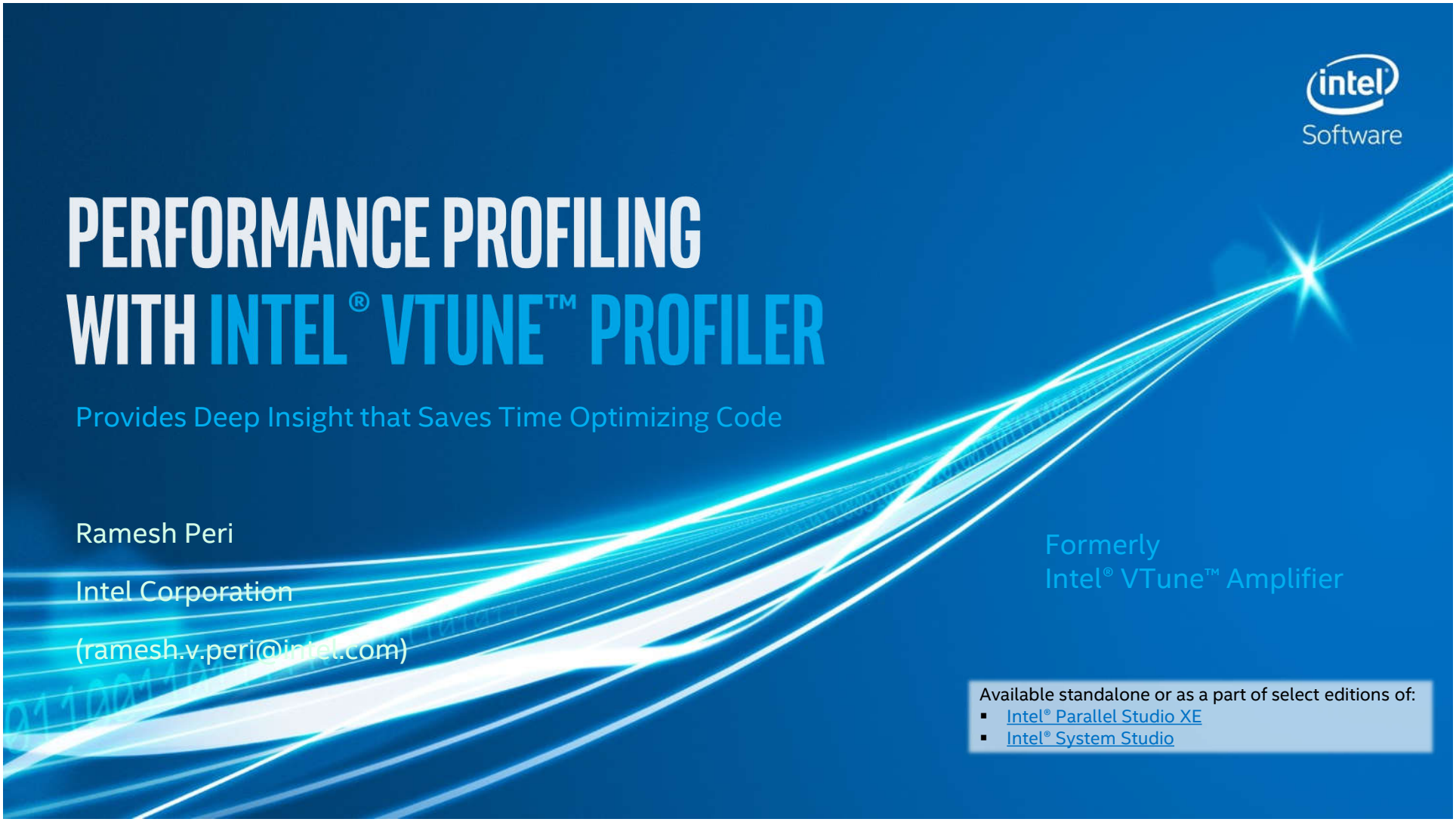

intel
Software
PERFORMANCE PROFILING
WITH INTEL® VTUNE™ PROFILER
Provides Deep Insight that Saves Time Optimizing Code
Ramesh Peri
Intel Corporation
(ramesh.v.peri@intel.com)
Formerly
Intel® VTune™ Amplifier
Available standalone or as a part of select editions of:
Intel® Parallel Studio XE
Intel® System Studio

