Counting Bits in an Array

Ray Seyfarth

August 7, 2011

Outline

1. Overview
2. Counting 1 bits in C
3. Counting 1 bit in assembly
4. Precomputing 1 counts for all patterns of bytes
5. Using the popcnt instruction

Overview of counting bits

- The basic goal is to count the number of 1 bits in an array
- Several solutions are examined in C and assembly
- In general C and assembly perform similarly
- On CPUs with the `popcnt` instruction, assembly rules

A simple C solution

```
long popcnt_array ( long *a, int size )
{
    int w, b;
    long word;
    long n;
    n = 0;
    for ( w = 0; w < size; w++ ) {
        word = a[w];
        n += word & 1;
        for ( b = 1; b < 64; b++ ) {
            n += (word >> b) & 1;
        }
    }
    return n;
}
```

- Checking every bit took 4.74 seconds to call popcnt_array 1000 times with 100000 longs (64 bits)

Ending the loop earlier

- A slightly better algorithm ends the inner loop when `word` = 0
- The time dropped to 3.34 seconds

```
long popcnt_array ( unsigned long *a, int size )
{
    int w, b;
    unsigned long word;
    long n;

    n = 0;
    for ( w = 0; w < size; w++ ) {
        word = a[w];
        while ( word != 0 ) {
            n += word & 1;
            word >>= 1;
        }
    }
    return n;
}
```

Counting 1 bits in assembly

- I unrolled the inner loop 64 times
- The code is too long to place in a slide
- I split each 64 bit word into 4 16 bit words in separate registers
- Then I added each bit of the four words into 4 different registers allowing out-of-execution, pipeline filling and parallelism
- It performed the test in 2.52 seconds, a bit better than C at 3.34
- I did have a function of 1123 bytes

Precomputing 1 counts for all pattern of bytes

```
long popcnt_array ( long *a, int size )
{
    int b;
    long n;
    int word;

    n = 0;
    for ( b = 0; b < size*8; b++ ) {
        word = ((unsigned char *)a)[b];
        n += count[word];
    }
    return n;
}
```

- The count array had a static initializer with 256 counts
- The time dropped to 0.24 seconds, 10 times faster than the last version
- I could only tie this code with an assembly version

Using the popcnt instruction

- Some newer computers (Intel Core i series and some Opterons) have a popcnt instruction which exactly matches the problem.‘
- After unrolling the loop 2 times, the operation took 0.04 seconds on a Core i7 at 3.4 GHz

```
.count_more:
        popcnt  rdx, [rdi+rcx*8]
        add     rax, rdx
        popcnt  r9, [rdi+rcx*8+8]
        add     r8, r9
        add     rcx, 2
        cmp     rcx, rsi
        jl      .count_more
        add     rax, r8
```