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!
- Working on a platform with a 2nd generation Intel® Core™ processor
- Using Intel® VTune™ Amplifier XE performance analyzer
  - >The performance information here applies to other tools (PTU, etc) but is focused on VTune Amplifier XE

How to Use this Presentation

- Read through the slides once, then again while collecting data
- Remember performance analysis is a process that may take several iterations
- Software Optimization should begin *after you have*:
  - >Utilized any compiler optimization options (/O2, /QxSSE4.2, etc)
  - >Chosen an appropriate workload
  - >Measured baseline performance

Using Intel® VTune™ Amplifier XE to Tune Software on the 2nd generation Intel® Core™ processor family

Software and Services Group

Ver. 1.01

http://software.intel.com/en-us/articles/using-intel-vtune-amplifier-xe-to-tune-software-on-the-2nd-generation-intel-core-processor-family/

Legal Disclaimer

INFORMATION IN 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 SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES 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.
- Intel® Hyper-Threading Technology requires a computer system with a processor supporting HT Technology and an HT Technology-enabled chipset, BIOS and operating system. Performance will vary depending on the specific hardware and software you use. For more information including details on which processors support HT Technology, see here
- 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 enabled for it. Functionality, performance or other benefits will vary depending on hardware and software configurations and may require a BIOS update. Software applications may not be compatible with all operating systems. Please check with your application vendor.
- 64-bit computing on Intel architecture requires a computer system with a processor, chipset, BIOS, operating system, device drivers and applications enabled for Intel® 64 architecture. Performance will vary depending on your hardware and software configurations. Consult with your system 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. Check with your PC manufacturer on whether your system delivers Intel Turbo Boost Technology. For more information, see http://www.intel.com/technology/turboboost."
- Intel, the Intel logo, VTune, inTru, and Core 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.
-

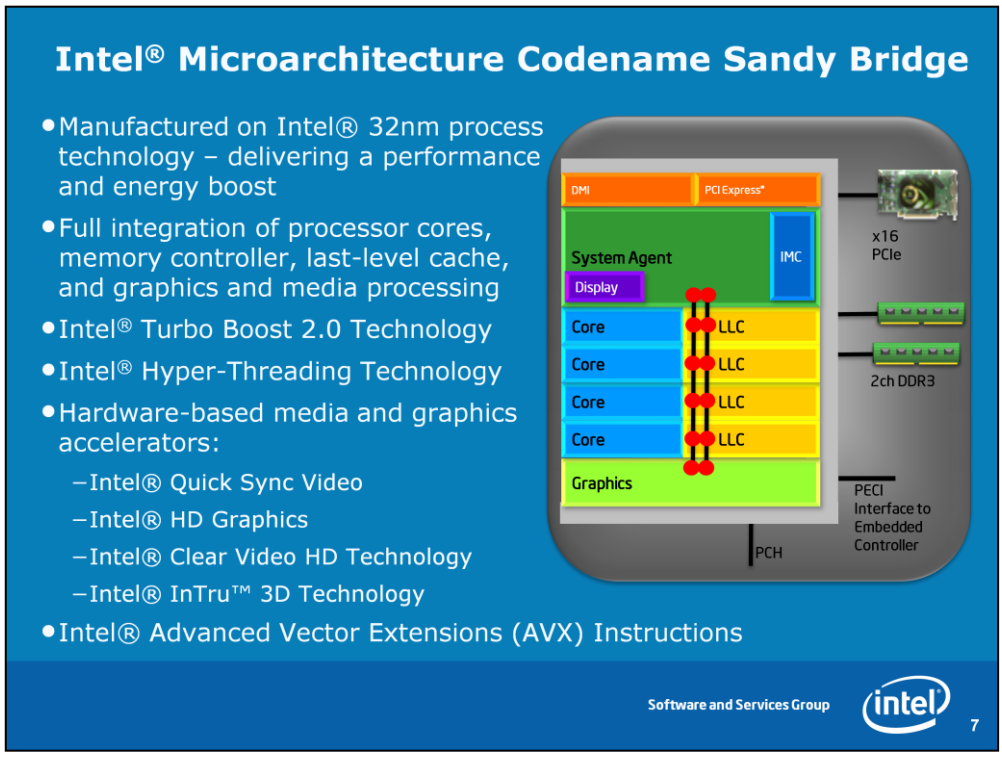
Optimization Notice

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

Agenda

- –Intel® Microarchitecture Codename Sandy Bridge
- –The new Intel® VTune™ Amplifier XE
- –The Software Optimization Cycle
  - >Hotspots
  - >Methods for Determining Efficiency
  - >Common Architectural Causes of Inefficiency:
    - **Cache Misses**
    - Other Data Access Issues
    - Execution Stalls
    - Branch Mispredicts
    - Front End Stalls

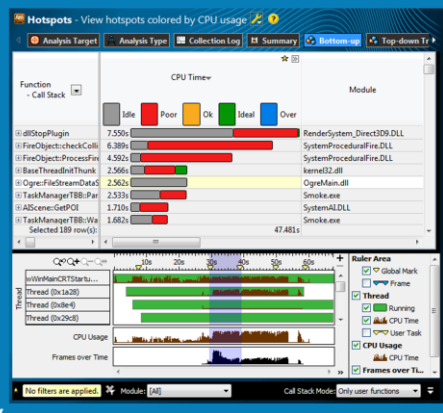


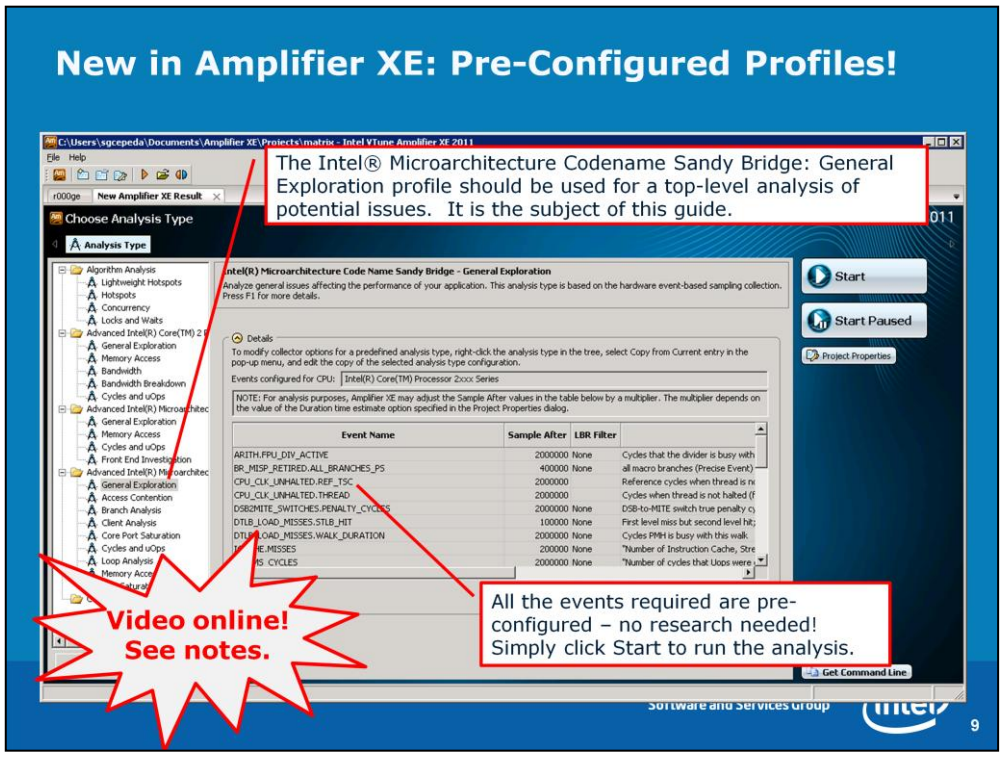
For how to tell which product numbers are Sandy Bridge, check here: http://www.intel.com/products/processor_number/about/core.htm

For more on these features see http://www.intel.com/consumer/products/processors/core-family.htm or http://www.intel.com/technology/architecture-silicon/2ndgen/index.htm

The New Intel® VTune™ Amplifier XE

- The latest version of VTune Analyzer was released in November 2010
- VTune Amplifier XE features:
  - Multiple Collection Types
    - >Hotspots (statistical call tree)
    - >Thread Concurrency
    - >Locks and Waits Analysis
    - >Event-based Sampling
  - Timeline View Integrated into all Analysis Types
  - Source/Assembly Viewing
  - Compatible with C/C++, Fortran, Assembly, .NET
  - Visual Studio Integration, Command-line, or Standalone interface for Windows* or Linux*



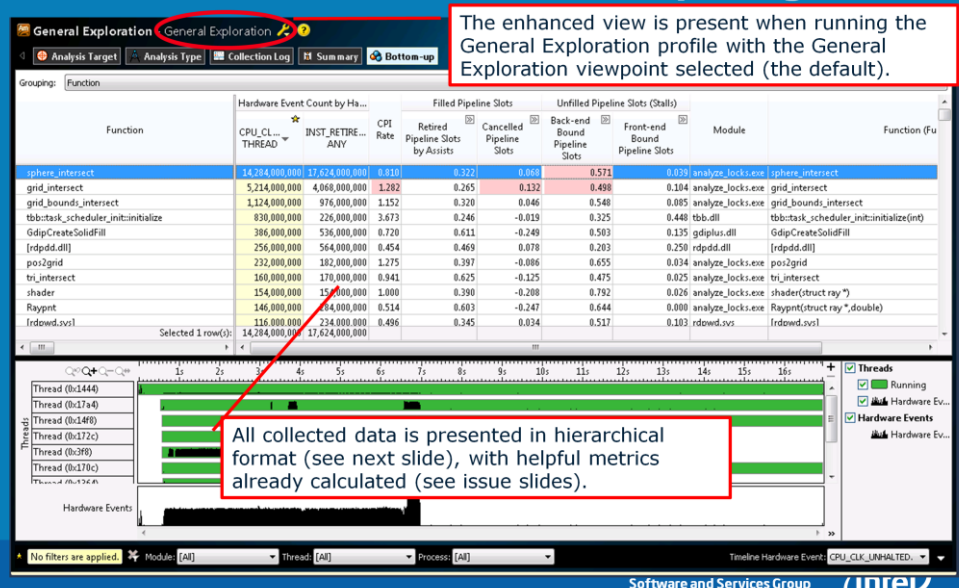


The logic for identifying issues on Microarchitecture Codename Sandy Bridge is embedded into the interface. All the formulas and metrics used are the same as the ones given in this guide. You no longer have to apply formulas and rules to the data yourself to figure out what it means – using this guide and the interface tuning features, you can easily pinpoint problems and possible solutions.

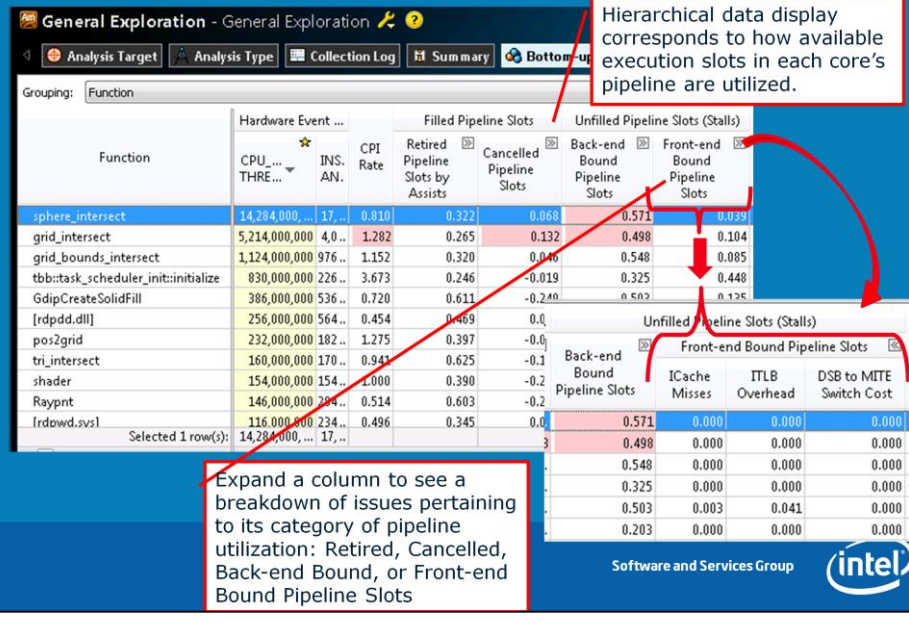
The formulas and metrics are only applied to the General Exploration profile, and the General Exploration viewpoint (which is automatic). For the other profiles, it will just show the raw data.

