11. Ассоциативная память. Аппаратная и программная реализации памяти с ассоциативным доступом. Области применения, ограничения, примеры реализации.

Теория

Ассоциативное ЗУ, АЗУ (англ. associative memory, content-addressable memory, CAM) — вид памяти, в котором адресация осуществляется на основе содержания данных, а не их местоположения, чем обеспечивается ускорение поиска необходимых записей.

АЗУ полезны и нужны для аппаратного ускорения поиска. Ведь любая задача поиска в конечном результате сводится к нахождению адреса ячейки памяти (для ускорения этого процесса данные сортируют, создают индексы и т.д.).

Логически АЗУ можно реализовать с помощью нейронной сети с обратной связью (например). Такая нейронная сеть будет выдавать нам нужные данные если на вход ей подать часть данных, либо зашумленные данные (используют для распознавания текстов).

Аппаратный ассоциативный массив

В отличие от обычной машинной памяти (памяти произвольного доступа, или RAM), в которой пользователь задает адрес памяти и ОЗУ возвращает слово данных, хранящееся по этому адресу, АП разработана таким образом, чтобы пользователь задавал слово данных, и АП осуществляла его поиск, чтобы выяснить, хранится ли оно где-либо в памяти. Если слово данных найдено, АП возвращает список одного или более адресов хранения, где слово было найдено (и в некоторых архитектурах также возвращает само слово данных или другие связанные части данных). Таким образом, АП — аппаратная реализация того, что в терминах программирования назвали бы ассоциативным массивом.

Практика (для более точного понимания)

Итак, допустим нам нужно поставить в соответствие две последовательности бит (возьмем 4-х разрядные числа для простоты):

1101 > 1000

1001 > 1100

Строим для каждого соответствия матрицу:

Метод построения предельно прост. Левый столбец — это первое значение, нижняя строка — это соответствие. На пересечениях мы выполняем логическую операцию AND текущих битов.

Матрицы соответсвий

1101 -> 1000

1	1	0	0	0
1	1	0	0	0
0	0	0	0	0
1	1	0	0	0
	1	0	0	0

1001 -> 1100

1	1	1	0	0
0	0	0	0	0
0	0	0	0	0
1	1	1	0	0
	1	1	0	0

Теперь объединим матрицы операцией OR (Rezult[i, j] = A1[i, j] OR A2[i, j]).

Ассоциативная матрица

1	1	0	0
1	0	0	0
0	0	0	0
1	1	0	0

Эта ассоциативная матрица(АМ) хранит наши два соответствия 1101 > 1000 и 1001 > 1100.

Теперь извлечем данные. Например, нам нужно найти чему соответствует 1001.

Запишем 1001 слева от АМ, и логически умножим получившийся столбец на каждый столбец АМ. В полученной матрице, просуммируем столбцы и запишем результаты в нижней строке.

Извлечение данных

1	1	1	0	0
0	0	0	0	0
0	0	0	0	0
1	1	1	0	0
	2	2	0	0

В полученной строке (2200) найдем наибольшее число (в нашем примере это будет 2) и в соответствующих битах запишем 1, а в остальных — 0. Таким образом получили число 1100. Именно это значение мы и записывали в матрицу.

Также можно получить обратное соответствие, если 1100 записать снизу и умножить на каждую строку, то по аналогии найдем в столбце исходное соответствие.

В примерах выше, мы работали с четырехбитными числами. Но самое интересное то, что данный подход можно реализовать в виде N-мерной матрицы и хранить (N-1) мерные данные. Например можно сделать 3-х мерную матрицу, которая будет помнить соответствия 2-х мерных черно-белых картинок. И самое главное — такие матрицы легко реализовать в железе.

Реализация на полупроводниках

Из-за того, что АП разработана, чтобы искать во всей памяти одной операцией, это получается намного быстрее, чем поиск в RAM фактически во всех приложениях поиска. Однако, есть и минус в большей стоимости АП. В отличие от чипа RAM, у которого хранилища простые, у каждого отдельного бита памяти в полностью параллельной АП должна быть собственная присоединенная схема сравнения, чтобы обнаружить совпадение между сохраненным битом и входным битом. К тому же, выходы сравнений от каждой ячейки в слове данных должны быть объединены, чтобы привести к полному результату сравнения слова данных. Дополнительная схема увеличивает физический размер чипа АП, что увеличивает стоимость производства. Дополнительная схема также увеличивает рассеиваемую мощность, так как все схемы сравнений активны на каждом такте. Как следствие, АП используется только в специализированных приложениях, где скорость поиска не может быть достигнута, используя другие, менее дорогостоящие, методы.

Альтернативные реализации

Для того, чтобы достигнуть другого баланса между скоростью, размером памяти и стоимостью, некоторые реализации эмулируют функции АП путём использования стандартного поиска по дереву или алгоритмов хеширования, реализованных аппаратно, также используя для ускорения эффективной работы такие аппаратные трюки, как репликация и конвейерная обработка. Эти проекты часто используются в маршрутизаторах.

Примеры приложений

Адресуемая содержанием память часто используется в компьютерных сетевых устройствах. Например, когда сетевой коммутатор (switch) получает фрейм данных на один из его портов, это обновляет внутреннюю таблицу с источником MAC-адреса фрейма и порта, на который он был получен. Потом он ищет MAC-адрес назначения в таблице, чтобы определить, на какой порт фрейм должен быть отправлен, и отсылает его на этот порт. Таблица MAC- адресов обычно

реализована на двоичной АП, таким образом порт назначения может быть найден очень быстро, уменьшая время ожидания коммутатора.

Недостатки

Самый главный недостаток — это низкий объем памяти матрицы. Если в нашу матрицу записать штук 5–6 соответствий, то система начнет путаться и выдавать неверные значения.

