

# Whitepaper on Characterizing NAV Execution Overhead

D. Uluski, M. Moffie and D. Kaeli  
Computer Architecture Research Laboratory  
Northeastern University  
Boston, MA  
kaeli@ece.neu.edu

## **Overview**

A major objective of this project is to allow computer systems that utilize software-based anti-virus applications, to run more efficiently. Security is an important issue for all computer users today, but many users are unhappy with the performance penalty they pay for security. A significant amount of overhead is introduced by running anti-virus software on a computer; this overhead is not unexpected. However, the amount of overhead introduced is so significant that many users will defer virus checking or totally disable their anti-virus software. Thus, it becomes necessary to address the performance issues associated with anti-virus software execution. Our research is aimed at identifying the main bottlenecks, from an architectural standpoint, present in the most commonly used anti-virus software. Our work will then develop a new class of architectural assists to provide hardware acceleration for anti-virus execution. Our approach is to identify the most commonly executed operations that the anti-virus program performs and then develop extension to the instruction set (e.g., Intel's IA32) and memory system to accelerate the performance of these repetitive tasks.

## **Evaluation Environment**

We have chosen to build our studies on top of the Virtutech Simics toolset, a full machine-state architectural simulator that emulates a faithful model of an Intel Pentium 4 system that runs Windows XP professional. We have built two nearly identical system images; one configuration that has Norton Anti-Virus 2004 installed and another that does not. We use these two configurations to study and compare instruction streams, in order to identify differences between these environments.

Virtutech Simics is an architectural simulator that we use to model the Windows XP environment on a standard personal computer. Simics allows us to study the instruction stream executed by the processor, as well as the memory hierarchy, I/O execution, and other workload elements. The simulated computer we use, which is known as DREDD, models an Intel Pentium 4 processor with a rather generic motherboard containing a model of the Intel 440BX chipset. We have installed Windows XP professional version 2002 and Norton Anti-Virus (NAV) 2004 version 10.0.0.109. The goal in modeling this type of machine is to capture anti-virus on a system that represents a very commonly used configuration that is on the market today, in an effort to make the results most applicable to current use of anti-virus programs.

## **Evaluation Studies**

We begin by studying NAV by focusing on on-access scanning. We are interested in identifying the computationally intensive portions of NAV during an on-access scan. To be able to identify these code sequences, we have developed a simple test program environment. The test program, which we call test.exe,

is a skeleton of a program compiled with Visual C++. test.exe is a console program that will load itself into memory, do nothing, and unload itself from memory.

We use the special assembly code sequence **xchg bx, bx** to generate a breakpoint within the simulated environment. When the simulator executes this instruction, simulation halts while maintaining the simulated machine state. We use this mechanism to set the boundaries of an execution data collection. When we reach our first breakpoint, we load a tracing tool into memory that will track the instruction stream of the machine. When we break execution a second time, the tracing is completed.

We then post process the traced execution to produce a number of statistics. For instance, we can determine the frequency of each instruction executed (by virtual address), identifying the hottest code sequences. We can then select the most frequently executed blocks of addresses and trace them back to where in the execution environment they came from (i.e., application, OS, NAV). Simics allows us to identify the source of an access by setting a virtual memory address breakpoint. When the instruction stream approaches this virtual address, the simulation is paused and we can display the instruction stream. We continue the simulation until the virtual address is hit again, thus constructing a block.

Once we have established the block responsible for a significant portion of overhead, it is important to connect the instructions to a specific piece of executable code. We study the instruction stream presented by the simulator and concatenate all opcodes up until the end of the basic block. We have developed a utility program to disassemble executables and search by opcode for any specific instruction sequence.

Since we are mainly interested in studying instructions introduced by the anti-virus program, we try to attribute blocks of instructions to the binaries associated with NAV. We scan every binary of NAV, looking for the identical sequence of instructions.