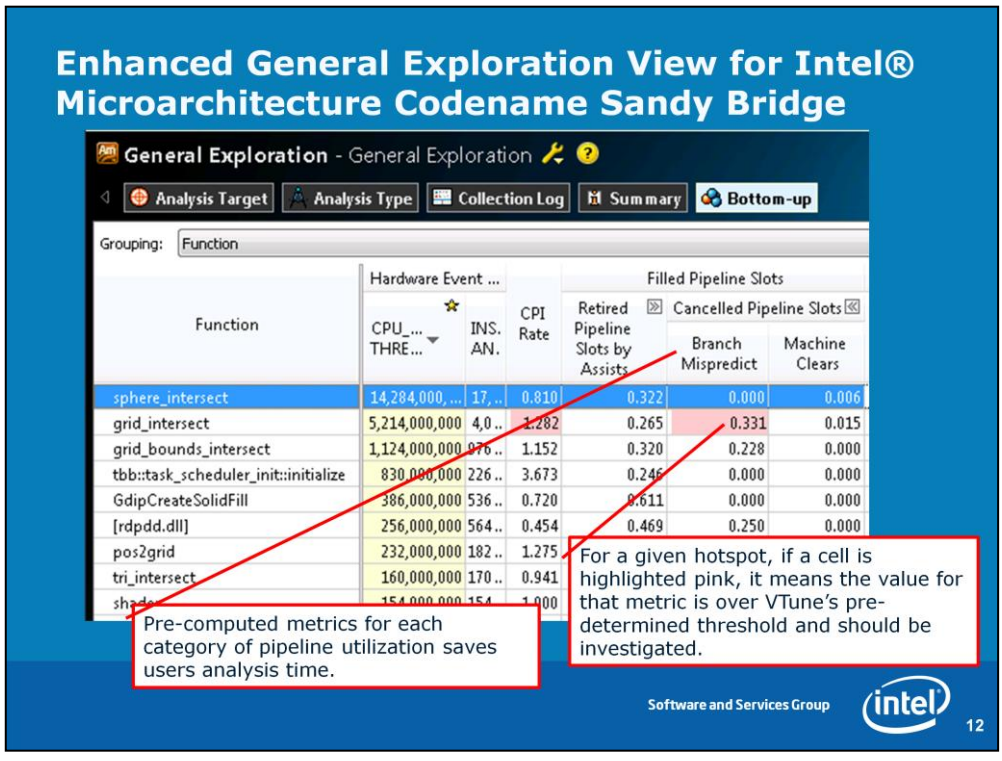
Also view our video demo of this interface at:
http://software.intel.com/en-us/videos/channel/parallel-programming/the-intel-vtune-amplifier-xe-analysis-and-results-interface-for-intel-microarchitecture-codename-sandy-bridge/1265162566001

Enhanced General Exploration View for Intel® Microarchitecture Codename Sandy Bridge



Enhanced General Exploration View for Intel® Microarchitecture Codename Sandy Bridge

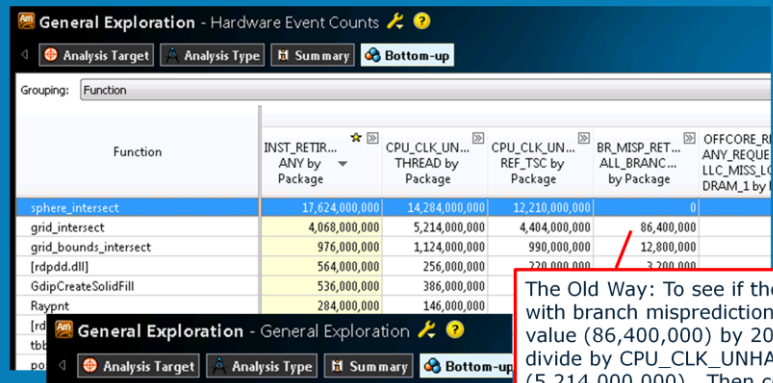


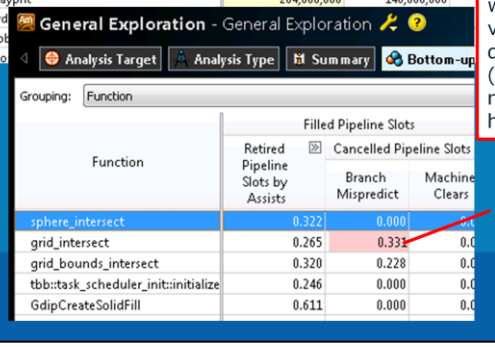


Note that issue highlighting occurs under 2 conditions:

1. The value for the metric is over VTune's pre-determined threshold
2. The associated function uses 5% or greater of the CPU clockticks sampled

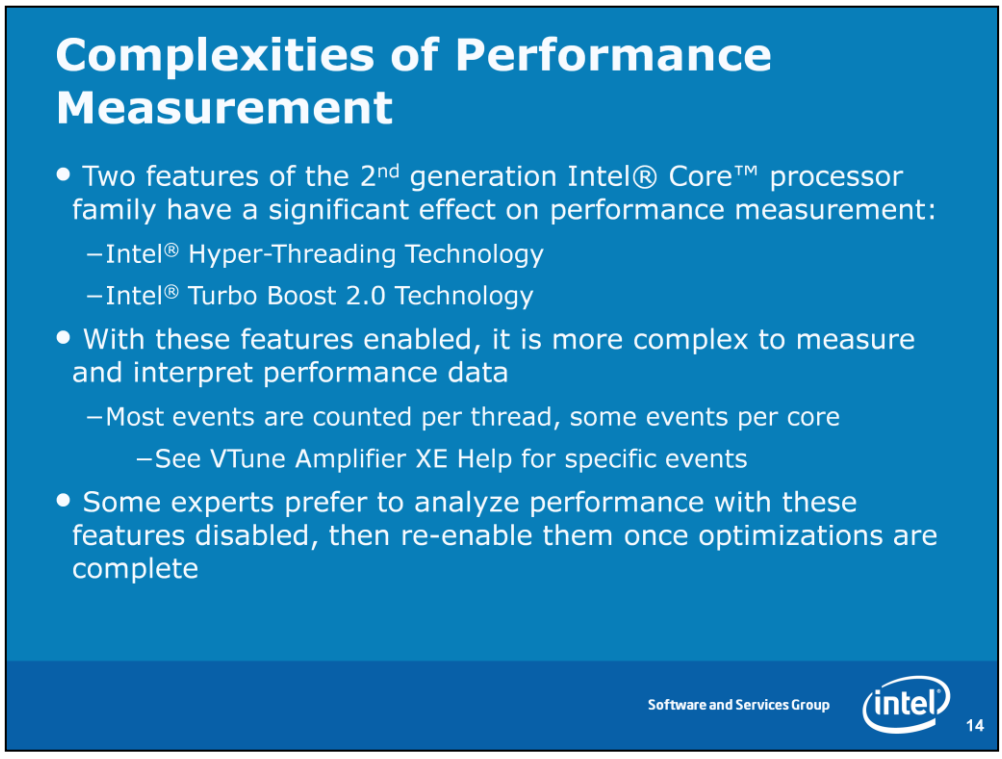
The Old Way vs. The New Way





The Old Way: To see if there is an issue with branch misprediction, multiply event value (86,400,000) by 20 cycles, then divide by CPU_CLK_UNHALTED.THREAD (5,214,000,000). Then compare the resulting value to a threshold. If it is too high, investigate.

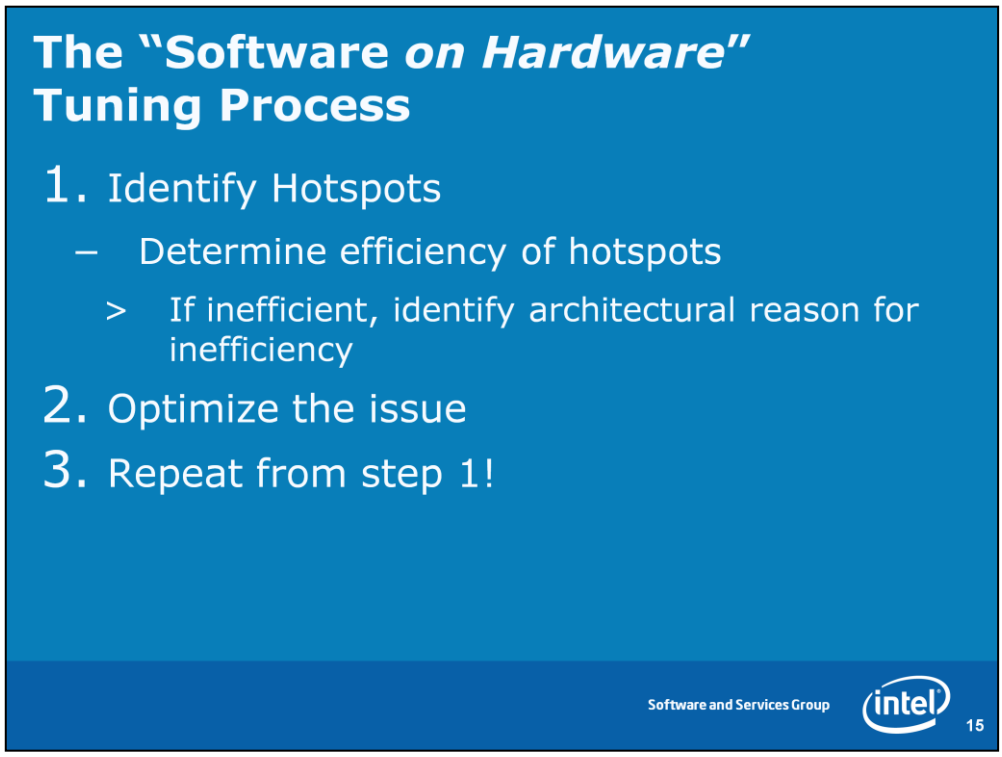
The New Way: Look at the Branch Mispredict metric, and see if any cells are pink. If so, investigate.



Both Intel® Hyper-Threading Technology and Intel® Turbo Boost 2.0 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!

