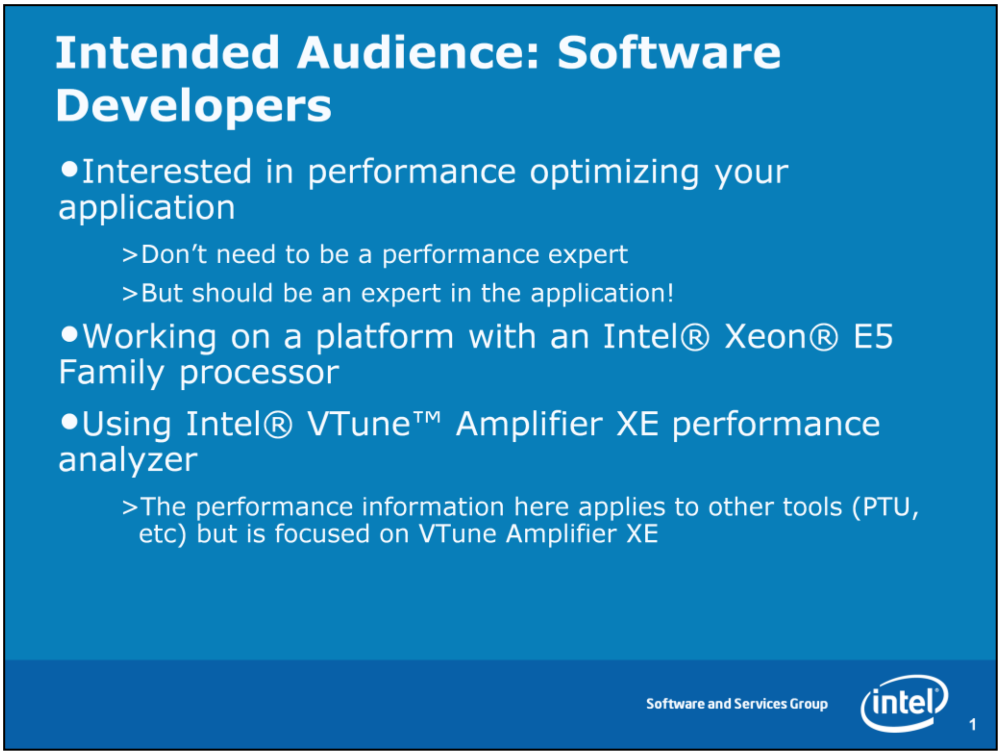
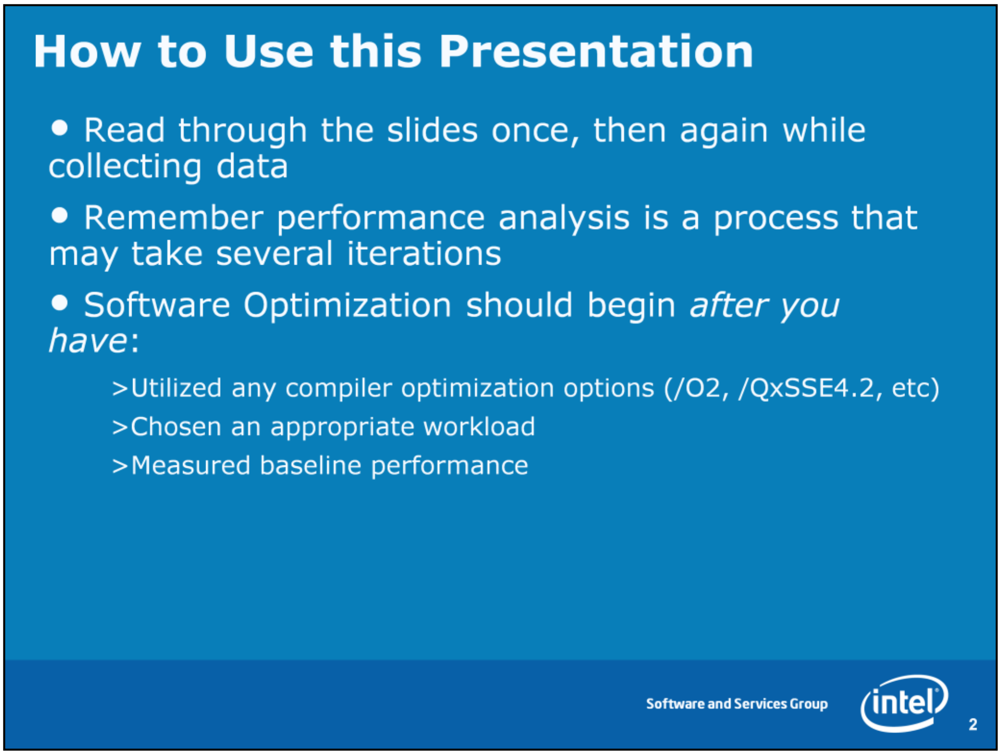


Intended Audience: Software Developers

- 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 an Intel® Xeon® E5 Family 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

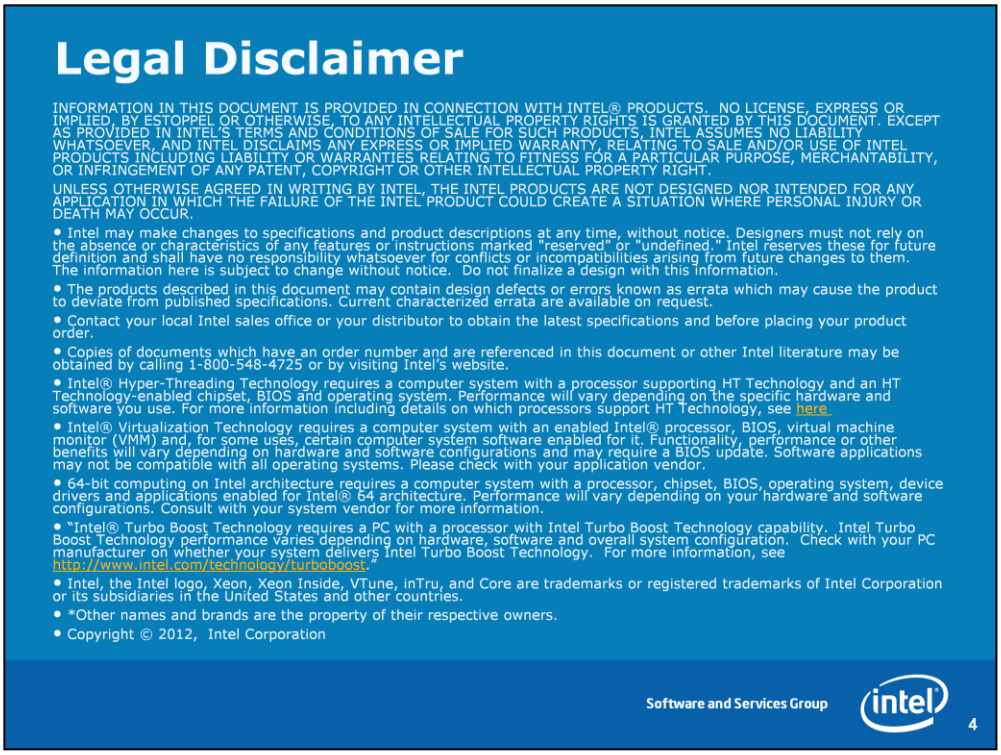
- Read through the slides once, then again while collecting data
- Remember performance analysis is a process that may take several iterations
- Software Optimization should begin *after you have*:
  - >Utilized any compiler optimization options (/O2, /QxSSE4.2, etc)
  - >Chosen an appropriate workload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

Using Intel® VTune™ Amplifier XE to Tune Software on the Intel® Xeon® Processor E5 Family

Software and Services Group

Ver. 1.0

For single-socket (desktop/client) systems containing Intel® microarchitecture code name Sandy Bridge processors, see our guide here: http://software.intel.com/en-us/articles/using-intel-vtune-amplifier-xe-to-tune-software-on-the-2nd-generation-intel-core-processor-family/

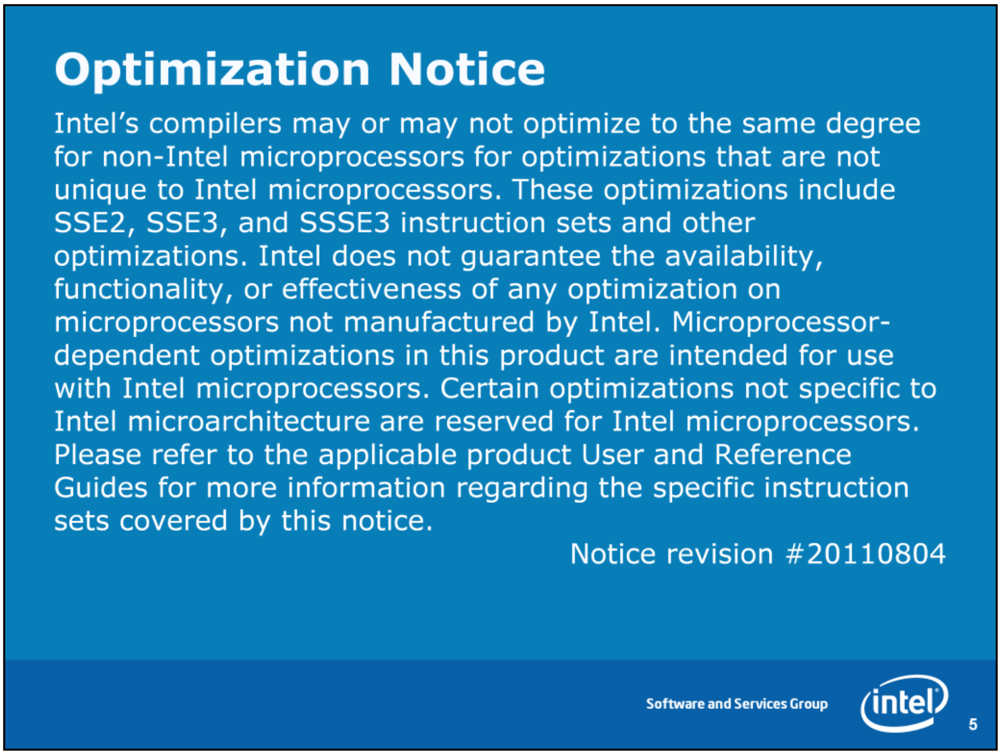


Legal Disclaimer

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.