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**
  - https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool
- **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

```c
#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 in_call=0;
  if (argc != 2) {
     printf("USAGE: %s <pgm-to-monitor>\n",argv[0]);
     exit(1);
  }
  switch(pid = fork()){
     case -1:
        perror("Eeror with fork");
        exit(1);
     case 0: /* in child */
        ptrace(PTRACE_TRACEME, 0, NULL, NULL);
        execvp(argv[1], argv+1);
     default: /* in parent */
        wait(&status);
        while(WIFSTOPPED(status) && WSTOPSIG(status) == SIGTRAP){
           if(!in_call){
              in_call=1;
              num_sys_calls++;
           } else
             in_call = 0;
           ptrace(PTRACE_SYSCALL, pid, NULL, NULL);
           wait(&status);
        }
  }
  printf("Number of System Calls=%d\n", num_sys_calls);
  return 0;
}
```

Set the child process to be traced

Launch the target binary

Toggle to distinguish syscall entry and exit

Continue child process until next syscall entry/exit

wait for child process

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**

```
ARITH.DIVIDER_ACTIVE
BACLEARS.ANY
BR_MISP_RETIRED.ALL_BRANCHES_PS
CPU_CLK_UNHALTED.ONE_THREAD_ACTIVE
CPU_CLK_UNHALTED.REF_TSC
CPU_CLK_UNHALTED.REF_XCLK
CPU_CLK_UNHALTED.THREAD
CPU_CLK_UNHALTED.THREAD_P
CYCLE_ACTIVITY.STALLS_L1D_MISS
CYCLE_ACTIVITY.STALLS_L2_MISS
CYCLE_ACTIVITY.STALLS_L3_MISS
CYCLE_ACTIVITY.STALLS_MEM_ANY
DSB2MITE_SWITCHES.PENALTY_CYCLES
DTLB_LOAD_MISSES.STLB_HIT
DTLB_LOAD_MISSES.WALK_ACTIVE
DTLB_STORE_MISSES.STLB_HIT
DTLB_STORE_MISSES.WALK_ACTIVE
EXE_ACTIVITY.1_PORTS_UTIL
EXE_ACTIVITY.2_PORTS_UTIL
EXE_ACTIVITY.BOUND_ON_STORES
```

There are typically ~500-600 events providing information about every aspect of the processor

Time Based vs Events Based Sampling

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

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

Some Profilers

Tool	Information
GProf	http://sourceware.org/binutils/docs/gprof/
Intel Vtune	https://software.intel.com/en-us/vtune
Linux Perf	https://perf.wiki.kernel.org/index.php/Main_Page
Visual Studio	https://docs.microsoft.com/en-us/visualstudio/profiling/?view=vs-2019
Xcode Instruments	https://help.apple.com/instruments/mac/current/#/dev7b09c84f5
ARM Forge	https://www.arm.com/products/development-tools/server-and-hpc/forge
HPCToolkit	http://hpctoolkit.org/
DynInst	https://www.dyninst.org/

