Процессоры с хранимым алгоритмом (контекстно-зависимой программой)

Н.В. Стрельцов

Формулируются основные принципы построения и классификации процессорных архитектур на концептуальном уровне. Исходя из общности организации вычислительных и биологических систем делается вывод о выборе дальнейшего направления развитии процессорных архитектур, а именно создании мультиклеточных процессоров с хранимым алгоритмом (контекстно-зависимой программой). Приводятся их основные характеристики.

1. Введение

«Фон-неймановская» эпоха в компьютерной индустрии завершается. В International Technology Roadmap for Semiconductors (ITRS) констатируется, что эволюционное развитие основных вариантов реализации этой модели процессора, сходит на нет [1]. Внедряемые решения не только перестали обеспечивать увеличение производительности процессоров в расчете на один транзистор, но и привели к непрерывному снижению этого показателя [2].

Выход из сложившейся ситуации – это поиск качественно нового («пост-неймановского») направления развития процессорных архитектур.

На данный момент, в качестве основного направления, в ITRS рассматривается многоядерность. Но, это не качественно новый шаг. Это экстенсивное и, в силу этого, временное направление развития все той же фон-неймановской модели.

Неизбежность смены архитектурной модели диктуется не только экономикой. Она следует из положений теории эволюционного развития компьютеров, декларирующей внешнее сходство эволюции биологических и вычислительных систем [3]. Но, если столь разные по физической реализации объекты эволюционируют похожим образом, то можно задаться вопросом – не является ли это сходство следствием общих, более глубоких и объективных закономерностей, лежащих в основе внутренней организации данных систем?

Если такие закономерности существуют и, если на их основе может быть построено общее, полное и систематизируемое множество моделей, отражающих основные принципы организации этих систем, то анализ этого множества позволил бы не только целенаправленно подойти к выбору нового направления развития процессорных архитектур, но и стал бы мощным инструментом исследования живой природы.

В настоящей статье рассматриваются принципы построения и классификации, указанного множества моделей, применительно к вычислительным системам. Полученные результаты проецируются на организацию биологических систем и на основании результатов их эволюции делается вывод о дальнейшем направлении развития процессорных архитектур.

2. Построение и классификация процессорных архитектур

2.1. Постановка задачи

Абстрагирование от всех особенностей реализации – обязательное условие подобной классификации. Из известных подходов [4] этому условию удовлетворяет, и то только частично, классификация Флинна [5]. Она рассматривает вычислительную систему как множество командных устройств (устройств управления), исполнительных устройств и устройств памяти, связанных потоками команд и данных. Параметрами классификации являются количественные оценки потоков команд и данных.

Очевидно, что для этой цели могут использоваться только те потоки, которые присутствуют абсолютно во всех известных системах и, про которые можно сказать, что они отражают их внутреннюю сущность.

К их числу, безусловно, относятся потоки команд, формируемые устройствами управления и поступающие в исполнительные устройства. Среди потоков разнообразных данных, общими для всех являются только потоки результатов, формируемые исполнительными устройствами при выполнении ими принятого поток команд. Потоки данных в том понимании, в котором они используются в классификации Флинна, а именно вызываемые («called») при выполнении команд, т.е. поступающие из памяти на обработку, присутствуют только у части систем.

Так, например, потоковая машина не имеет потоков данных, поступающих из памяти и используемых для выполнения очередной команды. Все данные, необходимые для исполнения команды, поступают вместе с ней, как составная часть командного слова.

Это концептуальное отличие архитектуры потоковых машин обеспечивается записью результатов выполненных команд непосредственно в поля аргументов тех команд, которые их используют. Команды исполняются после получения всех необходимых аргументов. Таким образом, очередность их исполнения непосредственно зависит от потока данных, формируемого исполнительным устройством.

Указанная зависимость имеет ключевое значение для выделения потоковых машин в отдельный вид. Только количественная оценка, без учета существующих зависимостей между потоками, в принципе, не может решить задачу систематизации подобных архитектур.

