Homework Assignment 4

CS/ECE 6810: Computer Architecture
Nov 28, 2018
Jake Pitkin, u0891770

Memory Systems

Due Date: December 11, 2018.
(120 points)

1. **Virtually Indexed Cache.** Referring to the lecture slide on virtual address translation and TLB, explain the challenges if the number of bits in the page offset does not equal the number of bits in the sum of “Index” and “Byte” (Please see `http://www.cs.utah.edu/~bojnordi/classes/6810/f18/slides/18-tlb.pdf`). You are asked to identify the issues and their corresponding solutions from the literature. **(20 points)**

   Let x be the number of page offset bits and y be the number of bits in the sum of ”Index” and ”Byte”. We will consider 3 cases:

   Case x = y: In this case they are equal, there is no challenge and we can index into the cache in parallel with page number translation by the TLB. This is a side effect of the page offset bits being identical in both the virtual and physical addresses so waiting for the TLB to execute won’t change the bits used for ”Index” and ”Byte”.

   Case x > y: Here we have more page offset bits than required for ”Index” and ”Byte”. We can just use a subset of the page offset bits for ”Index” and ”Byte” and this is fine for the same reasons as in case 1.

   Case x < y: When there are less page offset bits than required for ”Index” and ”Byte” an issue arises. As discussed in the lecture, in this case part of the ”Index” bits will come from the virtual address. This means the index into cache will be dependent on the virtual address. Two different virtual addresses could be mapped to a single physical address. As such, it’s possible for a given physical word to be put into the cache in more than one location.

   From the wikipedia on CPU caches, this is called aliasing. This is an issue as we now have cache coherence problems inside a single cache. Processors are designed to satisfy cache coherence under the assumption a given cache only contains a physical word once. There is also the issue of contiguous pages in virtual memory not always being contiguous in physical memory. Using the virtual address for caching could lead to poor performance as we overwrite contiguous pages in physical memory.

To solve this issue we must guarantee any given physical word doesn't exist in a cache in more than one location. But we still want to use the virtual address for some of the "Index" bits as it allows us to run the two streams of execution as described in the lecture. As hinted in lecture the solution is cache coloring.

From the wikipedia on cache coloring, it is the process of attempting to allocate free pages that are contiguous from the CPU cache's point of view. That is, normally a physically index CPU aims to place contiguous pages in physical memory in different positions in the cache. But with page coloring: *Physical memory pages are "colored" so that pages with different "colors" have different positions in CPU cache memory. When allocating sequential pages in virtual memory for processes, the kernel collects pages with different "colors" and maps them to the virtual memory. In this way, sequential pages in virtual memory do not contend for the same cache line* (from the wikipedia on cache coloring).

This solves our performance problem as sequential pages in virtual memory do not content for the same cache line. When using some of the virtual address bits for "Index" we use the lower bits. As such, contiguous pages in virtual memory will be assigned different positions in the cache. Additionally this solves the problem of aliasing. If two virtual addresses map to the same physical address, they will be assigned to the same line in the cache (as the physical address with have a given color) and we won't have duplicate entries and cache coherence problems.

References:

Lecture 18 video: https://web.microsoftstream.com/
video/8d07bb10-4791-4c46-bab8-1b2e2a7e1a09

Lecture 18 slides: http://www.cs.utah.edu/ bojnordi/classes/6810/f18/slides/18-tlb.pdf

Wikipedia on caches: https://en.wikipedia.org/wiki/CPU_cache

Wikipedia on cache coloring: https://en.wikipedia.org/wiki/Cache_coloring

2. **Virtual Memory and TLB.** Consider an operating system (OS) using 1KB pages for mapping virtual to physical addresses. Initially, the TLB is empty and all of the required pages for the user application as well as the page table are stored in the main memory. (There is no need to transfer data between main memory and the storage unit.) Every access to the TLB and main memory takes respectively 1 and 200 processor cycles. The TLB can store up 16 entries.

   i Assuming that the main memory size is 1MB and the page table has 2048 entries, show the number of required bits for the physical and virtual address fields. **(5 points)**

   1KB pages = 2^{10} = 10 offset bits to index
   2048 page table entries = 2^{11} = 11 bits to address
   1MB main memory with 1KB pages = $2^{20}/2^{10}$ = 10 bits to address

