Intended Audience: Software Developers

- Interested in performance optimizing your application
  - Don’t need to be a performance expert
  - But should be an expert in the application!

- Running on a platform using the Intel® Core™ i7 Processor Family
  - This includes Intel® Core™ i7 processors and Intel® Xeon® 5500 Series processors

- Using Intel® VTune™ Performance Analyzer
  - If you have not used VTune analyzer before see the How-To at the end of this presentation
  - The performance information here applies to other tools (Intel® PTU, etc) but is focused on VTune analyzer
How to Use this Presentation

- Read through the slides once, then again while collecting data
- **For slides 18-26, Notice whether the issue applies to single-socket systems, dual-socket systems, or both!**
- Remember performance analysis is a process that may take several iterations
- Use the backup VTune™ analyzer How-To section if needed
- **Software Optimization should begin after you have:**
  - Utilized any compiler optimization options (/O2, /QxSSE4.2, etc)
  - Chosen an appropriate workload
  - Measured baseline performance
Legal Disclaimer

This document is provided in connection with Intel® products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. Except as provided in Intel’s terms and conditions of sale for such products, Intel assumes no liability whatsoever, and Intel disclaims any express or implied warranty, relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright or other intellectual property right.

Unless otherwise agreed in writing by Intel, the Intel products are not designed nor intended for any application in which the failure of the Intel product could create a situation where personal injury or death may occur.

- Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked “reserved” or “undefined.” Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.
- The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.
- Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.
- Copies of documents which have an order number and are referenced in this document or other Intel literature may be obtained by calling 1-800-548-4725 or by visiting Intel’s website.
- Hyper-Threading Technology requires a computer system with a processor supporting HT Technology and an HT Technology-enabled motherboard. Intel processors and motherboards supporting HT Technology are available at the time of publication. For more information about system requirements and to find a HT Technology-enabled motherboard and processor, visit the Intel® Web site.
- Intel® Virtualization Technology requires a computer system with an enabled Intel® processor, BIOS, virtual machine monitor (VMM) and, for some uses, certain computer system software. Please refer to your system or application vendor for more information.
- Intel® Turbo Boost Technology requires a PC with a processor with Intel Turbo Boost Technology capability. Intel Turbo Boost Technology performance varies depending on hardware, software and overall system configuration. For more information, see the Intel® Turbo Boost Technology白皮书.
- Intel, the Intel logo, VTune, Xeon, and Core i7 are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
- *Other names and brands are the property of their respective owners.
- Copyright © 2009, Intel Corporation
Agenda

– Processor Features
– The Software Optimization Cycle
  > Hotspots
  > Methods for Determining Efficiency
  > Common Architectural Causes of Inefficiency:
    • Cache Misses
    • NUMA Distribution
    • Branch Mispredicts
    • Front-end Stalls
    • Address Aliasing
    • Long Latency Instructions and Exceptions
    • DTLB Misses
    • Server/HPC Issues
Features of the Intel® Core™ i7 Processor Family

• Integrated Memory Controller
• Intel® Hyper-Threading Technology
• Intel® Turbo Boost Technology
• Intel® Smart Memory Access
• Intel® QuickPath Interconnect
• 8 MB Intel® Smart Cache
• Intel® Wide Dynamic Execution
• Intel® HD Boost
• Quad-Core Processing
• Digital Thermal Sensor

• And... a new Performance Monitoring Unit!


For Description of these Features... see...

The following Features Greatly affect Performance Monitoring...
The New Performance Monitoring Unit
-(New Counters, New Meanings)
Intel® Wide Dynamic Execution
-(this includes the OOO capabilities of the processor)
Intel® Hyper-Threading Technology
-Greatly complicates things – if you can it is easier to turn it off during performance analysis
Intel® Turbo Boost Technology
-Greatly complicates things – if you can it is easier to turn it off during performance analysis
The Features of the Integrated Memory Controller/Smart Memory Access/QPI/8 MB Intel Smart Cache
-Adds More things which can be/need to be measured on Nehalem
The 3 Fixed Counters =

- INST_RETIRED.ANY (it is recommended to use INST_RETIRED.ANY_P instead)
- CPU_CLK_UNHALTED.THREAD
- CPU_CLK_UNHALTED.REF

Note: each of these counters has a corresponding Programmable version (which will use a general purpose counter)

- INST_RETIRED.ANY_P
- CPU_CLK_UNHALTED.THREAD_P
- CPU_CLK_UNHALTED.REF_P

Note: Vtune analyzer does not support sampling uncore events, except in the case of bandwidth (slide 26). New event capabilities like load latency from PEBS records are also not supported. Intel® PTU supports some of these capabilities – see slide 27.
Both Intel® Hyper-Threading Technology and Intel® Turbo Boost Technology can be enabled or disabled through BIOS on most platforms.