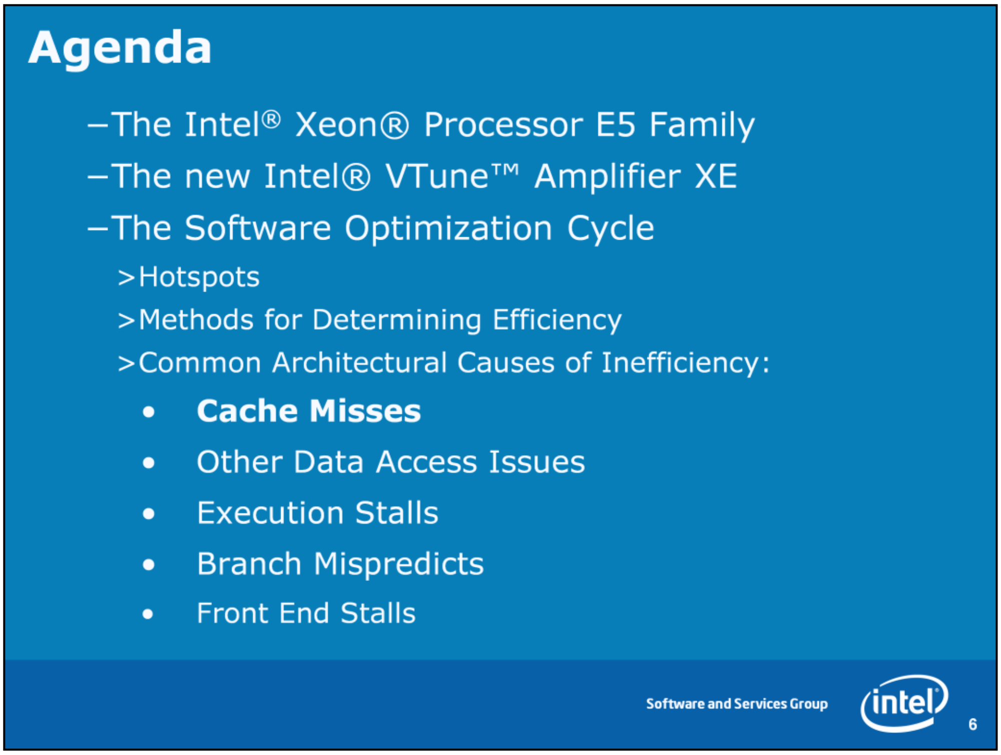
- Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever 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, Xeon, Xeon Inside, 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.
- Copyright © 2012, Intel Corporation



Optimization Notice

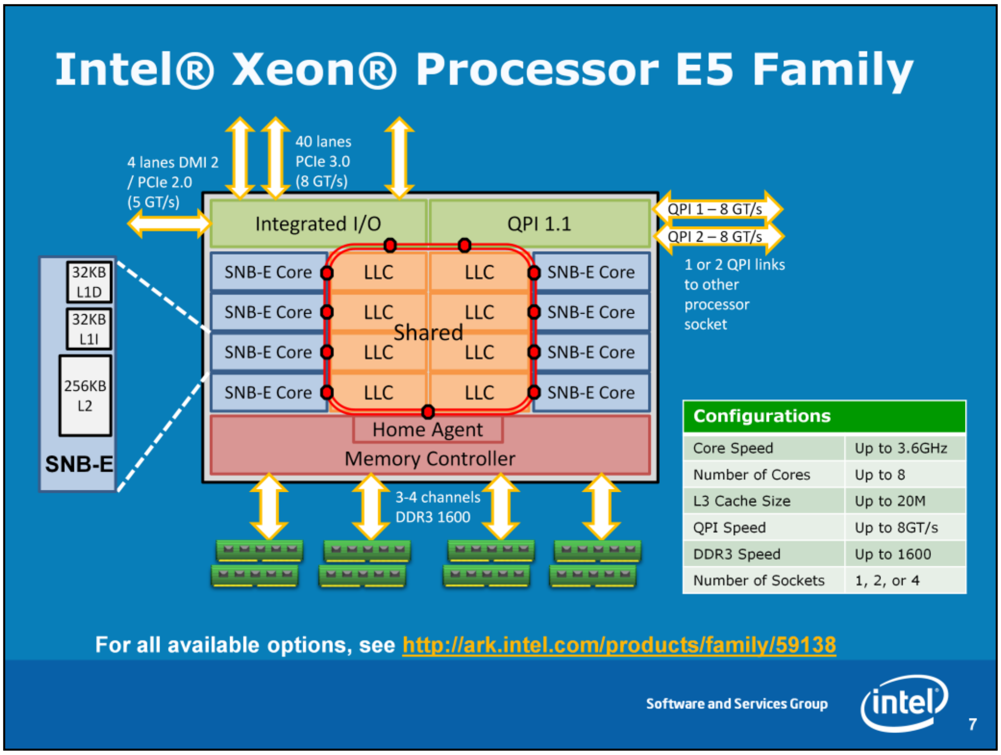
Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, 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

- –The Intel® Xeon® Processor E5 Family
- –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



Intel® Xeon® Processor E5 Family

4 lanes DMI 2 / PCIe 2.0 (5 GT/s)

40 lanes PCIe 3.0 (8 GT/s)

Integrated I/O

QPI 1.1

QPI 1 – 8 GT/s

QPI 2 – 8 GT/s

1 or 2 QPI links to other processor socket

32KB L1D

32KB L1I

256KB L2

SNB-E

SNB-E Core

LLC

Shared

Home Agent

Memory Controller

3-4 channels DDR3 1600

Configurations	
Core Speed	Up to 3.6GHz
Number of Cores	Up to 8
L3 Cache Size	Up to 20M
QPI Speed	Up to 8GT/s
DDR3 Speed	Up to 1600
Number of Sockets	1, 2, or 4

For all available options, see http://ark.intel.com/products/family/59138

Software and Services Group

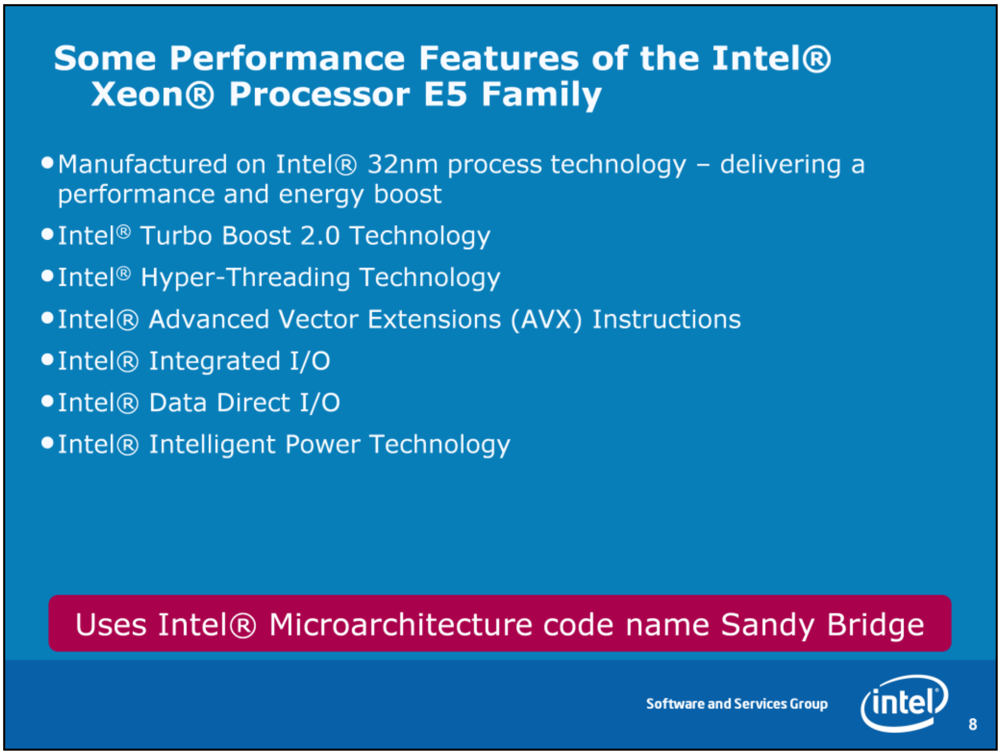
7

Intel® Xeon® E5-2400 Family – (formerly known as SNB-EN) - dual-socket, with only 1 QPI link and 3 mem channels (LGA1356 socket)

Intel® Xeon® E5-26xx Family – (formerly known as SNB-EP 2S) – dual socket, with 2 QPI links and 4 mem channels (LGA2011 socket)

Intel® Xeon® E5-46xx Family – (formerly known as SNB-EP 4S) – quad-socket, with 2 QPI links and 4 mem channels (LGA2011 socket)

Also available is the E3-1200 family, single-socket server processors based on Intel® microarchitecture code name Ivy Bridge. These are covered in the single-socket tuning guide for Intel® microarchitecture code name Sandy Bridge.



