The cost/benefit analysis of comparing before an assignment

devblogs.microsoft.com/oldnewthing/20180625-00

June 25, 2018

Raymond Chen

Consider these two code fragments:

```
// Fragment 1
x = newValue;

// Fragment 2
if (x != newValue) x = newValue;
```

Which is faster?

The answer to that question is “It depends.”

Fragment 2 introduces extra work (a compare and a branch), and depending on the pattern of comparison results, the branch predictor might or might not have trouble predicting the result of the comparison. From the branch predictor’s point of view, it would prefer that the value always be different or the value always be the same, since that lets it predict with 100% accuracy.

On the other hand, Fragment 2 has the advantage of not writing to memory unnecessarily.

For languages with a write barrier, this will result in extra work to track the memory modification as well as creating extra work for the garbage collector because it needs to scan all modified memory in old generations to see if they point to objects in newer generations.`

Even for languages without a write barrier, unnecessary writes to memory have a cost.

The first write to a copy-on-write page will trigger a copy.

Every write to a page makes the page dirty and means that it will have to be written to disk when paged out, even if the contents in the page file match the values in memory.

On a multiprocessor system, writing to a cache line causes the processor doing the writing to become the owner of the cache line. When multiple processors all write to the same cache line, you suffer from a phenomenon known as *cache line bouncing*, where ownership of the

cache line keeps moving between processors, and the cache lines in the other processors become continuously invalidated due to the write.

So what tips the balance one way or the other?

First consideration: How often is the write avoided?

If the value is usually changing, then the comparison doesn't save you much because you end up doing the write most of the time anyway.

Another consideration is whether the variable is in a page that is otherwise read-only. If so, then you may want to avoid the write to keep the page clean. (This may be worthwhile even if the write is avoided only part of the time, because the cost of writing a page to disk is relatively high on a CPU time scale.)

Similarly, is the write into a cache line that is otherwise read-only? Avoiding the write allows the cache line to remain valid on all other processors and avoids cache line bouncing.

Is this cache line even being shared by multiple processors in the first place? If the variable is used by only a single thread, then it won't be in the cache of other processors anyway. (Modulo false sharing.)

These are all considerations to take into account, but as with all performance-related issues, the only way to know for sure is to test it under representative loads. Still, these early considerations may let you filter out the cases where avoiding the write is unlikely to offer any performance benefit.

Raymond Chen

Follow