Существенное значение имеет и методология получения этой оценки. Известно, что ее корректность должна обеспечиваться измерительным инструментом, не зависящим от объекта измерения и равенством условий измерения. Применительно к количественной оценке потоков эти два требования можно сформулировать следующим образом:

а) на принятом уровне абстрагирования систем должны быть однозначно определены понятия «одиночный» и «множественный», действительные по отношению к любым потокам (команд или данных) и для любых архитектур;

б) все систематизируемые вычислительные системы должны рассматриваться на одном уровне абстрагирования.

Ни первое, ни второе требования в классификации Флинна не выполняются. Суть количественных показателей не определена. Их значения устанавливаются постфактум, путем сравнения классифицируемой системы с эталонными для каждого класса образцами и, таким образом, определения ее класса.

В результате, одна и та же характеристика – «множественный поток данных», в эталонных образцах классов MIMD (многопроцессорная система) и SIMD (матричный процессор), используется для описания двух принципиально разных явлений. Множества независимых потоков данных в многопроцессорной системе и потока векторных (многокомпонентных) данных в матричном процессоре.

Следует также отметить, что эталонный образец класса SIMD не соответствует концептуальному уровню. Модель функционирования матричного процессора, предусматривающая одновременное выполнение команды над векторным элементом потока данных, отражает одну из возможных реализаций векторных команд. Для других команд, например, скалярных – эта модель не применима. Следовательно, она не применима и на концептуальном уровне, на котором команда рассматривается в обобщенном виде.

Таким образом, для построения множества моделей, абстрагированных от особенностей реализации, и последующей классификации этого множества необходимо:

а) в качестве потоков данных рассматривать потоки, формируемые исполнительными устройствами;

б) ввести в качестве параметра классификации наличие функциональной зависимости между потоками;

с) однозначно определить понятия «одиночный» и «множественный», действительные по отношению к любым потокам (команд или данных) и для любых архитектур.

2.2. Основные понятия и определения

На принятом уровне абстрагирования любой процессор будет состоять из устройств управления и исполнительных устройств.

Под устройством управления (УУ) понимается источник потока команд, поступающих на исполнение. Каждая команда обладает целостностью. Она не может быть выдана несколькими УУ и исполнена по частям несколькими исполнительными устройствами.

Исполнительное устройство (ИУ) – это источник потока данных. ИУ принимает, полностью или частично, поступающий ему поток команд и исполняет его. Выполненная команда порождает один элемент потока данных, может быть пустой.

Все источники независимы, а именно процессы, протекающие в одном, непосредственно не связаны с одновременно протекающими процессами в других источниках.

Любой источник формирует только один (одиночный) поток, который может поступать в несколько приемников.

Два и более источников одного типа (УУ или ИУ) образуют множество источников. Формируемое ими множество потоков может использоваться как совокупность одиночных потоков, либо как единое целое – множественный поток. Как и одиночный, он также может поступать в несколько приемников.

Под потоком понимается непосредственная информационная связь между источником потока и его приемником. Элементы потока существуют только в процессе передачи. Поступая в приемник, они либо используются им, либо исчезают. Для сохранения и последующего использования элемент потока должен быть отчужден, т.е. изъят из потока и размещен вне источника и приемника в окружающей их общей среде (в данном случае, в памяти). Очевидно, что отчуждение разрывает непосредственную связь (поток) между источником и приемником. Такая информационная связь становится опосредованной.

Опираясь на эти понятия можно дать следующие определения архитектуры процессора и вычислительной системы.

Архитектура процессора – это абстрактная структура, отражающая организацию процессора и представленная в виде ориентированного графа, состоящего из множества вершин, содержащего не менее одного УУ и не менее одного ИУ, а также множества дуг, образованных потоками команд, исходящими из УУ, и потоками данных, исходящими из ИУ, и при этом:

а) соответствующий ему неориентированный граф – связный;

б) каждый исходящий поток команд является входящим хотя бы для одного ИУ;

с) каждое ИУ имеет хотя бы один входящий поток команд.

Архитектура вычислительной системы – это абстрактная структура, отражающая организацию системы и образованная несвязным множеством ориентированных графов, каждый из которых представляет архитектуру процессора.