Some Performance Features of the Intel® Xeon® Processor E5 Family

- Manufactured on Intel® 32nm process technology – delivering a performance and energy boost
- Intel® Turbo Boost 2.0 Technology
- Intel® Hyper-Threading Technology
- Intel® Advanced Vector Extensions (AVX) Instructions
- Intel® Integrated I/O
- Intel® Data Direct I/O
- Intel® Intelligent Power Technology

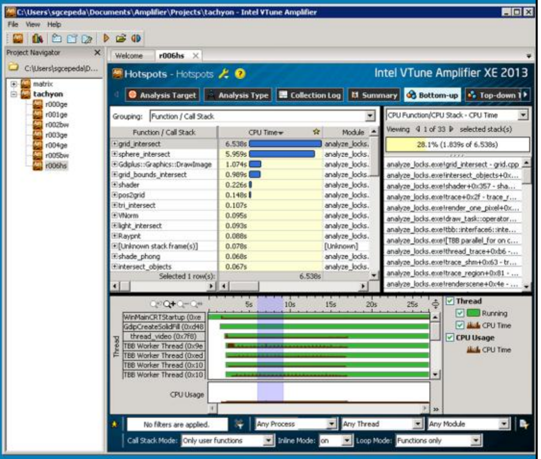
Uses Intel® Microarchitecture code name Sandy Bridge

For a complete listing of SKUs, see http://ark.intel.com/products/family/59138

The New Intel® VTune™ Amplifier XE

VTune Amplifier XE 2013 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, Java, Assembly, .NET
- Visual Studio Integration, Command-line, or Standalone interface for Windows* or Linux*



C:\Users\sgcepeda\Documents\Amplifier\Projects\tachyon - Intel VTune Amplifier

C:\Users\sgcepeda\D...

Grouping: Function / Call Stack

Function / Call Stack	CPU Time	Module
grid_intersect	6.538s	analyze_locks.
sphere_intersect	5.959s	analyze_locks.
Gdiplus::Graphics::DrawImage	1.074s	analyze_locks.
grid_bounds_intersect	0.989s	analyze_locks.
shader	0.226s	analyze_locks.
pos2grid	0.148s	analyze_locks.
tri_intersect	0.107s	analyze_locks.
VNorm	0.095s	analyze_locks.
light_intersect	0.093s	analyze_locks.
Raypnt	0.088s	analyze_locks.
[Unknown stack frame(s)]	0.078s	[Unknown]
shade_phong	0.068s	analyze_locks.
intersect_objects	0.067s	analyze_locks.
Selected 1 row(s):	6.538s	

CPU Function/CPU Stack - CPU Time

Viewing 1 of 33 selected stack(s)

28.1% (1.839s of 6.538s)

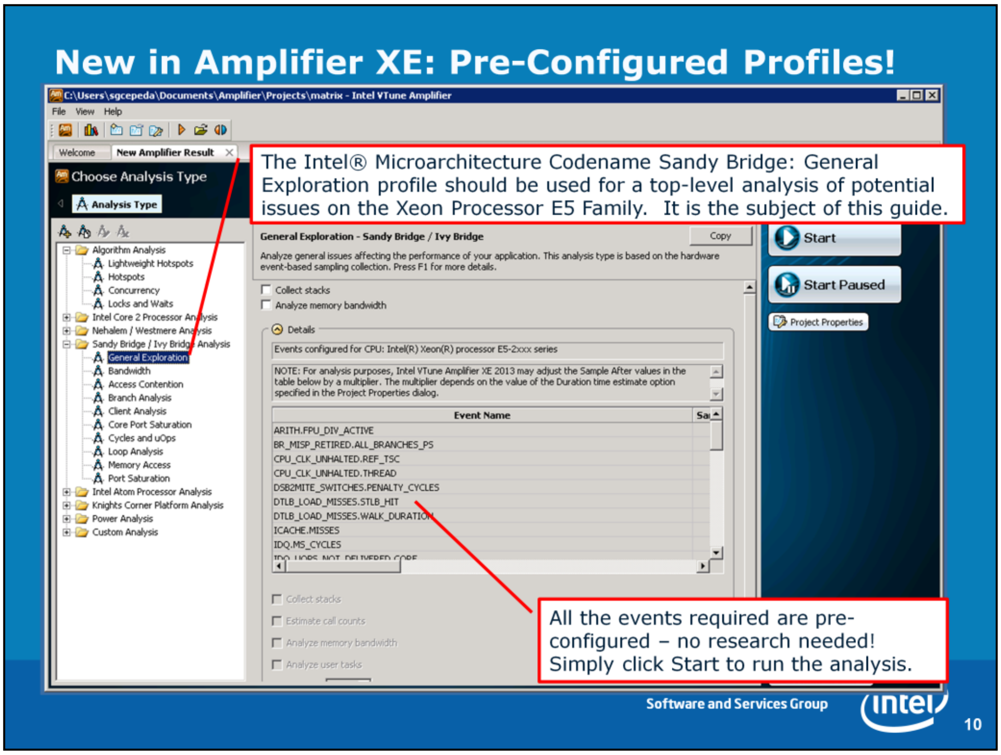
```
analyze_locks.exe!grid_intersect - grid.cpp
analyze_locks.exe!intersect_objects+0x...
analyze_locks.exe!shader+0x357 - sha...
analyze_locks.exe!trace+0x2f - trace_r...
analyze_locks.exe!render_one_pixel+0x...
analyze_locks.exe!draw_task::operator...
analyze_locks.exe!tbb::interface6::inte...
analyze_locks.exe![TBB parallel_for on c...
analyze_locks.exe!thread_trace+0xb6 -...
analyze_locks.exe!trace_shm+0x63 - tr...
analyze_locks.exe!trace_region+0x81 - ...
analyze_locks.exe!renderscene+0x4e - ...
```

```
WinMainCRTStartup (0xe
GdipCreateSolidFill (0xd48
thread_video (0x7f8)
TBB Worker Thread (0x9e
TBB Worker Thread (0xed
TBB Worker Thread (0x10
TBB Worker Thread (0x10
```

No filters are applied. | Any Process | Any Thread | Any Module

Call Stack Mode: Only user functions | Inline Mode: on | Loop Mode: Functions only

New in Amplifier XE: Pre-Configured Profiles!



The Intel® Microarchitecture Codename Sandy Bridge: General Exploration profile should be used for a top-level analysis of potential issues on the Xeon Processor E5 Family. It is the subject of this guide.

General Exploration - Sandy Bridge / Ivy Bridge

Analyze general issues affecting the performance of your application. This analysis type is based on the hardware event-based sampling collection. Press F1 for more details.

☐ Collect stacks
☐ Analyze memory bandwidth

Details

Events configured for CPU: Intel(R) Xeon(R) processor E5-2xxx series

NOTE: For analysis purposes, Intel VTune Amplifier XE 2013 may adjust the Sample After values in the table below by a multiplier. The multiplier depends on the value of the Duration time estimate option specified in the Project Properties dialog.

Event Name	Sa
ARITH.FPU_DIV_ACTIVE	
BR_MISP_RETIRED.ALL_BRANCHES_PS	
CPU_CLK_UNHALTED.REF_TSC	
CPU_CLK_UNHALTED.THREAD	
DSB2MITE_SWITCHES.PENALTY_CYCLES	
DTLB_LOAD_MISSES.STLB_HIT	
DTLB_LOAD_MISSES.WALK_DURATION	
ICACHE.MISSES	
IDQ.MS_CYCLES	
IDQ_UOPS_NOT_DELIVERED.CORE	

☐ Collect stacks
☐ Estimate call counts
☐ Analyze memory bandwidth
☐ Analyze user tasks

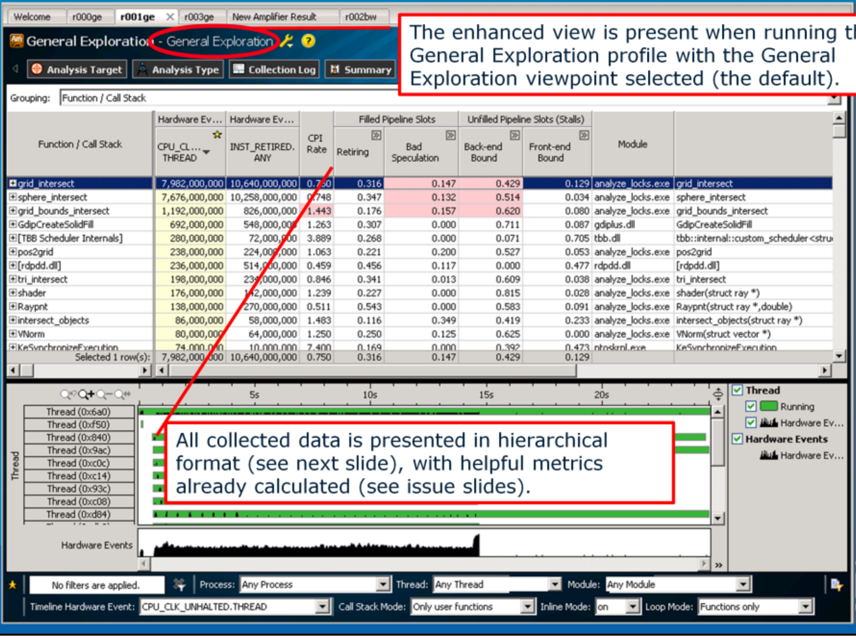
All the events required are pre-configured – no research needed! Simply click Start to run the analysis.