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

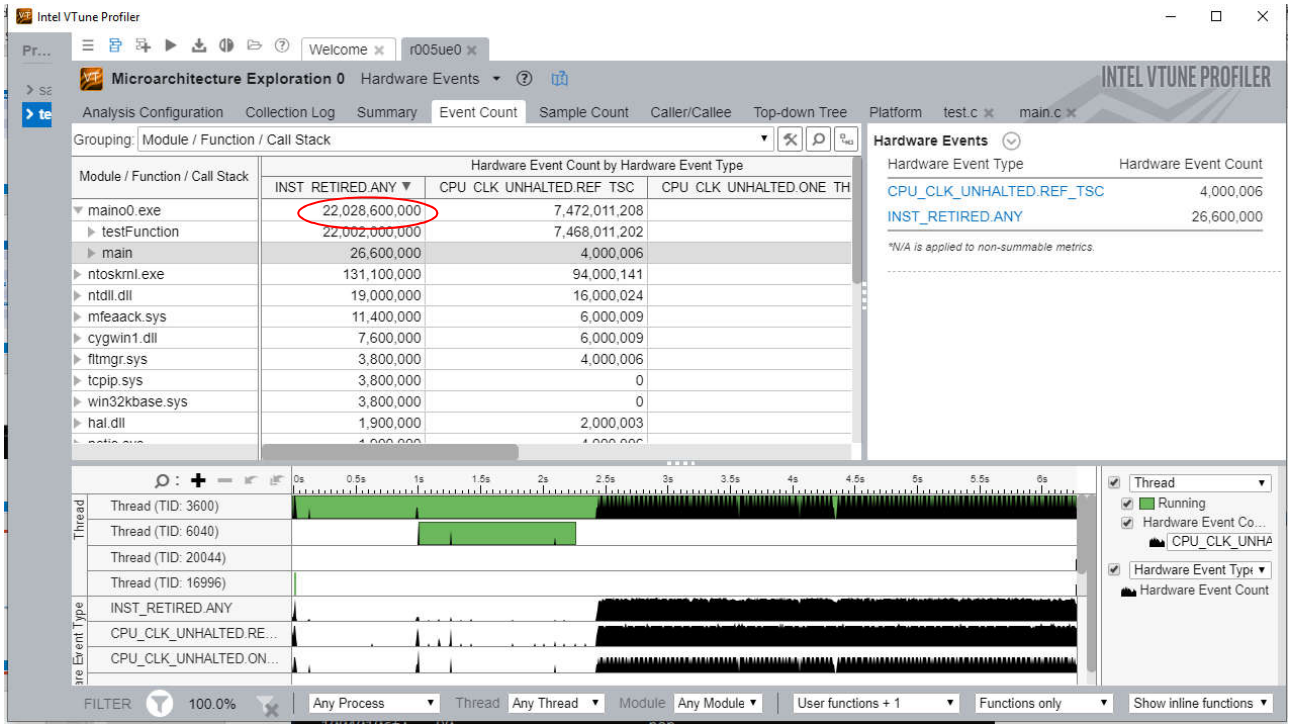
O0 Compilation

```
0000000100401080 <testFunction>:
   100401080:   55                      push   %rbp
   100401081:   48 89 e5                mov    %rsp,%rbp
   100401084:   48 83 ec 10             sub    $0x10,%rsp
   100401088:   48 89 4d 10             mov    %rcx,0x10(%rbp)
   10040108c:   89 55 18                mov    %edx,0x18(%rbp)
   10040108f:   c7 45 fc 00 00 00 00    movl   $0x0,-0x4(%rbp)
   100401096:   c7 45 f8 00 00 00 00    movl   $0x0,-0x8(%rbp)
   10040109d:   eb 1d                   jmp    1004010bc <testFunction+0x3c>
   10040109f:   8b 45 f8                mov    -0x8(%rbp),%eax
   1004010a2:   48 98                   cltq
   1004010a4:   48 8d 14 85 00 00 00    lea    0x0(,%rax,4),%rdx
   1004010ab:   00
   1004010ac:   48 8b 45 10             mov    0x10(%rbp),%rax
   1004010b0:   48 01 d0                add    %rdx,%rax
   1004010b3:   8b 00                   mov    (%rax),%eax
   1004010b5:   01 45 fc                add    %eax,-0x4(%rbp)
   1004010b8:   83 45 f8 01             addl   $0x1,-0x8(%rbp)
   1004010bc:   8b 45 f8                mov    -0x8(%rbp),%eax
   1004010bf:   3b 45 18                cmp    0x18(%rbp),%eax
   1004010c2:   7c db                   jl     10040109f <testFunction+0x1f>
   1004010c4:   8b 45 fc                mov    -0x4(%rbp),%eax
   1004010c7:   48 83 c4 10             add    $0x10,%rsp
   1004010cb:   5d                      pop    %rbp
   1004010cc:   c3                      retq
   1004010cd:   90                      nop
   1004010ce:   90                      nop
   1004010cf:   90                      nop
```

