

Chapters 25 & 26

Code Tuning Strategies and Techniques

By: David Harrison

What is Code Tuning?

Code Tuning is the modification of code in the interest of improving performance at the cost of readability and quality.

Performance Issues

Code performance is becoming an increasingly less important field. In many cases it is cheaper to just upgrade your hardware than it is to spend large amounts of cash on more efficient code. Unless you are dealing with restricted system specs such as hand held devices, your time will be better spent improving the softwares quality and features.

Before you Tune...

If performance has become a must solve situation for your code, there are other alternatives you should pursue before you dive into tuning your code.

Before you Tune... cont.

- Review Program Design
 - Is there a fault in your basic plan?
 - Forget to feed your code monkey?
- Review Class and Routine Design
 - Are you using bulky algorithms? (bubblesort)
 - Inefficient variable types? (floats)
- Operating-System Interactions
 - System calls can be slow and bulky.
 - Unnecessary I/O usage.
 - Calls to unavailable devices timing out.

Before you Tune... cont.

- Code Compilation
 - Good compilers improve machine code optimization.
- Hardware
 - Who doesn't want a new toy?

Code Tuning Fallacies

- **Less != Better**
 - Fewer lines of code does not execute faster.
- **Guidelines don't always apply**
 - Whenever you change languages, compilers, or libraries, different operations perform differently.
- **Don't optimize as you go**
 - Actually increases to efficiency are immeasurable until the program is complete.
- **Don't be wrong faster**
 - Quality and complete code should trump performance.

Code Tuning Techniques

- Logical
 - Use short-circuit logic to stop the code when you find a solution.
 - When performing series of tests, order them by anticipated frequency.
 - Compare similar logic structures such as if-then-else statements and case.
 - Use table lookups when possible to replace complicated expressions.
 - Evaluate functions only when needed.

Code Tuning Techniques

- Loops
 - Unswitch or move decisions outside of loops.
 - Jam two loops together if they are using the same counter.
 - Unroll loops to handle multiple cases in one pass.
 - When possible, move statements outside the loop to reduce work.
 - When performing a search loop, use sentinel values to force termination.
 - When embedding loops, the busiest loop should be placed on the interior.
 - Use lower level functions inside your loops when possible. (add instead of multiply)

Code Tuning Techniques

- Data Transformations
 - Replace Floats with Integers
 - When possible, avoid multidimensional arrays.
 - Minimize number of array references
 - Create supplementary indexes. (string length index)
 - Cache commonly used variables.

Code Tuning Techniques

- Expressions
 - Use algebraic identities to enable lower level functions.
 - Use lower level algorithms when possible.
 - Create constants at compile time that can reduce routine calls.
 - Avoid system routines when possible by creating your own simplified versions.
 - Avoid type conversion of your constants.
 - Compute results in advance and save them if they are to be reused.
 - Create new variables to replace commonly used subexpressions.

Code Tuning Techniques

- Routines
 - When making routines, place them in the in-line of your code.
- Recode in a Low-Level Language
 - Rewrite any hot-spots in your code in assembler.

While Tuning

While tuning code it is vital that you perform speed tests after every modification. These speeds should be recorded precisely and repeated in order to accurately measure the increases caused by your tuning. Look for areas in the code that are traversed most often and focus on them first. Tuning rarely used code will have nominal effects. Keep backups of your code; it's very possible your changes might make things worse.

Remember

Not all Code Tuning Techniques are effective in all languages or with all compilers. You must test the performance after each change.

Code Tuning creates messy code that is difficult to read and maintain. If something goes wrong in the future, good luck figuring out where in mess of code the errors are.

Code Tuning is a last desperation move to improve performance and should only be done after the program is complete.