Eliminating Clutter - Introduction*

Charles Severance
Kevin Dowd

We have looked at code from the compiler's point of view and at how to profile code to find the trouble spots. This is good information, but if you are dissatisfied with a code's performance, you might still be wondering what to do about it. One possibility is that your code is too obtuse for the compiler to optimize properly. Excess code, too much modularization, or even previous optimization-related "improvements" can clutter up your code and confuse the compilers. Clutter is anything that contributes to the runtime without contributing to the answer. It comes in two forms:

Things that contribute to overhead
Subroutine calls, indirect memory references, tests within loops, wordy tests, type conversions, variables preserved unnecessarily

Things that restrict compiler flexibility
Subroutine calls, indirect memory references, tests within loops, ambiguous pointers

It's not a mistake that some of the same items appear in both lists. Subroutine calls or if-statements within loops can both bite and scratch you by taking too much time and by creating *fences* — places in the program where instructions that appear before can't be safely intermixed with instructions that appear after, at least not without a great deal of care. The goal of this chapter is to show you how to eliminate clutter, so you can restructure what's left over for the fastest execution. We save a few specific topics that might fit here, especially those regarding memory references, for later chapters where they are treated as subjects by themselves.

Before we start, we'll remind you: as you look for ways to improve what you have, keep your eyes and mind open to the possibility that there might be a fundamentally better way to do something—a more efficient sorting technique, random number generator, or solver. A different algorithm may buy you far more speed than tuning. Algorithms are beyond the scope of this book, but what we are discussing here should help you recognize "good" code, or help you to code a new algorithm to get the best performance.

*Version 1.3: Aug 25, 2010 10:26 am +0000