Loop with a total of 11 Instructions

There are 2000 * 11 * 1M = 22Billion Instructions here

Vtune View



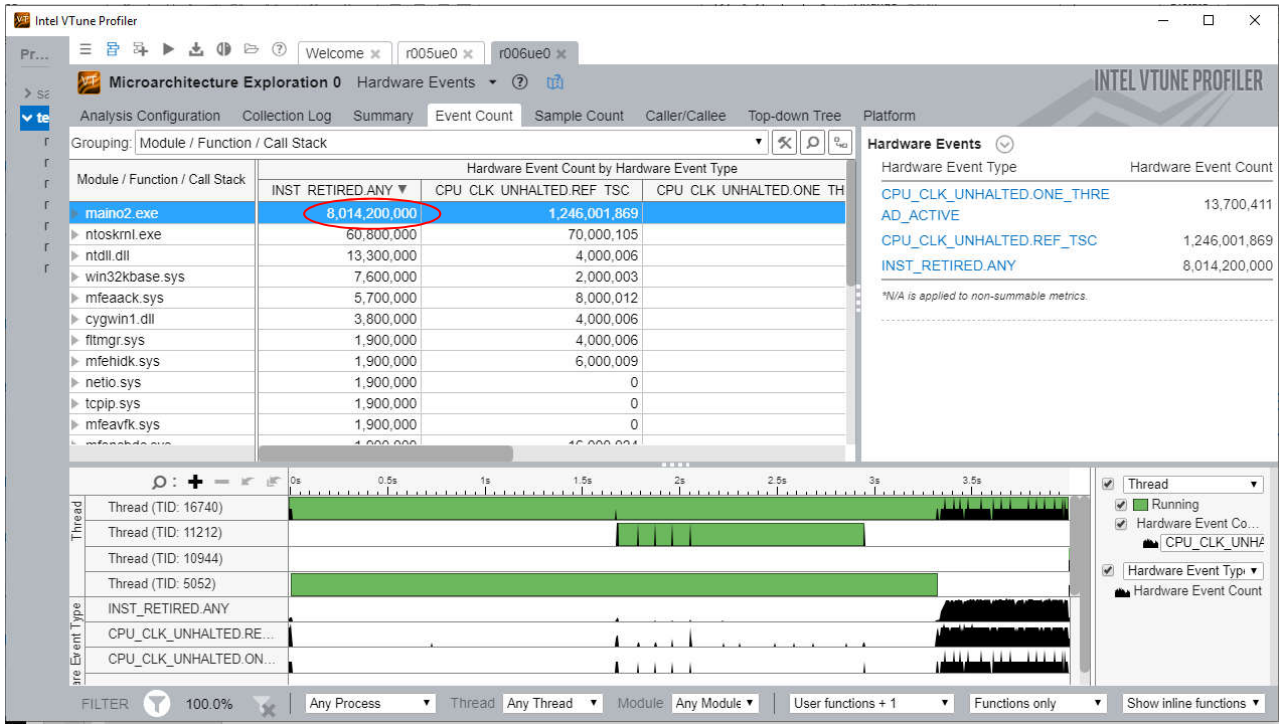
O2 Compilation

```
0000000100401080 <testFunction>:
   100401080:   85 d2                   test   %edx,%edx
   100401082:   7e 1c                   jle    1004010a0 <testFunction+0x20>
   100401084:   8d 42 ff                lea    -0x1(%rdx),%eax
   100401087:   48 8d 54 81 04          lea    0x4(%rcx,%rax,4),%rdx
   10040108c:   31 c0                   xor    %eax,%eax
   10040108e:   66 90                   xchg   %ax,%ax
   100401090:   03 01                   add    (%rcx),%eax
   100401092:   48 83 c1 04             add    $0x4,%rcx
   100401096:   48 39 d1                cmp    %rdx,%rcx
   100401099:   75 f5                   jne    100401090 <testFunction+0x10>
   10040109b:   c3                      retq
   10040109c:   0f 1f 40 00             nopl   0x0(%rax)
   1004010a0:   31 c0                   xor    %eax,%eax
   1004010a2:   c3                      retq
   1004010a3:   90                      nop
   1004010a4:   90                      nop
   1004010a5:   90                      nop
   1004010a6:   90                      nop
   1004010a7:   90                      nop
   1004010a8:   90                      nop
   1004010a9:   90                      nop
   1004010aa:   90                      nop
   1004010ab:   90                      nop
   1004010ac:   90                      nop
   1004010ad:   90                      nop
   1004010ae:   90                      nop
   1004010af:   90                      nop
```