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/the-intel-vtune-amplifier-xe-analysis-and-results-interface-for-intel-microarchitecture

Enhanced General Exploration View for Intel® Microarchitecture Codename Sandy Bridge



Grouping: Function / Call Stack

Function / Call Stack	Hardware Ev… CPU_CL… THREAD	Hardware Ev… INST_RETIRED. ANY	CPI Rate	Filled Pipeline Slots: Retiring	Filled Pipeline Slots: Bad Speculation	Unfilled Pipeline Slots (Stalls): Back-end Bound	Unfilled Pipeline Slots (Stalls): Front-end Bound	Module	
⊞grid_intersect	7,982,000,000	10,640,000,000	0.750	0.316	0.147	0.429	0.129	analyze_locks.exe	grid_intersect
⊞sphere_intersect	7,676,000,000	10,258,000,000	0.748	0.347	0.132	0.514	0.034	analyze_locks.exe	sphere_intersect
⊞grid_bounds_intersect	1,192,000,000	826,000,000	1.443	0.176	0.157	0.620	0.080	analyze_locks.exe	grid_bounds_intersect
⊞GdipCreateSolidFill	692,000,000	548,000,000	1.263	0.307	0.000	0.711	0.087	gdiplus.dll	GdipCreateSolidFill
⊞[TBB Scheduler Internals]	280,000,000	72,000,000	3.889	0.268	0.000	0.071	0.705	tbb.dll	tbb::internal::custom_scheduler<stru
⊞pos2grid	238,000,000	224,000,000	1.063	0.221	0.200	0.527	0.053	analyze_locks.exe	pos2grid
⊞[rdpdd.dll]	236,000,000	514,000,000	0.459	0.456	0.117	0.000	0.477	rdpdd.dll	[rdpdd.dll]
⊞tri_intersect	198,000,000	234,000,000	0.846	0.341	0.013	0.609	0.038	analyze_locks.exe	tri_intersect
⊞shader	176,000,000	142,000,000	1.239	0.227	0.000	0.815	0.028	analyze_locks.exe	shader(struct ray *)
⊞Raypnt	138,000,000	270,000,000	0.511	0.543	0.000	0.583	0.091	analyze_locks.exe	Raypnt(struct ray *,double)
⊞intersect_objects	86,000,000	58,000,000	1.483	0.116	0.349	0.419	0.233	analyze_locks.exe	intersect_objects(struct ray *)
⊞VNorm	80,000,000	64,000,000	1.250	0.250	0.125	0.625	0.000	analyze_locks.exe	VNorm(struct vector *)
⊞KeSynchronizeExecution	74,000,000	10,000,000	7.400	0.169	0.000	0.392	0.473	ntoskrnl.exe	KeSynchronizeExecution
Selected 1 row(s):	7,982,000,000	10,640,000,000	0.750	0.316	0.147	0.429	0.129		

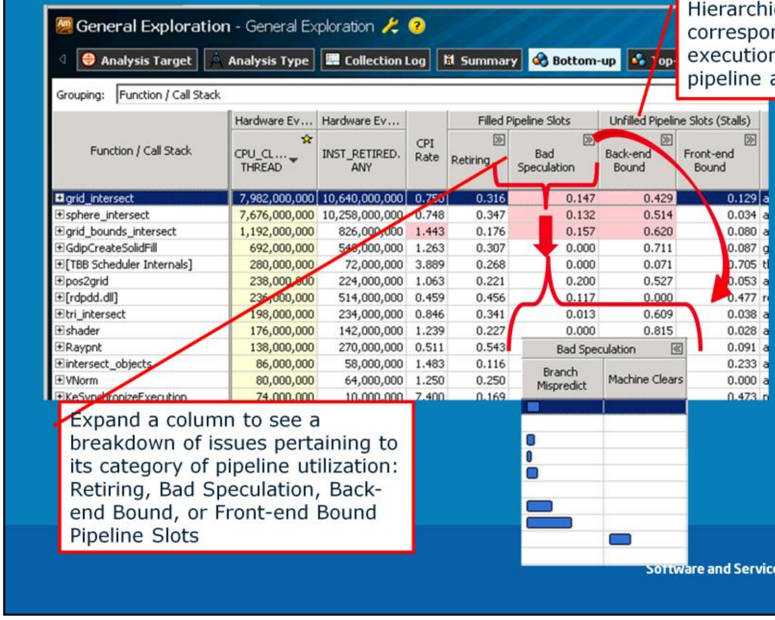
No filters are applied. Process: Any Process Thread: Any Thread Module: Any Module

Timeline Hardware Event: CPU_CLK_UNHALTED.THREAD Call Stack Mode: Only user functions Inline Mode: on Loop Mode: Functions only

The enhanced view is present when running the General Exploration profile with the General Exploration viewpoint selected (the default).

All collected data is presented in hierarchical format (see next slide), with helpful metrics already calculated (see issue slides).

Enhanced General Exploration View for Intel® Microarchitecture Codename Sandy Bridge



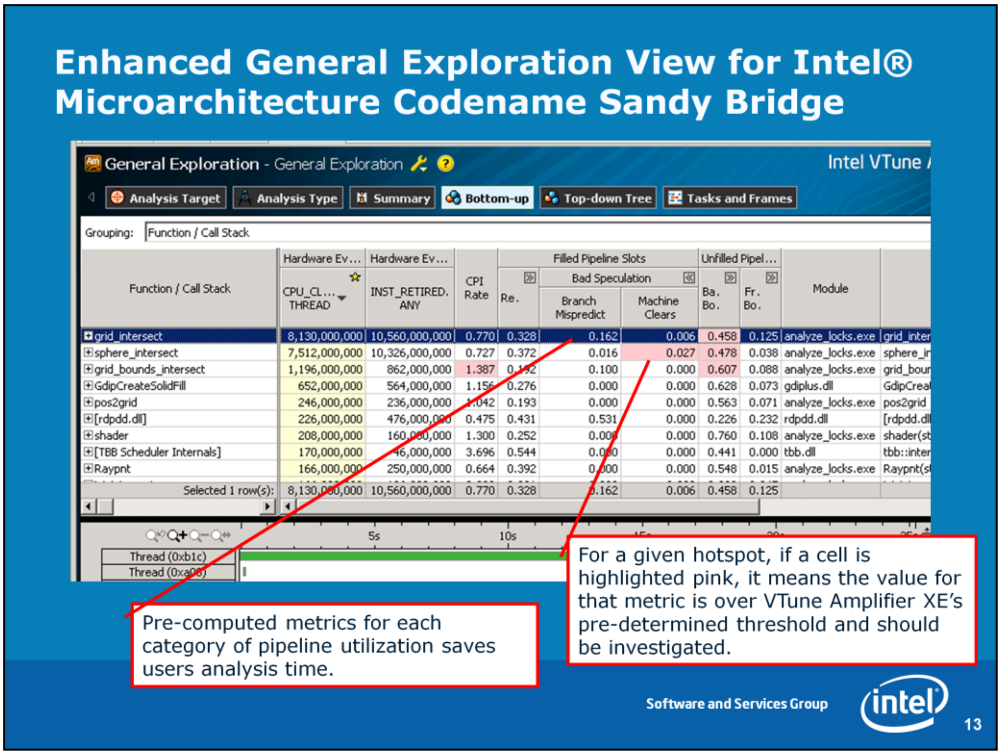
Grouping: Function / Call Stack

Function / Call Stack	Hardware Ev… CPU_CL… THREAD	Hardware Ev… INST_RETIRED.ANY	CPI Rate	Filled Pipeline Slots: Retiring	Filled Pipeline Slots: Bad Speculation	Unfilled Pipeline Slots (Stalls): Back-end Bound	Unfilled Pipeline Slots (Stalls): Front-end Bound
grid_intersect	7,982,000,000	10,640,000,000	0.750	0.316	0.147	0.429	0.129
sphere_intersect	7,676,000,000	10,258,000,000	0.748	0.347	0.132	0.514	0.034
grid_bounds_intersect	1,192,000,000	826,000,000	1.443	0.176	0.157	0.620	0.080
GdipCreateSolidFill	692,000,000	548,000,000	1.263	0.307	0.000	0.711	0.087
[TBB Scheduler Internals]	280,000,000	72,000,000	3.889	0.268	0.000	0.071	0.705
pos2grid	238,000,000	224,000,000	1.063	0.221	0.200	0.527	0.053
[rdpdd.dll]	236,000,000	514,000,000	0.459	0.456	0.117	0.000	0.477
tri_intersect	198,000,000	234,000,000	0.846	0.341	0.013	0.609	0.038
shader	176,000,000	142,000,000	1.239	0.227	0.000	0.815	0.028
Raypnt	138,000,000	270,000,000	0.511	0.543			0.091
intersect_objects	86,000,000	58,000,000	1.483	0.116			0.233
VNorm	80,000,000	64,000,000	1.250	0.250			0.000
KeSynchronizeExecution	74,000,000	10,000,000	7.400	0.169			0.473

Bad Speculation: Branch Mispredict	Bad Speculation: Machine Clears
▬	
▬	
▬	
▬	
▬	
▬	
	▬

Hierarchical data display corresponds to how available execution slots in each core's pipeline are utilized.

