**Доклад**

**На тему: Программное кэширование**

**Содержание**

1. Введение.
2. Программная предвыборка в процессорах К6+ и РIII+.
3. Предвыборка в процессорах AMD К6 и VIA C3.
4. Предвыборка в процессорах РIIIи Р4.
5. Pentium III.
6. Pentium 4.
7. Эффективность предвыборки в многозадачных системах.

**Введение**

Программному управлению кэшированием просто не повезло. Концепция "прозрачного" кэша, активно продвигаемая фирмой Intel, Абстрагировала программистов от потребностей аппаратной реализации кэш-контроллера и не предоставила им никаких рычагов управления последним. Впрочем, для достижения полной абстракции интеллектуальности кэш-контроллеров все же не хватило, и для системных программистов пришлось крохотную лазейку, позволив им, в частности, запрещать кэширование страниц памяти, принадлежащих периферийным устройствам.

До тех пор, пока большинство приложений перемалывало компактные, многократно обрабатываемые структуры данных, стратегия загрузки кэш-линеек по первому требованию вполне справлялась со своей задачей, но с появлением мультимедийных приложений стала "буксовать". Резко возросший объем обрабатываемых данных и распространение потоковых алгоритмов, обращающихся к каждой ячейке памяти лишь единожды, обернулся постоянными перезагрузками кэша, что ограничило производительность системы не быстродействием процессора, а пропускной способностью оперативной памяти. Впервые этой проблеме бросила вызов фирма AMD, включив в состав набора команд 3D Now! инструкцию prefetch, позволяющую программисту заблаговременно загружать в кэш ячейки памяти, к которым он расчитывает обратится в ближайшем будущем. Причем загрузка данных осуществляется без участия и остановки вычислительного конвейера! Это убивает двух зайцев сразу: во-первых, ручное управление кэш-контроллером позволяет выбрать оптимальную стратегию упреждающей загрузки данных, что существенно уменьшает количество кэш-промахов, а, во-вторых, с предвыборкой становится возможным загружать очередную порцию данных параллельно с обработкой предыдущей, маскируя тем самым латентность оперативной памяти.

Следом за К6, предвыборка (естественно в усовершенствованном варианте) появилась и в Pentium lll, да не одна, а с целой свитой команд ручного управления кэшированием - Intel явно не хотела отставать от конкурентов!

Совершенствование управления подсистемной памяти продолжилось и в Pentium 4. Помимо расширения набора команд, в нем реализован уникальный на сегодняшний день механизм аппаратной предвыборки с интеллектуальным алгоритмом упреждающей загрузки. Анализируя порядок, в котором приложение запрашивает данные из оперативной памяти, процессор пытается предсказать (приблизительно так же, как предсказывает направление условных переходов) адрес следующей обрабатываемой ячейки, чтобы спекулятивно загрузить ее в кэш задолго до реального затребования. Естественно, при всей прозрачности аппаратной предвыборки структуры данных желательно сделать так, чтобы процессор пореже ошибался в своих предсказаниях, а в идеале - не ошибался вообще.

**Программная предвыборка в процессорах К6+ и РIII+**

Поддержка программной предвыборки имеется как в К6/К7 (и совместимом с К6 микропроцессоре VIA C3), так и в Р!!!/Р4, однако, их реализации различны и к тому же не совместимы друг с другом. Процессоры от AMD(VIA) не понимают инструкций предвыборки процессоров Intel и, соответственно, наоборот.

Это печальное обстоятельство существенно снижает популярность предвыборки, поскольку программистам приходится либо писать два варианта кода: один для Intel, другой для AMD(VIA); либо ограничивать аудиторию пользователей одним из процессоров.

Поэтому к предвыборке целесообразно прибегать лишь в действительно крайних случаях, когда никакими другими путями обеспечить требуемое быстродействие уже не удается.

**Предвыборка в процессорах AMD К6 и VIA C3**

В К6/К7 и VIA C3 программная предвыборка осуществляется одной из двух инструкций: prefetch или prefetchw. Буква w в конце последней сообщает процессору, что загружаемые данные планируется модифицировать. Это отнюдь не означает, что данные, загружаемые посредством prefetch, модифицировать нельзя. Модифицировать их можно, но не желательно, т. к. в этом случае процессор вынужден совершать дополнительный цикл, изменяя атрибуты соответствующей кэш-линейки с эксклюзивной на модифицируемую.

