Способ организации персистентной кэш памяти для многозадачных, в том числе симметричных многопроцессорных компьютерных систем.

Борис Муратшин (zzeng@mail.ru),
Александр Артюшин (alexnikart@mail.ru)

Декабрь 7, 2005

В идеале память должна обеспечивать процессор данными таким образом, чтобы избежать простоев в ожидании данных. К сожалению, не придуманы еще методы компиляции, способные обеспечить надлежащий транспорт данных. В современных вычислительных системах уменьшение времени доступа достигается введением иерархического устройства памяти, когда каждый следующий уровень памяти больше и медленнее предшествующего, при этом адресуемым является лишь самый верхний уровень, а остальные запоминают в соответствии с некоторой стратегией последние обращения к данным и способны выдавать их быстрее, чем это делают вышестоящие уровни. Размер кэш памяти первого уровня типично равен 8K слов (у HP8500 - 1.5. мегабайта), второго и третьего (если есть) - значительно больше.

Когда инструкция в качестве операнда содержит виртуальный адрес памяти, этот адрес преобразуется в физический. По физическому адресу запрашиваются данные из памяти, если такие данные уже содержатся в кэш памяти, процессор получает их быстрее. Зададимся вопросом, почему кэш работает с физическими адресами? Проблема в том, что адресные пространства, предоставляемые современными операционными системами задачам, могут пересекаться. Можно было бы расширить виртуальный адрес с помощью идентификатора задачи и преобразовывать адрес в физический лишь после неуспеха поиска в кэш памяти, но это приведет к фактическому удвоению тега строк кэш памяти и существенному усложнению устройства кэш памяти всего лишь при небольшой экономии в преобразованиях адресов. Кроме того, преобразование адреса выполняется процессором, тогда как старшие уровни кэш памяти обычно из процессора вынесены.

Недостатки существующего подхода очевидны и большей частью заключаются в совместном использовании кэш памяти всеми выполняющимися задачами. При любой стратегии вытеснения старых данных, после потери задачей управления и его возврате после работы десятка других задач, личное содержи-

мое кэш памяти можно считать потерянным, и приходится начинать собирать его заново, возникает так называемая проблема холодного старта. При нормальном числе одновременно работающих задач, измеряемом десятками или даже сотнями, для того, чтобы сохранить хоть часть личных данных, приходится делать кэш память чудовищных размеров. Еще одна проблема связана со свопингом. Если при преобразовании адреса оказывается, что страница с таким виртуальным адресом отсутствует в физической памяти, она будет подгружена, при этом вытеснив какую-то другую. Очевидно, все линии кэш памяти, ссылающиеся на предыдущую страницу, должны быть аннулированы.

Преимущества существующего подхода иногда оказываются спорными. Ускоряется среднее время доступа к памяти, особенно при последовательном и локализованном обращении. При любом отличном от этого поведении, программа не должна рассчитывать на помощь со стороны кэш памяти. При этом иногда такое "среднее"поведение кэш памяти играет с программами дурную шутку, теряется смысл заботиться о "красоте"алгоритмов и пытаться оптимизировать программу вручную - кэш память всех подстрижет под одну гребенку, и, часто, "плохо"написанный код в результате работает быстрее "хорошего", что, согласимся, не очень правильно. Попытки же использовать особенности кэш памяти при оптимизации программ приводят к потере переносимости кода даже между версиями одного процессора.

Предпринимаются попытки сделать поведение кэш памяти более предсказуемым, например, в виде загрузки данных по предположению. Так, компилятор добавляет команды загрузки данных в кэш память до того, как они могут реально понадобиться (SPARC V9, IBM POWER3 и HP PA-8xxx). Не менее полезной является возможность принудительно инвалидировать линию кэш памяти, если ее данные заведомо не пригодятся в ближайшем будущем. Некоторые системы (TMS320C6xxx) позволяют конфигурировать кэш память как прямо адресуемую сверхоперативную память, но в такой ситуации возникают существенные трудности с ее совместным использованием различными задачами.

Отдельных слов заслуживает проблема поддержания идентичности (когерентности) кэш памяти при совместном использовании памяти несколькими процессорами. Поскольку когерентность поддерживается с помощью межмодульных пересылок, а количество пересылок существенно не линейно по отношению к числу процессоров в системе, объективно существует предел сверху числа процессоров. Отметим, что большая часть упомянутых пересылок является бесполезной т.к. в дальнейшем не будет использована, но это издержки кэширования как такового и, частично, плата за кэширование в физических адресах, порождающее излишние зависимости между процессорными модулями. И если бы при прочих равных нам удалось избавиться от заведомо бесполезных пересылок, это позволило бы существенно поднять предел числа процессоров.