Следовательно, что обмен данными между процессорами в вычислительной системе может осуществляться только через отчуждение элементов потока данных.

2.3. Классификация процессорных архитектур

Расширив нотацию, предложенную М. Флинном, архитектуру процессора можно описать используя заглавные буквы для последовательного описания источников команд (SI или MI) и данных (SD или MD), а в скобках, после описания источников, указывая строчными буквами входящие потоки, т.е. потоки от которых зависит формирование исходящего потока. Например, MI(si,md)MD(si);SIMD(si,md);MI(mi)MD(si).

Такое двухкомпонентное описание архитектуры, в котором множество источников одного типа рассматривается как единое целое, предполагает ее гомогенность и симметричность. А именно, указанная в описании зависимость распространяется на все множество (гомогенность), а при множественности обоих типов и зависимости одного из них от одиночного потока другого, размерности множеств источников считаются равными (симметричность).

Архитектуру, имеющую двухкомпонентное описание, будем называть базовой.

Очевидно, что в реальных условиях не все процессорные архитектуры обладают гомогенностью и симметричностью. Но, можно показать, что любая из них является частным случаем одной из базовых архитектур.

Рассмотрим фон-неймановскую модель процессора. Описание и исполнение алгоритма в ней реализуется упорядоченной последовательностью команд, изменяющих состояние памяти (общей среды функционирования), т.е. оно основано на отчуждении получаемых результатов.

Следовательно, поток данных, как непосредственная связь, в этой модели отсутствует и связность архитектуры в ней обеспечивается только независимыми потоками команд. Это ограничивает фон-неймановскую архитектуру четырьмя видами показанными на рис.1.

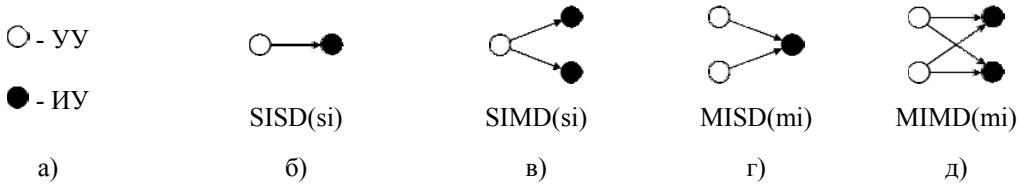

Рис. 1. Графическое и формульное описание возможных видов фон-неймановской архитектуры: а) условные обозначения; б) монопроцессор; в) процессор, использующий сопроцессоры; г) многоядерный процессор, имеющий одно ИУ; д) многоядерный процессор, имеющий несколько ИУ, каждое из них которых может выполнять команды любогоУУ.

В отличии от классификации Флинна, независимость источников объединяет архитектуру процессоров таких машин как, например, векторно-конвейерные и матричные, в один вид с классической фон-неймановской машиной. Дальнейшая их классификация должна проводиться уже внутри вида по типу операций (скалярные, векторные). Среди векторных машин – по способу реализации (векторно-конвейерные, матричные).

Рассматривая фон-неймановскую архитектуру как отправную точку, можно путем последовательного усложнения зависимостей между потоками построить полное множество возможных видов базовых архитектур. Это множество делится на два основных класса [6]:

- архитектуры с хранимой программой (контекстно-свободной программой) – 28 видов;
- архитектуры с хранимым алгоритмом (контекстно-зависимой программой) – 50 видов.

Принципиальное отличие данных классов, как показано на рис.2, заключается в характере зависимости потоков команд. Поток команд в архитектурах с хранимой программой либо независим как, например, в фон-неймановских архитектурах или в синпьютере [7], либо зависит только от потока данных. В архитектурах с хранимым алгоритмом он всегда зависит от самого себя или от самого себя и от потока данных как, например, в описываемых ниже мультиклеточном и структурно-ориентированном процессорах.

