How-to measure CPU cycles using the time stamp counter (TSC) register

Sujesha Sudevalayam

March 27, 2014

Abstract

To compare CPU performance of different implementations of the same algorithm, or to compare performance of different algorithms that perform a certain task, we may need to measure the CPU cycles incurred for the implementation(s) and algorithm(s) concerned. Piece-by-piece comparison of individual steps in an algorithm may also be done in a simulation module, to suggest which alternative implementation has lower CPU overhead than others.

In this document, we present collated information regarding how-to use the model-specific TSC register to count the number of ticks or CPU cycles spent for any target set of instructions. For ease of use, we make available snippets of code that can be pasted into existing code to quickly perform measurement. We also present a sample use-case at the end, of measuring CPU cycles incurred for MD5 hashing and outline our learnings.

1 Motivation

In a prototype implementation or simulation module, we may need to perform comparison of individual steps, to suggest which alternative implementation has lower CPU overhead than others. When a fine measurement of number of CPU cycles [**?**] is needed for a piece of code, or when we need to compare the number of CPU cycles that two sets of instructions might take, we need a method to measure the CPU cycles expended by the system-under-test for processing, in each case. For such cases, the timestamp counter (TSC) can be accessed before and after the target set of instructions, and the difference between the two acccessed values is the number of CPU cycles expended.

The timestamp counter is a 64-bit model-specific register (MSR) present in Intel machines, beginning with the Pentium processor[**?**]. It is incremented on every clock tick, and is set to 0 upon machine reset. It can be accessed using the RDTSC (read time stamp counter) instruction in Microsoft VC++ environments. However, in Linux environment, the GCC compiler has no in-built RDTSC instruction, because of which we need to write a piece of assembly code to directly access the 64-bit value. The value of the time stamp counter needs to be loaded into the EDX and EAX registers (high-order 32 bits into EDX, low-order into EAX) and then read up from them into a local variable. Since the Linux GCC compiler does not have the RDTSC instruction, its asm opcode (0x31) has to be used so as to allow the processor to understand the instruction even if the compiler does not.

In this document, we present collated information regarding how-to use the model-specific TSC register in Linux environment, to count the number of ticks or CPU cycles spent for any target set of instructions. Our contributions in this document are the following:-

- Collation of information regarding CPU cycle measurement
- Provision of re-usable code-snippets for quick CPU cycle measurement for any target set of instructions
- Example usecase: Measurement of CPU cycles incurred for MD5 hashing.

In the next section, we present a brief survey of existing profiling tools for Linux, and delineate how they fail to meet the needs of monitoring CPU cycles. The rest of the document is organized as follows. Section 3 demonstrates how to access the time stamp counter in C code, and presents code snippets that can be quickly re-used. In Section 4, we present a sample usecase wherein, the time stamp counter is accessed to measure the number of CPU cycles incurred for MD5 hashing of given content. Section 5 concludes.

2 Other Profiling Tools

3 Counting CPU cycles using RDTSC command

```
// Hello.java
import javax.swing.JApplet;
import java.awt.Graphics;

public class Hello extends JApplet {
    public void paintComponent(Graphics g) {
        g.drawString("Hello, world!", 65, 95);
    }
}
```

4 Measuring CPU cycles for MD5 hashing

5 Conclusions