1999

p 8q4
1 AP

Comparative Architectures / IAP Q1 Extended Answer

A new microprocessor has a 64KB write-back L1 D-cache that is 2 way set-associative with 64 byte cache lines. How might knowledge of these cache parameters be exploited in order to improve program performance? [5 marks]

Knowing the size of the cache may help determine the best implementation of an algorithm to use. For example, a table-lookup approach may prove more efficient than a computational one if the table can be designed such that its working set fits in the cache.

Since the cache has only limited associativity, care should be taken two ensure that data structures that are frequently accessed together are not aligned modulo 32KB. Two structures that are thus aligned are no problem, but a third would cause thrashing of the L1 cache.

It makes sense to group data that is frequently accessed together into the same 64 byte cache line; after causing the line to be fetched into the L1, there is little cost in accessing its other elements. This can have consequences for the datastructures used in a program. For example, using a hash table whereby a cache line's worth of entries are held in each hash bucket is likely to be more efficient than a linked list of entries.

Since the cache is write-back, multiple stores to the same location will typically be merged in the cache. Were the cache write-through, such practise is probably best avoided.

What pros and cons might the designers have considered when selecting the cache line size? [5 marks]

Increasing the block size can reduce the number of compulsory misses, since more data is fetched into the cache with each miss. Since each bus transaction is longer, any transaction overhead (latency) is amortized over a greater number of cycles, thus increasing bandwidth.

However, increasing the line size can increase the number of conflict misses a cache experiences. It can also increase the load latency of the system: An instruction may have to wait several cycles until the actual word it references is returned by the memory system (though this can be usually be solved by a combination of critical-word-first cache line wrapping and early instruction restart.) Also, a load that references another cache line will have to wait until the whole of the previous line is returned before it can be supplied with the data it requires.

The processor's L1 D-cache is indexed by virtual address and tagged by physical address. Why would the designers have done this? Since the processor has an 8KB page size, their decision is likely to have impact on the operating system virtual memory system. Explain why this is so, and state why the designers may have considered the possibility of increasing the cache's associativity. [5 marks]

By using the virtual address as an index, the cache access can be started in parallel with the D-TLB access, rather than having to wait until after it (thus reducing the load-to-use latency). By the time the TLB access is complete, the cache will have accessed both of the possible candidate lines that the load may have referenced (2 way). The translated physical address is then compared against the tags of of the two possibilities, and one of them selected, or a miss declared.

With a physically tagged, virtually indexed cache, care must be taken to avoid virtual address aliasing. For the 64KB 2 way cache described, the operating system must ensure that all virtual addresses that map to the same physical page must have the same alignment modulo 32KB (4 pages). Without this assurance, it would be possible for two or more non-coherent copies of the same physical page to exist in the cache.

Rather than modifying the operating system to ensure this condition is met, the chip designers may have considered making the cache 8-way associative, thus avoiding the aliasing problem. Presumably, this would have resulted in a slower cycle time, and was thus discounted.

The processor's physical address space is 64 bits wide. Calculate the number of bits of RAM required to implement the D-cache, including tag, data and status bits. [5 marks]

There are 64KB/64 = 1024 cache lines in the cache, organised as 512 sets of two lines.

Thus, 6+9 = 15 bits are used as a direct index, leaving 49 bits for the tag.

Since the cache is write back, a minimum of two bits of status information must be maintained (typically this will be used to implement a Modified, Exclusive, Shared, Invalid scheme (MESI)).

Number of bits of SRAM required to implement: 1024 * [64*8 + 49 + 2] = 576512 bits