Loop with a total of 4 Instructions

There are 2000 * 4 * 1M = 8 Billion Instructions here

Vtune View



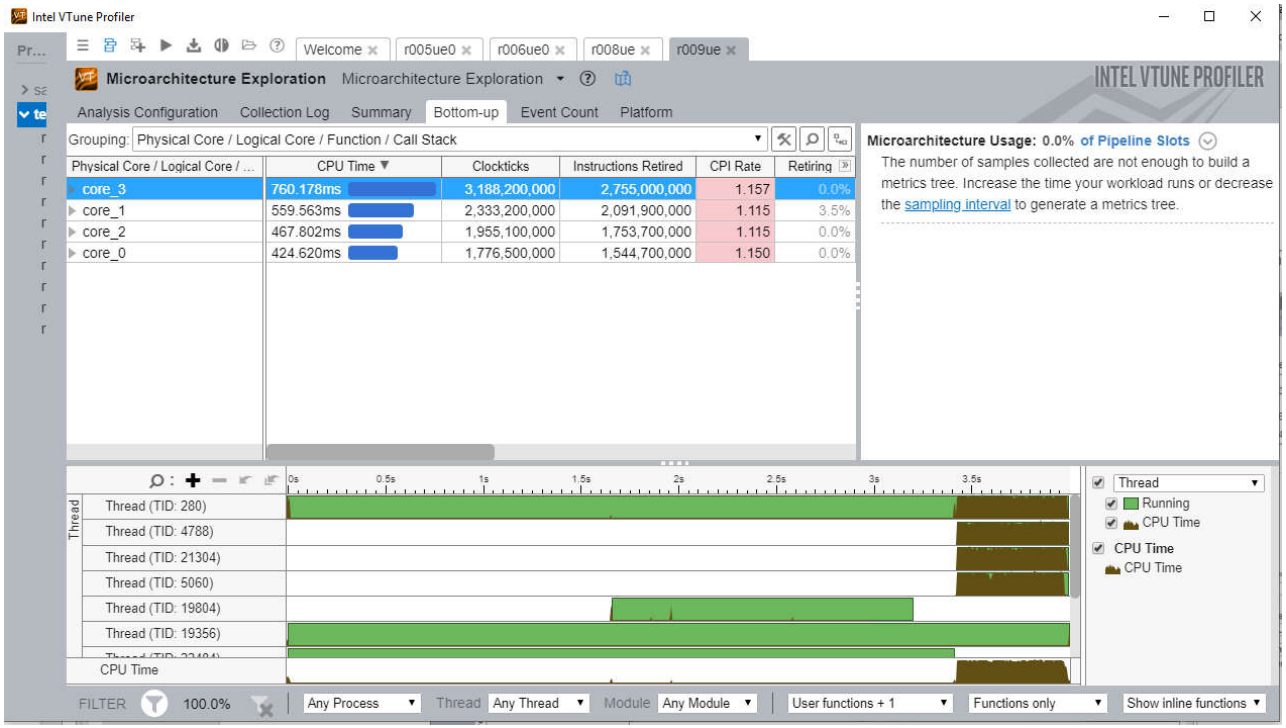
Parallel Program

```
#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



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.

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