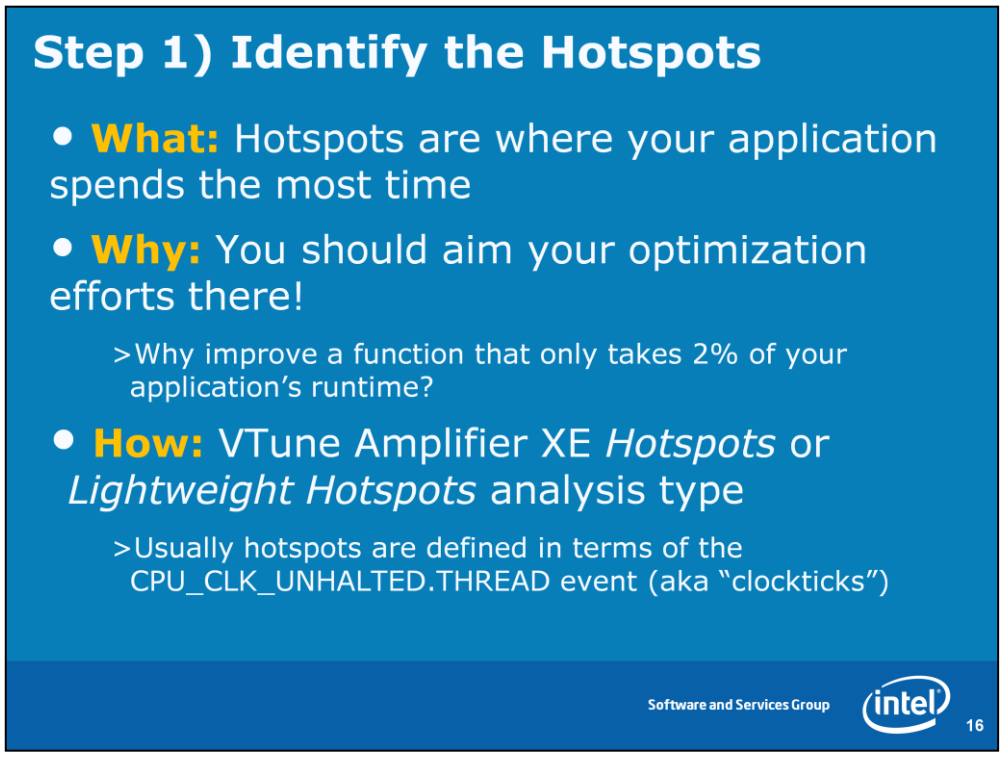


Note: While VTune Amplifier XE's Concurrency, Timeline and Locks and Waits features can also be helpful in threading an application, this slideset is not aimed at the process of introducing threads.

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

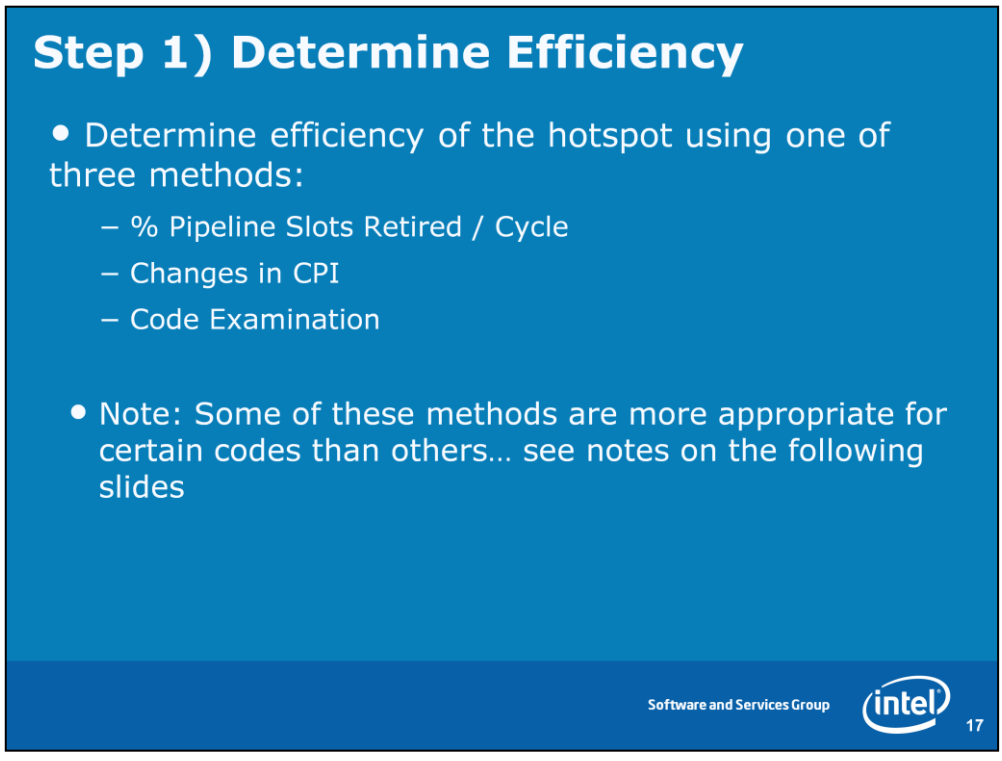
However, we *do* recommend that you follow a top-down process when optimizing: beginning with system tuning (if appropriate), then algorithmic tuning, then microarchitectural tuning. The name of Software on Hardware tuning just means we are tuning software for specific hardware.

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 2nd generation Intel® Core™ 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.

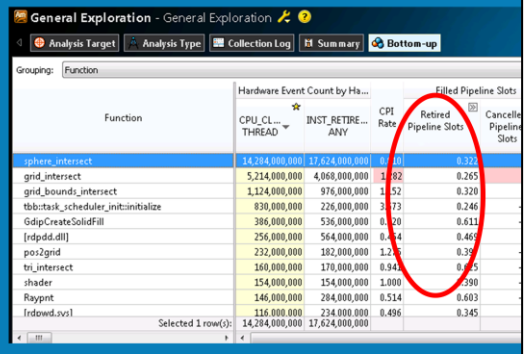
There is also a CPU_CLK_UNHALTED.REF counter, which counts unhalted clockticks per thread, at the reference frequency for the CPU. In other words, the CPU_CLK_UNHALTED.REF counter should not increase or decrease as a result of frequency changes due to Turbo Mode 2.0 or Speedstep Technology.



% Pipeline Slots Retired and Changes in CPI methods rely on VTune Amplifier XE's event-based sampling. The Code Examination method relies on using VTune Amplifier XE's capability as a source/disassembly viewer.

Efficiency Method 1: % Retired Pipeline Slots / Cycle

- **Why:** Helps you understand how efficiently your app is using the processors
- **How:** General Exploration profile, Metric: *Retired Pipeline Slots*
- **What Now:**
  - For a given hotspot:
  - In general, > 90% retiring (.9 or higher) is good. Go to efficiency method 3.
  - 50 - 90% for client apps - Consider investigating stall reduction using the following issue slides.
  - < 60% for server apps – consider investigating stall reduction.



Formula:

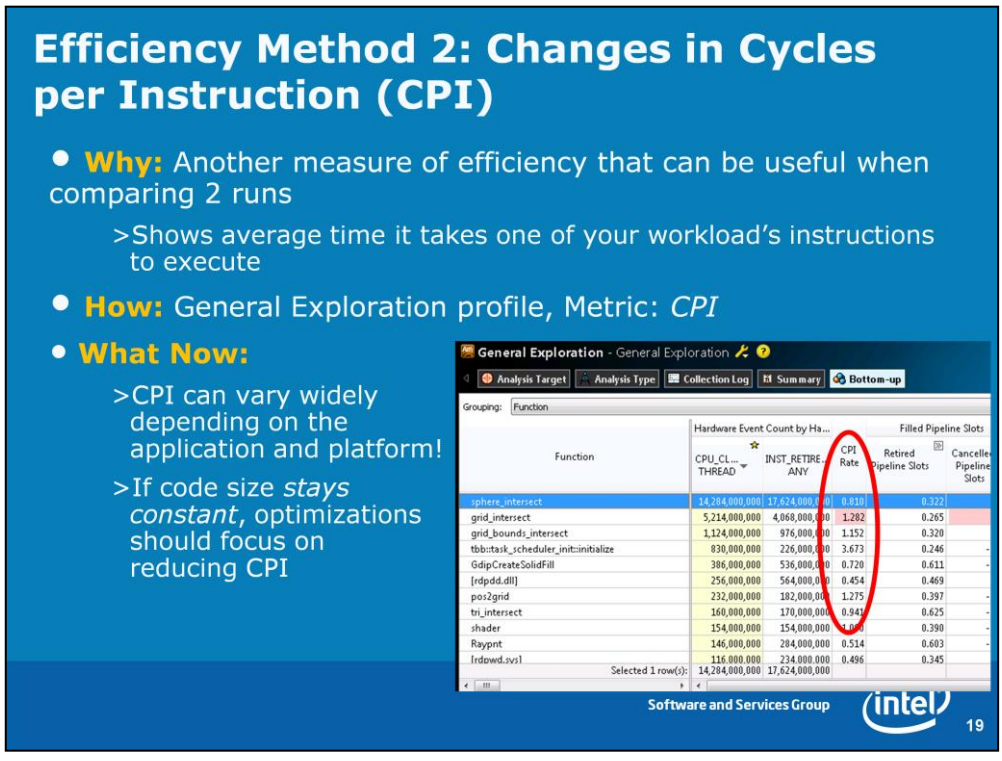
(UOPS_RETIRED.RETIRE_SLOTS/
(4*CPU_CLK_UNHALTED.THREAD))

Thresholds: Investigate if -

% Retiring < .9

This metric is based on the fact that when operating at peak performance, the pipeline on a 2nd generation Core CPU should be able to retire 4 micro-operations per clock cycle (or "clocktick"). The formula looks at "slots" in the pipeline for each core, and sees if the slots are filled, and if so, whether they contained a micro-op that retired.

The thresholds are general guidelines. Depending on the domain, some applications can run with less slots allocated retiring than the thresholds above and still be very efficient. For example, it is common for database workloads to be running with only 20-25% of allocated slots retiring per clocktick (due to heavy I/O).



Formula:

CPU_CLK_UNHALTED.THREAD/INST_RETIRED.ANY

Threshold:

In the interface, CPI will be highlighted if > 1. This is a very general rule based on the fact that some very well tuned apps achieve CPIs of 1 or below. However, many apps will naturally have a CPI of over 1 – it is very dependent on workload and platform. It is best used as a comparison factor – know your app's CPI and see if over time it is moving upward (that is bad) or reducing (good!).

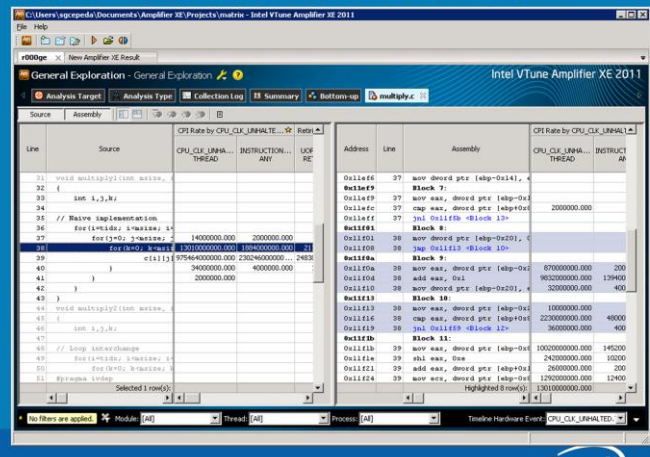
Note that CPI is a ratio! Cycles per instruction. So if the code size changes for a binary, CPI will change. In general, if CPI reduces as a result of optimizations, that is good, and if it increases, that is bad. However there are exceptions! Some code can have a very low CPI but still be inefficient 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).

