CodeTEST Instrumentation Overhead

Limiting Instrumentation Impact

This document discusses possible techniques for limiting the overhead incurred by CodeTEST instrumentation of source code.

Contents

Assumptions

It is assumed that the reader of this document is familiar with the basic operation of CodeTEST Tools as well as the environment in which CodeTEST Tools are used.

Acronyms and Abbreviations

Acronym/Abbreviation	Meaning
ctcc	CodeTEST Instrumenter driver
tag	Value inserted by CodeTEST Instrumenter at a point of instrumentation

Purpose

A key part of the CodeTEST Software Analysis Tool is the instrumentation of source code. This instrumentation process inserts small pieces of code into the source code during the build

CodeTEST Development Team

Revision 3

freescale semiconductor

process. This additional code introduces some overhead. This paper outlines several strategies that can be employed to help minimize instrumentation overhead in three areas.

- Code bloat – The physical size of the resulting executable can grow due to the instrumentation.
- Target runtime performance – The performance of the code running on target can be affected by the addition of instrumentation.
- Compilation time – The time required to build the code with instrumentation may increase.

Factors Affecting Overhead

There are several factors that can influence the amount of overhead introduced by source code instrumentation. These factors are not necessarily mutually exclusive and can affect each other. These factors include:

- Amount of source code being instrumented
- Complexity and constructs employed within the source code
- The instrumentation level
- Possible usage of compiler optimizations

Techniques to Reduce Overhead

Proper Level of Instrumentation

CodeTEST instrumentation can be added at branch points, function entries, function exits, and dynamic memory allocations/deallocations. Which sections get instrumented depends upon the level of instrumentation.

When performance measurements are the desired data set, set the coverage level to 1. The Instrumenter driver command is:

```
ctcc -CTtag-level=1
```

Change the instrumentation level only when coverage measurements are needed. The instrumentation levels and their corresponding command line settings are listed in order from least to most overhead.

`tag-level=0`	(no instrumentation)
`tag-level=1`	(performance)
`tag-level=2`	(statement/block coverage)
`tag-level=B`	(decision coverage)
`tag-level=A`	(modified condition/decision coverage)

The complexity of the code can have a significant impact on the amount of instrumentation that gets added to an application. Small functions with many branch points will get more instrumentation than large functions with few branch points. Adding more instrumentation to a given application will result in more overhead when running and compiling the application. It is best to specify the least amount of instrumentation that is required to obtain the desired results.

Instrumenting for dynamic memory allocations can be performed independent of the other levels of instrumentation. If only memory data is required, set `tag-level=0`.

Note: See the *Instrumenter Reference Manual* for more information on instrumentation levels.

Selective Instrumentation

Often, not all blocks of code need to be measured. In such cases, selectively instrumenting the source code can significantly reduce the overhead incurred by instrumentation. Selective instrumentation can be done at the file level or function/class level.

Selective instrumentation at the file level can be accomplished in three ways:

- Modifying the build process, such as a makefile, to pass only selected files to the Instrumenter.
- Instructing the CodeTEST Instrumenter driver to not pass certain files to the Instrumenter using a command line switch.

  ```
  -CTexclude-from-tagging=<comma_separated_list_of_filenames>
  ```

- Instructing the CodeTEST Instrumenter to selectively instrument a given file or set of files. This can be accomplished in several ways using a command line switch.

  ```
  -CTno-tag-files=<comma_separated_list_of_filenames>
  -CTonly-tag-files=<comma_separated_list_of_filenames>
  ```

Selective instrumentation at the function/class level can be accomplished in two ways:

- Instructing the CodeTEST Instrumenter to not instrument given functions using a command line switch.

  ```
  -CTno-tagging=<comma separated list of functions>
  ```

- Use pragmas within the source code to control instrumentation.

  ```
  #pragma tagging OFF  /*turns off instrumentation*/
  #pragma tagging ON   /*turns on instrumentation*/
  ```

Note: See the *Instrumenter Reference Manual* for more information on selective instrumentation.

Merging Coverage Data Sets

When taking coverage measurements on a large system, it can be advantageous to break the coverage measurement into subsets of data that can later be merged into one large data set.

1. Build the application as usual, without any CodeTEST instrumentation.
2. Save the resulting object modules in a different location, so they are not overwritten.
3. Build the application with CodeTEST instrumentation.
4. Replace one (or more) of the normal object modules with an Instrumenter module(s).
5. Build the final application.
6. Run the application and test it while collecting a new set of CodeTEST coverage data.
7. Repeat steps 4 – 6 using different instrumented object modules each time. Be sure to create a new data set each time, so you do not override the previously collected coverage data set.
8. After all instrumented modules have been tested, merge all of the collected data sets into one large coverage data set. This will result in one data set that includes all of the coverage results.

Note: See the *CodeTEST Tools User's Guide* for more information on merging coverage data.

Tag Format

Tag format refers to the way in which a CodeTEST instrumentation tag is written into the instrumented source code. This can be controlled by command line switches. In general, writing directly to a memory address can be more efficient than using a function call. Depending upon the environment, some run-time performance may be gained by having the tags written directly to memory. Note that many RTOS implementations require the use of function calls to write tags, and this method will not work. A full description on how to control the tag format can be found in the *Instrumenter Reference Manual*.

Tag Port Locations (Hardware Probes only)

The location of the tag ports can affect run-time performance. Different memory locations and devices may be faster or slower, depending upon the environment. For example, using a flash memory address can result in slow write cycles, and thus slow down the process of writing out tags. Using a faster memory location, such as an unused chip select or SRAM, can result in the process running faster.

Be aware however, that faster accesses can, in some cases, affect the CodeTEST Hardware Probe's data collection. Errors could include "FIFO overflows" if the tag rate is too great. If this occurs, use methods previously described to reduce the amount of instrumentation added to the application.

Remove Extraneous Build Output Information

Often when first installing the CodeTEST Tools, the `-CTv` and `-CTkeep` options are used to help debug the instrumentation/build process. Removing these switches may slightly speed up the build time.