоматной моделью

Алгоритм параллельного сложения чисел, названный алгоритмом сдваивания, состоит в последовательности выполнения следующих шагов. «Разобьём все слагаемые на пары и осуществим суммирование двух чисел внутри каждой пары. Все эти операции независимы. Полученные частные суммы также разобьём на пары и снова осуществим суммирование двух чисел внутри каждой пары. Снова все операции независимы. Вся сумма будет получена через $\log_2 n$ шагов» [1].

Усложним несколько ситуацию, предположив, что массив чисел формируется динамически и параллельно самому процессу суммирования. Подобная постановка задачи дана в [28]. Возьмем ее за основу и рассмотрим реализацию динамического суммирования чисел в соответствии с алгоритмом сдваивания, когда числа поступают в массив в произвольные моменты времени.

На рис. 1. представлена структурная модель алгоритма сдваивания. Параллельная программа, построенная по этой схеме, выполняет параллельное суммирование массива чисел, которые поступают в него по мере их готовности. На схеме блоки ConfigData и FInToList отвечают за моделирование процесса поступления данных в массив. Из них только блок FInToList является процессом. Он помещает в некоторые временные моменты числа, которые заданы объектом ConfigData, в массив, названный в [28] асинхронным списком (на самом деле в программе это обычный список).

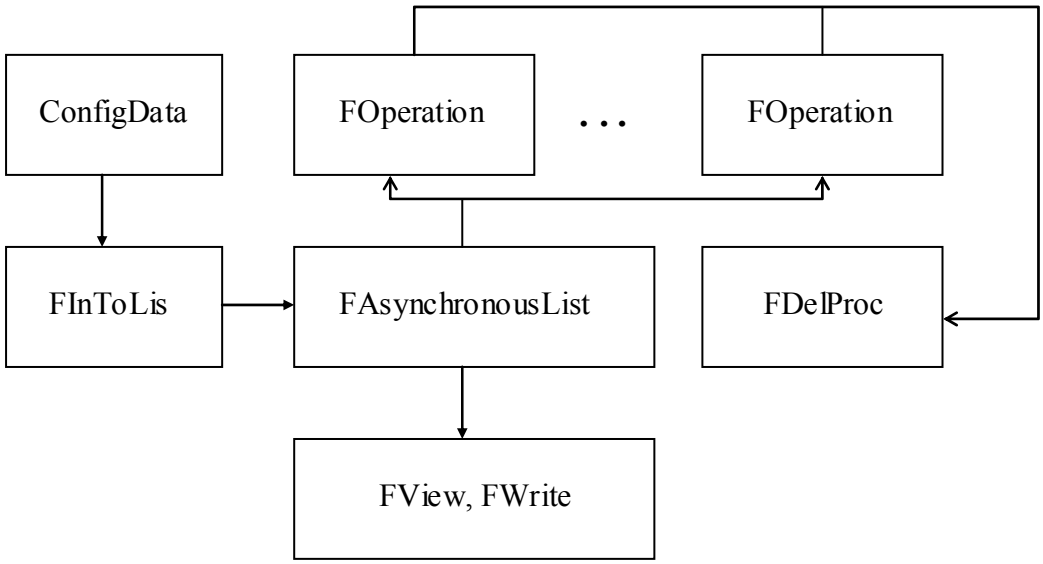

Рис. 1. Структурная модель алгоритма сдваивания.

Процесс, представленный на схеме блоком FAsynchronousList, анализирует асинхронный список (АС) и порождает процессы – FOperation, которые выполняют операцию суммирования пар чисел.

Процесс FDelProc удаляет процессы типа FOperation, которые завершили свою работу. Делает это он, проверяя активность списка созданных процессов. Неактивные процессы удаляются. Т.е. это своего рода «мусорщик сбора неактивных процессов»

Процессы FView и FWrite отображают текущее состояние асинхронного списка соответственно в окне отображения приложения и в файл на диске.