Инструкция prefetch просто инициирует запрос ячейки памяти, точно также, как это делает любая команда, обращающаяся к памяти, но, в отличие от последней, prefetch не помещает загружаемые данные ни в какой регистр, более того, она вообще не дожидается конца загрузки этих данных, тут же возвращая управление. Преждевременное завершение инициатора запроса еще не освобождает кэш-контроллер от обязанности выполнения этого запроса, но если запрошенная ячейка уже находится в кэше первого уровня, ничего не происходит и инструкция prefetch ведет себя аналогично команде NOP (нет операции). В противном случае кэш-контроллер обращается к кэшу второго уровня, а если искомой ячейки не оказывается и там - к оперативной памяти (кэшу третьего уровня), целиком заполняя соответствующие кэш-строки кэшей всех нижестоящих уровней. Поскольку кэш-контроллер работает независимо от вычислительного конвейера процессора, предвыборка позволяет загружать очередную порцию данных параллельно с обработкой предыдущей. Если время загрузки данных не превышает времени их обработки, то простоя процессора вообще не происходит - вычислительны конвейер работает безостановочно, а время доступа к памяти полностью маскируется.

Инструкция prefetchw работает аналогично prefetch, но автоматически присваивает загружаемой ячейке статус модифицируемой. Если строку действительно планируется модифицировать, это экономит 15-25 тактов процессорного времени. Однако, если вы не уверены, будет ли строка меняется, лучше загрузите ее как эксклюзивную, т. к. выгрузка модифицируемой, но реально не модифицированной строки в оперативную память обойдется намного дороже.

Модифицируемые же кэш-строки независимо от того, были ли они реально модифицированы или нет, всегда вытесняются в оперативную память или кэш вышестоящего уровня, что требует определенного количества тактов процессора.

Несмотря на то, что AMD позиционирует команды предвыборки как аппаратно-независимые, они таковыми не являются, поскольку количество байт, загружаемых инструкциями prefetch и prefetchw, определяются размерами кэш-линий процессора, а их длинна различна: 32 байта для AMD K6 (VIA C3) и 64 байта для Athlon/Duron. Соответственно, различны оптимальный шаг и минимальная дистанция предвыборки.

В этом свете становится очень интересным следующее высказывание AMD, почерпнутое из руководства по оптимизации под Athlon: "Инструкции PREFETCHNTA/T0/T1/T2 из ММХ-расширения аппаратно зависимы. Если вы, господин разработчик, нуждаетесь в совместимости с 25 миллионами уже проданных процессоров AMD-K6-2 и AMD-K6-III, вместо инструкций предвыборки нового расширения ММХ, пользуйтесь командами PREFETCH/W из расширения 3Dnow!"

Вот хорошая демонстрация искусства умолчания! Если уж бросать камень в огород Intel, то не лишнее бы отметить, что, во-первых, и собственные инструкции предвыборки аппаратно-зависимы, а, во-вторых, процессорами Pentium они оно не поддерживаются. Так что никаких преимуществ у AMD`шной предвыборки перед Intel нет.

**Предвыборка в процессорах Р!!! и Р4**

В процессорах Р!!! и Р4 программная предвыборка осуществляется следующими инструкциями : prefetchnta, prefetcht0, prefetcht1, prefetcht2. Суффикс указывает на тип загружаемых данных, что определяет уровень кэш-иерархии, в которую эти данные будут загружены. Так NTA расшифровывается как Non-TemporAl [Data] - не временные данные, т.е. данные, многократное использование которых планируется. Соответственно Т0, Т1, Т2 обозначает временные данные, использовать которые планируется неоднократно.

Какой бы командой предвыборка ни осуществлялась, кэш-линейкам, загружаемым из основной памяти, всегда присваивается эксклюзивный статус. При предвыборке линеек из кэша второго уровня их прежний статус сохраняется. Возможность загрузки кэш-линейки с автоматической установкой статуса модифицируемой в процессорах Pentium не реализована. Однако ввиду многоступенчатой схемы буферизации записи, изменение атрибутов кэш-линеек происходит в основном, а не в дополнительном, как в К6/Athlon, цикле обмена, т.е. без ущерба для производительности.

Причем в отличии отprefetch/w,инструкции prefetchnta/t0/t1/t2 не приказывают, а рекомендуют осуществить предвыборку. Процессор отклоняет рекомендацию и не осуществляет предвыборку, если :

1. запрошенные данные уже содержатся в кэше соответствующей или ближайшей к процессору иерархии;
2. сведения о странице, к которой принадлежат загружаемые данные, отсутствуют в DTLB (Data Translation Look aside Buffer - Буфере Ассоциативной Трансляции;)
3. подсистема памяти процессора занята перемещением данных между L1- и L2- кэшем;
4. запрошенные данные принадлежат региону некэшируемой памяти(странице с атрибутами UC и USWC);
5. данные не могут быть загружены из-за ошибки доступа (при этом исключение не вырабатывается);
6. инструкция предвыборки предваряется префиксом LOCK (в этом случае генерируется исключение "неверный опкод");

Во c остальных случаях предвыборка выполняется. Алгоритм ее выполнения аппаратно-зависим и сильно варьируется от одной модели процессора к другой, поэтому, поведение "предвыборных" команд на Р!!! и Р4 ниже мы рассмотрим по отдельности

# **Pentium lll**

Инструкция prefetchnta загружает данные в кэш первого уровня, минуя второй. Действительно, данные, повторное обращение к которым не планируется, целесообразно помещать в кэш самой ближайшей к процессору иерархии, не затирая содержимое остальных, т.к. оно может еще пригодится, а вот однократно используемые данные после их вытеснения из L1-кэша, из L2-кэша затребованы уж точно не будут. Инструкция prefetcht0 загружает данные в кэш-иерархии обоих уровней. Данные, обращение к которым происходит многократно, будучи загруженными в L2-кэш, окажутся как нельзя кстати, когда будут вытесненными из L1-кэша.