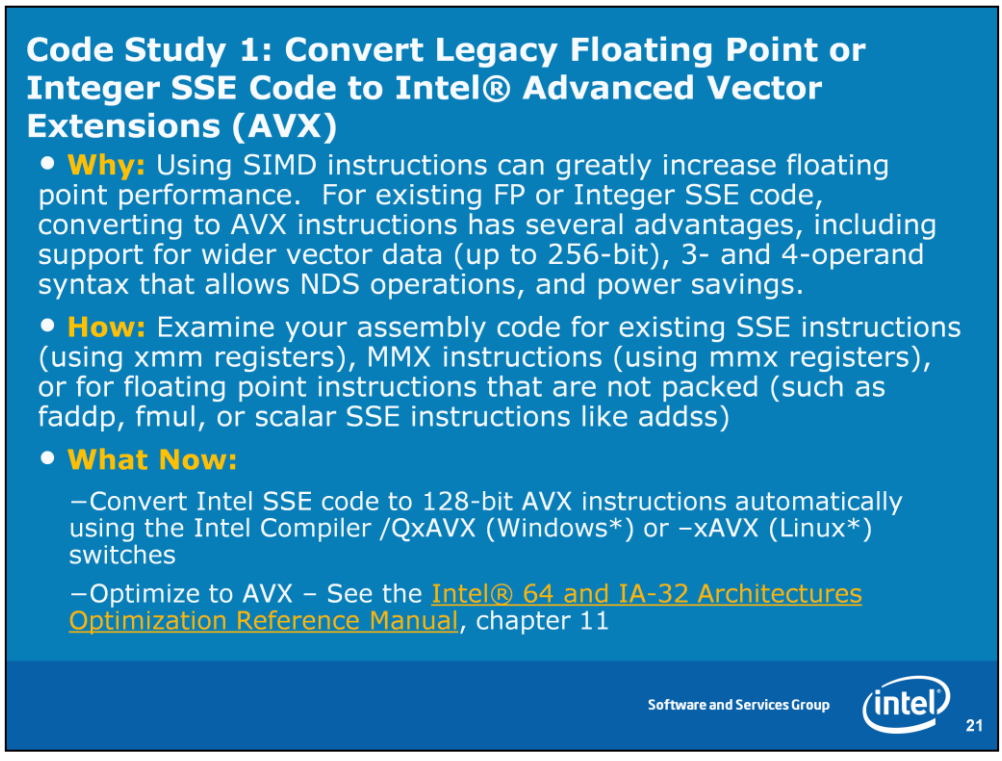
Note: Optimized code (i.e: SSE instructions) may actually lower the CPI, and increase stall % – but it will increase the performance. CPI is just a general efficiency metric – the real measure of efficiency is work taking less time.

Efficiency Method 3: Code Examination

- **Why:** Methods 1 and 2 measure how long it takes instructions to execute. The other type of inefficiency is executing too many instructions.
- **How:** Use VTune Amplifier XE's capability as a source and disassembly viewer
- **What Now:**
  - Failure to utilize modern instructions results in larger code size
  - See next 2 slides for potential issues



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 2 easy-to-detect issues and suggest how they may be fixed using new features of Intel® Microarchitecture Codename Sandy Bridge.



For more on AVX, see: http://software.intel.com/en-us/articles/intel-avx-new-frontiers-in-performance-improvements-and-energy-efficiency/

SSE instructions will look like: addps xmm4, xmm5.

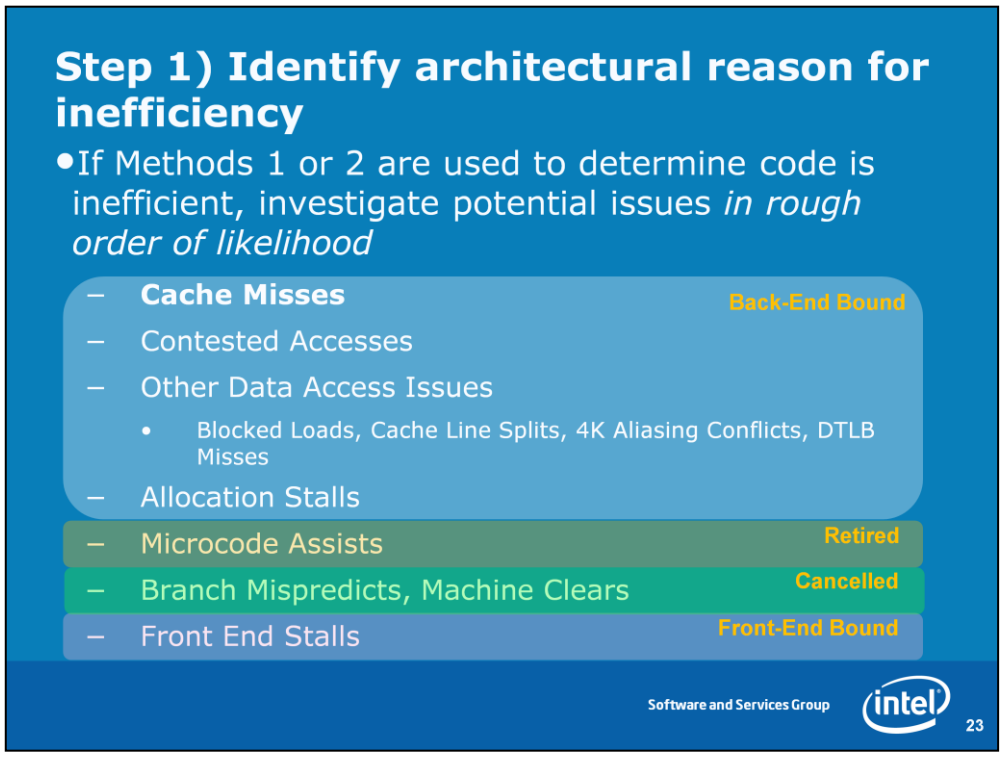
+ add**s**s is a **s**(calar) Intel® SSE instruction – which is better than x87 instructions- but is not as good as packed SSE instructions.

Code Study 2: Take Advantage of Improvements in Intel® Advanced Encryption Standard (AES) Instructions

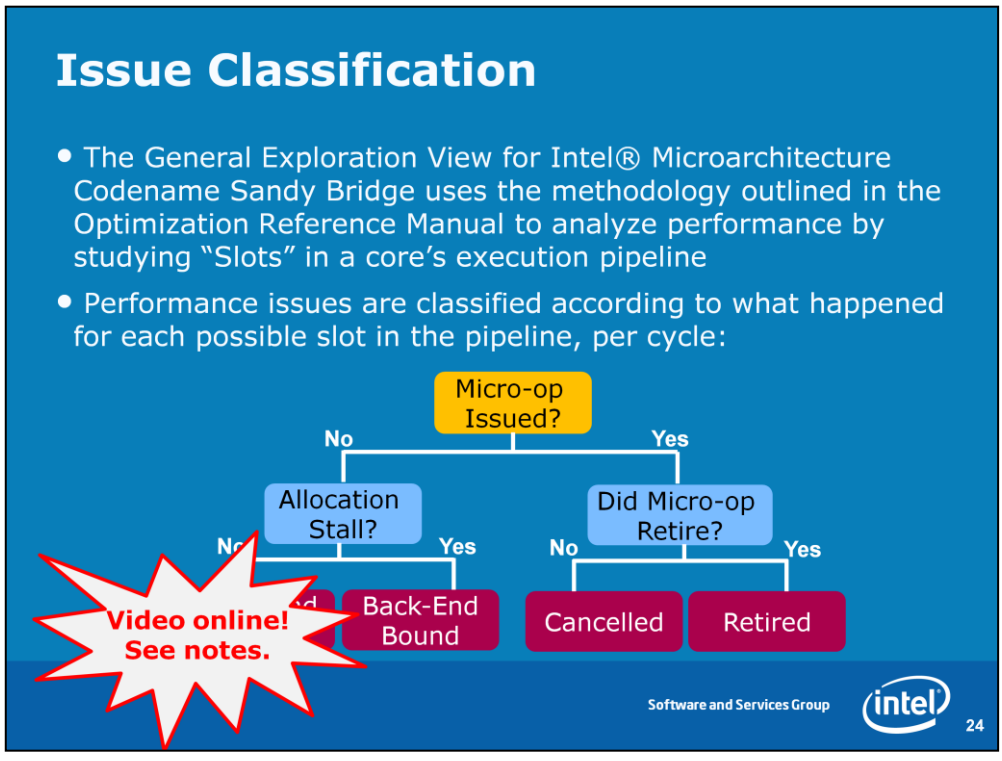
- **Why:** Existing AES instruction throughput has been improved on Sandy Bridge microarchitecture, which can result in significant performance increases in parallel encryption/decryption.
- **How:** If the application's functionality is in the domain of encryption/decryption, check to see if AES instructions are being used. Blocks of aes instructions with xmm registers as operands may be using parallel modes:

```
aesenc  %xmm5, %xmm7
aesenc  %xmm5, %xmm8
aesenc  %xmm5, %xmm9
aesenc  %xmm5, %xmm10
```

- **What Now:**
  - If AES is being used in parallel modes (such as ECB, CTR, and CBC-Decrypt), increase performance by redefining the number of blocks to be processed in parallel. (8 on Sandy Bridge compared to 4 on Westmere).
  - If AES is not being used and the application does any encryption or decryption, try it! See the Intel® Advanced Encryption Standard (AES) New Instructions Set

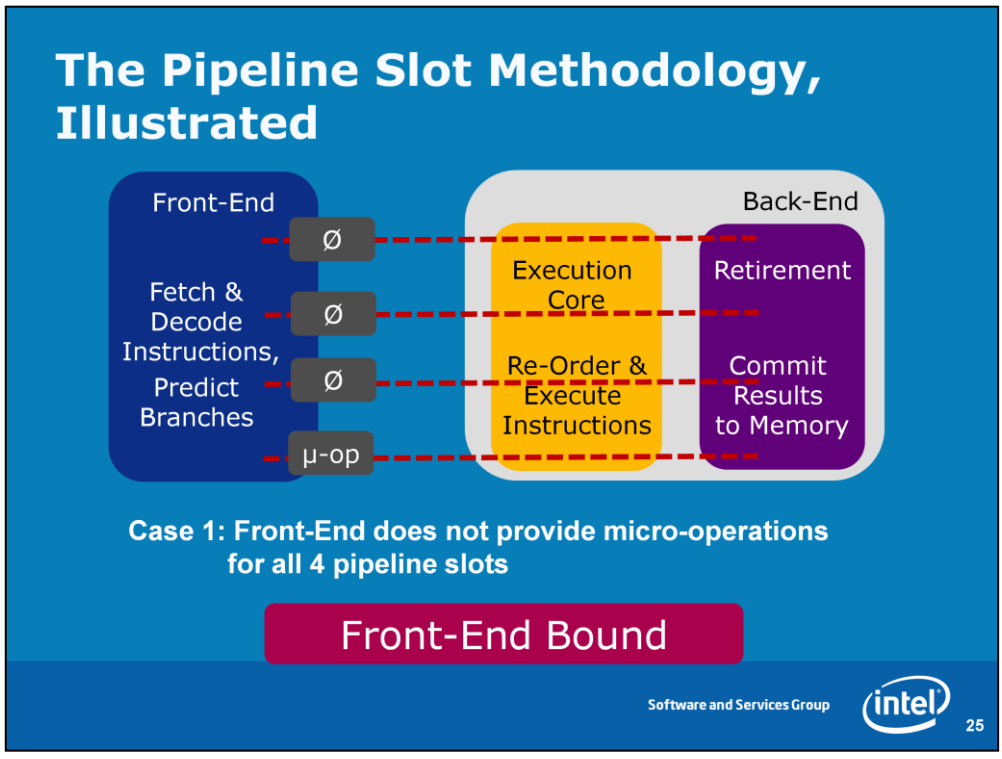


These are issues that result in inefficient pipeline use and high CPI. In addition to being in rough order of likelihood, these issues have been classified into the 4 categories of pipeline slot usage identified in the Intel® 64 and IA-32 Architectures Optimization Reference Manual .
The General Exploration profile also groups metrics according to these 4 categories. If desired, you can see how your application used the available pipeline slots per cycle using the first 4 metrics to the right of CPI: Retired Pipeline Slots, Cancelled Pipeline Slots, Back-End Bound Pipeline Slots, and Front-End Pipeline Slots.