Собственно к алгоритму сдваивания имеют отношения только процессы FAsynchronousList и FOperation. Все остальные – служебные и составляют их окружение. Одни из них реализуют процесс динамического формирования асинхронного списка, другие отвечают за корректное удаление неактивных процессов, третьи выполняют отображение текущего состояния АС в разных формах.

Алгоритмические модели структурных блоков в рамках КА-технологии представляют собой конечные автоматы в форме автоматов Мили. Все автома-

ты работают параллельно. Далее мы рассмотрим только автоматы, реализующие собственно алгоритм сдваивания. В форме графов автоматов они представлены на рис. 2.

Автомат FAsynchronousList функционирует следующим образом. Находясь в начальном состоянии «st», он ожидает сигнала о начале работы (предикат x5 примет значение «истинно»). Дождавшись, он переходит в состояние «w», где анализируется число элементов в списке – за это отвечает предикат x1 и условие завершения работы – x4. Если элементов в списке меньше двух и нет условия завершения работы, то выполняется переход по петле с действием y9. Данное действие наращивает счетчик «холостых» проходов, отражающих непрерывное число автоматных тактов, в течение которых в списке меньше двух элементов. Его значение определяет момент перехода в заключительное состояние – «00».

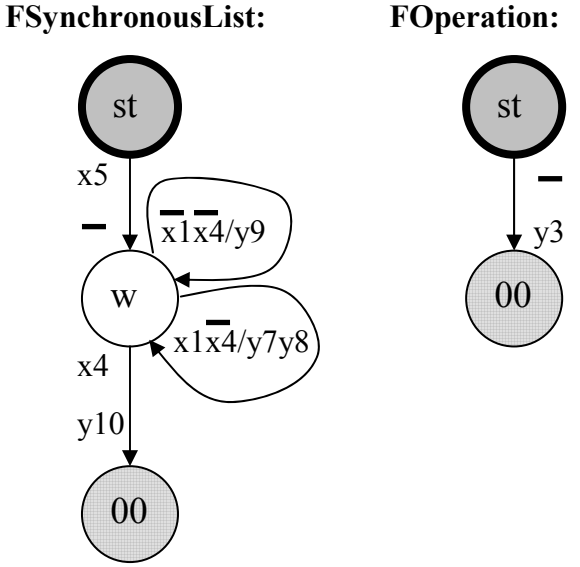

Рис. 2. Автоматные модели алгоритма сдваивания.

Если предикат x1 принял истинное значение, то для каждой пары элементов АС [параллельные] действия y7, y8 порождают операцию – автомат FOperation. При этом счетчик «холостых проходов» сбрасывается в ноль, а элементы данных, для которых порождена операция, исключаются из списка. Адреса процессов-операций помещаются в список созданных процессов, который затем обрабатывает автомат FDelProc. Мы его не рассматриваем, но функция его проста – проверить активность созданных процессов и удалить неактивные.

Автомат FOperation выполняет один переход и завершает на этом работу, переходя в заключительное состояние «00». Его действие y3 реализует операцию суммирования и помещает результат в тот же асинхронный список.

Ниже приведен листинг протокола для тестового набора данных – 44, 94, 6, 55, 13, 42, 18, 67. Он отражает текущее состояние асинхронного списка, когда а) данные в него поступают во времени, и протокол, когда б) данные поступили в список одновременно, т.е. в течение одного дискретного такта времени. И в том и другом случае результат один и тот же, что может служить тестом правильном работы приложения в любом режиме (речь о режимах поступления данных в список).

```
а)асинхронное поступление данных в список
    1:
    2:44.,94.,
    3:
    4:138.,
    5:138.,6.,
    6:
    7:144.,
    8:144.,
    9:144.,
   10:144.,
   11:144.,
   12:144.,55.,13.,42.,18.,67.,
   13:
   14:199.,55.,85.,
   15:85.,
   16:85.,254.,
   17:
   18:339.

б)одновременное поступление данных в список
    1:
    2:44.,94.,6.,55.,13.,42.,18.,67.,
    3:
    4:138.,61.,55.,85.,
    5:
    6:199.,140.,
    7:
    8:339.
```