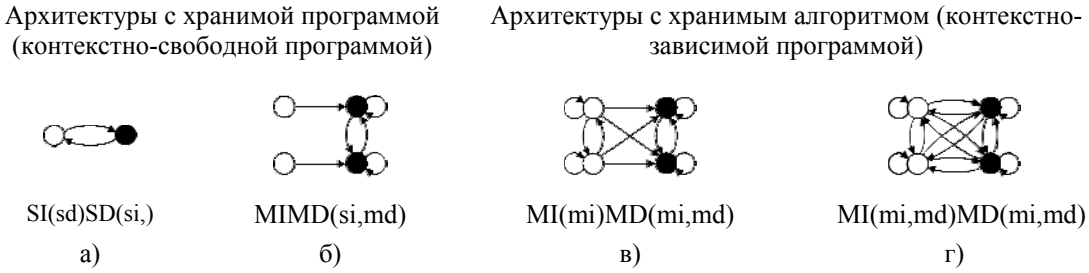

Рис. 2.Примеры архитектур с хранимой программой и хранимым алгоритмом: а) процессор, поток команд которого зависит от потока данных; б) синпьютер; в) мультиклеточный процессор; г) структурно-ориентированный процессор.

Процессорные архитектуры с хранимым алгоритмом при вычислениях оперируют не отдельными командами, а контекстно-зависимыми последовательностями команд, которые получаются, например, при использовании языка триад в качестве машинного языка. Наиболее явно отличия между этими подходами видны, если провести следующий мысленный эксперимент. А именно, поменять местами две команды. В фон-неймановском процессоре обе команды, если будут выбраны, то и будут выполнены. Если же система команд построена на базе триад, то выборка команды не гарантирует ее исполнение в «чужом» контексте. Например, команда может ждать результата, который в данной последовательности не вырабатывается.

Предложенная трактовка понятия «процессор» позволяет провести прямые аналогии между организацией вычислительных и биологических систем.

Во-первых, пару, состоящую из устройства управления и исполнительного устройства связанных потоком команд, можно рассматривать как модель простейшей одноядерной клетки на

архитектурном уровне. Соответственно, процессор можно рассматривать как примитивный одноклеточный или многоклеточный организм – «особь», состоящую из одной или более клеток, непосредственно связанных между собой потоками команд и данных. Также как и особь в биологии, процессор – это неделимая и целостная организационная единица. Выход из строя хотя бы одного устройства управления, исполнительного устройства или связи, приводят к потере работоспособности всего процессора. Для сравнения, выход из строя процессора в мультипроцессорной или многомашинной системе влияет на систему, а не на работоспособность других процессоров этой системы, т.е. на объект другого организационного уровня, частью которого он является.

Во-вторых, наличие непосредственных связей между устройствами процессора ограничивает организацию вычислительных систем из клеток всего тремя (!) формами – монопроцессор (одноклеточная особь), система процессоров (колония или сообщество), мультиклеточный процессор (многоклеточная особь).

В-третьих, принципиально возможные функциональные зависимости между потоками команд и данных позволяют построить всего два (!) класса процессорных архитектур. Первый можно рассматривать как «прокариотов», а второй как «эукариотов».

Как известно, качественное развитие живых существ, включая появление разума, связано с появлением эукариотов. Следовательно, если последнее предположение верно, то качественно новую архитектуру необходимо искать во втором классе. И, наоборот, если реализация какого–либо вида архитектуры, входящего во второй класс, обеспечивает качественно новые возможности, то можно говорить о правильности той концепции, на которой построена система классификации и проведены аналогии с миром живых существ.

3. Мультиклеточный процессор вида MI(mi)MD(mi,md)

Мультиклеточный процессор вида MI(mi)MD(mi,md) – первый параллельный процессор из класса архитектур с хранимым алгоритмом. Он образован множеством независимо, но согласованно, функционирующих клеток, связанных коммутационной средой.

Машинный язык процессора является развитием языка триад, используемого для промежуточного представления программы после «front-end» фазы в процессе компиляции.

Текст программы на языке триад не связан каким-либо образом с количеством клеток. Эта «ресурсная» независимость, неупорядоченность команд внутри их контекстной последовательности (линейного участка) и рассылка всем клеткам всех получаемых результатов обеспечивают «естественную» реализацию параллелизма (без решения задачи распараллеливания), а также эффективное масштабирование процессора [8].