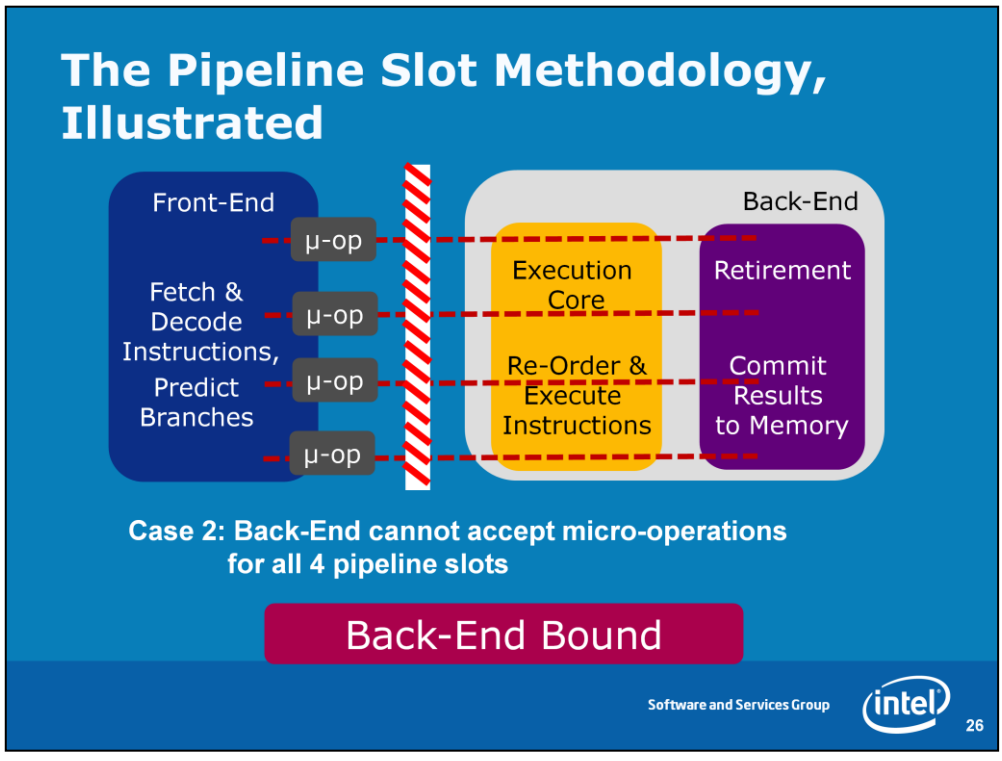
Note that the way this methodology allows us to classify what percentage of all pipeline slots end up in each category, for each cycle and for each core. It is possible that for a given dataset, there may be a significant percentage of pipeline slots in multiple categories that merit investigation. Ideally a large percentage of slots will fall into the "Retired" category, but even then, it may be possible to make your code more efficient.

For a complete description of this methodology, see the Intel® 64 and IA-32 Architectures Optimization Reference Manual, Appendix B.3. You can also view our 10-minute video, which describes the methodology in more detail, here: http://software.intel.com/en-us/videos/channel/parallel-programming/performance-analysis-methodology-for-intel-microarchitecture-codename-sandy-bridge/1265132823001.

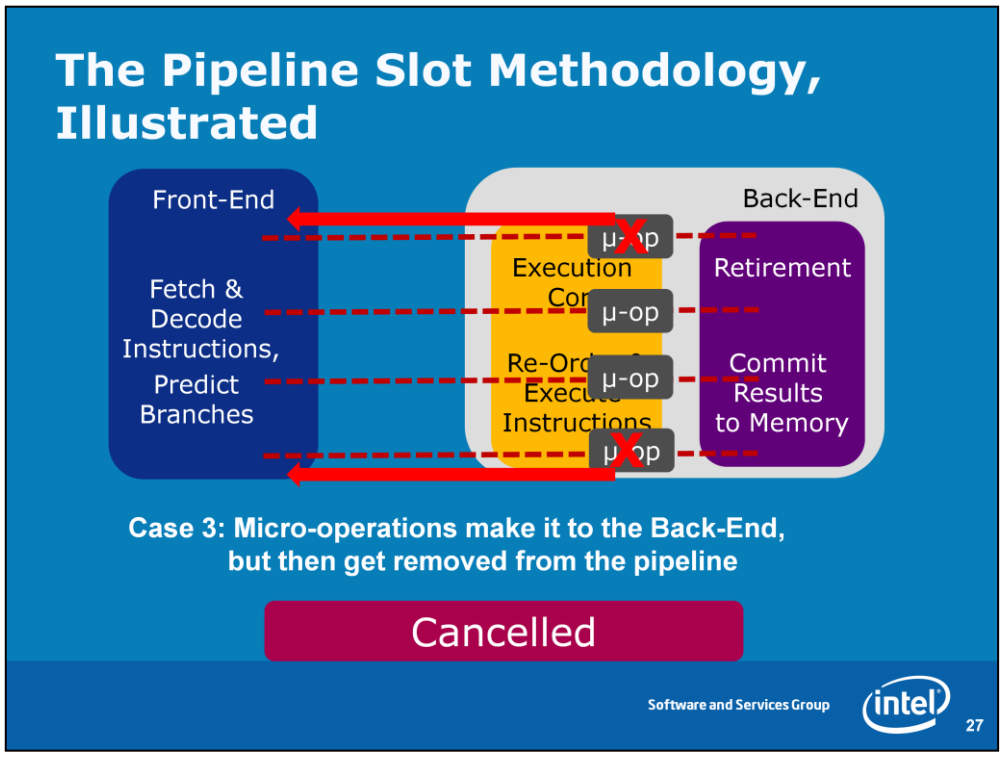


The Front-End consists of several different structures. It is responsible for fetching instructions, decoding them into micro-operations, and then delivering those micro-operations to the Back-End of the pipeline. For Intel® Microarchitecture Codename Sandy Bridge, a maximum of 4 micro-operations can be delivered to the Back-End portion of the pipeline per cycle (per core).

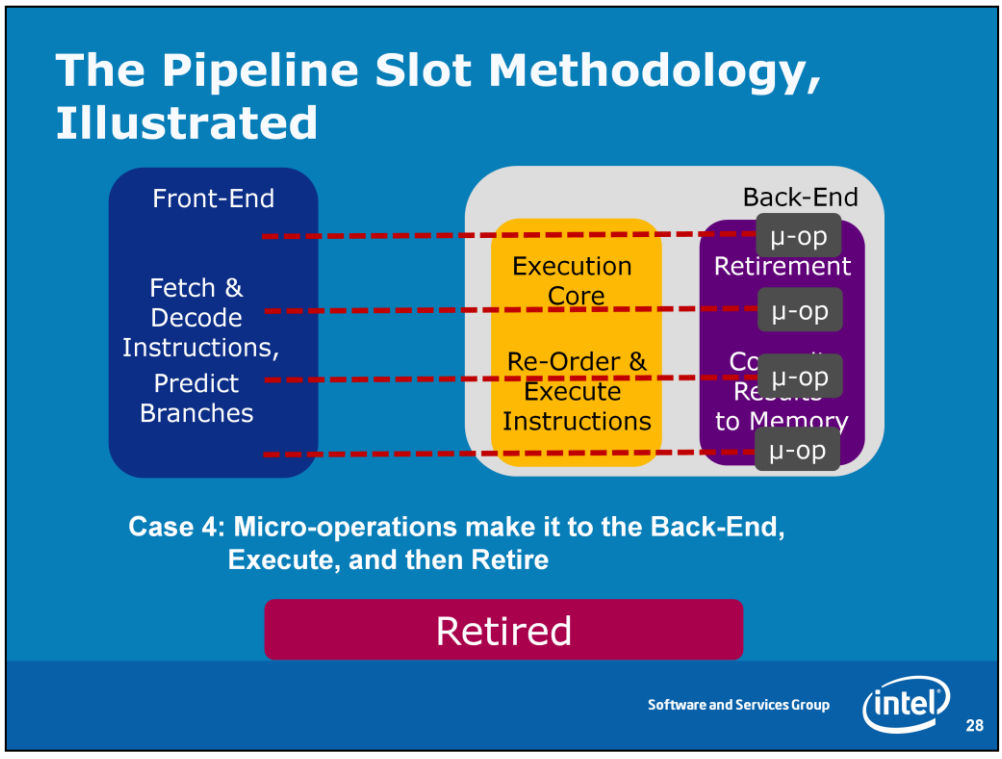
Front-End issues are generally caused by delays in fetching code (due to caching or ITLB issues) or in decoding instructions (due to specific instruction types or queueing issues). Front-End issues are generally resolved by compiler techniques.



The Back-End of the pipeline is responsible for accepting micro-operations from the Front-End, then re-ordering them as necessary to schedule their execution in the various execution units, retrieving needed operands from memory, executing the operations, then committing the results to memory. If the Back-End is not able to accept micro-operations from the Front-End, it is generally because internal queues are full. Most of the time this is due to data access issues – the Back-End's structures are being taken up by micro-operations that are waiting on data from the caches.



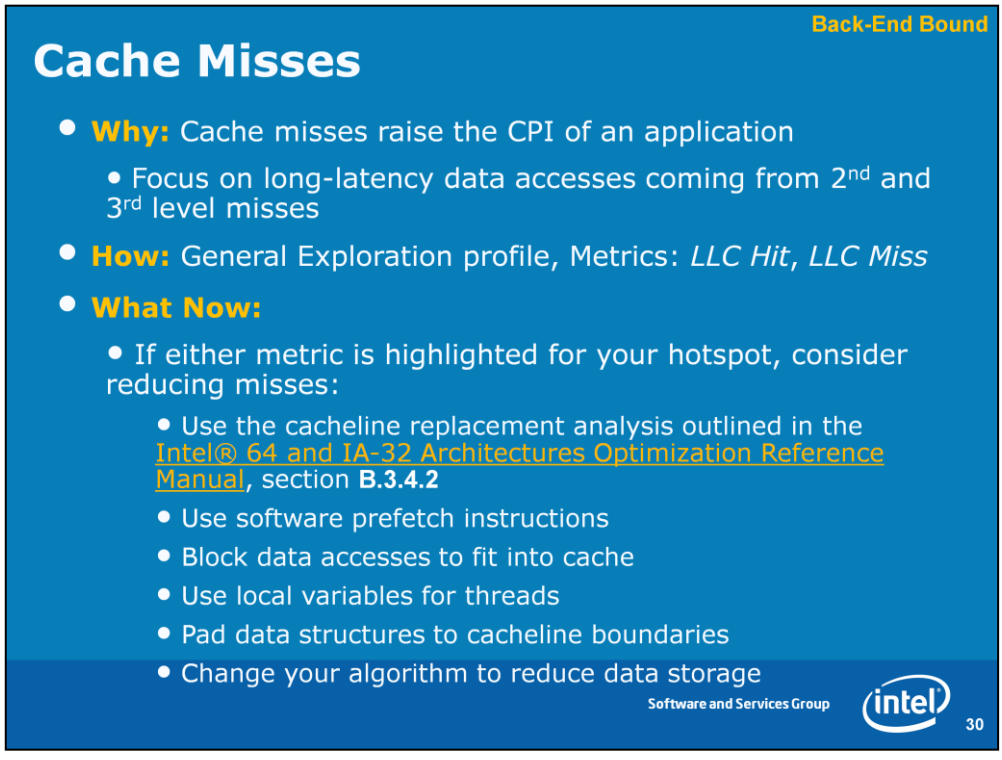
Cancelled micro-operations most likely happen because the Front-End mis-predicted a branch. This is discovered in the Back-End when the branch operation is executed. At this point, if the target of the branch was incorrectly predicted, the micro-operation and all subsequent incorrectly predicted operations are cancelled and the Front-End is re-directed to begin fetching instructions from the correct target.



