PERFORMANCE PROFILING
WITH INTEL® VTUNE™ PROFILER
Provides Deep Insight that Saves Time Optimizing Code

Ramesh Peri
Intel Corporation
(ramesh.v.peri@intel.com)
Agenda

- Definition of Profiling
- Kinds of Profilers
  - Instrumentation based
  - Sampling based
    - Time Based
    - Event Based
- Profiling a serial Program
- Profiling a parallel Program
- Live Vtune Demo
About Myself

• **Work at Intel® Corporation**
  • Been there for 20 years
  • Our office is on South Mopac Expway
  • I am always looking for intern candidates with good background in compilers and systems

• **I got my Ph.D from University of Virginia**

• **My background is in software tools**
  • worked on compilers, debuggers, profilers, binary analysis tools.

• **I am interested in Robotics and mentor my daughter’s Robotics team to compete in First FTC.**
  • Last year our team won FLL World championship 2nd place in Houston
Profiling

Profiling is defined as the process of **collecting events** of interest in the platform and **finding causes(s)** of those events with the intent of **understanding aspect(s)** of the platform.
Profiling in a Computing Platform

Collecting Events

- An Instruction Retired
- A Page Fault

Finding Causes

- Function Executed
- A network Packet Arrived

Understanding Aspects

- Why is my program running slow/fast
- Why is my platform consuming more/less energy than expected
- Is my program consuming more/less Memory than expected
Profilers for Software

Instrumentation based Profilers

Sampling Based Profilers

Tools touch every aspect of computing platforms:
architecture, firmware, VMs, Operating Systems, Compilers, Applications, GUIs, Databases, Cloud, Machine Learning/AI.
Instrumentation Based Profilers

Insert “Instrumentation” code at places of interest in the program

```
for (i=0;i<n;i++) {
    call loop_count(loop_id);
    <body of the loop>
}

void count_loop(int loop_id) {
    loop_count[loop_id]++;
}
```

Things to pay attention

- Manage storage for loop count array
- Different loop invocations vs. total loop invocations
Instrumentation Based Profilers

Tools provide an API to insert arbitrary code to monitor events of interest

• Pin

• DynInst
  • https://www.dyninst.org/

• DynamoRIO
  • https://dynamorio.org/
Ptrace

The `ptrace()` system call provides a means by which one process (the "tracer") may observe and control the execution of another process (the "tracee"), and examine and change the tracee's memory and registers.

This is the mechanism used by debuggers and profilers to control behavior of another process.
Ptrace based Profiling tool

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/ptrace.h>

int main ( int argc, char * argv[] )
{
    int status;
    pid_t pid;
    int num_sys_calls = 0;
    int fn_call = 0;
    if (argc != 2) {
        printf("USAGE: %s <pgm-to-monitor>\n", argv[0]);
        exit(1);
    }
    switch(pid = fork()){
    case -1:
        perror("Error with Fork");
        exit(1);
    case 0: // in child */
        ptrace(PTRACE_TRACEME, 0, NULL, NULL);
        execvp(argv[1], argv);
        default: // in parent */
        wait(&status);
        while(WIFSTOPPED(status) && WSTOPSIG(status) == SIGTRAP) {
            if(fn_call){
                in_calls;
                num_sys_calls++;
            } else{
                fn_call = 0;
                ptrace(PTRACE_SYSCALL, pid, NULL, NULL);
                wait(&status);
            }
        }
    printf("Number of System Calls=X\n", num_sys_calls);
    return 0;
}
Exercises

• Develop a strace like tool which prints all the syscalls along with the time stamp
• Develop a tool which counts the number times a given function is called in a target binary
Sampling Based Profilers

Stop program execution periodically and save the program of the program which is later processed to get information about program execution behavior.

Two Types of Sampling profilers

• Event Based Sampling
• Time Based Sampling
Time Based Sampling

Stop program execution after a “fixed time” period and save the program state which is mapped to a program construct like function or file using debug information to provide a histogram of samples

Fixed time period is provided by Operating system timers and is accessible at the user level
Event Based Sampling

Stop program execution after a “fixed number of events” and save the program state which is mapped to a program construct like function or file using debug information to provide a histogram of samples

Events counting capabilities are provided hardware inside the processor and is accessible through a device driver
History

- First Appeared in Pentium Processors in early 1990s
- Early Intel Pentium processors – 2 x PMCs as MSRs readable with RDMSR in ring 0
- Terje Mathisen reverse engineered EMON – “Pentium Secrets: Undocumented features of the Intel Pentium can give you all the information you need to optimize Pentium code” Byte Magazine, July 1994, Page 191

Intel Pentium with MMX Technology (P55C) – New CPU instructions:
- RDPMC Read Performance Monitoring Counter
- RDTSC – Read Time Stamp Counter
Kinds of Events

Many kinds of events are provided by the hardware to be monitored. A complete list is available in the Architecture manual of the processor.