Указанные особенности также делают ненужными все те методы (суперскалярность, широкое командное слово, суперконвейер, спекулятивное и предикатное исполнение и т.п.), которые обеспечивая быстродействие фон-неймановской модели, резко усложняли ее организацию. Отказ от этих методов и использование децентрализованной организации позволяет резко уменьшить сложность мультиклеточного процессора и, соответственно, снизить трудозатраты и повысить качество проектирования.

Одновременно с этим, по сравнению с традиционными фон-неймановскими решениями, улучшаются и характеристики процессора. Первые оценки позволяют говорить о росте производительности в 2-4 раза и снижении энергопотребления в 10 – 15 раз.

Система команд мультиклеточного процессора фактически является аппаратной реализацией входного языка программирования, опирающейся не на внешнюю форму, а на сущность языковых выражений.

В результате:

- Сохраняется все программное обеспечение, созданное на традиционных императивных языках высокого уровня.
- Процесс компиляции с языка высокого уровня, фактически, ограничивается начальной машинно-независимой («front-end») фазой, что позволяет резко сократить затраты на разработку компиляторов.

- Исчезает понятие «программирование на ассемблере», поскольку язык процессора не наглядный и поэтому практически «не программируемый».
- Наиболее эффективной формой существования программ становится исходный текст и, соответственно, программы становятся открытыми.

Следует отметить, что ни одна, из ранее созданных процессорных архитектур, не обеспечивает комплексного решения тех проблем, которые решаются мультиклеточным процессором. Это позволяет говорить о качественно новом направлении в построении микропроцессоров..

Дальнейшее развитие мультиклеточной обработки связано с расширением понятия «линейный участок» до понятия «линейный фрагмент», который отличается от участка тем, что не все выбранные команды фрагмента обязательно будут исполнены. Это обеспечивается зависимостью потока команд от самого себя и от потока данных, которая позволяет реализовать логические операторы и операторы цикла как целостные языковые конструкции (идеология структурного программирования). Архитектура подобного структурно-ориентированного процессора будет иметь вид – MI(mi,md)MD(mi,md).

4. Заключение

Создание архитектур с хранимым алгоритмом решает целый ряд проблем компьютерной индустрии, которые принципиально не могут быть решены в рамках традиционной фоннеймановской модели или других известных моделей. Это уменьшение сложности и стоимости проектирования процессора, при одновременном повышении его технических характеристик. Устранение семантического разрыва и, как следствие, сокращение затрат на создание как компиляторов, так и программного обеспечения в целом. Обеспечение «естественной» реализации параллелизма и эффективного динамического реконфигурирования процессора.

Комплексное решения указанных проблем позволяет говорить о качественно новом направлении в построении микропроцессоров..

Литература

1. International Technology Roadmap for Semiconductors 2005 Edition. System Drivers [htpp://itrs.net/Links/2005ITRS/SysDrivers2005.pdf].
2. Затуливетер Ю.С., Фищенко Е.А. Многоядерное будущее многопроцессорной архитектуры ПС – 2000 // Труды III Международной научной конференции «Параллельные вычисления и задачи управления» - Москва, 2-4 октября 2006, стр.319-336.
3. Цилькер Б.Я., Орлов С.А. Организация ЭВМ и систем. - СПб,: Питер, 2004.
4. Классификации архитектур вычислительных систем [http://parallel.ru].
5. Flynn M. Very high-speed computing system // Proc. IEEE. 1966. N 54. P.1901-1909.
6. Стрельцов Н.В. Классификация процессорных архитектур // Доклады Международной научной конференции «Суперкомпьютерные системы и их применение» SSA’2004, Минск, 26-28 октября, 2004, стр.67-72.
7. Streltsov N. et al. A Novel MIMD Processor Targeting High Performance Low Power DSP Applications // International Signal Processing Conference, Dallas, 1-3 April, 2003.
8. Стрельцов Н.В. Реализация параллелизма в мультиклеточном процессоре // Труды III Международной научной конференции «Параллельные вычисления и задачи управления» - Москва, 2-4 октября 2006, стр.337-347.