   Virtual address = Virtual Page No + offset = 11 + 10 = 21 bits
   Physical address = Page frame No + offset = 10 + 10 = 20 bits

   ii Assume that the user application only generates five memory requests to the virtual addresses `0000`, `0004`, `0008`, `0800`, and `0804` (all in hexadecimal); the first request is ready at time 0; and the processor generates every next request immediately after serving the current one. Please find the execution time with and without using TLB in the proposed system and compute the attainable speedup due to using TLB. **(15 points)**

Assumptions: There is no mention of caching the words once they have been accessed. I will assume that once the virtual address has been translated to a physical address we will always have to access memory to get the data (as there is no caching). I'll also assume searching the page table takes negligible time as it's not mentioned in the problem.

Without a TLB: every access will have to read the page table in main memory and the data in main memory as there is no caching:

Virtual Address	**Virtual Page No**	**Offset**	**Access Cycles**
0000	0	0	400
0004	0	8	400
0008	0	16	400
0800	2	0	400
0804	2	8	400

$\boxed{\text{2,000 cycles with no TLB}}$

With a TLB: every access will still have to read main memory for the data as there is no caching, but we can cache address translations in the TLB:

Virtual Address	**Virtual Page No**	**Offset**	**Access Cycles**
0000	0	0	400
0004	0	8	201
0008	0	16	201
0800	2	0	400
0804	2	8	201

$\boxed{\text{1,403 cycles with a TLB}}$

Speedup: to calculate speedup we compare the ratio of access cycles for not using a TLB and using a TLB:

$\boxed{\text{old time / new time = 2,000 / 1,403 = 1.4255 or a 42.55\% speedup}}$

3. **DRAM Address Mapping.** Consider a simple in-order DRAM command scheduler. Initially, all DRAM banks are precharged and the scheduling queue contains seven read requests to the following physical addresses: `00040108`, `01040101`, `FF042864`, `A5181234`, `A5184321`, `00161804`, and `01040104` (all in hexadecimal). Using the following address mapping scheme, show all the required commands issued by the controller to serve the memory requests. **(20 points)**

row (10)	bank (4)	rank (2)	channel (0)	column (16)

First I will calculate the row, bank, rank, and column from each physical address:

Physical Address	row	bank	rank	column
00040108	0	1	0	264
01040101	4	1	0	257
FF042864	1020	1	0	10340
A5181234	660	6	0	4660
A5184321	660	6	0	17185
00161804	0	5	2	6148
01040104	0	1	0	260

Physical Address	Commands
00040108	
01040101	
FF042864	
A5181234	
A5184321	
00161804	
01040104	

4. **DRAM Row (Page) Management.** A computer system includes DRAM and CPU. The DRAM subsystem comprises a single channel/rank/bank. CPU generates a sequence of memory requests to rows `X` and `Y`. Table below shows the accessed rows by the memory requests and their arrival times at the DRAM controller. Assume that the DRAM bank is precharged initially; the DRAM scheduling queue has infinite size; and the memory interface must enforce the following timing constraints (all in ns): tCAS = 3, tRAS = 20, tRP = 20, tRCD = 10, and tBURST = 4.

Accessed Row	Arrival time (ns)
X	10
X	70
Y	80
X	85
Y	110
X	210

The goal is to evaluate two row management policies, namely *open-page* and *closed-page*. Show all the necessary transactions on the command, address, and data buses for each policies. Discuss which policy performs better for the given example. (Note: you are allowed to reorder requests already waiting in the memory controller.) **(20 points)**

Cmd --

Addr --

Data --

Answer

5. **Data TLB.** Consider the following pseudo code, in which **X** and **Y** are allocated as contiguous integer arrays in memory and are aligned to the 4KB boundaries. Assume that **k** and **l** are allocated in the register file with no need for memory accesses. We execute the code on a machine with virtual memory where the integer type (**int**) is 4 bytes wide. Assume that the system include a direct-mapped D-TLB with 1024 entries for serving all data accesses only. Initially, the D-TLB is empty. Find the hit rate of the D-TLB when running the code.

   (20 points)

```
#define M 1024
int X[M*M];
int Y[M*M];
int k, l;
  for ($k = 0; k < M; k++$) do
    for ($l = 0; l < M; l++$) do
      Y[l*M+k] = X[k*M+l];
    end for
  end for
```

Answer