In general, having as many pipeline slots retiring per cycle as possible is the goal. Only one issue is identified for this category – which deals with how to get micro-operations to this stage faster.

Step 2) Optimize the Issue

- For each potential issue, there are several important pieces of information:
  - Why? – Why you should be concerned about this potential problem.
  - How? – Which profile and metric to use in the Amplifier XE interface. If the data is highlighted, then it should be investigated.
  - What Now? – Helps you move to Step 2 of the Tuning Process (Optimize the Issue). Gives suggestions for follow-up investigations or optimizations to try.
  - Event Names and Metric Formulas are given in the Notes. These are not included on the slide because they are already embedded in the Amplifier XE logic. You only need to use the pre-configured profiles and metrics pointed out in order to know if you may have a problem.



Formulas:

% of cycles spent on memory access (LLC misses):

(MEM_LOAD_UOPS_MISC_RETIRED.LLC_MISS_PS * 180) / CPU_CLK_UNHALTED.THREAD

% of cycles spent on last level cache access (2nd level misses that hit in LLC):

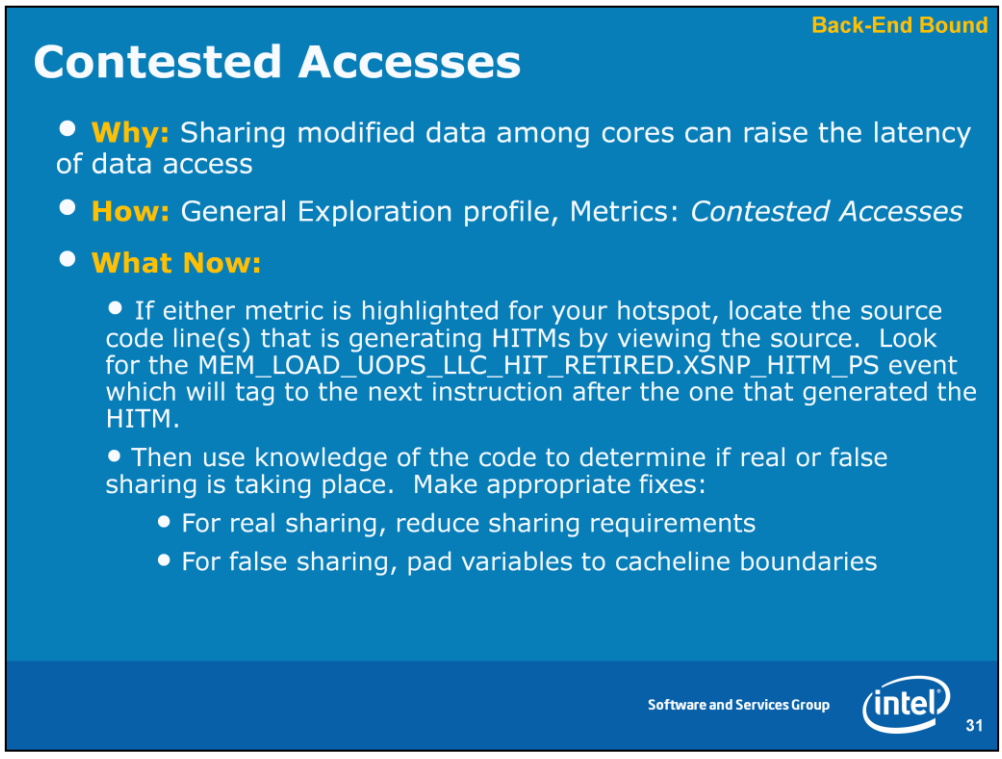
((MEM_LOAD_RETIRED.L3_HIT_PS * 26) + (MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HIT_PS * 43) +
(MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HITM_PS * 60)) / CPU_CLK_UNHALTED.THREAD

Thresholds: Investigate if –

% cycles for LLC miss ≥ .2,

% cycles for LLC Hit ≥ .2

Doing the cacheline replacement analysis study (the first suggestion) can be very helpful in resolving these issues. The approach is to figure out at what level most of the application's loads are being satisfied, then look one level above. This is because if loads are coming from a particular level of cache or memory, it is because the data has been replaced at the level above (for example, if loads are coming from memory, the data has been replaced in the LLC). The study shows how to identify the areas of your code that are causing the replacements to occur. Once these are identified, you can try changing the algorithm, doing non-temporal stores, or one of the other suggestions above.



Formula:

% of cycles spent accessing data modified by another core:

(MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HITM_PS * 60) / CPU_CLK_UNHALTED.THREAD

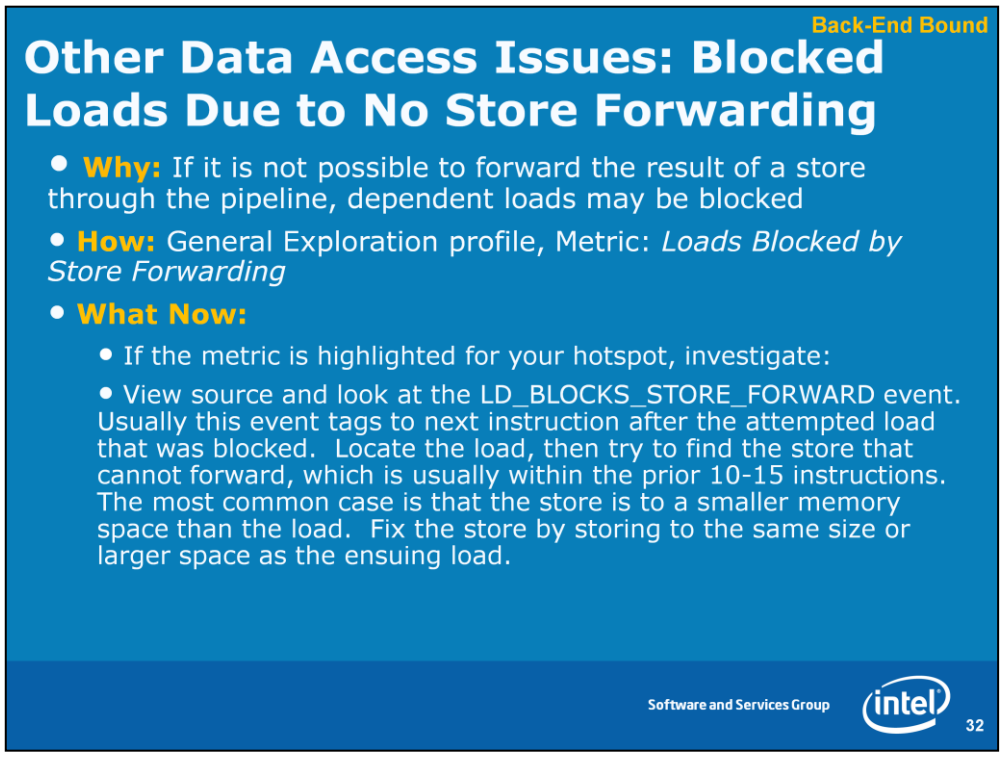
Thresholds: Investigate if –

% cycles accessing modified data > .05

This metric is also called write sharing. It occurs when one core needs data that is found in a modified state in another core's cache. This causes the line to be invalidated in the holding core's cache and moved to the requesting core's cache. If it is written again and another core requests it, the process starts again. The cacheline ping pong-ing between caches causes longer access time than if it could be simply shared amongst cores (as with read-sharing).

Write sharing can be caused by true sharing, as with a lock or hot shared data structure, or by false sharing, meaning that the cores are modifying 2 separate pieces of data stored on the same cacheline.

Note that in the case of real write sharing that is caused by a lock, Amplifier XE's Locks and Waits analysis should also indicate a problem. This hardware-level analysis will detect other cases as well though (such as false sharing or write sharing a hot data structure).



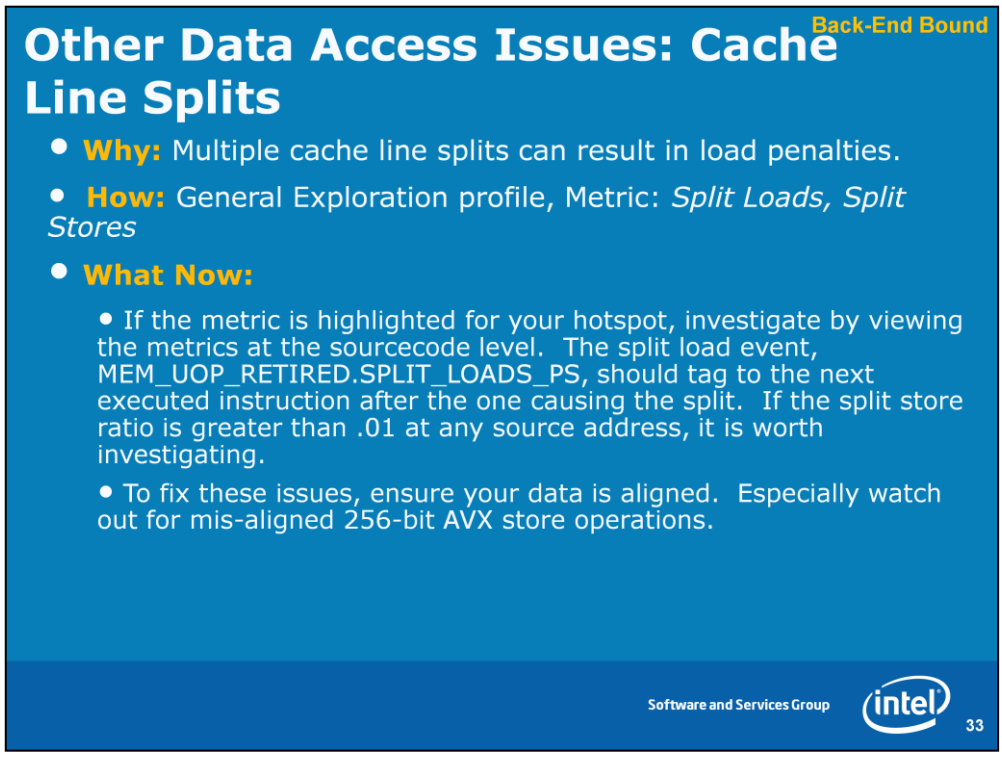
Formula:

Blocked Store Forwarding Cost = (LD_BLOCKS_STORE_FORWARD * 13) / CPU_CLK_UNHALTED.THREAD

Threshold: Investigate if –

Cost ≥ .05

Store forwarding occurs when there are two memory instructions in the pipeline, a store followed by a load from the same address. Instead of waiting for the data to be stored to the cache, it is "forwarded" back along the pipeline to the load instruction, saving a load from the cache. Store forwarding is the desired behavior, however, in certain cases, the store may not be able to be forwarded, so the load instruction becomes blocked waiting for the store to write to the cache and then to load it.



A cacheline split is any load or store that traverses a 64-byte boundary.