- For x86 it is in Vol3b of the IA32/64 architecture manual.

There are typically ~500-600 events providing information about every aspect of the processor.
Time Based vs Events Based Sampling

<table>
<thead>
<tr>
<th>Software Collector</th>
<th>Hardware Collector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uses OS interrupts</td>
<td>Uses the on chip Performance Monitoring Unit (PMU)</td>
</tr>
<tr>
<td>Collects from a single process tree</td>
<td>Collect system wide or from a single process tree.</td>
</tr>
<tr>
<td>~10ms default resolution</td>
<td>~1ms default resolution (finer granularity - finds small functions)</td>
</tr>
<tr>
<td>Either an Intel® or a compatible processor</td>
<td>Requires a genuine Intel® processor for collection</td>
</tr>
<tr>
<td>Call stacks show calling sequence</td>
<td>Optionally collect call stacks</td>
</tr>
<tr>
<td>Works in virtual environments</td>
<td>Works in a VM only when supported by the VM (e.g., vSphere*, KVM)</td>
</tr>
<tr>
<td>No driver required</td>
<td>Uses Intel driver or perf if driver not installed</td>
</tr>
</tbody>
</table>

No special recompiles - C, C++, C#, Fortran, Java, Python, Assembly
# Some Profilers

<table>
<thead>
<tr>
<th>Tool</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>GProf</td>
<td><a href="http://sourceware.org/binutils/docs/gprof/">http://sourceware.org/binutils/docs/gprof/</a></td>
</tr>
<tr>
<td>Xcode Instruments</td>
<td><a href="https://help.apple.com/instruments/mac/current/#/dev7b09c84f5">https://help.apple.com/instruments/mac/current/#/dev7b09c84f5</a></td>
</tr>
<tr>
<td>DynInst</td>
<td><a href="https://www.dyninst.org/">https://www.dyninst.org/</a></td>
</tr>
</tbody>
</table>
Sample Program

main.c

```c
#include <stdlib.h>
#include <stdio.h>

int testFunction(int*, int);

void main() {
   int length = 2000;
   int sum = 0;
   int *A = (int *)malloc(length*(sizeof(int)));
   for (int i = 0; i < length; i++) {
      A[i] = i;
   }
   for (int i=0;i<1000000;i++)
      sum+=testFunction(A,length);
   printf("%d\n",sum);
}
```

test.c

```c
int testFunction(int* input, int length) {
   int sum = 0;
   for (int i = 0; i < length; ++i) {
      sum += input[i];
   }
   return sum;
}
```

Can be compiled gcc –O? –o main.exe -g
O0 Compilation

There are $2000 \times 11 \times 1M = 22\text{Billion} \text{ Instructions here}$
Vtune View
O2 Compilation

There are 2000 * 4 * 1M = 8 Billion Instructions here
Vtune View
Parallel Program

```c
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

void main(int argc, char *argv[])
{
    int sum=0;
    if (argc==2) {
        int num = atoi(argv[1]);
        omp_set_num_threads(num);
    }
    #pragma omp parallel for reduction(+:sum)
    for (int i=0; i<100000; i++)
        for (int j=0; j<10000; j++)
            sum+=i+j;
    printf("sum = %d\n",sum);
}
```

- Sets the number of threads
- Body of the parallel loop which will be split among the threads
Vtune View

Microarchitecture Exploration

Microarchitecture Usage: 0.0% of Pipeline Slots

The number of samples collected are not enough to build a metrics tree. Increase the time your workload runs or decrease the sampling interval to generate a metrics tree.
Live Vtune Demo
Resources

Intel® VTune™ Profiler – Performance Profiler
- Product page – overview, features, FAQs...
- Training materials – tech briefs, documentation, eval guides...
- Reviews
- Support – forums, secure support...

Additional Analysis Tools
- Intel® Inspector – memory and thread checker/debugger
- Intel® Advisor – vectorization optimization and thread prototyping
- Intel® Trace Analyzer and Collector – MPI Analyzer and Profiler

Additional Development Products
- Intel® Software Development Products

Webinars
Free in-depth presentations
- Register
- View Archives
Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

Copyright © 2018, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804