INTEL® PROCESSOR TRACE TECHNOLOGY FOR ANOMALY DETECTION

Artem Kashkanov <artem.kashkanov@intel.com>,
Stanislav Bratanov,
Ludmila Pakhomova,
Artyom Shatalin
Periodic events stability make sense!

VR

Picture generation jitter = Angry user
Lazy transactions handling = Lost money
Packet not processed in time = Data corruption

Frame1 Frame2 Frame3 Frame4 Frame5 Frame6 Frame7 Frame8
Agenda

- Glitch and Jitter detection
- Intel® Processor Trace Technology for time critical code analysis
- Anomaly detection with Intel® VTune™ Amplifier
- Challenges
- Summary
Definition of Glitch and Jitter

Jitter – is a time deviation of the code instance duration

Glitch - is a short-lived fault in a system.
How can we find Glitch and Jitter?

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistical sampling</td>
<td>Great for steady-state</td>
</tr>
<tr>
<td></td>
<td>Can’t catch short-lived or infrequent issues</td>
</tr>
<tr>
<td>Code instrumentation</td>
<td>Good solution for debug purpose</td>
</tr>
<tr>
<td></td>
<td>Need deep code modification</td>
</tr>
<tr>
<td>Control flow recording</td>
<td>Precise/fine-grain Information</td>
</tr>
<tr>
<td></td>
<td>Big overhead, or not?</td>
</tr>
</tbody>
</table>
What is Intel® Processor Trace (Intel PT)?

HW feature for recording a full control flow with precise time and small overhead

**HANDLE**
- Indirect and conditional jumps, interrupts, exceptions, far branches..
- Asynchronous events, transactions
- VM Entries/exits

**CPL, CR3 or IP ranges filtering**

Precise timestamp events

**SUPPORTED BY**
- GDB* and WinDBG*
- Linux* perf, simple-pt
- Lauterbach JTAG*
- Intel® VTune™ Amplifier

≥Goldmont
What is Intel® Processor Trace (Intel PT)?

+ At least 1-bit per event
+ CPU State information

= Enough data for control-flow deconstruction

```
Assembly
mov  eax,offset BasicBlock
call eax
...

BasicBlock:
Loop1:
..do stuff..
jnz  Loop1

Loop2:
..more stuff..
jz   Loop2
ret
```

PT Log

<table>
<thead>
<tr>
<th>TIP: BasicBlock</th>
</tr>
</thead>
<tbody>
<tr>
<td>CYC: 8 cycles</td>
</tr>
<tr>
<td>TNT: 111110</td>
</tr>
<tr>
<td>CYC: 18 cycles</td>
</tr>
<tr>
<td>TNT: 110</td>
</tr>
<tr>
<td>CYC: 16 cycles</td>
</tr>
<tr>
<td>TIP: CALL NLST</td>
</tr>
<tr>
<td>CYC: 2 cycles</td>
</tr>
</tbody>
</table>
Quick re-cap: Intel® VTune™ Amplifier

Accurately profile C, C++, Fortran, Python, Go, Java, or a mix of coding languages.

Provides diverse data to optimize for CPU, memory, and storage.

Delivers fast answers. Rich analysis turns data into insight that saves time optimizing code.

FREE for academic and open-source!

Quick re-cap: Intel® VTune™ Amplifier

Accurately profile C, C++, Fortran, Python, Go, Java, or a mix of coding languages.

Provides diverse data to optimize for CPU, memory, and storage.

Delivers fast answers. Rich analysis turns data into insight that saves time optimizing code.

FREE for academic and open-source!


VTune Amplifier support Intel® Processor Trace technology!
“Jitter” example

Given:

- Loop with 1000 iterations
- Most times FastFrame() called, but randomly SlowFrame() executed
- 998 Fast and 2 Slow iterations
“Jitter” example

Given:

- Loop with 1000 iterations
- Most times FastFrame() called, but randomly SlowFrame() executed
- 998 Fast and 2 Slow iterations
Sampling-based Advanced Hotspot Analysis result

No Slow functions detected!
Intel PT hotspots analysis results

Set AMPLXE_EXPERIMENTAL=full-intel-pt

Correct functions detected!

2 Slow + 998 Fast
But

In general, we have no explicit Fast and Slow functions!

Just a code region of interest
Load Intel PT details by selection

<table>
<thead>
<tr>
<th>Module / Function / Call Stack</th>
<th>CPU Time</th>
<th>Clockticks</th>
<th>Instructions Retired</th>
<th>Estimated Call Count</th>
<th>Wait Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>jitter</td>
<td>26.629ms</td>
<td>106,730,069</td>
<td>128,366,509</td>
<td>1,020,015</td>
<td></td>
</tr>
<tr>
<td>Outer</td>
<td>19.768ms</td>
<td>79,228,464</td>
<td>110,013,624</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>rdsc</td>
<td>4.519ms</td>
<td>18,113,654</td>
<td>12,215,652</td>
<td>1,017,999</td>
<td></td>
</tr>
<tr>
<td>FastFrame</td>
<td>2.282ms</td>
<td>9,147,684</td>
<td>6,000,958</td>
<td>998</td>
<td></td>
</tr>
<tr>
<td>SlowFrame</td>
<td>0.051ms</td>
<td>202,432</td>
<td>120,026</td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

---

**Grouping:**
- Module
- Function
- Call Stack

**Options:**
- View Source
- What's This Column?
- Hide Column
- Show All Columns
- Select All
- Collapse All
- Expand Selected Rows
- Copy Rows to Clipboard
- Copy Cell to Clipboard
- Export to CSV

**Thread:**
- jitter (TID: 17498)

**Load Intel Processor Data by Selection**
- Filter In by Selection
- Filter Out by Selection
All region instances list in details tab

