Choose Concurrency-Friendly Data Structures

By Herb Sutter, June 27, 2008

Post a Comment

Linked Lists and Balanced Search Trees are familiar data structures, but can they make the leap to parallelized environments?

What is a high-performance data structure? To answer that question, we're used to applying normal considerations like Big-Oh complexity, and memory overhead, locality, and traversal order. All of those apply to both sequential and concurrent software.

But in concurrent code, we need to consider two additional things to help us pick a data structure that is also sufficiently concurrency-friendly:

- In parallel code, your performance needs likely include the ability to allow multiple threads to use the data at the same time. If this is (or may become) a high-contention data structure, does it allow for concurrent readers and/or writers in different parts of the data structure at the same time? If the answer is, "No," then you may be designing an inherent bottleneck into your system and be just asking for lock convoys as threads wait, only one being able to use the data structure at a time.
- On parallel hardware, you may also care about minimizing the cost of memory synchronization. When one thread updates one part of the data structure, how much memory needs to be moved to make the change visible to another thread? If the answer is, "More than just the part that has ostensibly changed," then again you're asking for a potential performance penalty, this time due to cache sloshing as more data has to move from the core that performed the update to the core that is reading the result.

It turns out that both of these answers are directly influenced by whether the data structure allows truly localized updates. If making what appears to be a small change in one part of the data structure actually ends up reading or writing other parts of the structure, then we lose locality; those other parts need to be locked, too, and all of the memory that has changed needs to be synchronized.

To illustrate, let's consider two common data structures: linked lists and balanced trees.