CS61c Summer 2014 Discussion 8 – More Caches & Review

1 Cache Question (From Fa11 Midterm)

Take a look at the following C function `sum_iter` run on a 32-bit MIPS machine. On this system, these structs are aligned to two-word boundaries since `sizeof(struct Node) = 8`. Assume the total space taken up by the linked list is greater than (and a multiple of) the cache size.

```
struct Node {
  int n;
  struct Node *next;
};
int sum_iter(struct Node *head) {
  int sum = 0;
  while(head != NULL) {
    sum += head -> n;    // load from head+0
    head = head -> next; // load from head+4
  }
  return sum;
}
```

Given a direct-mapped data cache with this T:I:O configuration: 12:13:7

- How many words are in a block?
- How many bytes of data does this cache hold?
- What is the lowest possible cache hit rate for the while loop in `sum_iter`?
- What is the highest possible cache hit rate for the while loop in `sum_iter`?
- To achieve this maximum hit rate, we obviously could have every Node next to every other node, like an array. However, thats too strict a constraint–we can still achieve this hit rate if thats not the case. What is the loosest constraint for how the Nodes are distributed in memory to get the best hit rate?

2 Cache Performance

1. Suppose our processor has separate L1 instruction cache and data cache. Our ideal base CPI is 2 clock cycles. Memory accesses take 100 cycles. Our I$ miss rate is 3% while our D$ miss rate is 10%. 40% of our instructions are loads or stores. What is our processor's actual CPI?
2. To improve the performance of this processor, we add a unified L2 cache between the L1 caches and memory. Our L2 cache has a hit time of 10 cycles and a global miss rate of 2%. What is the new actual CPI?
3. Suppose we have an unrelated processor with a two-level cache. The L1$ has a hit time of 1 cycle and a miss rate of 10%. The L2$ has a hit time of 3 cycles and a miss penalty of 100 cycles. If we need our AMAT to be $\leq$ 2 cycles, what should the L2$ miss rate be?

3 Cache Question (From Su98 Final)

- A 32 KiB cache has a 16 B block size and is 4-way set-associative. What is the T:I:O breakdown?
- In a 2-way set-associative cache, three addresses A, B, and C all have the same index but distinct tags. What is a minimum sequence of accesses which, if repeated, will maximize the miss rate in the cache if it uses the LRU replacement policy?

- If the above sequence is repeated for a long period of time, what will the miss rate be if the cache uses an LRU replacement policy?
- If the hit time is 1 cycle, and the miss penalty is 3 cycles, what will be the AMAT for the LRU replacement policy using the above sequence?
- If the sequence is repeated for a long period of time, will the miss rate be improved if "random" is used as the replacement policy?

4 Review: MIPS Mystery Question (From Fa10 Midterm)

This assembly function takes no arguments and returns some value. Write this answer as binary number. What does it mean?

```
Address             Instruction
0x08001000 mystery: addiu $sp, $sp, -4
0x08001004          sw $ra, 0($sp)
0x08001008          addiu $v0, $zero, 0
0x0800100c          jal inner
0x08001010          lw $ra, 0($sp)
0x08001014          addiu $sp, $sp, 4
0x08001018          jr $ra
0x0800101c   inner: lw $v0, 4($ra)
0x08001020          jr $ra
```

5 Review: Programming in C Question (From Sp14 Midterm)

A colleague of yours has implemented some homebrew C99 string manipulation functions, while steadfastly refusing to use any standard libraries, but they might be buggy! Your job is to find the incorrect lines and give correct replacement lines.

```
/** Converts the string S to lowercase */
void string_to_lowercase(char *s) {
  for(char c = *s; c != '\0'; s++) {
    if(c >= 'A' && c <= 'Z') {
      s += 'a' - 'A';
    }
  }
}

/** Returns the number of bytes in S before, but not counting, the null terminator. */
size_t string_length(char *s) {
  char *s2 = s;
  while(*s2++);
  return s2 - s - 1;
}

/** Return the number of odd numbers in a number array */
uint32_t number_odds(uint32_t *numbers, uint32_t size) {
  uint32_t odds = 0;
  for (uint32_t i = 0; i < size; i++)
    odds += *numbers+i && 1;
  return odds;
}
```