Список литературы

1. Воеводин В.В. Математические проблемы параллельных вычислений. http://parallel.ru/info/voevodin.doc
2. Минский М. Вычисления и автоматы. М.: Мир, 1971. 364 с.
3. Любченко В.С. Фантазия или программирование? // Мир ПК. 1997. № 10. С. 116-119. http://www.osp.ru/pcworld/1997/10/116.htm
4. RS-триггер, как мини модель параллельной системы. Форум сайта SoftCraft. http://www.softcraft.ru/forum/viewtopic.php?t=40
5. Армстронг Дж.Р. Моделирование цифровых систем на языке VHDL: Пер с англ./М.: Мир, 1992. 175 с.
6. Глушков В.М. Синтез цифровых автоматов. М.: Физматгиз, 1962.
7. Мелихов А.Н. Ориентированные графы и конечные автоматы. М.: Наука, 1971. 416 с.
8. Баранов С.И. Синтез микропрограммных автоматов. Л.: Энергия, 1979. 232 с.
9. Питерсон Дж. Теория сетей Петри и моделирование систем: Пер с англ. М.: Мир, 1984. 264 с.
10. Любченко В.С. Параллельные сортировки: быстрее, проще... умнее // Открытые системы. 2005. № 5. http://www.osp.ru/os/2004/05/049.htm
11. Рамбо Дж., Якобсон А., Буч Г. UML: специальный справочник. СПб.: Питер, 2002. 656 с.
12. Любченко В.С. Аппаратно-программный комплекс для выращивания кристаллов методом ОТФ // Тезисы докладов. IX Национальная конференция по росту кристаллов НКР-2000. М.: ИК РАН, 2000. С. 61.
13. Шалыто А.А. SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998. 628 с.
14. Любченко В. С. Автоматная модель параллельных вычислений // Высокопроизводительные параллельные вычисления на кластерных системах: Материалы четвертого Международного научно-практического семинара. Самара, 2005. С. 181-186.

15. Любченко В.С. Эксперименты над абстрактными машинами. // Мир ПК. 2002. № 2, 3. http://www.osp.ru/pcworld/2002/02/150.htm, http://www.osp.ru/pcworld/2002/03/152.htm
16. Любченко В.С. От машины Тьюринга к машине Мили. // Мир ПК. 2002. № 8. http://www.osp.ru/pcworld/2002/08/130.htm
17. Любченко В. С. К решению проблемы обедающих философов Дейкстры // Высокопроизводительные параллельные вычисления на кластерных системах: Материалы четвертого Международного научно-практического семинара. Самара, 2005. С. 186-193.
18. Хоар Ч. Взаимодействующие последовательные процессы: Пер. с англ. М.: Мир, 1989. 264 с.
19. Элементы параллельного программирования / Под ред. В.Е. Котова. М.: Радио и связь, 1983. 240 с.
20. Котов В.Е. Введение в теорию схем программ. Новосибирск.: Наука, 1978. 258 с.
21. Любченко В.С. Автоматные схемы программ (с исторической ремаркой автора). http://www.softcraft.ru/auto/ka/ash/ash.shtml
22. Любченко В.С. О борьбе с рекурсией. http://www.softcraft.ru/auto/ka/recwar.shtml
23. Любченко В.С. Задача о преступниках. http://www.softcraft.ru/auto/ka/crimes/crimes.shtml
24. Мазин М., Шалыто А. Преступники и автоматы // Мир ПК. 2004. № 9.
25. Тоффоли Т., Марголус Н. Машины клеточных автоматов: Пер. с англ. М.: Мир, 1991. 280 с.
26. Малышкин В.Э. Основы параллельных вычислений. Часть 1. http://www.ssga.ru/metodich/paral1/contents.html
27. Тема «Разделение множеств (написание реальных ПП)» на форуме сайта SoftCraft. http://www.softcraft.ru/forum/viewtopic.php?p=2227.
28. Легалов А.И. Использование асинхронных списков в потоковой модели вычислений. http://www.softcraft.ru/parallel/alist/index.shtml.