Technical Report ARWSE-TR-14025

CODE SPEED MEASURING FOR VC++

Tom Nealis

October 2015

U.S. ARMY ARMAMENT RESEARCH, DEVELOPMENT AND ENGINEERING CENTER

Weapons and Software Engineering Center

Picatinny Arsenal, New Jersey

Approved for public release; distribution is unlimited.
The views, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

The citation in this report of the names of commercial firms or commercially available products or services does not constitute official endorsement by or approval of the U.S. Government.

Destroy this report when no longer needed by any method that will prevent disclosure of its contents or reconstruction of the document. Do not return to the originator.
CODE SPEED MEASURING FOR VC++

It’s often important to know how fast a snippet of code executes. This information allows the coder to make important decisions about the code that they create. Perhaps the coder is trying to find out what function call is taking so long or which code executes faster between different ways of accomplishing the same task. Regardless of the reason, faster code is typically more efficient and inherently will use less power.
<table>
<thead>
<tr>
<th>CONTENTS</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Methodology</td>
<td>1</td>
</tr>
<tr>
<td>Conclusions</td>
<td>3</td>
</tr>
<tr>
<td>Distribution List</td>
<td>5</td>
</tr>
</tbody>
</table>
INTRODUCTION

For measuring code snippets on a Windows machine with C++, it’s convenient to use the built-in, high-resolution timestamps. For native code, the function QueryPerformanceCounter is available to retrieve this timestamp. There are other versions of this function for managed and kernel-mode code, but for the purposes of this discussion, the report will concentrate on native code.

METHODOLOGY

In order to access this Application Program Interface (API), the coder is only required to include windows.h. The usage of this API can be easily understood from the following snippet of code taken from the Microsoft Developer Network (MSDN).

```
LARGE_INTEGER StartingTime, EndingTime, ElapsedMicroseconds;
LARGE_INTEGER Frequency;

QueryPerformanceFrequency(&Frequency);
QueryPerformanceCounter(&StartingTime);

// Activity to be timed
QueryPerformanceCounter(&EndingTime);

// We now have the elapsed number of ticks, along with the
// number of ticks-per-second. We use these values
// to convert to the number of elapsed microseconds.
// To guard against loss-of-precision, we convert
// to microseconds *before* dividing by ticks-per-second.
//
// ElapsedMicroseconds.QuadPart *= 1000000;
ElapsedMicroseconds.QuadPart /= Frequency.QuadPart;
```

QueryPerformanceFrequency must be called in order to get the performance-counter frequency. The frequency is returned in counts/ticks per second. This frequency does not change, so this function only needs to be called once. As the previous code shows, a timestamp is retrieved by calling QueryPerformanceCounter before and after the code to be measured.

Depending on the units desired, the value of the number of ticks is then multiplied by the appropriate value ($10^6$ for µs, $10^3$ for ms, etc.). The previous example is 1000000 in order to get the value in µs. The result is then divided by the number of ticks per second.

As a simple example of this, consider the following program:
int _tmain(int argc, _TCHAR* argv[])
{
    LARGE_INTEGER frequency;
    QueryPerformanceFrequency(&frequency);

    LARGE_INTEGER starting_time, ending_time, elapsed_microseconds;
    QueryPerformanceCounter(&starting_time);

    //put code to measure speed of here!!!
    Sleep(100);

    QueryPerformanceCounter(&ending_time);
    elapsed_microseconds.QuadPart = ending_time.QuadPart - starting_time.QuadPart;

    //this time is in micro seconds
    auto time_elapsed = static_cast<double>((elapsed_microseconds.QuadPart * 1000000.0) / frequency.QuadPart);

    printf("Time Elapsed: %4.2f\n", time_elapsed);
    printf("All done!\n");
    getchar();
    return 0;
}

All this program is going to do is measure a Sleep(100). Keep in mind that Windows is not considered a real time operating system and does not guarantee times like the amount of time a program is told to sleep. But as one can see by the output produced in figure 1, it is pretty close.

![Figure 1](image.png)

```
Figure 1
Output
```

The program was told to sleep 100 ms and 99.898 ms was measured.
CONCLUSIONS

Speed measuring in a C++ Windows environment is a relatively easy way for a programmer to gauge the relative efficiency of two comparable sets of code. Using the performance-counters provided in the Windows Application Program Interface, the coder can quickly find performance bottlenecks, measure the impact that a new piece of code will have on an existing code base, or compare the efficiency difference between two code snippets.
DISTRIBUTION LIST

U.S. Army ARDEC
ATTN: RDAR-EIK
    RDAR-WSF-M, T. Nealis
Picatinny Arsenal, NJ  07806-5000

Defense Technical Information Center (DTIC)
ATTN: Accessions Division
8725 John J. Kingman Road, Ste 0944
Fort Belvoir, VA  22060-6218

GIDEP Operations Center
P.O. Box 8000
Corona, CA 91718-8000
gidep@gidep.org

Approved for public release; distribution is unlimited.
REVIEW AND APPROVAL OF ARDEC TECHNICAL REPORTS

Code Speed Measuring for VC++4

Title

Thomas M. Nealis
Author/Project Engineer

X8048 31

Extension Building

Date received by LCSD

Report number (to be assigned by LCSD)

RDAR-WSF-M

Author's/Project Engineers Office
(Division, Laboratory, Symbol)

PART 1. Must be signed before the report can be edited.

a. The draft copy of this report has been reviewed for technical accuracy and is approved for editing.

b. Use Distribution Statement A, B, C, D, E, F, or X for the reason checked on the continuation of this form. Reason: Operational Use

1. If Statement A is selected, the report will be released to the National Technical Information Service (NTIS) for sale to the general public. Only unclassified reports whose distribution is not limited or controlled in any way are released to NTIS.

2. If Statement B, C, D, E, F, or X is selected, the report will be released to the Defense Technical Information Center (DTIC) which will limit distribution according to the conditions indicated in the statement.

c. The distribution list for this report has been reviewed for accuracy and completeness.

Patricia Alameda

DIVISION CHIEF

(Date)

PART 2. To be signed either when draft report is submitted or after review of reproduction copy.

This report is approved for publication.

Patricia Alameda

DIVISION CHIEF

(Date)

Andrew Pskowski

RDAR-CIS

(Date)

LCSD 49 supersedes SMCAR Form 49, 20 Dec 05