Contact with the system vendor or manufacturer for the specifics of any platform before attempting this. Incorrectly modifying bios settings from those supplied by the manufacturer can result in rendering the system completely unusable and may void the warranty.

Don’t forget to re-enable these features once you are through with the software optimization process!
The Software Optimization Process

1. Identify Hotspots
   - Determine efficiency of hotspots
     > If inefficient, identify architectural reason for inefficiency

2. Optimize the issue

3. Repeat from step 1!

Note: While VTune™ analyzer can also be helpful in threading an application, this slideset is not aimed at the process of introducing threads.

The process described here could be used either before or after threading.

Intel® Thread Profiler is a feature of Intel® VTune™ Performance Analyzer for Windows that is downloaded and installed separately. It can help determine the efficiency of a parallel algorithm.

Remember for all upcoming slides – that you should only focus on hotspots! Only try to determine efficiency, identify causes, and optimize in hotspots!
For the Intel® Core™ i7 Processor Family, the CPU_CLK_UNHALTED.THREAD counter measures unhalted clockticks on a per thread basis. So for each tick of the CPU's clock, the counter will count 2 ticks if Hyper-Threading is enabled, 1 tick if Hyper-Threading is disabled. There is no per-core clocktick counter.

There is also a CPU_CLK_UNHALTED.REF counter, which counts unhalted clockticks per thread, at the reference frequency for the CPU. In other words, the CPU_CLK_UNHALTED.REF counter should not increase or decrease as a result of frequency changes due to Turbo Mode or Speedstep Technology.
% Execution Stalled and Changes in CPI methods rely on Vtune analyzer’s sampling capability. The Code Examination method relies on using Vtune analyzer’s source/disassembly view feature.
The UOPS_EXECUTED.CORE_STALL_CYCLES counter measures when the EXECUTION stage of the pipeline is stalled. This counter counts per CORE, not per thread. This has implications for doing performance analysis with Intel® Hyper-Threading enabled. For example, if Intel® Hyper-Threading is enabled, you won’t be able to tell if the stall cycles are evenly distributed across the threads or just occurring on one thread. For this reason it can be easier to interpret performance data taken while Intel® Hyper-Threading is disabled. (And enabled again after optimizations are completed).

Note: Optimized code (i.e: SSE instructions) may actually lower the CPI, and increase stall % – but it will increase the performance. CPI and Stalls – is just general guidance of efficiency... the real measure of efficiency is work taking less time... work is not something that can be measured by the CPU.
Note that CPI is a ratio! Cycles per instruction. So if the code size changes for a binary, CPI will change. In general, if CPI reduces as a result of optimizations, that is good, and if it increases, that is bad. However there are exceptions! Some code can have a very low CPI but still be inefficient because more instructions are executed than are needed. This problem is discussed using the Code Examination method for determining efficiency.

Another Note: CPI will be doubled if using Intel® Hyper-threading. With Intel® Hyper-Threading enabled, there are actually 2 different definitions of CPI. We call them "Per Thread CPI" and "Per Core CPI". The Per Thread CPI will be twice the Per Core CPI. Only convert between per Thread and per Core CPI when viewing aggregate CPIs (summed for all logical threads).

Note: Optimized code (i.e: SSE instructions) may actually lower the CPI, and increase stall % – but it will increase the performance. CPI and Stalls – is just general guidance of efficiency… the real measure of efficiency is work taking less time… work is not something that can be measured by the CPU
This method involves looking at the disassembly to make sure the most efficient instruction streams are generated. This can be complex and can require an expert knowledge of the Intel instruction set and compiler technology. What we have done is describe how to find the 2 most easy-to-detect and easy-to-fix issues in the next 2 slides.
+ (p)acked SSE instructions (i.e.: add\textbf{ps}) work on multiple data elements at a time – therefore are generally faster than (s)calar SSE Instructions (i.e.: add\textbf{ss}), which work on just one element at a time, or the generally slower x87 instructions (i.e.\!: faddp), which work on one element at a time from registers on a stack.