<table>
<thead>
<tr>
<th>Ipt Region / Ipt Region Instance / Function / Call Stack</th>
<th>Instructions Retired</th>
<th>Estimated Call Count</th>
<th>Total Iteration Count</th>
<th>Clockticks</th>
</tr>
</thead>
<tbody>
<tr>
<td>▼ IPT_MARK_4007a0</td>
<td>18,346,612</td>
<td>1,019,997</td>
<td>1,018,998</td>
<td>27,465,425</td>
</tr>
<tr>
<td>▼ 340</td>
<td>180,011</td>
<td>10,002</td>
<td>10,001</td>
<td>300,604</td>
</tr>
<tr>
<td>▼ 719</td>
<td>180,023</td>
<td>10,002</td>
<td>10,001</td>
<td>287,822</td>
</tr>
<tr>
<td>▼ 384</td>
<td>18,011</td>
<td>1,002</td>
<td>1,001</td>
<td>43,930</td>
</tr>
<tr>
<td>▼ 708</td>
<td>18,023</td>
<td>1,002</td>
<td>1,001</td>
<td>39,213</td>
</tr>
<tr>
<td>▼ 696</td>
<td>18,011</td>
<td>1,002</td>
<td>1,001</td>
<td>39,127</td>
</tr>
<tr>
<td>▼ 57</td>
<td>17,999</td>
<td>1,002</td>
<td>1,001</td>
<td>38,647</td>
</tr>
<tr>
<td>▼ 627</td>
<td>18,011</td>
<td>1,001</td>
<td>1,001</td>
<td>38,598</td>
</tr>
<tr>
<td>▼ 385</td>
<td>18,023</td>
<td>1,002</td>
<td>1,001</td>
<td>36,954</td>
</tr>
<tr>
<td>▼ 730</td>
<td>18,011</td>
<td>1,002</td>
<td>1,001</td>
<td>36,854</td>
</tr>
<tr>
<td>▼ 332</td>
<td>18,011</td>
<td>1,001</td>
<td>1,001</td>
<td>36,675</td>
</tr>
<tr>
<td>▼ 254</td>
<td>18,011</td>
<td>1,002</td>
<td>1,001</td>
<td>34,536</td>
</tr>
<tr>
<td>▼ 678</td>
<td>18,011</td>
<td>1,002</td>
<td>1,001</td>
<td>34,305</td>
</tr>
<tr>
<td>▼ 998</td>
<td>18,023</td>
<td>1,002</td>
<td>1,001</td>
<td>34,066</td>
</tr>
<tr>
<td>▼ 814</td>
<td>18,010</td>
<td>1,001</td>
<td>1,001</td>
<td>33,908</td>
</tr>
</tbody>
</table>

All region instances

Our 2 Slow Instances!

Slow instances!
Compare Fast and Slow region instances

<table>
<thead>
<tr>
<th>Ipt Region / Ipt Region Instance</th>
<th>Actions Retired</th>
<th>Estimated Call Count</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPT_MARK_4007a0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>340</td>
<td>180,011</td>
<td>10,002</td>
<td></td>
</tr>
<tr>
<td>719</td>
<td>180,023</td>
<td>10,002</td>
<td></td>
</tr>
<tr>
<td>rdtsc</td>
<td>120,000</td>
<td>10,000</td>
<td></td>
</tr>
<tr>
<td>[Loop at line 57 in ]</td>
<td>60,002</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>SlowFrame</td>
<td>11</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>GenerateFrame</td>
<td>10</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>384</td>
<td>18,011</td>
<td>1,002</td>
<td></td>
</tr>
<tr>
<td>708</td>
<td>18,023</td>
<td>1,002</td>
<td></td>
</tr>
<tr>
<td>696</td>
<td>18,011</td>
<td>1,002</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>17,999</td>
<td>1,002</td>
<td></td>
</tr>
<tr>
<td>627</td>
<td>18,011</td>
<td>1,002</td>
<td></td>
</tr>
</tbody>
</table>

Sort from high to low

Sort from low to high
Another example: bfutils/bfrun interpreter

- Helloworld.bfk: 142 instructions retired:
  - 130: Generate string “Hello world!” in memory
  - 12: print chars to console
Compare Fast and Slow region instances
Source and assembly view

Metrics per Basic Block
Challenges

- Intel® Processor Trace collection generates huge amount of data: 200+MB/sec per core, so billions events processing can take a lot of time
  - Tools address this by using circular buffer

- Intel® Processor Trace collection distinguishes data by cores and processes, but not by threads
  - Tools address this by catching context switches from OS
Summary

- Intel® Processor Trace (Intel PT) is great for fine-grain analysis
- Use open-Source tools with Intel PT support
- Use Intel® VTune™ Amplifier for anomaly detection

In case of any questions/feedback – feel free to contact me: Artem.Kashkanov@intel.com

Follow the link:
Legal Disclaimer and 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.

Copyright © 2017, 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
BackUp: Main Links

- Intel® VTune™ Amplifier product page
- Intel Processor Trace Technology
  - Intel Software Developer’s Manual Vol 3 Chapter 36
- A reference Processor Trace decode library (libipt)
BackUp: Linux/Windows

- Intel Processor Trace on Linux
- Adding processor trace to Linux describes the Linux perf Processor trace implementation.
- Reference documentation for PT on Linux perf
- simple-pt is an alternative reference PT implementation. It is implemented on Linux, but can be also used as a starting point to implement PT on other OS.
- The GNU debugger gdb support PT on Linux for backward debugging
- A Windows windbg processor trace plugin for debugging on Windows with PT
- A reference Processor Trace decode library
- A plugin for Linux crash to dump PT buffers (look for ptdump)