Microprocessor Architecture

FROM SIMPLE PIPELINES TO CHIP MULTIPROCESSORS

Jean-Loup Baer
University of Washington, Seattle

CAMBRIDGE
UNIVERSITY PRESS

on the device to proceed simultaneously with the access of a cached row. Finally, by transferring data on both edges of the clock, the data rate can be doubled, yielding the DDR SDRAMs (DDR stands for double data rate).

6.4.2 Improving Memory Bandwidth

Improving memory bandwidth means that we want to transfer as much information per (bus) cycle as possible. With a given memory data bus width, the DRAM chips should be organized optimally to provide a bus width of data per request. With current bus widths of 64 or 128 bits, the DRAM organization shown in Figure 6.11, whereby a chip yields one bit of output per request, needs to be revisited so that it yields a larger number of bits per request. Indeed, expansion is needed also so that the number of chips does not increase unduly and to ease the modularity of capacity expansion.

A first improvement is to allow more than one data output pin per chip, so that consecutive bits in a row can be delivered at each cycle from a single chip. Chips are organized into circuit board modules called *dual inline memory modules* (DIMMs), an improvement over SIMMs, where S stands for single, by doubling the number of connections on using the two edges of the physical connectors. As an example, a 256 Mbit SDRAM found in the marketplace could consist of four chips of 64 Mbit each. The 64 Mbit chips can be either (i) 8 K rows and 2 K columns of 4 bits (dubbed 16 $\times$ 4 in the commercial literature), or (ii) 8 K rows of 1 K columns of 8 bits (8 $\times$ 8), or (iii) 8 K rows of 512 columns of 16 bits (4 $\times$ 16). Generally, the cost of the module increases with the width it can deliver because of the larger number of data lines needed. Ideally, the number of bits that can be read on a read request should be equal to the number of bits that can be transmitted on the memory data bus. If the width is too narrow, a *reassembling buffer* must be included in the memory controller to concatenate the results of consecutive requests.

The number of chips that can be included on a DIMM is limited by physical constraints; 4 to 16 chips is a good order of magnitude. Because main memory is a resource that users will often want to expand, having very large capacity DIMMs means that jumping from one configuration to the next will involve large increments in capacity, and hence cost, thus also placing a user-motivated limit on the overall capacity of DIMMs.

Main memory is further divided into *banks* that can receive independent commands from the memory controller. Banks can be *interleaved* in several ways, depending on the mapping of memory addresses to banks. A popular mapping is to have banks interleaved by units of bus width. For example, if a memory has two banks and the data bus is 64 bits (8 bytes) wide, bank 0 stores all double words (8 bytes) at addresses 0, 16, 32, ..., while bank 1 stores those at addresses 8, 24, ... If the cache line size of the higher-level cache is greater than 8 bytes, then both banks can proceed in parallel to fulfill the request (except of course for the serialization on the bus). Alternatively, a bank can serve the whole request for a cache line, and the

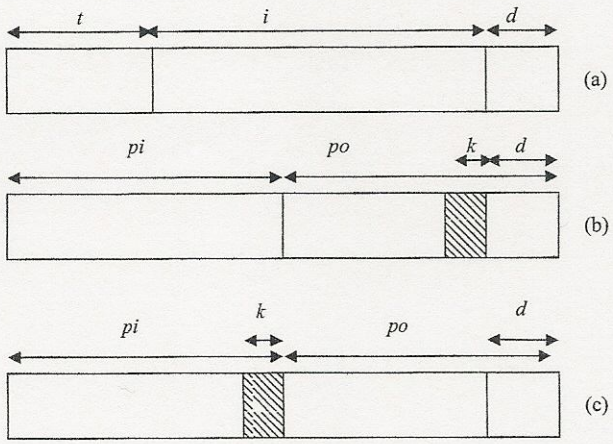

Figure 6.12. Interleaving of external banks: (a) address as seen from the cache, with d the line offset, i the set index, and t the tag; (b) line interleaving, with po the page offset, pi the page index, and k the bank index; (c) page interleaving.

next bank will then serve the next (addresswise) line. In the first case, because the bus is in general faster than the DRAM even in page mode, the number of banks should be larger than the number of cycles it takes to deliver data to the bus. Banks are also interesting for writes, for they allow writes in independent banks to proceed in parallel.