Теперь обратимся к сути данного изобретения. Основными идеями являются:

1. разделение данных на глобальные и локальные, которое может быть сделано на основании значения виртуального адреса

2. кэширование в виртуальном адресном пространстве

3. ситуация, когда каждая задача работает со своей кэш памятью, при этом в отношении локальных данных постороннее влияние исключено,

4. буфер данных кэш памяти разбит на две равные страницы, при смене задач происходит переключение страниц, причем новая задача немедленно продолжает выполнение с одной страницей, а из второй страницы в фоновом режиме выгружаются старые данные и загружаются новые

5. глобальные данные кэшируются с тегом, расширенным идентификатором, предоставляемым операционной системой, например, номером процесса текущей задачи

Пусть вычислительная система содержит не менее одного процессора. Пусть вычислительная система предназначена для параллельного выполнения ряда процессов, каждый из которых может содержать более одной параллельно выполняющейся задачи (thread). Данные, доступ к которым имеет лишь одна задача, мы будем называть локальными, а данные, к которым имеют доступ все задачи некоторого процесса, глобальными. Разделение данных между процессами допускают, но требует особой обработки, например, такие данные размещают в некэшируемые сегменты основной памяти, не подвергают свопированию или доступ к ним осуществляют через системные вызовы ...

Загрузчик операционной системы, предназначенный для создания образа программы в основной памяти, размещает глобальные данные процесса в сегменты памяти с легко идентифицируемыми виртуальными адресами, например, результат операции битового '&' должен быть истинным при сочетании адреса с некоторой битовой маской, которая может быть задана как аппаратно, так и программно. Ядро операционной системы, далее называемое как ядро, ответственно за планирование задач по процессорам и их переключение.

Каждый процессор в вычислительной системе содержит одинаковую по размеру и одинаково устроенную локальную кэш память и аппаратуру для работы с ней, в дальнейшем именуемую как кэш. Кэш состоит из ряда логических блоков, как то:

1. две равные по размеру страницы данных, одна из которых в дальнейшем будет называться активной, другая теневой (пассивной)

2. блок доступа к страницам данных

3. блок управления глобальными данными

4. страницу глобальных данных

5. не менее одного канала прямого доступа к памяти (DMA)

В зависимости от реализации, кэши данных и команд могут быть расположены как на одной (как на активной, так и на теневой) странице, используя общие пути данных, так и на разных, полностью или частично дублируя весь описываемый механизм кэш памяти. Блок доступа к страницам данных реализует одну из

известных стратегий и алгоритм кэширования, например, он может быть полностью ассоциативным со сквозной буферизованной записью и LRU вытеснением. Блок управления глобальными данными предназначен для поддержания их когерентности как внутри процессорного модуля вычислительной системы, так и в масштабах всей вычислительной системы. Страницу глобальных данных разделяют между всеми задачами, но эта страница не обязана совпадать по размеру с локальными страницами и используется непосредственно блоком управления глобальными данными в качестве ассоциативной памяти. Каналы прямого доступа к памяти предназначают для копирования данных из кэш памяти и обратно в кэш, причем каждый из имеющихся каналов работает независимо, в каждый конкретный момент со своей и только своей областью теневой страницы кэш памяти.

Активную страницу кэш памяти выделяют для использования активной в данный момент задачей, причем содержимое этой страницы является контекстом кэш памяти этой задачи. Ядро содержит механизм сохранения контекста кэш памяти предыдущей задачи из теневой страницы и загрузки контекста кэш памяти следующей задачи туда же. В момент смены активной задачи ядром производят обращение страниц кэш памяти, т.е. активная страница становится теневой, а теневая - активной, для этого используют флаг из управляющего регистра процессора, инверсия значения которого приводит к соответствующему изменению работы кэш памяти, значение этого флага может меняться как аппаратным, так и программным путем . К этому моменту теневая страница должна содержать полностью загруженный ядром контекст кэш памяти новой активной задачи. Сразу после смены активной задачи ядром, начинают сохранять контекст кэш памяти из новой теневой страницы. После того, как ядро выберет следующую задачу для исполнения на данном процессоре, ее начинают загружать в теневую страницу контекст кэш памяти этой новой задачи. Загрузку и выгрузку содержимого теневой страницы имеют возможность производить одновременно со сдвигом по времени, достаточным для исключения их столкновения.

При занесении линии в страницу глобальных данных в качестве тега используют виртуальный адрес, расширенный предоставляемым операционной системой идентификатором, например, номером процесса текущей задачи, что делает этот виртуальный адрес уникальным в масштабах всей операционной системы. Идентификатор берут из выделенного регистра процессора, задаваемого операционной системой. В момент запроса процессором глобальных данных, отсутствующих в странице глобальных данных, производят их выборку из основной памяти, размещение в активной странице и регистрацию в глобальной странице. В случае, когда глобальная страница переполнена, действия зависят от реализации, например, данному значению отказывают в кэшировании или вытесняют им другое глобальное значение одновременно из активной и глобальной страниц, важно лишь, чтобы все глобальные данные локальных страниц были синхронны со значениями из глобальной страницы. В процессе загрузки кэшем данных блок управления глобальными данными пропускает без изменения локальные данные и приводит глобальные в соответствие со значениями из глобальной страницы. В случае многопроцессорной системы блок управления глобальными данными реализует некоторый протокол поддержки распределенных транзакций для под-

держания общесистемной когерентности кэш памяти. Совпадает ли содержимое глобальных страниц кэш памяти разных процессорных модулей, зависит от реализации протокола распределенных транзакций.