Expand a column to see a breakdown of issues pertaining to its category of pipeline utilization: Retiring, Bad Speculation, Back-end Bound, or Front-end Bound Pipeline Slots



Enhanced General Exploration View for Intel® Microarchitecture Codename Sandy Bridge

Grouping: Function / Call Stack

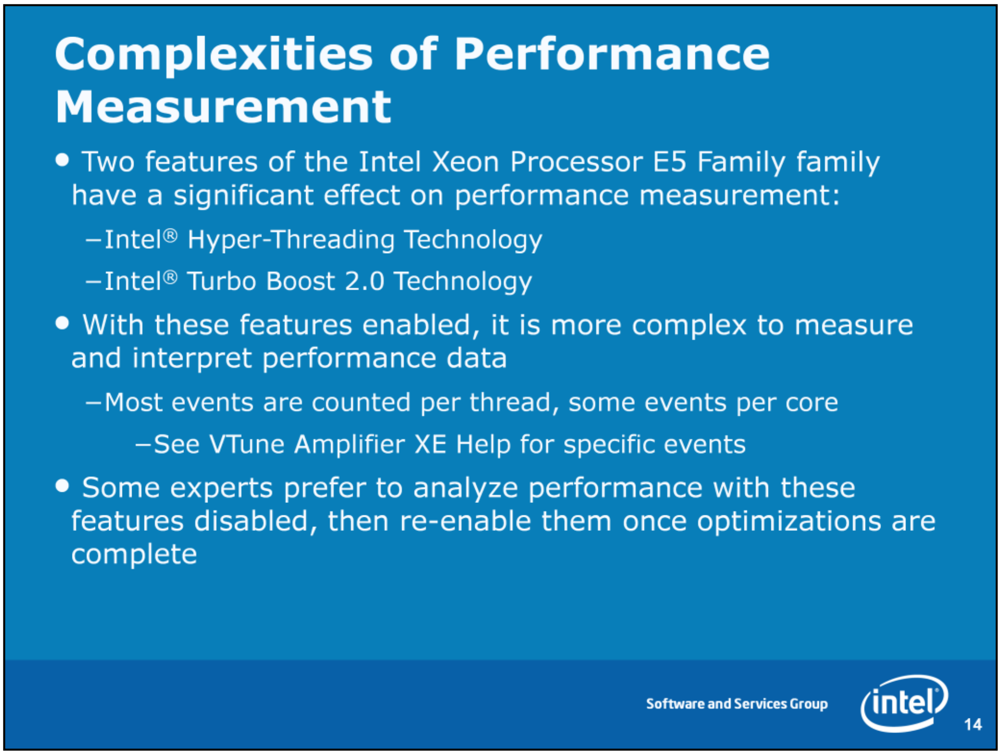
	Hardware Ev…	Hardware Ev…		Filled Pipeline Slots			Unfilled Pipel…			
					Bad Speculation					
Function / Call Stack	CPU_CL… THREAD	INST_RETIRED. ANY	CPI Rate	Re.	Branch Mispredict	Machine Clears	Ba. Bo.	Fr. Bo.	Module	
grid_intersect	8,130,000,000	10,560,000,000	0.770	0.328	0.162	0.006	0.458	0.125	analyze_locks.exe	grid_inter
sphere_intersect	7,512,000,000	10,326,000,000	0.727	0.372	0.016	0.027	0.478	0.038	analyze_locks.exe	sphere_ir
grid_bounds_intersect	1,196,000,000	862,000,000	1.387	0.192	0.100	0.000	0.607	0.088	analyze_locks.exe	grid_bour
GdipCreateSolidFill	652,000,000	564,000,000	1.156	0.276	0.000	0.000	0.628	0.073	gdiplus.dll	GdipCreat
pos2grid	246,000,000	236,000,000	1.042	0.193	0.000	0.000	0.563	0.071	analyze_locks.exe	pos2grid
[rdpdd.dll]	226,000,000	476,000,000	0.475	0.431	0.531	0.000	0.226	0.232	rdpdd.dll	[rdpdd.dll
shader	208,000,000	160,000,000	1.300	0.252	0.000	0.000	0.760	0.108	analyze_locks.exe	shader(st
[TBB Scheduler Internals]	170,000,000	46,000,000	3.696	0.544	0.000	0.000	0.441	0.000	tbb.dll	tbb::inter
Raypnt	166,000,000	250,000,000	0.664	0.392	0.000	0.000	0.548	0.015	analyze_locks.exe	Raypnt(s
Selected 1 row(s):	8,130,000,000	10,560,000,000	0.770	0.328	0.162	0.006	0.458	0.125		

For a given hotspot, if a cell is highlighted pink, it means the value for that metric is over VTune Amplifier XE's pre-determined threshold and should be investigated.

Pre-computed metrics for each category of pipeline utilization saves users analysis time.

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



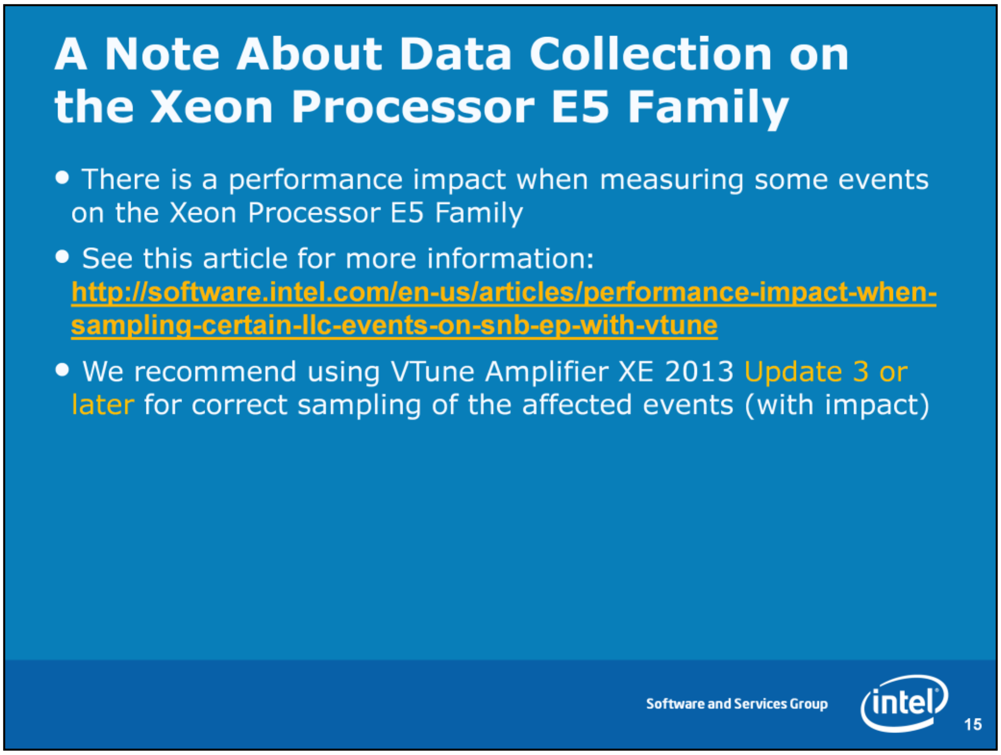
Complexities of Performance Measurement

- Two features of the Intel Xeon Processor E5 Family family have a significant effect on performance measurement:
  - Intel® Hyper-Threading Technology
  - Intel® Turbo Boost 2.0 Technology
- With these features enabled, it is more complex to measure and interpret performance data
  - Most events are counted per thread, some events per core
    - See VTune Amplifier XE Help for specific events
- Some experts prefer to analyze performance with these features disabled, then re-enable them once optimizations are complete

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!



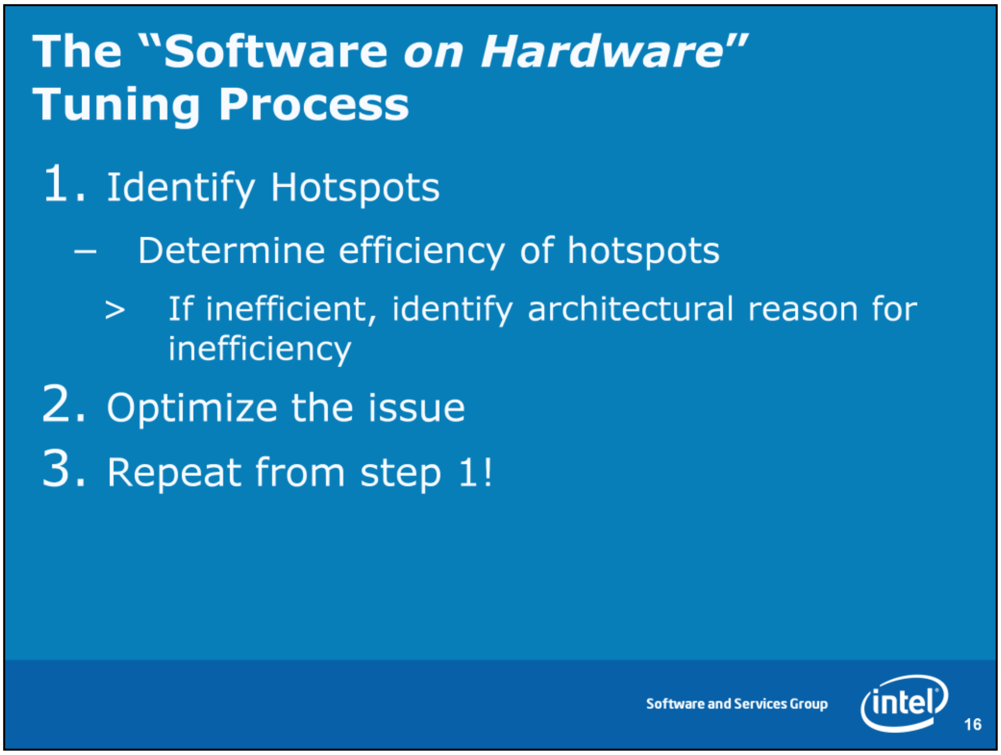
A Note About Data Collection on the Xeon Processor E5 Family

- There is a performance impact when measuring some events on the Xeon Processor E5 Family
- See this article for more information: http://software.intel.com/en-us/articles/performance-impact-when-sampling-certain-llc-events-on-snb-ep-with-vtune
- We recommend using VTune Amplifier XE 2013 Update 3 or later for correct sampling of the affected events (with impact)

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!