Инструкции prefetcht1 и prefetcht2 загружают данные в один лишь кэш второго уровня, не помещая их в кэш первого. Поскольку выгрузка буферов записи происходит в кэш второго уровня, минуя первый, то предвыборку соответствующих линеек в L1-кэш осуществлять нецелесообразно. Вот тут-то и пригодится prefetcht1/t2!

Размер загружаемых данных равен длине кэш-линеек соответствующей кэш-иерархии и составляют 32 байта ( в Plll и L2-, и L1- кэши имеют линейки по 32 байта).

# **Pentium 4**

Ни одна из команд предвыборки Р4 не позволяет загружать данные в кэш первого уровня. Все – и временные, и не временные данные помещаются лишь в кэш второго уровня – создатели процессора решили поступить именно так. Эффективность такой стратегии не бесспорна, но в любом случае время доступа к кэшу второго уровня намного меньше времени доступа к оперативной памяти, поэтому даже такая предвыборка значительно лучше, чем ничего.

Возникает вопрос: если все команды предвыборки помещают загружаемые данные в кэш второго уровня, то какая между ними разница? Между командами prefetcht0, prefetcht1 и prefetcht2 – действительно никакой. А вот команда prefetchnta отличается тем, что помещает загружаемые данные не в любой, а исключительно в первый банк кэша второго уровня (восьми-ассоциативный L2-кэш Р4 содержит восемь таких банков), благодаря чему prefetchnta никогда не вытесняет более 1/8 объема кэша второго уровня. Однократно используемы данные, как уже говорилось выше, действительно не должны вытеснять многократно используемые данные из верхних кэш-иерархий, но в Р4 такое вытеснение все же происходит, и предотвратить его, увы, нельзя. Причем вытесняются отнюдь не те ячейки, к которым дольше всего не было обращений, а линейки фиксированного банка, возможно интенсивно используемые обрабатывающем их приложением! Словом, в Р4 программная предвыборка реализована далеко не наилучшим образом.

Размер загружаемых данных равен длине линеек кэша второго уровня, что составляет 128 байт.

Различия в реализации предвыборки на Р!!! и Р4 существенно затрудняют оптимизацию приложений, поскольку каждый процессор требует к себе индивидуального подхода. Для достижения максимальной эффективности все критические процедуры рекомендуется реализовывать как минимум в двух вариантах – отдельно для Р!!! и отдельно для Р4. В противном случае, либо Р!!! чрезвычайно тормозить, либо Р4 не раскроет подлинного потенциала своей производительности. Учитывая существование К6/Athlon, вариантов реализации набирается уже четыре. Не слишком ли много головной боли для программистов? Нет, это вовсе не призыв к отказу от предвыборки – ведь программисты, как и комсомольцы, легкими путями не избалованы.

**Эффективность предвыборки в многозадачных системах**

Процессы, исполняющиеся в многозадачных системах, владеют кэш-памятью не единолично, а вынуждены делить ее между собой? Снижает ли это эффективность предвыборки? Эффективность предвыборки в кэш первого уровня – однозначно нет. Промежуток времени между переключениями задач – это целая вечность для процессора, соответствующая, по меньшей мере, миллионам тактов. В любом случае, независимо от того, будет ли вытеснено содержимое L1-кэша или нет, - предвыборка позволяет конвейеризировать загрузку данных из памяти, предотвращая тем самым возможное падение производительности.

С L2-кэшем ситуация не так однозначна. Если оптимизируемый алгоритм позволяет распараллелить загрузку данных с их обработкой, то состояние L2-кэша вообще не играет никакой роли, поскольку быстродействие программы ограничивается именно скоростью вычислений, а не пропускной способностью подсистемной памяти. Однако если время обработки данных меньше времени их загрузки из основной памяти, падения производительности никак не избежать. Предвыборка, конечно, увеличит производительность программы и в этом случае, но, увы, ненамного – максимум в два-три раза.

С другой стороны, одновременное выполнение двух или более приложений, интенсивно обменивающихся с памятью, на рабочих станциях случается очень редко (для серверов, правда, это – норма жизни). В большинстве случаев пользователь активно работает лишь с одним приложением, другие же находятся в фоне и довольствуются минимальным количеством памяти, а порой и вовсе «спят», не трогая L2-кэш и практически не снижая эффективности предвыборки.