Контекст кэш памяти каждой задачи хранят в структуре ядра, описывающей поведение данной задачи.

Деятельность кэш памяти может быть приостанавливаю изменением значения еще одного флага. Деятельность кэш памяти в отношении глобальных данных приостанавливают изменением значения еще одного флага. При возникновении в процессоре аппаратного прерывания, например в случае отсутствия нужной страницы в физической памяти, приостанавливают деятельность кэша вплоть до выхода этого процессора из режима прерывания.

Активную страницу выполняют дополнительно имеющей режим прямой адресации. Таким образом, каждой задаче при желании вместо буфера кэш памяти (или его части) выделяют собственную область сверхоперативной памяти.

Что дает нам такая организация? Первое, сверхоперативная память может иметь либо собственное адресное пространство и собственные инструкции для доступа, либо иметь выделенную в виртуальном адресном пространстве область адресов. В любом случае, наличие такой памяти никак не учитывается в языках высокого уровня. Одним из объективных методов ее использования можно считать использование такой памяти компилятором для хранения временных переменных, возникающих при генерации кода. Увы, при генерации кода никогда не возникает потребности хранить тысячи временных переменных, поэтому необходимы области применения оставшейся сверхоперативной памяти. Существует как минимум три варианта.

1. Размещение в этой памяти стека. Действительно, в стеке оказываются локальные переменные (и временные в том числе), принадлежащие только данной задаче (в традиционной схеме одна задача может передать другой адрес переменной из своего стека, но подобная техника является скорее ошибкой проектирования и создает больше проблем, чем их решает), эти данные сильно локализованы, кэширование вершины стека всегда актуально, объем актуальной для кэширования вершины стека (объем локальных данных в последних 2-3 вызовах процедур) обычно не очень велик. При переполнении стека происходит исключение, операционная система выделяет дополнительный буфер для сохранения при переключении задачи и сдвигает данные так, что буфер кэш памяти выступает в роли окна ускоренного доступа. Если происходит обращение за пределы этого окна, то при преобразовании виртуального адреса в физический, выдается адрес, соответствующий нужному месту в образе стека данной задачи в основной памяти.

2. Явное обращении за этой памятью к операционной системе и ручное управлением ее содержимым.

3. Внесение изменений в языки высокого уровня и/или разработку новых методов компиляции с учетом особенностей архитектуры. Особенно актуально это для архитектур с явным параллелизмом уровня команд (ILP),

где кэширование как таковое может оказаться недопустимым т.к. делает поведение системы непредсказуемым.

Второе, явное разделение данных на глобальные и локальные позволяет программисту явно контролировать межпроцессорные пересылки, необходимые для поддержания когерентности системы. Возможен весь спектр, если мы запрещаем кэширование глобальных данных, синхронизация осуществляется на уровне доступа к физической памяти - не такой уж плохой вариант, если таких данных мало, а если глобальных данных вообще нет, то поддержка когерентности как таковая просто не нужна. Если же все данные глобальные, мы имеем примерно ту же ситуацию, что и в традиционной системе с кэшированием физических адресов. Здесь важно, чтобы программист всегда отдавал себе отчет, с какими именно данными он работает, и имел возможность выбора, например, в виде двух системных вызовов выделения памяти - локальной и разделяемой.

Третье, в сохраняемый контекст задачи могут быть включены и некоторые дополнительные данные как, например, таблица историй переходов и кэш физических адресов (TLB). Некоторые системы хранят таблицу историй переходов вместе с кэшем инструкций, в любом случае наличие такой личной таблицы позволит без особых издержек (в MIPS R10000 такая таблица содержит 512 элементов) повысить вероятность правильного предсказания перехода для спекулятивного выполнения. Типичный размер TLB - сотни записей. Его также не составит труда разместить в сохраняемом контексте задачи с тем лишь отличием, что при подкачке страниц операционная система должна следить за тем, чтобы содержимое TLB выгруженных и неактивных в данный момент задач продолжало соответствовать действительности.

Список литературы

[1] Б. Муратшин, А. Артюшин. Способ организации персистентной кэш памяти для многозадачных, в том числе симметричных многопроцессорных компьютерных систем и устройство для его осуществления. Новосибирск, Июль 2002, патент на изобретение N2238584.