Banks can be grouped in banks of banks, or external banks. Within an external bank, the internal banks are interleaved so that they can deliver a request to the cache as fast as possible. The interleaving of the external banks is not as crucial, but some mappings are better than others. In Figure 6.12, we show how the address is seen from the cache's viewpoint and with two different natural interleavings of external memory banks.

As can be seen, in both types of interleaving the bank index overlaps with the cache set index. This will happen for all reasonable higher-level cache and DRAM page sizes. The consequence is that in case of cache conflict misses, the missing line will have the same bank number as the replaced line, and therefore the same bank will be accessed if there is line writeback (although this might be alleviated by the presence of write buffers) and also if the conflicting misses are occurring in a Ping-Pong fashion. In both cases, a full penalty of precharge and row and column access is likely, because the high-order bits in the addresses of the conflicting lines (those called *pi* at the left of the page offset) are likely to be different. A possible solution is to have the bank index be the result of a hashing (a single XOR operation is sufficient) of k bits in the *tag* field with the original k bits at the left of the page offset (those indicated in Figure 6.13). The advantage is that the cache conflicting misses will in all likelihood map to different banks and the contents of the pages remain the same (only the bank index changes). Moreover, because the original bank index is used in the hashing function, there is a uniform distribution of addresses in banks.

We have centered our discussion within the context of a single request from the processor, or rather the higher-level cache, to main memory. However, several

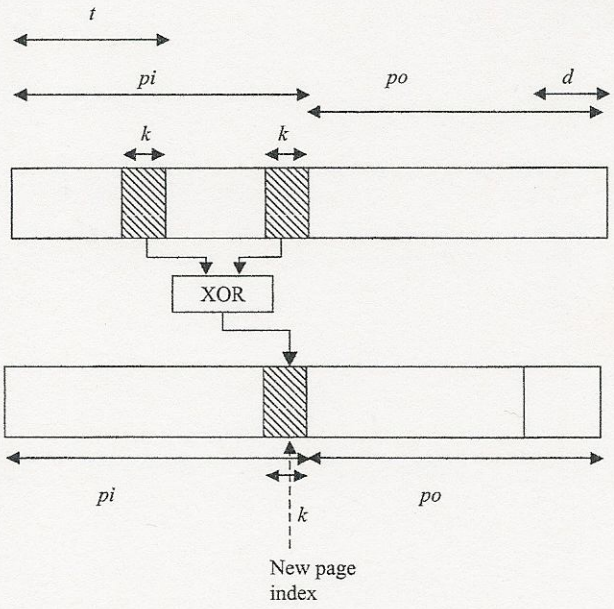

Figure 6.13. Interleaving scheme to reduce bank conflicts (adapted from Zhang et al. [ZZZ00])

requests could be pending, for the caches can be lockup-free. Moreover, I/O should be done concurrently, and main memory can also be shared by several processors. Holding the processor–memory bus busy during the whole latency of a request is not efficient in these conditions. Waiting times for the bus can be reduced by using a *split-transaction* bus. As its name indicates, with a split-transaction bus a request, for example a read transaction, can be decomposed into sending the address on the bus, releasing the bus while the DRAM reads the data, requesting the bus when the data are ready in the DRAM buffer, and, when this last request is satisfied, sending the data on the bus. While the data are read from the DRAM, other requests can be sent on the bus and queued at the memory controller, or data read from another transaction can be sent on the bus. If address lines and data lines of the bus are distinct, then two transactions can be in progress concurrently. The address for transaction A can be sent at the same time as the data for transaction B.

The presence of a split-transaction bus complicates the design of the memory controller. Buffering is needed for incoming transactions and possibly for read–write data. Of course, with this complication comes an advantage, namely, transactions do not have to be processed in order. An intelligent controller could, for example, delay prefetching transactions in favor of processing a real read miss as soon as possible. The controller could also take advantage of hits in row buffers, delaying temporarily a request that would destroy the locality of other transactions. However, because the read requests are processed out of order, they must be tagged so that the recipients can know when their requests are fulfilled.

Although interleaving and split-transaction buses improve the occupancy of the bus, they do not reduce the latency of single requests. Latency speedup requires faster devices and faster buses.