The "Software *on Hardware*" Tuning Process

1. Identify Hotspots
   – Determine efficiency of hotspots
     > If inefficient, identify architectural reason for inefficiency
2. Optimize the issue
3. Repeat from step 1!

Software and Services Group

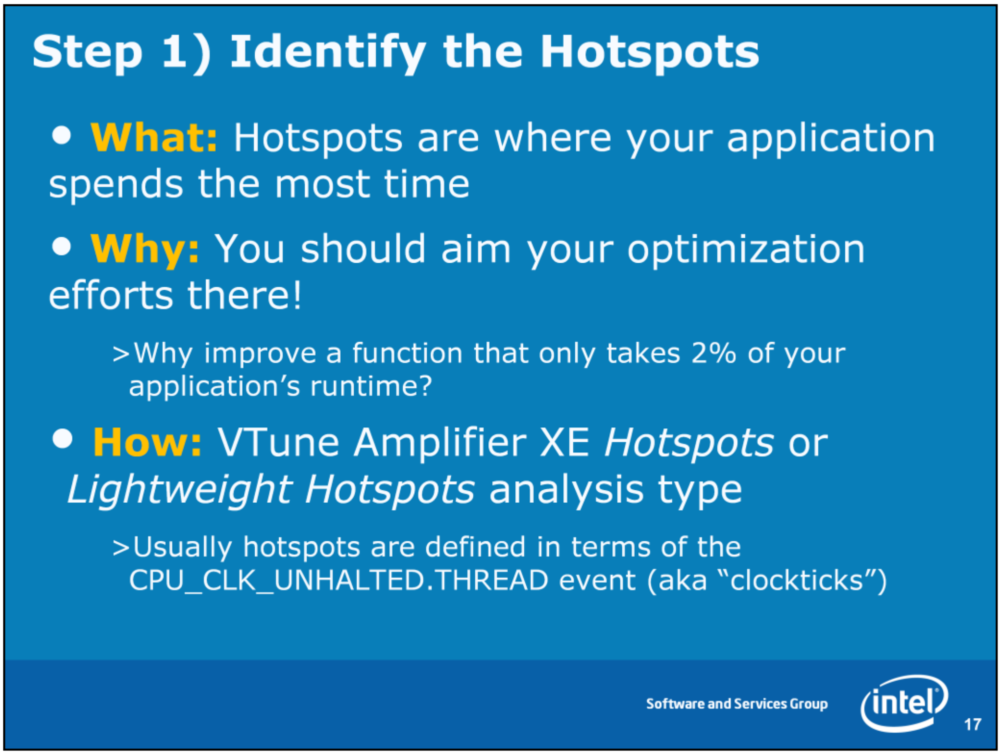
16

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!

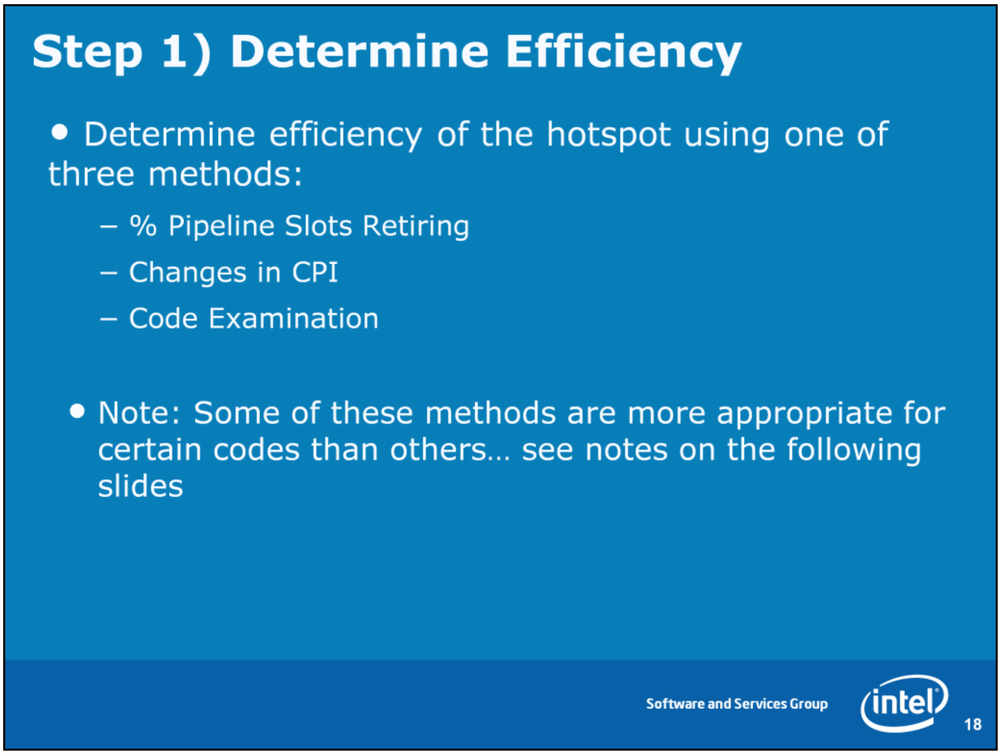


Step 1) Identify the Hotspots

- **What:** Hotspots are where your application spends the most time
- **Why:** You should aim your optimization efforts there!
  - >Why improve a function that only takes 2% of your application's runtime?
- **How:** VTune Amplifier XE *Hotspots* or *Lightweight Hotspots* analysis type
  - >Usually hotspots are defined in terms of the CPU_CLK_UNHALTED.THREAD event (aka "clockticks")

For the Xeon processor E5 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.



Step 1) Determine Efficiency

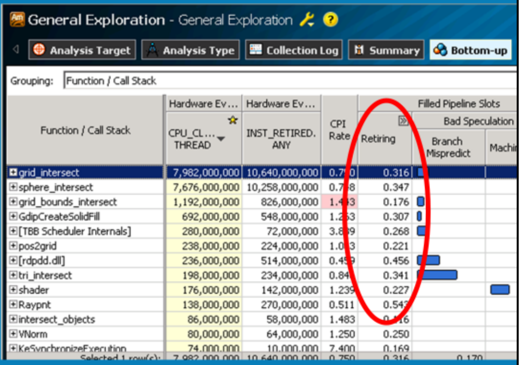
- Determine efficiency of the hotspot using one of three methods:
  - % Pipeline Slots Retiring
  - Changes in CPI
  - Code Examination

- Note: Some of these methods are more appropriate for certain codes than others… see notes on the following slides

% 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: % Retiring Pipeline Slots

- **Why:** Helps you understand how efficiently your app is using the processors
- **How:** General Exploration profile, Metric: *Retiring*
- **What Now:**
  - For a given hotspot:
  - In general, > 75% retiring (.8 or higher) is good. Go to efficiency method 3.
  - < 75% for client or HPC apps - Consider investigating stall reduction using the following issue slides.
  - < 50% for server apps – consider investigating stall reduction.



Grouping: Function / Call Stack

Function / Call Stack	Hardware Ev…	Hardware Ev…	CPI Rate	Filled Pipeline Slots		
					Bad Speculation	
	CPU_CL… THREAD	INST_RETIRED. ANY		Retiring	Branch Mispredict	Machi
⊞grid_intersect	7,982,000,000	10,640,000,000	0.750	0.316		
⊞sphere_intersect	7,676,000,000	10,258,000,000	0.748	0.347		
⊞grid_bounds_intersect	1,192,000,000	826,000,000	1.443	0.176		
⊞GdipCreateSolidFill	692,000,000	548,000,000	1.263	0.307		
⊞[TBB Scheduler Internals]	280,000,000	72,000,000	3.889	0.268		
⊞pos2grid	238,000,000	224,000,000	1.063	0.221		
⊞[rdpdd.dll]	236,000,000	514,000,000	0.459	0.456		
⊞tri_intersect	198,000,000	234,000,000	0.846	0.341		
⊞shader	176,000,000	142,000,000	1.239	0.227		
⊞Raypnt	138,000,000	270,000,000	0.511	0.543		
⊞intersect_objects	86,000,000	58,000,000	1.483	0.416		
⊞VNorm	80,000,000	64,000,000	1.250	0.250		
⊞KeSynchronizeExecution	74,000,000	10,000,000	7.400	0.169		
Selected 1 row(s):	7,982,000,000	10,640,000,000	0.750	0.316	0.170	

Formula:

