**Иерархия в многопроцессорных системах**

Азин С.Н.

Рассматривается аппаратный механизм управления процессорной матрицей и создание логических внутренних и внешних потоков данных в многопроцессорной структуре.

Описывается вычислительная система (ВС) с распределенной памятью, которая состоит из произвольного количества абсолютно идентичных, примитивных фон Неймановских процессоров (далее элемент) с простейшим набором команд и полной страницей памяти (например, 64кб для 16-разрядного процессора). Система имеет структуру иерархической пирамиды. Один из элементов является основным или ведущим ("император"), он имеет каналы программного, прямого доступа к памяти (ПДП) всех элементов нижнего яруса, они, в свою очередь, имея аналогичные каналы, распределяют между собой управление следующим ярусом и т.д. до самого низа пирамиды, по этим каналам происходит загрузка программ и данных. Имея доступ к регистрам управления подчиненного элемента, командир может обращаться к памяти всех нижних элементов до основания пирамиды, минуя непосредственных командиров. Наличие ПДП дает командиру полную власть над подчиненным, например, чтобы остановить выполнение программы достаточно установить ловушку в вектор прерывания и вызвать его и, заменив ловушку новым указателем, запустить другую программу. Командный элемент может наблюдать за работой программы подчиненного элемента, например, считывая значения указателей буферов или стека и даже помогать ему, взяв часть нагрузки на себя. Система имеет выход наружу канала ПДП в память "императора" для первоначальной загрузки системы, характерно отсутствие какой бы то ни было энергонезависимой памяти, не нужен даже начальный загрузчик, достаточно чтобы все элементы стартовали с команды WAIT. Механизм ПДП должен иметь аппаратную систему сменных приоритетов, чтобы изолировать программное ядро от несанкционированных действий внешней программы. Более того, максимальный приоритет совсем запрещает ПДП (например, у "императора") при этом система становиться недоступной для постороннего вмешательства – вирусов, несанкционированного клонирования и т.п., в этом случае открыть систему можно только "убив" ее выключением питания.

Получился простой и удобный механизм управления любым количеством элементов, но на самом деле это ничего не значит т.к., чтобы система заработала, требуется написать для нее программу и загрузить (считать) данные. Тут встает философский вопрос, почему параллельные вычисления практически ограничиваются конвейерами и матрицами с жестким алгоритмом (сетевые вычислители в расчет не берем т.к. эффективность их близка к нулю)? Причина, по всей видимости, в том, что человек не способен создавать параллельные программы, синхронизировать 3-4 процесса - это предел его возможностей. Идея заключается в том, что программист должен работать не с последовательностью операций, а оперировать взаимосвязями потоков, тогда он может писать хоть на Бейсике или Фортране, подразумевая под переменными - потоки данных. Потоки данных могут быть внешними, например, оцифрованный сигнал с микрофона, видео сигнал или магистральный канал оптического волокна, и внутренними, например, мантисса числа с плавающей запятой. При описании потока достаточно определить два его параметра, тип числа (например, int или float) и его интенсивность (количество байт в одном значении деленное на периодичность в тактах системы). Если, например, заданы два потока А-int и В-long, то при связи B=sinA, интенсивность потока В в два раза больше А. Тут нужна вычислительная система, способная легко создавать достаточное количество таких потоков, устанавливать связи между ними и самостоятельно согласовывать их интенсивности. Для получения множества потоков удобнее всего использовать стробоскопический общий канал (ОК). С помощью установки регистров управления, программируется его цикличность и каждому потоку (и элементу) назначаются тайм-слоты из этого цикла для чтения или записи, как это делается в телекоммуникационных каналах связи. Через один луч канала можно пропускать сколько угодно потоков, все определяется размерами цикла. Ясно, что ОК не может быть бесконечно длинным из-за ограниченной нагрузочной способности. Назовем сегментом элементарный фрагмент пирамиды, состоящий из нескольких элементов с одним командиром и объединенных одним ОК, количество элементов в сегменте равно нагрузочной способности ОК. В идеале сегмент расположен на одном чипе. Для связи с другими сегментами не обязательно выводить ОК наружу сегмента, просто одному или нескольким элементам назначаются функции узлов связи. Узел связи отличается от остальных элементов только тем, что его интерфейсные регистры подключены не к каналам ПДП нижнего яруса и не к множественному потоку данных, а к аналогичным регистрам соседних сегментов. Как соединять, это отдельный вопрос, я думаю лучше всего сверху в низ по пирамиде параллельно ПДП, чтобы диспетчеризацией потоков занимались сегменты верхнего уровня. Тогда пространственное соединение ОК сегментов можно реализовать в виде: параллельных каналов, решетки, звезды, дерева и т.д., в зависимости от требований алгоритма. Задачей узлов связи является сортировка и переадресация потоков. Если применить классификацию Флина, то ОК выполняет функции одиночного потока данных, а элементы - множественного потока команд. Все элементы нижнего яруса пирамиды, кроме узлов связи, имеют свободными выходы своих интерфейсных регистров, к которым подключается множественный двунаправленный поток данных, это любые внешние устройства, архивная память, диски, терминалы, каналы связи и т.д. Есть еще один момент, предположим, нагрузочная способность ОК равна 10, значит в сегменте 10 элементов. Пусть для простоты, за один такт ОК элемент выполняет одну команду и каждому элементу назначено по одному тайм-слоту, значит цикл равен 10 тактам. Получается, что каждый элемент может выполнить цикл только из 10 команд, этого явно недостаточно для большинства задач. Поэтому не имеет смысла гнаться за скоростью канала, кстати, при уменьшении скорости ОК увеличивается его нагрузочная способность, тут есть оптимум.