*Advanced Note: Another potential optimization to look for in Intel VTune Performance Analyzer on the Intel® Core i7 Processor family is sequences of instructions such as - movsd,movss,movhps,shufpd to load 16b of data – which could be replaced with the now faster movups/movupd instructions.

See slides 4-7 at http://intel.wingateweb.com/US08/published/sessions/NGMS002/SF08_NGMS002_100r.pdf for more information.

Note: When possible the Intel Compiler uses movups/movupd if you use /QxSSE4.2
Code Examination Issue 2: Failure to use New Intel® SSE4.2 instructions

- Why: New Intel® Streaming SIMD Extensions (SSE4.2) can optimize XML parsing, string parsing/searching, encryption/decryption, CRC checksum, I-SCSI, RDMA, and bit-pattern algorithms.

- How: If the functionality is in the domain of Intel SSE4.2 – drill down to Source View, & Examine the assembly for Intel SSE4.2. (which use xmm# or rm# registers and one of the following instructions:

| PCMPISTRI | CRC32 |
| PCMPESTRI | POPCNT |
| PCMPISTRM | PCMPGTQ |
| PCMPESTR  |       |

- What Now: If Intel SSE4.2 is not used, try it! Possible ways:
  - Libraries: i.e. Intel® Integrated Performance Primitives and/or Intel® Math Kernel Library
  - Intel Compiler 11.0: Use /QxSSE4.2 (Linux: -xSSE4.2)
    Note: Some LIBM and CRT library functions have been enhanced in the 11.0 compiler to use Intel SSE4.2 (strieq for example)
  - Switches for GCC (-mSSE4.2) or MS compiler (/arch:SSE2)
  - Rewrite in Assembly/Intrinsics
These are issues that result in stall cycles and high CPI. It is important to go through these in order, as we have placed the most likely problem areas in front.
Note: For these formulas, 2nd level misses and 3rd level misses are mutually exclusive. 3rd level misses do NOT include 2nd level misses. Another way to think of this is that 3rd level misses = memory hits, and 2nd level misses = 3rd level hits.
Note: This slide assumes a NUMA distribution of 80% local access on a dual-socket platform. This applies to third-level misses – when a data access is not found in the L3 cache, it must be brought from either local or remote memory. The latency will be different depending on which place the data is found – approximately 180 cycles for local memory and 320 cycles for remote memory (for a dual-socket platform). Using an estimation of 80% local access, the value of 210 cycles is used in the 3rd level miss cycles formula.

See the next slide for more information on NUMA distribution, and modify as needed according to your system’s distribution.
20-30% remote accesses may be considered normal on certain types of workloads where there is heavy data sharing (in particular update sharing). For example, database workloads. Likewise 20% remote access may be unacceptable in certain types of workloads featuring heavy array computation, such as HPC workloads. For HPC workloads remote access should be much less frequent- between 0-5%.
Branch Mispredicts

- **Why:** Mispredicted branches cause pipeline inefficiencies due to wasted work or instruction starvation (while waiting for new instructions to be fetched)
- **How:** UOPS_ISSUED.ANY, UOPS_RETIRED.ANY, BR_INST_EXEC.ANY, BR_MISP_EXEC.ANY, RESOURCESTALLS.ANY
- **What Now:**
  > 
  > ((UOPS_ISSUED.ANY/UOPS_RETIRED.ANY) − 1) tells you the fraction of execution that was wasted (due to mispredictions)
  > Instruction starvation = (UOPS_ISSUED.CORESTALL_CYCLES - RESOURCESTALLS.ANY)/CPU_CLK_UNHALTED.THREAD
  > If you have significant wasted work (> .1) or instruction starvation (> .1), examine branch misprediction percentage ((BR_MISP_EXEC.ANY/BR_INST_EXEC.ANY) * 100)
  > Try to reduce misprediction impact with better code generation (compiler, profile-guided optimizations, or hand-tuning)