Formulas:

Split Load Cost = (MEM_UOP_RETIRED.SPLIT_LOADS_PS * 5) / CPU_CLK_UNHALTED.THREAD

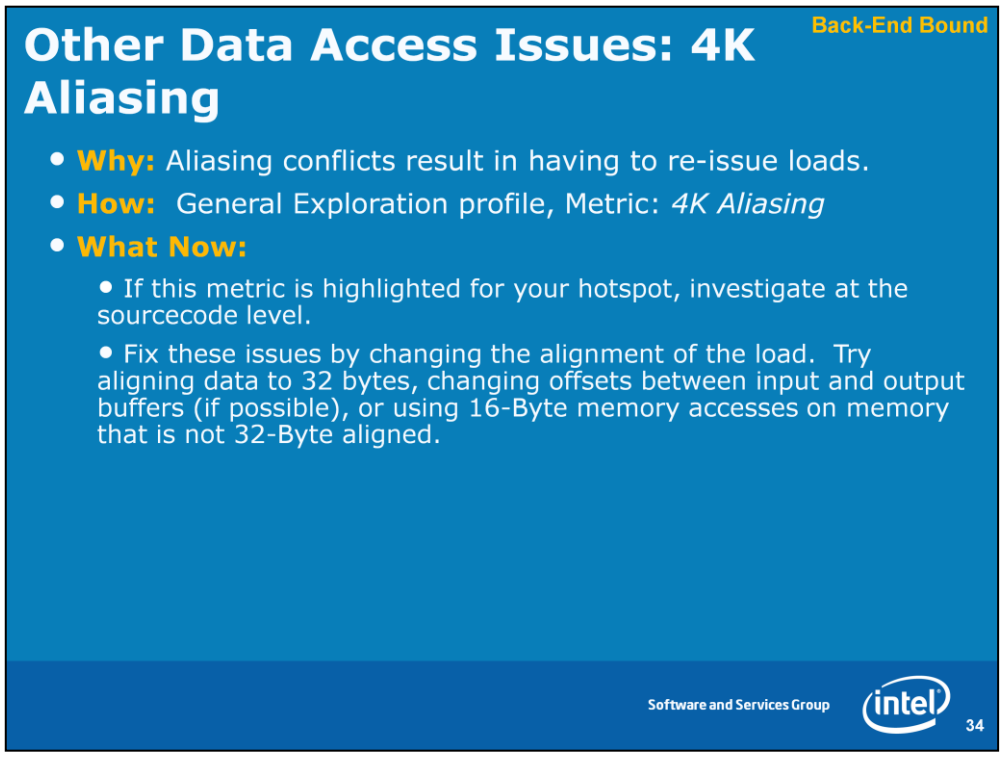
Split Store Ratio = MEM_UOP_RETIRED.SPLIT_STORES_PS / MEM_UOP_RETIRED.ANY_STORES_PS

Thresholds: Investigate if –

Split load cost ≥ .1 or

Split store ratio is > 0.01

Beginning with the Intel® Core™ architecture, the penalty for cacheline splits has been reduced to only 5 cycles. However, if there are repeated splits occurring, the penalty can grow, and even just a 5-cycle increase in latency can make a difference in application performance.



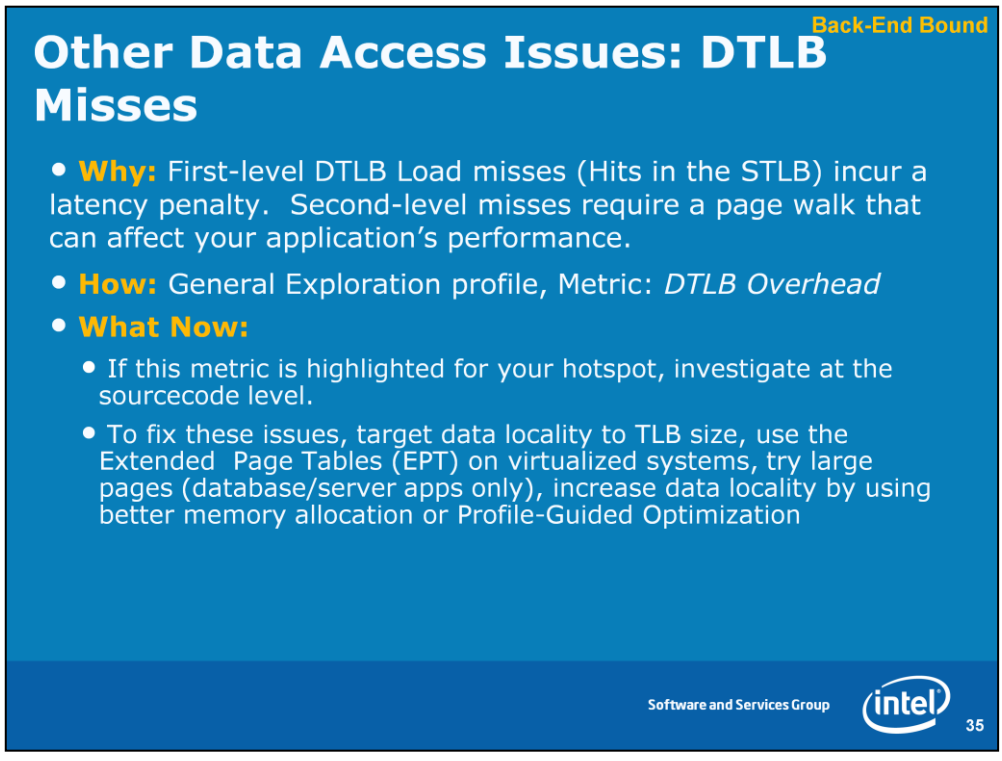
Formula:

Aliasing Conflicts Cost = (LD_BLOCKS_PARTIAL.ADDRESS_ALIAS * 5) / CPU_CLK_UNHALTED.THREAD

Threshold: Investigate if

Aliasing conflicts cost ≥ .1

This occurs when a load is issued after a store and their memory addresses are offset by (4K). When this is processed in the pipeline, the issue of the load will match the previous store (the full address is not used at this point), so pipeline will try to forward the results of the store and avoid doing the load (this is store forwarding). Later on when the address of the load is fully resolved, it will not match the store, and so the load will have to be re-issued from a later point in the pipe. This has a 5-cycle penalty in the normal case, but could be worse in certain situations, like with un-aligned loads that span 2 cache lines.



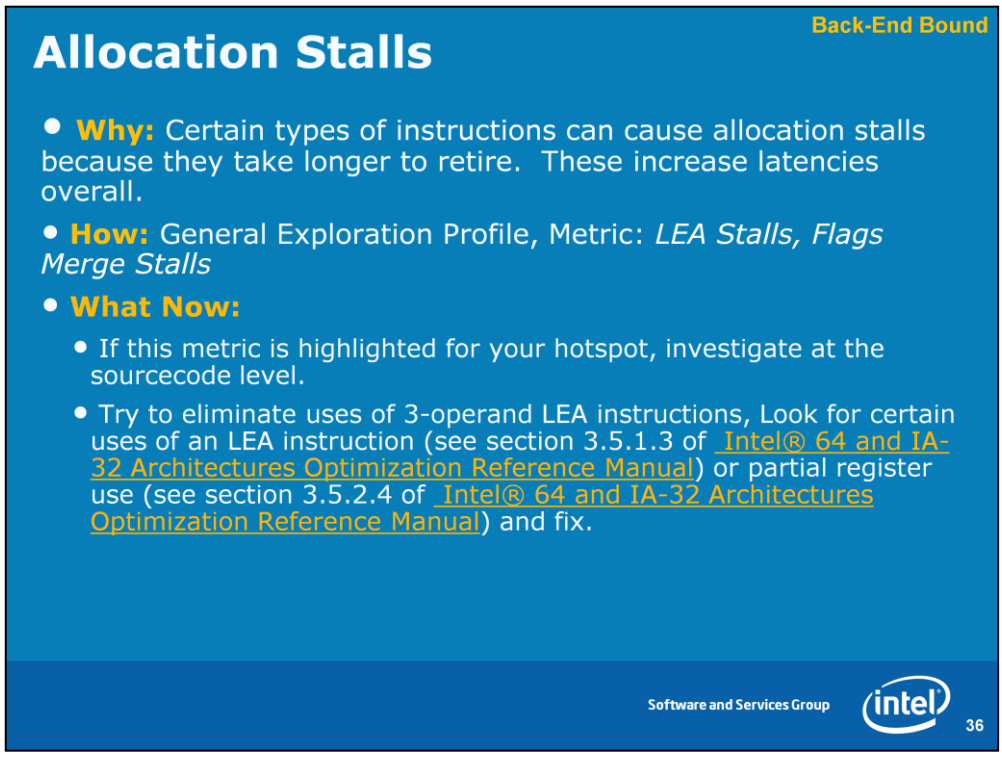
Formula:
DTLB Overhead = ((DTLB_LOAD_MISSES.STLB_HIT * 7) + DTLB_LOAD_MISSES.WALK_DURATION) / CPU_CLK_UNHALTED.THREAD

Threshold: Investigate if-

DTLB Overhead ≥ .1

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



Formulas:
Flags Merge Stalls =
PARTIAL_RAT_STALLS.FLAGS_MERGE_UOP_CYCLES/CPU_CLK_UNHALTED.THREAD

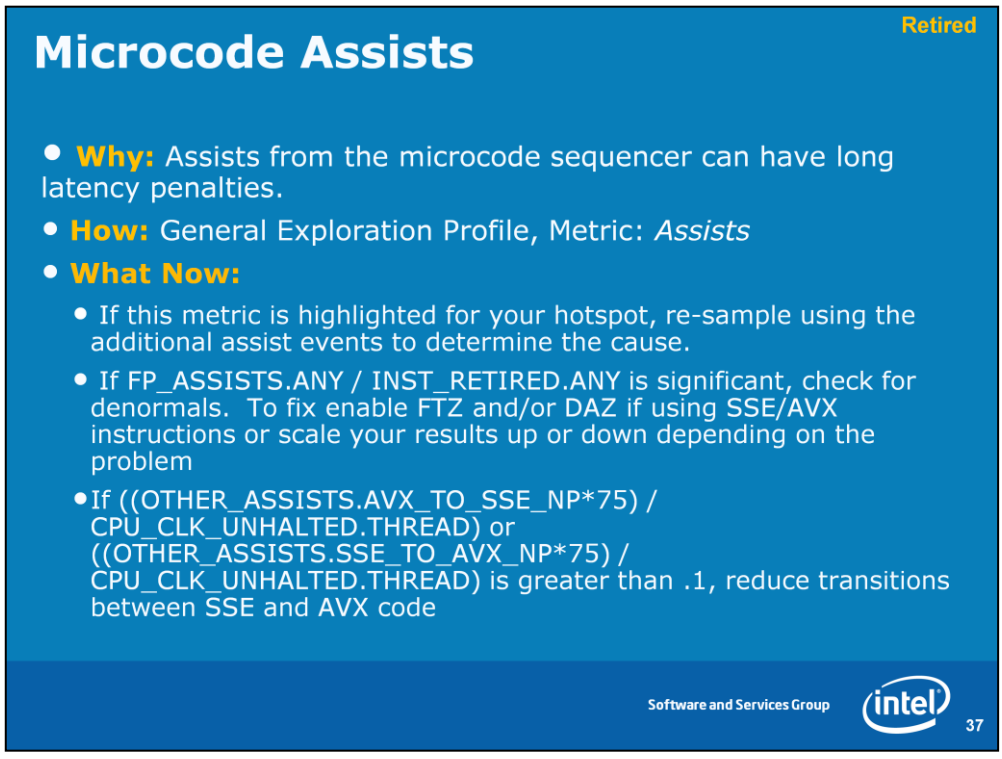
LEA Stalls =
PARTIAL_RAT_STALLS.SLOW_LEA_WINDOW/CPU_CLK_UNHALTED.THREAD

Threshold: Investigate if-

Flags Merge Stalls > .05

LEA Stalls > .05

Long-latency instructions cause the Back-end to refuse instructions from the front-end (allocation stalls).



