The cost-benefit analysis of bitfields for a collection of booleans

devblogs.microsoft.com/oldnewthing/20081126-00

November 26, 2008

Raymond Chen

Consider a class with a bunch of `BOOL` members:

```
// no nitpicking over BOOL vs bool allowed
class Pear {
 …
 BOOL m_peeled;
 BOOL m_sliced;
 BOOL m_pitted;
 BOOL m_rotten;
 …
};
```

You might be tempted to convert the `BOOL` fields into bitfields:

```
class Pear {
 …
 BOOL m_peeled:1;
 BOOL m_sliced:1;
 BOOL m_pitted:1;
 BOOL m_rotten:1;
 …
};
```

Since a `BOOL` is typedef'd as *INT* (which on Windows platforms is a signed 32-bit integer), this takes sixteen bytes and packs them into one. That's a 93% savings! Who could complain about that?

How much did that savings cost you, and how much did you save anyway?

Let's look at the cost of that savings. Code that updated the plain `BOOL` `m_sliced` member could do it by simply storing the result into the member. Since it was a normal field, this could be accomplished directly:

```
  mov [ebx+01Ch], eax ; m_sliced = sliced
```

On the other hand, when it's a bitfield, updating it becomes trickier:

```
add eax, eax         ; shift “sliced” into the correct position
xor eax, [ebx+01Ch] ; merge “sliced” with other bits
and eax, 2
xor [ebx+01Ch], eax ; store the new bitfield
```

Exercise: Figure out how the above trick works.

Converting a `BOOL` to a single-bit field saved three bytes of data but cost you eight bytes of code when the member is assigned a non-constant value. Similarly, extracting the value gets more expensive. What used to be

```
push [ebx+01Ch]      ; m_sliced
call _Something@4    ; Something(m_sliced);
```

becomes

```
mov  ecx, [ebx+01Ch] ; load bitfield value
shl  ecx, 30         ; put bit at top
sar  ecx, 31         ; move down and sign extend
push ecx
call _Something@4    ; Something(m_sliced);
```

The bitfield version is bigger by nine bytes.

Let’s sit down and do some arithmetic. Suppose each of these bitfielded fields is accessed six times in your code, three times for writing and three times for reading. The cost in code growth is approximately 100 bytes. It won’t be exactly 102 bytes because the optimizer may be able to take advantage of values already in registers for some operations, and the additional instructions may have hidden costs in terms of reduced register flexibility. The actual difference may be more, it may be less, but for a back-of-the-envelope calculation let’s call it 100. Meanwhile, the memory savings was 15 byte per class. Therefore, the breakeven point is seven. If your program creates fewer than seven instances of this class, then the code cost exceeds the data savings: Your memory optimization was a memory de-optimization.

Even if you manage to come out ahead in the accounting ledger, it may be a win of just a few hundred bytes. That’s an awful lot of extra hassle to save a few hundred bytes. All somebody has to do is add an icon to a dialog box and your savings will vanish.

When I see people making these sorts of micro-optimizations, sometimes I’ll ask them, “How many instances of this class does the program create?” and sometimes the response will be, “Oh, maybe a half dozen. Why do you ask?”

But wait, there’s more. Packing all these members into a bitfield has other costs. You lose the ability to set a hardware write breakpoint on a specific bit, since hardware breakpoints are done at the byte level (at a minimum). You also lose atomicity: An update to `m_sliced` will interfere with a simultaneous update to `m_peeled` on another thread, since the update process merges the two values and stores the result non-atomically. (Note that you also lose atomicity if you had used a byte-sized `bool` instead of a 32-bit `BOOL` because some CPU architectures such as the original Alpha AXP cannot access memory in units smaller than a `DWORD`.)

These are just a few things to take into account when considering whether you should change your fields to bitfields. Sure, bitfields save data memory, but you have to balance it against the cost in code size, debuggability, and reduced multithreading. If your class is going to be instantiated only a few times (and by “a few” I’m thinking less than a few thousand times), then these costs most likely exceed the savings.

Raymond Chen

Follow