Согласовать скорости программных процедур и потоков, задача не реальная, поэтому между ними в каждом элементе должны быть эластичные кольцевые буфера типа "очередь". Достаточно двух таких буферов, один для всех входных потоков и один для выходных, они занимают всю свободную память элемента. Данные в буфере однозначно привязаны к данным канала, поэтому нет необходимости предпринимать какие-то действия для их идентификации, но при переходе в другой сегмент, это не очевидно. Эта проблема лежит на совести программистов, которые будут реализовывать ядро системы. Можно в лоб пересчитать индексы при переходе, можно снабдить потоки синхронизирующими метками, а можно индексы передавать в потоке вместе с данными.

Принципиальной и специфической особенностью данной системы реального времени является то, что она должна работать быстрее поступления данных, иначе произойдет переполнение буферов и сбой. Эту ситуацию можно обнаружить заранее, если командир обнаружит накопление данных в буфере одного из элементов (это может быть прерывание по переходу красной зоны), у него есть некоторое время, чтобы избежать сбоя. В этом случае необходимо либо останавливать задачу и перекомпоновывать систему, либо решать проблему в ходе выполнения, например путем подключения резервных элементов.

Скорости потоков данных, в принципе, ограничены только физическими возможностями канала, поскольку всегда можно организовать стробоскоп из нескольких элементов, которые будут по очереди брать (выставлять) данные с канала. Увеличивая количество элементов в стробоскопе легко достичь предельных скоростей. Более того, если канал не удовлетворяет требованиям по скорости, достаточно сделать стробоскопический коммутатор и распределить поток между параллельными структурами. Этот способ применим также для оптимизации обработки потоков, т.е., чтобы не простаивали короткие процедуры, длинные запускаются через стробоскоп, стробоскоп может быть временным, т.е. имея несколько резервных элементов, можно подключать их к отстающим процедурам в ходе выполнения. Имеется несколько вариантов организации конвейеров типа конвейера арифметики плавающей.

Пирамида ВС не обязательно должна быть правильной формы, и может иметь ответвления произвольной высоты и ширины (даже удаленные, например, по оптическому волокну) и вообще может быть спроектирована с учетом выполняемой задачи, здесь открывается большой простор для фантазии, например, N+1 мерная пирамида с N ортогональными ОК. Уровень интеллектуальных возможностей системы зависит от высоты пирамиды, а производительность от ширины, т.к. ядро работает в верхних слоях, а основная задача в основании. Ядро занимается загрузкой, планированием, разбором конфликтов подчиненных и статистикой их загруженности. В задачу ядра входит также тестирование ВС – определение конфигурации конкретной системы и обнаружение и изоляция дефектных элементов. Ядро ВС, в отличие от UNIX, работает в фоновом режиме, кроме чисто командных элементов, в которых основная задача вообще не выполняется.

Очевидно, что работа в реальном времени сильно экономит оперативную память. Когда обрабатывается начало большого сегмента данных, конец может вообще не понадобиться и грузить его в свопинг или память, по крайней мере, не экономно. Аналогично с приложениями, если приложение выполнило львиную долю своей работы и крутиться в малом цикле упорядочивания данных, большая часть программного кода уже (или еще) не нужна и, как результат, никому не нужные страницы гуляют по системе между КЭШем, памятью, свопингом и файловой системой. Одноуровневая память AS/400 пытается полумерами частично решить эти проблемы, но зачем для этого городить огород из 64-х разрядного адреса. В нашем случае эта ситуация отслеживается полностью, поскольку все процедуры активны, находятся под наблюдением ядра и всегда могут отрапортовать – “Задание выполнено, жду указаний или перезагрузки”. В некотором смысле, разбиение приложений на процедуры аналогично разбиению сегмента на страницы или файла на кластеры. Кроме того, нужно заметить, что многозадачные операционные системы типа UNIX, которые неплохо справляются с проблемами баз данных и секретарей, являются тормозом для целого ряда весьма актуальных задач - задачи управления в реальном масштабе времени, задачи искусственного интеллекта, машинного зрения и т.д.

В заключении хочу подчеркнуть, что система команд элемента должна быть максимально простой, фактически это интеллектуальная страница памяти. Никаких команд умножения и деления, тем более плавающей арифметики не требуется, поскольку ничего не стоит достичь любой скорости вычислений простой перекомпоновкой элементов, к тому же выгодно комбинировать сразу весь комплекс вычислений, а не отдельные арифметические действия. При этом получается большая экономия в операциях, например, достаточно одной операции нормировки на весь комплекс, при этом устраняется накопленная ошибка. К тому же вы можете работать с любым числом плавающей запятой, а не с числом процессора плавающей арифметики. Ориентация на режим реального времени накладывает специфические требования на конструкцию процессорного элемента, например, бесполезность стека и вложенных прерываний, вместо стека целесообразно иметь аппаратные эластичные буфера. Имеются некоторые соображений по поводу системы команд и регистровой архитектуры элемента с двумя стробоскопическими ОК, селектированым каналом ПДП с защитой по приоритетам и двумя эластичными, кольцевыми буферами вместо аппаратного стека, но это тема отдельного разговора.