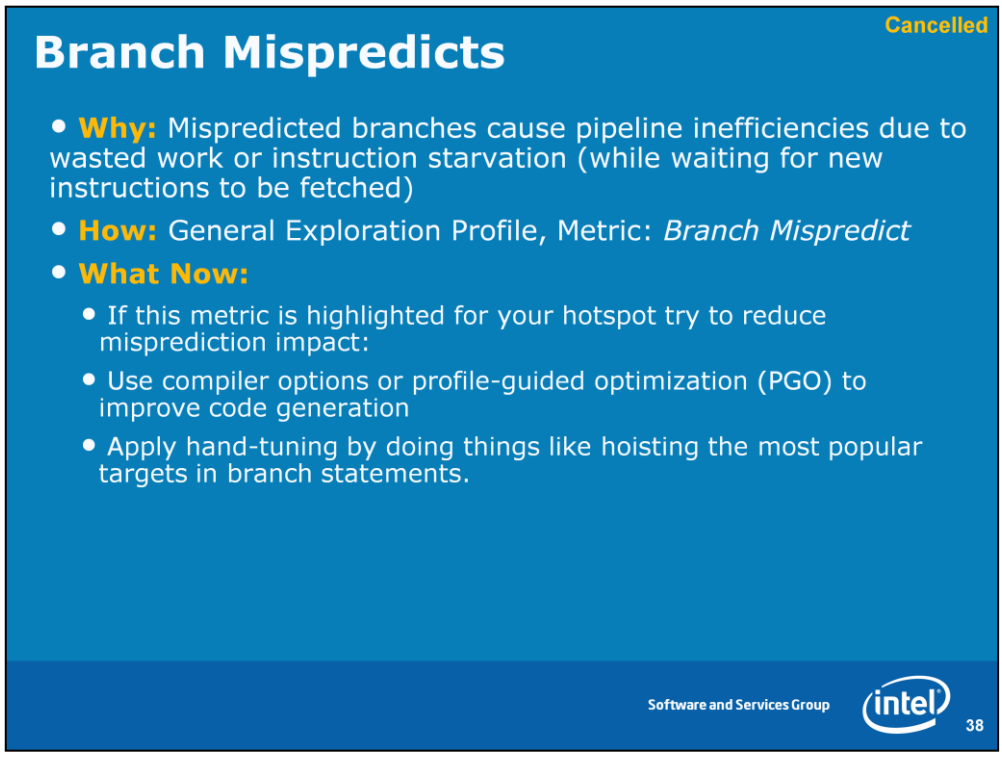
Formula:

Assist % = IDQ.MS_CYCLES / CPU_CLK_UNHALTED.THREAD

Threshold: Investigate if –

Assist Cost ≥ .05

There are many instructions that can cause assists when there is no performance problem. If you see MS_CYCLES it doesn't necessarily mean there is an issue, but whenever you do see a significant amount of MS_CYCLES, check the other metrics to see if it's one of the problems we mention.



Formula:

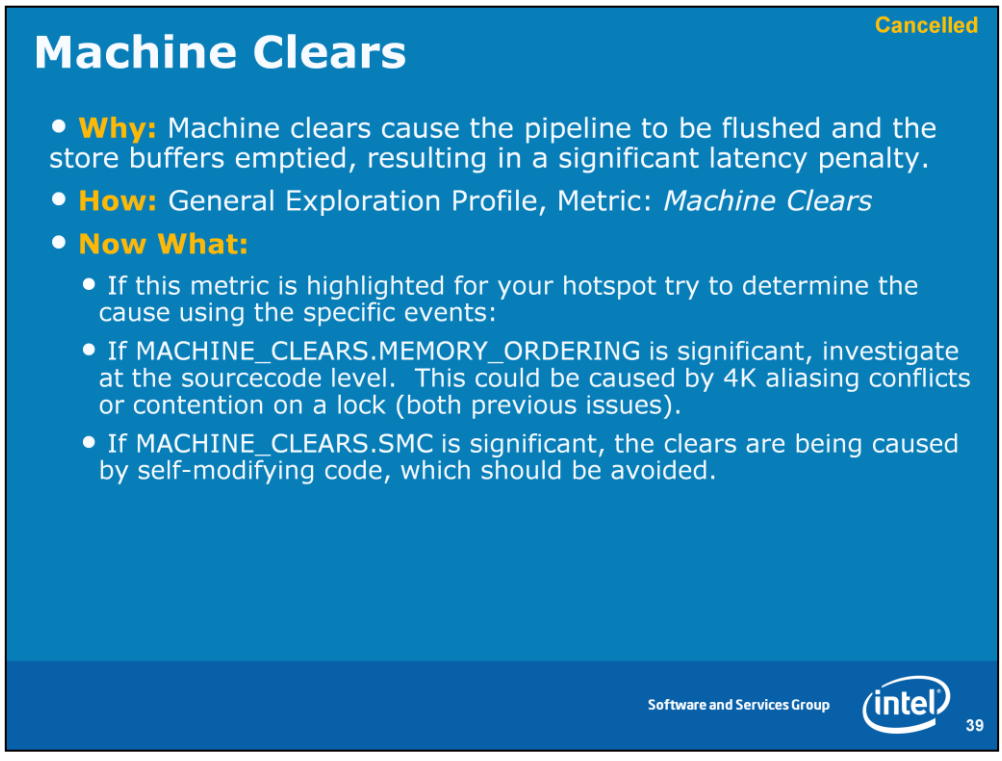
Mispredicted branch cost: (20* BR_MISP_RETIRED.ALL_BRANCHES_PS)/ CPU_CLK_UNHALTED.THREAD

Threshold: Investigate if -

Cost is ≥ .2

Note that all applications will have some branch mispredicts - it is not the number of mispredicts that is the problem but the impact.

To do hand-tuning, you need to locate the branch causing the mispredicts. This can be difficult to track down due to the fact that this event will normally tag to the first instruction in the correct path that the branch takes.



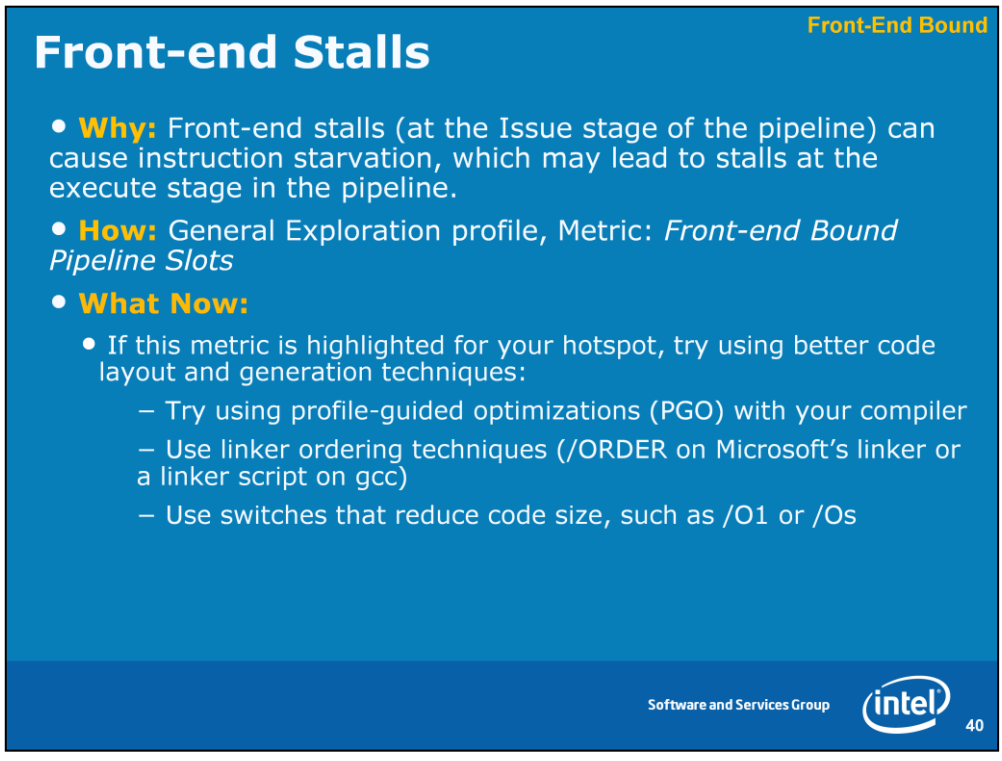
Formula:

Machine Clear cost: ((MACHINE_CLEARS.MEMORY_ORDERING + MACHINE_CLEARS.SMC + MACHINE_CLEARS.MASKMOV) * 100) / CPU_CLK_UNHALTED.THREAD

Threshold: Investigate if -

Cost is ≥ .02

Machine clears are generally caused by either contention on a lock, or failed memory disambiguation from 4k aliasing (both earlier issues). The other potential cause is self-modifying code (SMC).



Formula:

IDQ_UOPS_NOT_DELIVERED.CORE / (CPU_CLK_UNHALTED.THREAD * 4)

Threshold: Investigate if –

Front-End Bounc uOps ≥ .15

Assists or excessive Branch Mispredicts, all on previous slides, could be the reason for front-end issues, so check for and resolve those problems first. This issue may also be caused by instruction cache misses (on server apps), which are generally fixed by better code layout.

Good Luck!
For more information:

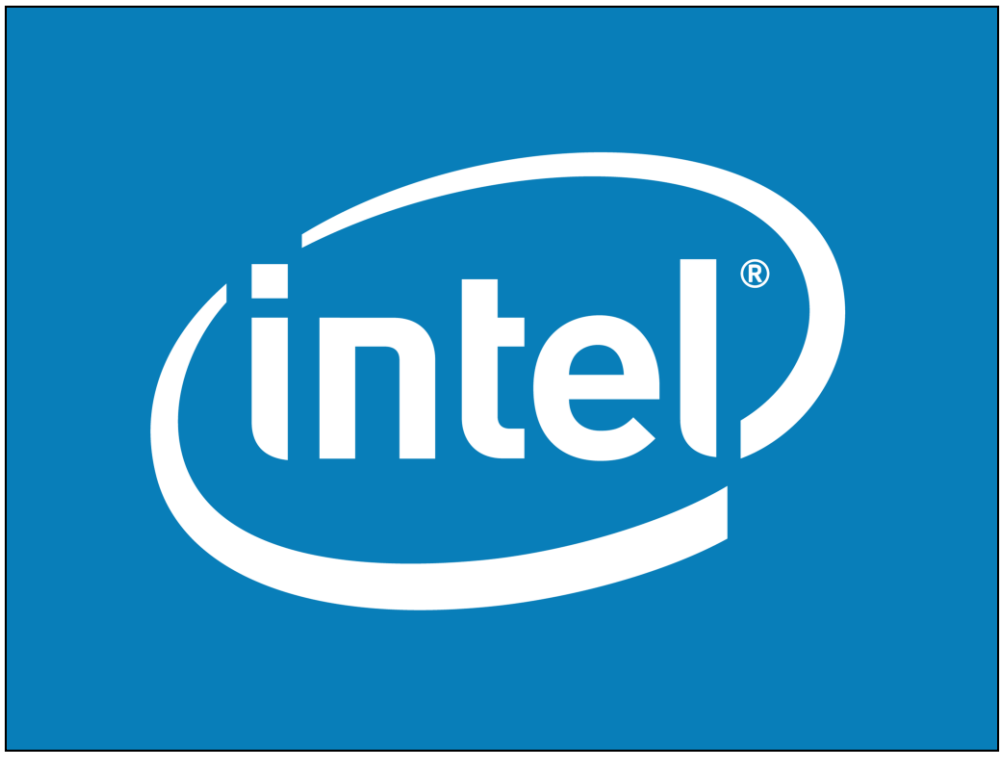
VTune User Forums:
http://software.intel.com/en-us/forums/intel-vtune-performance-analyzer/

VTune Amplifier XE Videos:
http://software.intel.com/en-us/articles/intel-vtune-amplifier-xe/

Intel® 64 and IA-32 Architecture Software Developer's Manuals:
http://www.intel.com/products/processor/manuals/index.htm

Optimization Guide for Intel® Microarchitecture Codename Nehalem:
http://software.intel.com/file/15529

For optimization of the integrated graphics controller on
Intel® Microarchitecture Codename Sandy Bridge:
www.intel.com/software/gpa


intel