Note that all applications will have some branch mispredicts. As much as 10% of branches being mispredicted could be considered normal, depending on the application. However, anything above 10% should be investigated. With branch mispredicts, it is not the number of mispredicts that is the problem but the impact. The equations above show how to determine the impact of the mispredicts on the front-end (with instruction starvation) and the back-end (with wasted execution cycles given to wrongly speculated code).
The pipeline for the Intel® Core™ i7 processor family consists of many stages. Some stages are collectively referred to as the “front-end” – these are the stages responsible for fetching, decoding, and issuing instructions and micro-operations. The back-end contains the stages responsible for dispatching, executing, and retiring instructions and micro-operations. These 2 parts of the pipeline are de-coupled and separated by a buffer (in fact buffering is used throughout the pipeline) – so, when the front-end is stalled, it does NOT necessarily mean the back-end is stalled. Front-end stalls are a problem when they are causing instruction starvation, meaning no micro-operations are being issued to the back-end. If instruction starvation occurs for a long enough period of time, the back-end may stall.
This counter only measures 4K False Store Forwarding... but if 4k False Store Forwarding is detected your code is probably being impacted by Set Associative Cache issues, 4K false Store Forwarding, and potentially even Memory bank issues – all resulting in more cache misses – and slower data access.

4k False Store Forwarding:
In order to speed things up - when the CPU is processing Stores and Loads it uses a subset of the address to determine if the store forward optimization can occur – if the bits are the same it starts to process the load. Later on – it will fully resolve the addresses and determine that the load is not from the same address as the store, resulting in the processor needing to fetch the appropriate data.

Set Associative Cache issue.
Set Associative Caches are organized into ways.... If 2 data elements are in memory at addresses which are \(2^N\) apart such that they fall in the same set – then they will take 2 entries in that set – on a 4 way set associative cache - the 5\(^{th}\) access that tries to occupy the same slot will force one of the other entries out of the cache. Turning a large Multi KB/MB cache into a N entry cache.
FP_ASSIST.ALL: It would be nice to use this counter – but that counter only works for x87 code (Some of the manuals misreport that this event works on SSE code)

UOPS_DECODED.MS: Counts the number of Uops decoded by the Microcode Sequencer, MS. The MS delivers uops when the instruction is more than 4 uops long or a microcode assist is occurring

There is no public definitive list of events/instructions that will cause this to fire – if there is a high % of uop_dedoded.ms – than try to deduce why the processor is issuing extra instructions or causing an exception in your code.

1 UOP – does not necessarily equal to one – cycle – but the result is close enough to determine if it is affecting you overall time.
On target data locality to TLB size: this is accomplished via data blocking and trying to minimize random access patterns.

Note: this is more likely to occur with server applications or applications with a large random dataset
There are many methods for diagnosing I/O latency issues, and none are perfect. The methodology above is one suggestion that can be effective in certain cases. Besides the methodology above, Vtune analyzer’s call graph functionality can be used to judge execution time for a function that could be affected by I/O, and O/S counters can also be used.

To diagnose latency of individual disks or arrays, use perfmon on Windows* systems and sar on Linux* systems.
For feedback on this presentation, contact Eric.W.Moore@intel.com or Shannon.G.Cepeda@intel.com.

Developers are encouraged to ask questions on this presentation in the VTune User Forum (link on slide).

Additional contributors(reviewers): David Levinthal, Michael Chynoweth, Vish Viswanathan
Intel® VTune™
Performance Analyzer
How-To

Instructions apply to versions 9.0 update 9 and above
Screen captures taken in Windows® environment, but apply to Linux® as well
Getting the Most out of VTune

- Make sure your application is compiled in Release Mode
- Ensure you have symbols for your application by compiling with ...
- Have a reproducible “workload” for your application and run it the same way every time you profile
  > Make sure you are running the parts of the application you want to optimize!
How To Set Up a Basic Sampling Project

- File → New Project
- Choose Sampling Wizard
- Fill in Name and Location and hit OK
How To Set Up a Basic Sampling Project

- Select the appropriate type of profiling
- Enable generation of tuning advice if desired
- Hit Next
How To Set Up a Basic Sampling Project

- Select the application to profile, fill in command-line arguments if needed, and select the working directory
- Leave the Run Activity when done with wizard box checked
- Hit Next
How To Set Up a Basic Sampling Project

• Select additional Modules of Interest if needed
  – Only needed if your application launches other processes you want to sample
• Hit Next
How To Set Up a Basic Sampling Project

- Leave the Sampling Mechanism radio button at default
  - Change Stop collection checkbox to sample application for fixed time period if needed
- Hit Finish
How To Set Up a Basic Sampling Project

- Vtune™ analyzer will launch your application
- Wait for the data collection to complete and load
- Optionally rename the Activity using the Tuning Browser
- View the data (see next How-To)
How To Interpret the Data

- Understand the interface

Tuning Browser allows access to each sampling activity

Center pane has spreadsheet-like view of all data

Event pane shows list of events and their values for the highlighted process in the center pane
How To Interpret the Data

- The Sampling Wizard profiles using 2 events:
  - CPU_CLK_UNHALTED – CPU cycles when a core is active
  - INST RETIRED – instructions executed
- All active processes on the system will be sampled!