Также нельзя записать соответствия в которых все нули или единицы (в нашем случае 1111 и 0000).

Ссылки

https://habr.com/ru/post/50541/

https://ru.wikipedia.org/wiki/%D0%90%D1%81%D1%81%D0%BE%D1%86%D0%B8%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C

12. Транзакционная память. Аппаратная и программная реализации транзакционных механизмов. Области применения, преимущества и недостатки.

Теория

1 опр: В компьютерных технологиях, программная транзакционная память (англ. software transactional memory, STM) представляет собой механизм управления параллелизмом, аналогичный механизму транзакций баз данных для управления доступом к совместно используемой памяти в параллельных вычислениях. Это альтернатива для синхронизации на основе блокировки. Транзакция в этом контексте является частью кода, который выполняет считывание и запись в разделяемую (совместно используемую) память. Считывание и запись логически происходит в единичный момент времени, а промежуточные состояния невидимы для других (результативных) транзакций.

2 опр: Транзакционная память — технология синхронизации конкурентных потоков. Она упрощает параллельное программирование, выделяя группы инструкций в атомарные транзакции. Конкурентные потоки работают параллельно, пока не начинают модифицировать один и тот же участок памяти.

Плюсы транзакционной памяти:

- относительная простота использования (заключение целых методов в блок транзакции);

- полное отсутствие блокировок и взаимных блокировок;
- повышение уровня параллелизма, а следовательно, и производительности.

Минусы:

- при неправильном использовании возможно падение производительности и некорректная работа;
- ограниченность применения — в транзакции нельзя выполнять операции, действие от которых невозможно отменить;
- сложность отладки — поставить breakpoint внутри транзакции невозможно.

Программные реализации

C/C++ (GCC 4.7+)

Начиная с версии 4.7, GCC поддерживает транзакционную память. Реализация представляет собой библиотеку времени выполнения libitm, для компиляции указывается флаг -fgnu-tm (-mrtm, -mhle). Библиотека разрабатывалась с оглядкой на черновик транзакционных конструкций для C++ (предлагается включение в стандарт языка).

Большинство реализаций аппаратной транзакционной памяти используют принцип наибольшего усилия. Поэтому практичные реализации используют объединение технологий аппаратурной и программной транзакционной памяти. Такие системы называют системами «гибридной транзакционной памяти». К ним относится, в частности, реализация GCC.

Также программные реализации есть на Haskell, Scala и Clojure (подробнее см в ссылках habr)

Аппаратные реализации

IBM BlueGene/Q (PowerPC A2) 2011

Суперкомпьютер Sequoia с архитектурой BlueGene/Q стал первой коммерческой системой с аппаратной поддержкой транзакционной памяти. Технология работает в 32-мегабайтном кэше второго уровня процессора PowerPC A2 (PowerPC BQC 16C). Данные в кэше имеют метку “версия”, и кэш способен хранить несколько версий одних и тех же данных.

Программа сообщает процессору о начале транзакции, делает свою работу и сообщает, что транзакцию нужно завершить (commit). В случае если другие потоки изменяли те же данные — создавая версии — кэш отклоняет транзакцию и поток

пытается провести её вновь. Если других версий данных не появилось, данные модифицируются и транзакция завершается успешно.

Технология версионных меток в PowerPC A2 также используется для спекулятивного выполнения. Вместо ожидания новой версии данных поток может выполнить вычисления с имеющимися данными, спекулятивно производя полезную работу. Если данные были такими же, как и после обновления, поток завершает (commit) работу из кэша. Производительность выше: поток выполнил работу до получения финального значения. В противном случае результаты спекулятивной работы отклоняются и поток производит вычисления с корректными значениями.

Поддержка транзакционной памяти — в некотором роде логическое расширение возможности, давно присутствующей в процессорах PowerPC — “load-link/store-conditional”, или LL/SC. LL/SC — примитивная операция, которая может использоваться как строительный блок для всех потокобезопасных конструкций. Первая часть LL/SC — load-link — используется программой для получения данных из памяти. Далее поток изменяет данные и записывает их обратно в память с помощью store-conditional. Команда завершается успешно, если данные не менялись. В противном случае программа повторяет действия с данными с начала.

Транзакционная память — LL/SC на стероидах: потоки выполняют операции LL на множестве различных областей памяти, а операция завершения транзакции атомарно изменяет все области памяти или отменяет транзакцию (как SC).

Intel Haswell (x86) 2013

В 2012 компания Intel объявила о введении расширений транзакционной синхронизации (Transactional Syncrhonization Extensions, TSX) в набор инструкций x86. Расширения позволяют программистам помечать отдельные участки кода как транзакции.

В 2013 с выходом поколения процессоров Haswell аппаратная поддержка транзакционной памяти впервые становится доступной на потребительском уровне.

Haswell управляет наборами чтения и записи с гранулярностью линии кэша, отслеживая адреса кэша данных L1. Конфликты определяются с помощью протокола когерентности кэша. Что логично предположить, поскольку задачи определения конфликтов транзакций и поддержки когерентности кэшей очень близки: если значение меняется в одном потоке, но не в других, то что-то неладно.

TSX состоит из двух частей. Аппаратное исключение блокировок (Hardware Lock Elision, HLE) предоставляет простую конвертацию программ на основе блокировок

в транзакционные программы, причём полученные программы будут обратно совместимы с существующими процессорами. Ограниченная транзакционная память (Restricted Transactional Memory, RTM) является более полной реализацией транзакционной памяти.

Ещё аппаратные реализации см по ссылке habr.

Ссылки

https://habr.com/ru/post/221667/

https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%B0%D1%8F_%D1%82%D1%80%D0%B0%D0%BD%D0%B7%D0%B0%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C