## **Results**

In our experiments we have found many blocks that can be attributed to the on-access scanning within NAV code. We have a fairly extensive collection of hot loop execution traces from NAV. Below is an example of a block of NAV that we have captured.

```
<cs:0xf67855cc> <l:0xf67855cc> <p:0x067d75cc> 0f 85 b7 00 00 00    jne 0xf6785689
<cs:0xf6785689> <l:0xf6785689> <p:0x067d7689> ff 45 08      inc dword ptr 0x8[ebp]
<ss:0x8118767c> <l:0x8118767c> <p:0x0198767c> Vani WB Read  4 bytes 0x90
<ss:0x8118767c> <l:0x8118767c> <p:0x0198767c> Vani WB Write 4 bytes 0x91
<cs:0xf678568c> <l:0xf678568c> <p:0x067d768c> 8b 45 08      mov eax,dword ptr 0x8[ebp]
<ss:0x8118767c> <l:0x8118767c> <p:0x0198767c> Vani WB Read  4 bytes 0x91
<cs:0xf678568f> <l:0xf678568f> <p:0x067d768f> 3b 45 0c      cmp eax,dword ptr 0xc[ebp]
<ss:0x81187680> <l:0x81187680> <p:0x01987680> Vani WB Read  4 bytes 0x1076
<cs:0xf6785692> <l:0xf6785692> <p:0x067d7692> 0f 86 1a ff ff ff    jbe 0xf67855b2
<cs:0xf67855b2> <l:0xf67855b2> <p:0x067d75b2> 8b 45 08      mov eax,dword ptr 0x8[ebp]
<ss:0x8118767c> <l:0x8118767c> <p:0x0198767c> Vani WB Read  4 bytes 0x91
<cs:0xf67855b5> <l:0xf67855b5> <p:0x067d75b5> 8b 4d 14      mov ecx,dword ptr 0x14[ebp]
<ss:0x81187688> <l:0x81187688> <p:0x01987688> Vani WB Read  4 bytes 0x811876b0
<cs:0xf67855b8> <l:0xf67855b8> <p:0x067d75b8> 66 0f b6 94 06 05 2a 00 00 movzx dx, byte ptr
0x2a05[esi][eax]
<ds:0xe196aa9e> <l:0xe196aa9e> <p:0x09adda9e> Vani WB Read  1 bytes 0x8b
<cs:0xf67855c1> <l:0xf67855c1> <p:0x067d75c1> 66 3b 51 02      cmp dx,word ptr 0x2[ecx]
<ds:0x811876b2> <l:0x811876b2> <p:0x019876b2> Vani WB Read  2 bytes 0x66
<cs:0xf67855c5> <l:0xf67855c5> <p:0x067d75c5> 8d 84 06 05 2a 00 00 lea eax,0x2a05[esi][eax]
```

We have studied this pattern in some amount of detail. The instruction at virtual address 0xf67855b8 (movzx dx, byte ptr 0x2a05[esi][eax]) loads opcodes sequentially of the executable that we are running. This block of anti-virus code is processing blocks of opcodes within an executable and searching for patterns.

### **Ongoing Work**

In related work, we are performing similar studies for a number of popular anti-virus programs. We have found some major similarities in the workloads, though each takes a slightly different approach to scanning.

We are currently focusing our work in the following areas:

1. Obtaining more accurate overhead measurements.
2. Developing architectural extensions to accelerate anti-virus workloads.
3. Developing performance models to evaluate these performance benefits quantitatively.

In order to be able to develop a precise estimate of anti-virus overhead, we will need to distinguish between the different execution streams of the different processes running on the simulated machine. Those processes include the anti-virus software processes, the test program being executed and OS background processes. Although we have access to the state of all simulated hardware including the processor itself, the processor has no knowledge of the processes being executed. We will need to get a better understanding of how processes are being switched in windows and what support is offered by the hardware.

To more develop a better understanding of what is happening in NAV and other anti-virus programs, will need to understand the events that trigger on-accesses scanning and also determine how frequently the anti-virus code is being executed. Although we are able to identify frequently executed code found in anti-virus executables (.dll and .sys files), we still lack a more fundamental understanding of the underlying behavior of the flow of this code.