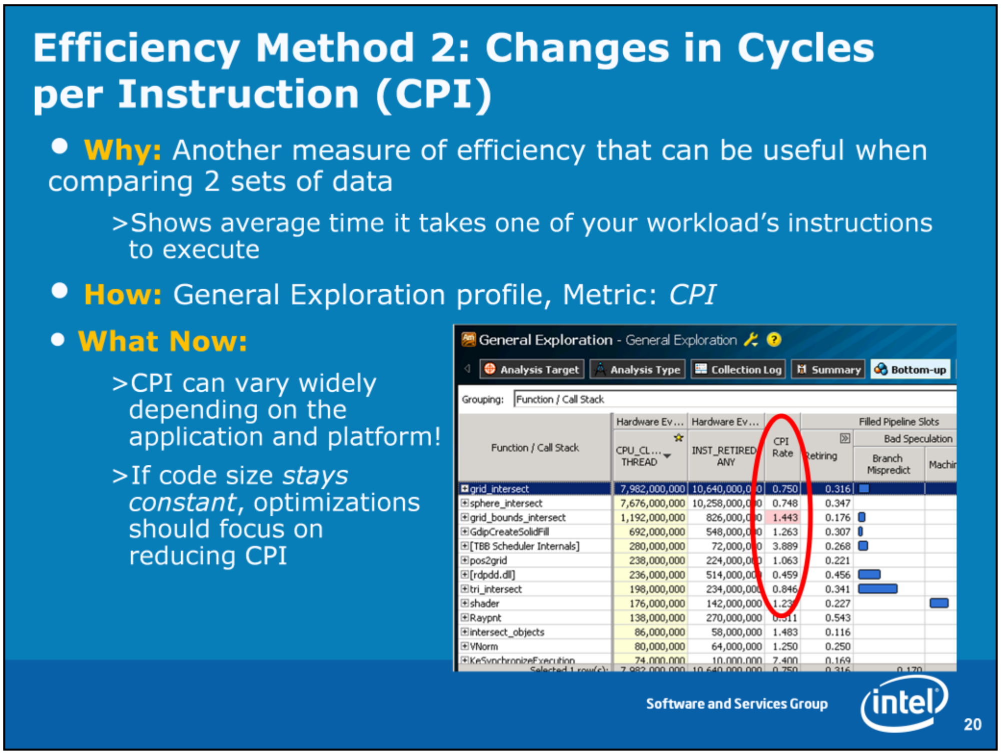
(UOPS_RETIRED.RETIRE_SLOTS/ (4*CPU_CLK_UNHALTED.THREAD))

Thresholds: Investigate if -

% Retiring < .5

This metric is based on the fact that when operating at peak performance, the pipeline on a Xeon E5 Family 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).



Efficiency Method 2: Changes in Cycles per Instruction (CPI)

- **Why:** Another measure of efficiency that can be useful when comparing 2 sets of data
  - >Shows average time it takes one of your workload's instructions to execute
- **How:** General Exploration profile, Metric: *CPI*
- **What Now:**
  - >CPI can vary widely depending on the application and platform!
  - >If code size *stays constant*, optimizations should focus on reducing CPI

Grouping: Function / Call Stack

Function / Call Stack	Hardware Ev…	Hardware Ev…			Filled Pipeline Slots	
					Bad Speculation	
	CPU_CL… THREAD	INST_RETIRED ANY	CPI Rate	Retiring	Branch Mispredict	Machin
grid_intersect	7,982,000,000	10,640,000,000	0.750	0.316	▬	
sphere_intersect	7,676,000,000	10,258,000,000	0.748	0.347		
grid_bounds_intersect	1,192,000,000	826,000,000	1.443	0.176	▬	
GdipCreateSolidFill	692,000,000	548,000,000	1.263	0.307	▬	
[TBB Scheduler Internals]	280,000,000	72,000,000	3.889	0.268	▬	
pos2grid	238,000,000	224,000,000	1.063	0.221		
[rdpdd.dll]	236,000,000	514,000,000	0.459	0.456	▬	
tri_intersect	198,000,000	234,000,000	0.846	0.341	▬	
shader	176,000,000	142,000,000	1.23	0.227		▬
Raypnt	138,000,000	270,000,000	0.511	0.543		
intersect_objects	86,000,000	58,000,000	1.483	0.116		
VNorm	80,000,000	64,000,000	1.250	0.250		
KeSynchronizeExecution	74,000,000	10,000,000	7.400	0.169		
Selected 1 row(s):	7,982,000,000	10,640,000,000	0.750	0.316	0.170	

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!).

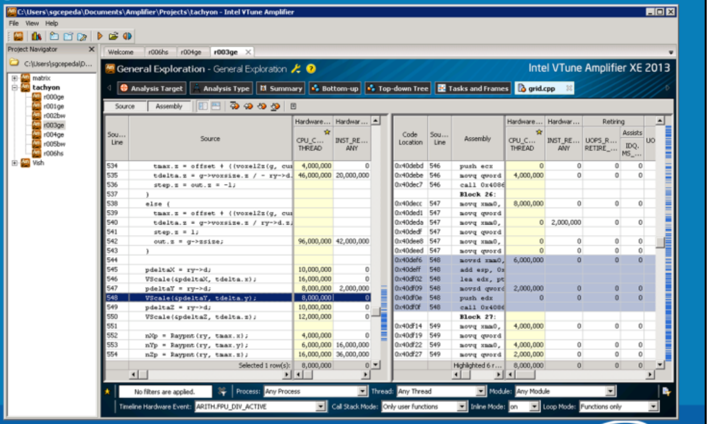
Note that CPI is a ratio! Cycles per instruction. So if the code size changes for a binary, CPI will change. In general, if CPI reduces as a result of optimizations, that is good, and if it increases, that is bad. However there are exceptions! Some code can have a very low CPI but still be inefficient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



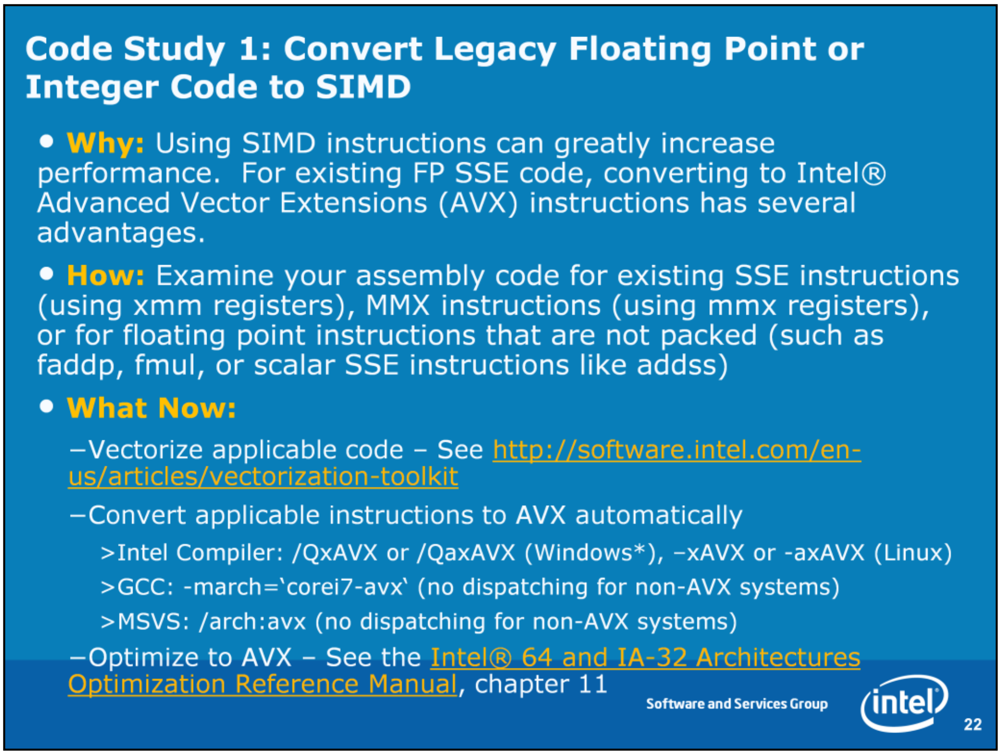
Sou... Line	Source	Hardware...	Hardwar...
		CPU_C... THREAD	INST_RE... ANY
534	`tmax.z = offset + ((voxel2z(g, cu`	4,000,000	0
535	`tdelta.z = g->voxsize.z / - ry->d.`	46,000,000	20,000,000
536	`step.z = out.z = -1;`		
537	`}`		
538	`else {`		
539	`tmax.z = offset + ((voxel2z(g, cu`		
540	`tdelta.z = g->voxsize.z / ry->d.z;`		
541	`step.z = 1;`		
542	`out.z = g->zsize;`	96,000,000	42,000,000
543	`}`		
544			
545	`pdeltaX = ry->d;`	10,000,000	0
546	`VScale(&pdeltaX, tdelta.x);`	16,000,000	0
547	`pdeltaY = ry->d;`	8,000,000	2,000,000
548	`VScale(&pdeltaY, tdelta.y);`	8,000,000	0
549	`pdeltaZ = ry->d;`	10,000,000	0
550	`VScale(&pdeltaZ, tdelta.z);`	12,000,000	0
551			
552	`nXp = Raypnt(ry, tmax.x);`	4,000,000	0
553	`nYp = Raypnt(ry, tmax.y);`	6,000,000	16,000,000
554	`nZp = Raypnt(ry, tmax.z);`	16,000,000	36,000,000
	Selected 1 row(s):	8,000,000	0

Code Location	Sou... Line	Assembly	Hardware...	Hardwar...	Retiring		UO
						Assists	
			CPU_C... THREAD	INST_RE... ANY	UOPS_R... RETIRE_...	IDQ. MS_...	
0x40debd	546	`push ecx`	0	0	0	0	
0x40debe	546	`movq qword`	4,000,000	0	0	0	
0x40dec7	546	`call 0x4086`					
		Block 26:					
0x40decc	547	`movq xmm0,`	8,000,000	0	0	0	
0x40ded1	547	`movq qword`					
0x40deda	547	`movq xmm0,`	0	2,000,000	0	0	
0x40dedf	547	`movq qword`					
0x40dee8	547	`movq xmm0,`	0	0	0	0	
0x40deed	547	`movq qword`	0	0	0	0	
0x40def6	548	`movsd xmm0,`	6,000,000	0	0	0	
0x40deff	548	`add esp, 0x`					
0x40df02	548	`lea edx, pt`					
0x40df09	548	`movsd qword`	2,000,000	0	0	0	
0x40df0e	548	`push edx`	0	0	0	0	
0x40df0f	548	`call 0x4086`					
		Block 27:					
0x40df14	549	`movq xmm0,`	4,000,000	0	0	0	
0x40df19	549	`movq qword`					
0x40df22	549	`movq xmm0,`	4,000,000	0	0	0	
0x40df27	549	`movq qword`	2,000,000	0	0	0	
		Highlighted 6 r...	8,000,000	0	0	0	

No filters are applied. Process: Any Process Thread: Any Thread Module: Any Module

Timeline Hardware Event: ARITH.FPU_DIV_ACTIVE Call Stack Mode: Only user functions Inline Mode: on Loop Mode: Functions only

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.



