Крис Касперски

ПРОБЛЕМЫ ТЕСТИРОВАНИЯ ОПЕРАТИВНОЙ ПАМЯТИ

kk@sendmail.ru

Крис Касперски
kk@sendmail.ru

Проблемы тестирования оперативной памяти

Разгон памяти – весьма радикальное средство увеличения производительности, но и чрезвычайно требовательное к качеству модулей памяти. Впрочем, некачественные модули могут сбоить даже в штатном режиме без всякого разгона. Последствия таких ошибок весьма разнообразны: от аварийного завершения приложения до потери и/или искажения обрабатываемых данных.

Судя по всему, приобретение "битой" памяти – отнюдь не редкость и со сбоями памяти народ сталкивается достаточно регулярно. Забавно, но подавляющее большинство разработчиков программного обеспечения начисто игнорируют эту проблему, заявляя, что всякое приложение вправе требовать для своей работы исправного "железа". Теоретически оно, может быть и так, но на практике факт исправности железа предстоит еще подтвердить.

Причем, популярные диагностические утилиты (такие, например, как Check It) для этой цели абсолютно не пригодны. За исключением совсем уж клинических случаев, тест проходит без малейших помарок, но стоит запустить тот же Word или Quake, как система мгновенно виснет. Меняем модуль память – все работает на ура. Выходит, виновата все же память? Тогда почему это не обнаружил Check It?!

Причина в том, что далеко не всякая неисправность чипа памяти приводит к его немедленному отказу. Чаще всего дефект проявляется лишь при определенном стечении ряда обстоятельств. Тяжеловесное приложение, гоняющее память "и в хвост, и в гриву", имеет все шансы за короткое время "подобрать" нужную комбинацию, провоцирующую сбой. Популярные диагностические программы, напротив, тестируют весьма ограниченный спектр режимов в весьма щадящих условиях. Соответственно, и вероятность обнаружить ошибку в последнем случае значительно ниже.

Выход – разрабатывать собственную тестирующую программу. Во-первых, необходимо учитывать, что вероятность сбоя тесно связана с температурой кристалла. Чем выше температура – тем вероятнее сбой. А температура в свою очередь зависит от интенсивности работы памяти. При линейном чтении ячеек, микросхема памяти за счет пакетного режима успевает несколько приостыть, поддерживая внутри себя умеренную температуру. Действительно, при запросе одной ячейки, вся DRAM-страница читается целиком, сохраняясь во внутренних буферах, и до тех пор пока не будет запрошена следующая страница этого же банка, никаких обращений к ядру памяти не происходит!

Поэтому, прежде чем приступать к реальному тестированию, память необходимо как следует прогреть, читая ее с шагом, равным длине DRAM-банка. Это заставит ядро данного банка работать максимально интенсивно, на каждом шаге выполняя процедуру чтения и восстанавливающей записи данных. Не стоит те-

стировать несколько банков одновременно. Во-первых, это несколько снизит температуру "накала" каждого из них, а, во-вторых, перепад температур внутри кристалла увеличивает вероятность обнаружения сбоя. (Вообще-то, микросхеме при этом приходится по-настоящему туго, но она обязана выдержать такой режим работы, в противном случае, ее место – на свалке).

Ага, модуль памяти нагрелся так, что не удержишься рукой. Самое время приступать к настоящим тестам. Заполняем DRAM-страницу контрольной последовательностью чисел (далее по тексту – шаблоном), переключаем страницу, чтобы гарантированно обновить ячейки памяти (в противном случае микросхема может возвратить содержимое своих буферов, не обращаясь к матрице памяти). Вновь переключаем страницу назад и проверяем, что мы записали. Это может выглядеть приблизительно так:

```
for (a=0; a < DRAM_BANK_SIZE; a += DRAM_PAGE_SIZE)
{
WriteTemplate(a);               // записываем шаблон

x = (DRAM_BANK_SIZE-a);         // переключаем DRAM-страницу

CompareTemplate(a);             // проверяем, что мы записали
}
```

Листинг 1. *Упрощенный пример реализации функции тестирования памяти.*

Причем, к шаблону предъявляются весьма жесткие требования. Во-первых, он должен тестировать каждый бит ячейки, причем на оба значения – единицу и нуль, поскольку, "битые" ячейки матрицы могут давать либо "всегда ноль", либо "всегда единица". Во-вторых, крайне желательно, чтобы во всем восьмерном слове соседние биты имели противоположные значения. Такая комбинация создает наибольший уровень помех и тем самым провоцирует систему на ошибку. Третье, шаблон должен обеспечивать выявление ошибок адресации, т. е. микросхема возвратила содержание ячейки не той строки и/или столбца.

Поскольку, все эти требования взаимоисключающие, для тестирования потребуется несколько шаблонов. Только не забывайте время от времени выполнять "холостое" чтение для поддержания температуры микросхемы на максимально достижимом уровне, иначе эффективность теста начнет падать.

Остается обсудить лишь последовательность перебора станиц. Первое, что приходит на ум тривиальный последовательный перебор, затем – хаотичное обращения к страницам по случайному шаблону. Достаточно ли этого для выявления всех типов ошибок? К сожалению, нет. Многие (если не все) современные контроллеры памяти самостоятельно определяют предпочтительный порядок обработки запросов. Возьмем, например, листинг рассмотренный выше.

Контролер, проанализировав очередь запросов, видит, что двойного переключения страниц можно избежать, если... не выполнять повторное чтение из матрицы памяти, а возвратить процессору содержимое буфера!

Похоже, есть только один путь обхитрить контроллер – проверять ячейки не сразу после записи, а спустя некоторое время, когда внутренние буфера контроллера будут гарантированно перекрыты последующими запросами. Это же, кстати, позволяет выявить ошибки регенерации, когда из-за каких-то дефектов заряд с ячейки матрицы стекает раньше, чем ее успевают регенерировать.