<table>
<thead>
<tr>
<th>Process</th>
<th>CPU_CLK_UNHALTED</th>
<th>INST RETIRED</th>
<th>Clocks per Instruction</th>
<th>CPI automatically calculated</th>
<th>Events columns show actual numbers of events, calculated by multiplying the number of samples times sampling frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>services.exe</td>
<td>111</td>
<td>0.49%</td>
<td>70.12%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AcroRead.exe</td>
<td>66</td>
<td>0.33%</td>
<td>70.21%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vnunetv.exe</td>
<td>57</td>
<td>0.29%</td>
<td>70.21%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pdq_decode</td>
<td>46</td>
<td>0.23%</td>
<td>70.21%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>inc_field.exe</td>
<td>45</td>
<td>0.23%</td>
<td>70.21%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S24e-Mon.exe</td>
<td>38</td>
<td>0.19%</td>
<td>70.21%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ICDESK.exe</td>
<td>37</td>
<td>0.19%</td>
<td>70.21%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ianvsv.exe</td>
<td>35</td>
<td>0.18%</td>
<td>70.21%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Explorer.exe</td>
<td>25</td>
<td>0.13%</td>
<td>70.21%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BESClient.exe</td>
<td>23</td>
<td>0.12%</td>
<td>70.21%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How To Interpret the Data

- In order to minimize impact to the system, samples are only taken periodically.
- For an event like CPU_CLK_UNHALTED (an active CPU cycle), a sample might be taken only after every 2,000,000 events.
  - The sampling frequency (i.e., 2,000,000) is known as the SAV (Sample After Value).
  - Different events may have different SAVs.
  - For a particular event, you can choose to concentrate on either samples or events to determine hotspots.

<table>
<thead>
<tr>
<th>Process</th>
<th>CPU_CLK_UNHALTED samples</th>
<th>INST_RETIRED samples</th>
<th>Clocks per Instruction</th>
<th>CPU_CLK_UNHALTED %</th>
<th>INST_RETIRED %</th>
<th>CPU_CLK_UNHALTED events</th>
<th>INST_RETIRED events</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobilePlayerExample</td>
<td>9,490</td>
<td>4,101</td>
<td>2.72</td>
<td>49.75%</td>
<td>70.21%</td>
<td>121,000,000</td>
<td>21,426,000,000</td>
</tr>
<tr>
<td>ThunderboltExample</td>
<td>111</td>
<td>71</td>
<td>1.52</td>
<td>9.62%</td>
<td>5.08%</td>
<td>227,000,000</td>
<td>12,000,000</td>
</tr>
</tbody>
</table>

9,490 samples * 2,000,000 SAV = 18,980,000,000 events occurred
How To Interpret the Data

Navigating through the data

Double-click a row in the center pane to drill down to a new view of the data.

Functions will be sorted in descending order; functions with the most samples of the CPU_CLK_UNHALTED event are called “hotspots”.

Drill down to thread, then module, then hotspot view.
How To Interpret the Data

- Double-click a function in the hotspot view to view source code

Switch between source, assembly, and combined views

View samples for each line of code
How To Set Up Event-Based Sampling

- Set up a project using the Sampling Wizard in the same way as a Basic project until step 3
- Check the *Modify default configuration* box
How To Set Up Event-Based Sampling

- Continue through the next screens and click Finish
- In the Advanced Activity Configuration dialog, choose Sampling from the Data Collectors listbox and hit the Configure button
How To Set Up Event-Based Sampling

- In the Configure Sampling dialog, go to the Events tab
- Select Events from the list, and click Add
- An appropriate SAV is determined based on the CPU frequency, but it can be changed if desired
- Vtune analyzer will run the application more than once depending on the number of events selected

Two new events have been added, and SAV values automatically set
Sometimes more than one run of the application will be needed
How To Set Up Event-Based Sampling

• After selecting your events, hit OK on the Configure Sampling dialog
• Hit OK on the Advanced Activity Configuration dialog
• VTune™ analyzer will run your application 1 or more times while collecting data
• After the data loads, rename the activity if desired and analyze the results