

PUTTING NEW TECHNOLOGY IN PERSPECTIVE

William Wong /// EMBEDDED/SOFTWARE/SYSTEMS EDITOR

OPTIMIZATION BREEDS APPLICATION EFFICIENCY

Linker and compiler work on different optimizations to create faster and smaller applications.

To create efficient applications written in a high-level language like C++, developers usually turn to compiler optimizations. Green Hills Software's latest toolchain includes a number of new compiler and linker optimizations. Together they deliver applications that are smaller and faster than the competition. Let's take a peek under the hood to see how two of these optimization techniques operate.

The linker optimization is called CodeFactor. In the simplest terms, it locates common blocks of code with an application, converts them into functions, and replaces the code with calls to the new functions as shown in the figure. Of course, it must submit to a cost/benefit analysis. Replacing a 2-byte instruction with a 4-byte call gives a net loss. Still, relatively small, duplicate blocks of a dozen instructions, depending upon the processor architecture, are very common.

As with most optimizations, CodeFactor is not free. A brute-force algorithm to find duplicate sequences is an N^2 problem. Green Hills cuts this down significantly, making CodeFactor computationally practical in large programs.

While CodeFactor's primary task is to minimize code size, it often results in faster code. It may seem counter-intuitive due to the added overhead of a call and return instruction, but keep in mind that the code involved is small.

The factored routines fit easily in a cache. They're likely to be called frequently and will typically be found in the cache. Moving code into common routines makes it more likely that the cache can keep the routines around longer. Ultimately, the level of speed improvement or overhead depends on the processor architecture and cache organization.

Code replacement isn't always easy. Details such as register usage and stack parameters must be addressed. CodeFactor's results may vary, but a 10% code compaction is attainable.

THE MAGIC OF WIZARDS • Most compiler optimizations are normally selected statically by the programmer. Green Hills' optimization wizards provide a more interactive approach, using information about the application and feedback from the developer. The CodeBalance Wizard is one of a number included in the latest compiler.

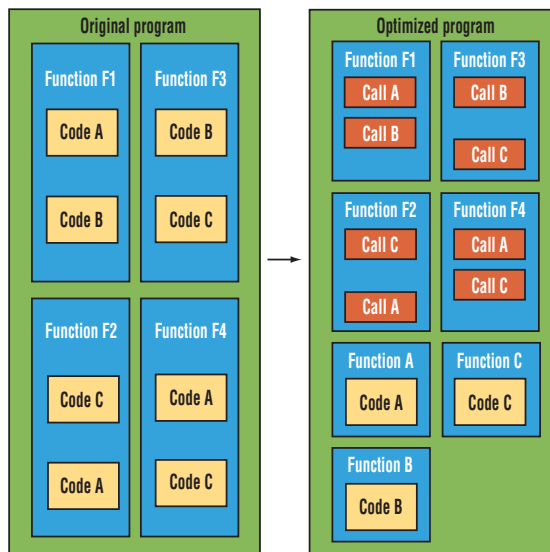
CodeBalance is well suited for embedded applications. It's designed to selectively optimize applications based on performance profile information. That information is

obtained from a prior run of an instrumented version of the program. It allows the wizard to determine which parts of the application should be optimized for speed versus space.

Running this information through the wizard results in a configuration file that specifies the suggested optimization on a "per function" basis. Developers can modify this configuration file. For example, certain functions may be required to quickly respond to events, even though profiling information may not indicate this as an important factor.

This wizard's tradeoff is very important when a developer needs to specify a maximum code size. Hardware and cost limitations are the main reasons for requiring a max code size. It is possible to experiment with different configuration files to get the best setup for a particular application.

All of these new features can be found in Green Hills' latest C and C++ compilers. The compilers are also bundled with the MULTI® Integrated Development Environment. **ED**



The Green Hills linker scans a program's object code for common code blocks and converts them into calls and new functions. Minor alterations to the code may be necessary to accommodate the subroutine call and the stack configuration.

GREEN HILLS SOFTWARE
www.ghs.com

ED Online 5681

Copyright © 2003 by Penton Media, Inc.