CS152/252A
Spring 2026

Memory Hierarchy
Discussion 4 Solutions

1 Prefetching Basics

1.1 Define and give a formula to compute each prefetching metric.

Accuracy: Did we use what was prefetched? Measured as (`Useful Prefetches / Total Prefetches`)

Coverage: Is the prefetcher covering all accesses? Measured as (`Useful / Total unique accesses`)

Timeliness: Is the prefetch on time (not too early / too late)? Measured as (`On-time / Total prefetches`)

1.2 What are the memory access patterns for:

Instruction prefetching: Often sequential with control flow jumps

Data prefetching: Much more irregular (loops, pointer chasing, etc)

1.3 Which prefetching algorithm(s) are most appropriate for:

Instruction memory: OBL, prefetch-on-miss

Data memory: Strided, spatial memory streaming, pointer chasing

2 Software Prefetching

For the following program, assume 128B cache lines (each row fits entirely in a cache line). Without the prefetch, the inner loop takes 50 cycles. The L1 miss penalty is 40 cycles. What should OFFSET be to minimize the total program cycles?

```
int A[N][M]; // N=32, M=32
int sum = 0;

for (int j = 0; j < M; j++) {
  for (int i = 0; i < N; i++) {
    // prefetches from (A + M*i + j + OFFSET)
    prefetch(&A[i][j] + OFFSET);
    sum += A[i][j];
  }
}
```

With perfect prefetching: 50-40=10 cycles per iteration. Prefetched data takes 40 cycles to return, so we need to fetch 4 iterations in advance. OFFSET = 4*32 = 128

3 Linear vs Hierarchical Page Tables

3.1 Consider 4 GiB (32-bit) of addressable virtual memory, 4 KiB pages, 4-byte PTEs.

Bits in the page offset: $\log_2$(page size in bytes) = $\log_2(4096) = 12$

Bits in the virtual page number: Size of memory address - page offset bits = 32 - 12 = 20

Number of pages: 2^{20} pages

3.2 Consider a linear page table for a process with only 1 page mapped to physical memory (paged in)

Number of valid page table entries (PTEs): 1 valid PTE

Total size of the page table: $2^{20} * 4B = 4MiB$

3.3 Consider a 2-level page table for a process with only 1 page mapped to physical memory (paged in). Assume that VPN bits are split equally between the two levels.

Number of valid page table entries (PTEs): 2 valid PTEs

Total size of the page table structure:

Lvl 1 Page Table = 2^{10} PTEs $*$ 4 = 4KiB
Lvl 2 Page Table = 2^{10} PTEs $*$ 4 = 4KiB
$\rightarrow$ 8KiB total

4 Page Size Tradeoffs

What are the benefits of a larger page size? What are the benefits of a smaller page size?

Larger: Smaller page tables, fewer page table walks required, smaller TLB required for same coverage

Smaller: Less internal fragmentation, lower page fault penalty (less traffic to load new page on fault)

5 Hierarchical Page Table Example

Assume: 8-bit virtual addresses, 32-bit words, 32-bit PPNs, 16-byte pages, two-level page table, LRU 4-entry TLB. At the beginning, the TLB is empty and the free pages list contains 0x9, 0x5, 0xA, 0x7, 0x1, 0x3, 0xB, 0xD, 0xE, and 0xF in that order. PTBR is set to 0.

5.1 How many bytes of virtual memory are addressable?

$2^8 = 256$ Bytes

5.2 How many bytes of physical memory are addressable?

PPN = 32 bits; 2^{32} pages $*$ 16 B/page = 64 GiB

5.3 Why might DRAM size > virtual address space size be useful?

Multiple processes resident in main memory

5.4 Divide the virtual address into:

Offset bits: vaddr[3:0] - $\log_2(16) = 4$-bits

Second level index bits: vaddr[5:4]
8-bit virtual address - 4 bits of offset = 4 bits
4 bits / 2 levels of index bits = 2-bits per level

Top level index bits: vaddr[7:6] - 2-bits as explained above

5.5 How many entries are in the…

Top level table? 4 - 2^2 ($2^{\#\text{ of index bits}}$)

Second level tables (each)? 4

5.6 Fill in the tables. The first address translation is given as an example.

Free pages: 0x9, 0x5, 0xA, 0x7, 0x1, 0x3, 0xB, 0xD, 0xE, 0xF

Execution Behavior:

Virtual Address	Index1	Index2	TLB hit / miss	Page hit / page fault	Physical address
0x68	0x1	0x2	miss	hit	0x128
0x14	0x0	0x1	miss	hit	0x134
0x6C	0x1	0x2	hit	hit	0x12C
0x90	0x2	0x1	miss	fault	0x090
0x74	0x1	0x3	miss	hit	0x114
0xE4	0x3	0x2	miss	fault	0x0A4
0x18	0x0	0x1	miss	hit	0x138
0xD0	0x3	0x1	miss	fault	0x070

TLB:

VPN	~~0x6~~ 0x1	~~0x1~~ 0xE	~~0x9~~ 0xD	0x7
PPN	~~0x12~~ 0x13	~~0x13~~ 0x0A	~~0x9~~ 0x07	0x11

Note: Because the TLB has an LRU replacement policy, the access to `0xE4` replaces the TLB entry for VPN=`0x1` because we had the `0x6C` access more recently than the access to `0x14`

Memory Contents:

Address	Content
0x00	0x06
0x04	0x04
0x08	0x02
0x0C	0x05
0x10	
0x14	
0x18	
0x1C	
0x20	0x08
0x24	0x09
0x28	
0x2C	
0x30	
0x34	
0x38	
0x3C	
0x40	
0x44	
0x48	0x12
0x4C	0x11
0x50	
0x54	0x07
0x58	0x0A
0x5C	
0x60	
0x64	0x13
0x68	
0x6C	