Code Study 1: Convert Legacy Floating Point or Integer Code to SIMD

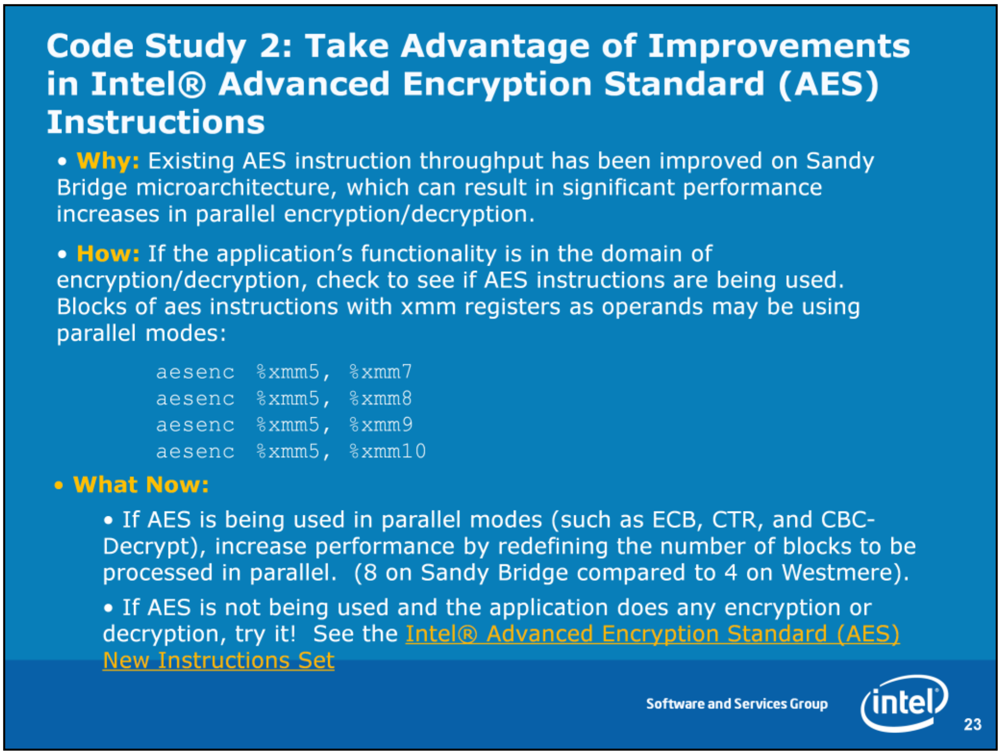
- **Why:** Using SIMD instructions can greatly increase performance. For existing FP SSE code, converting to Intel® Advanced Vector Extensions (AVX) instructions has several advantages.
- **How:** Examine your assembly code for existing SSE instructions (using xmm registers), MMX instructions (using mmx registers), or for floating point instructions that are not packed (such as faddp, fmul, or scalar SSE instructions like addss)
- **What Now:**
  - –Vectorize applicable code – See http://software.intel.com/en-us/articles/vectorization-toolkit
  - –Convert applicable instructions to AVX automatically
    - >Intel Compiler: /QxAVX or /QaxAVX (Windows*), –xAVX or -axAVX (Linux)
    - >GCC: -march=‵corei7-avx‵ (no dispatching for non-AVX systems)
    - >MSVS: /arch:avx (no dispatching for non-AVX systems)
  - –Optimize to AVX – See the Intel® 64 and IA-32 Architectures Optimization Reference Manual, chapter 11

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

A few notes:

Integer vector code is not supported by AVX in the first generation, but integer code can still use 128-bit SSE instructions.

The /Qax and –ax versions of the Intel Compiler switches support automatic dispatching, to create a AV-optimized codepath for appropriate systems and a default codepath otherwise.



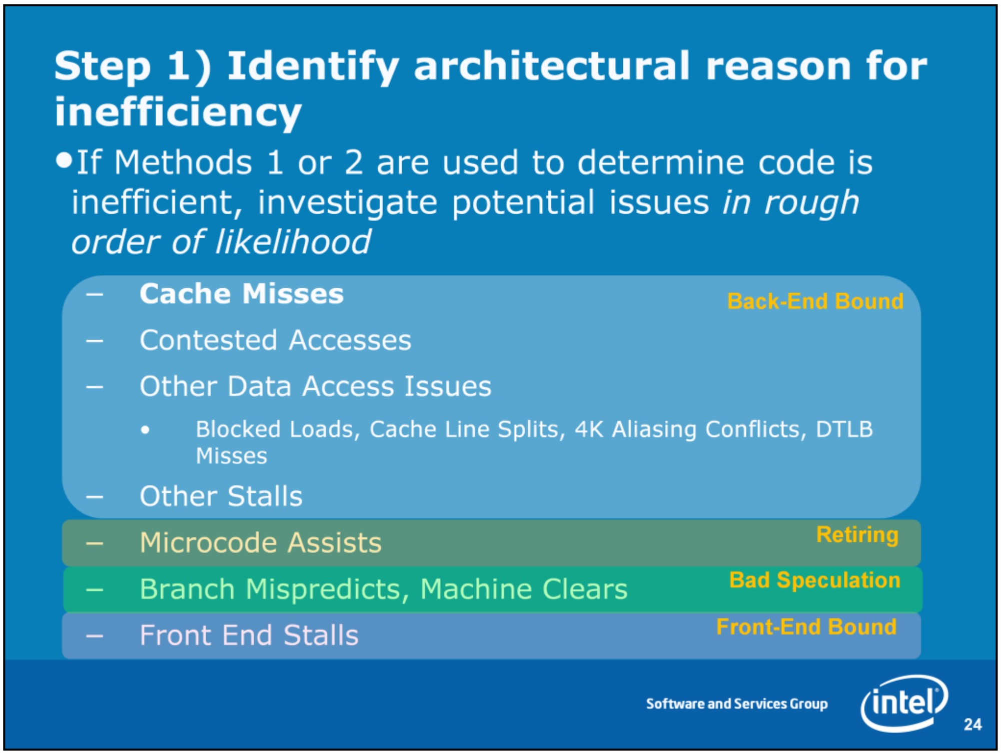
Code Study 2: Take Advantage of Improvements in Intel® Advanced Encryption Standard (AES) Instructions

- **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



Step 1) Identify architectural reason for inefficiency

- If Methods 1 or 2 are used to determine code is inefficient, investigate potential issues *in rough order of likelihood*

Back-End Bound

- – **Cache Misses**
- – Contested Accesses
- – Other Data Access Issues
  - Blocked Loads, Cache Line Splits, 4K Aliasing Conflicts, DTLB Misses
- – Other Stalls

Retiring

- – Microcode Assists

Bad Speculation

- – Branch Mispredicts, Machine Clears

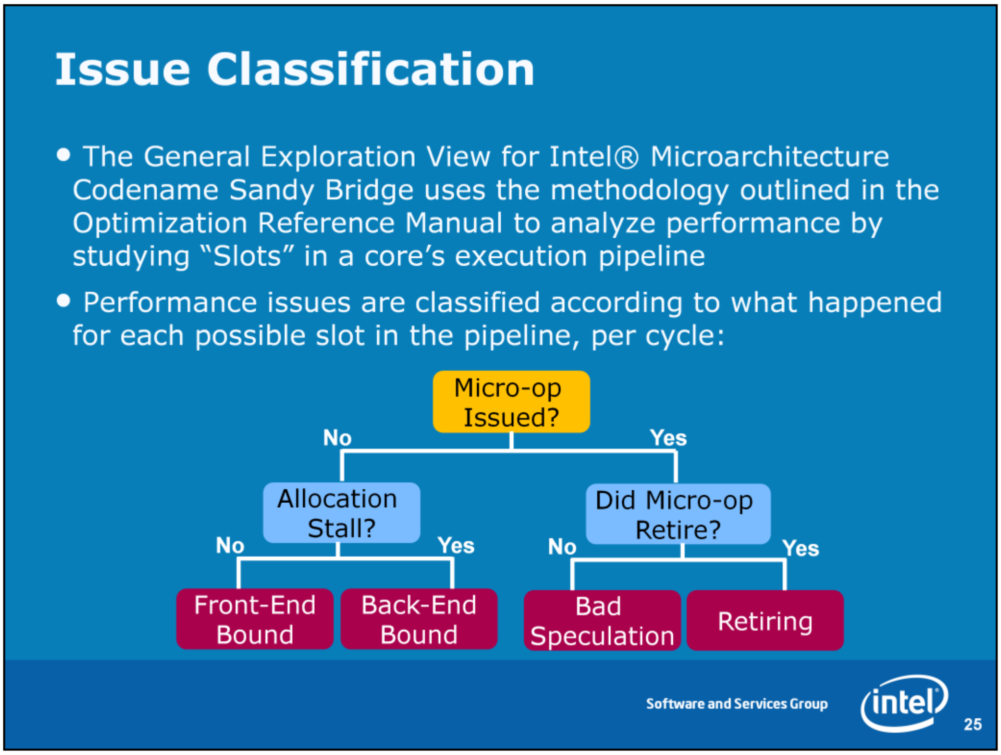
Front-End Bound

- – Front End Stalls

Software and Services Group

24

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.



Issue Classification

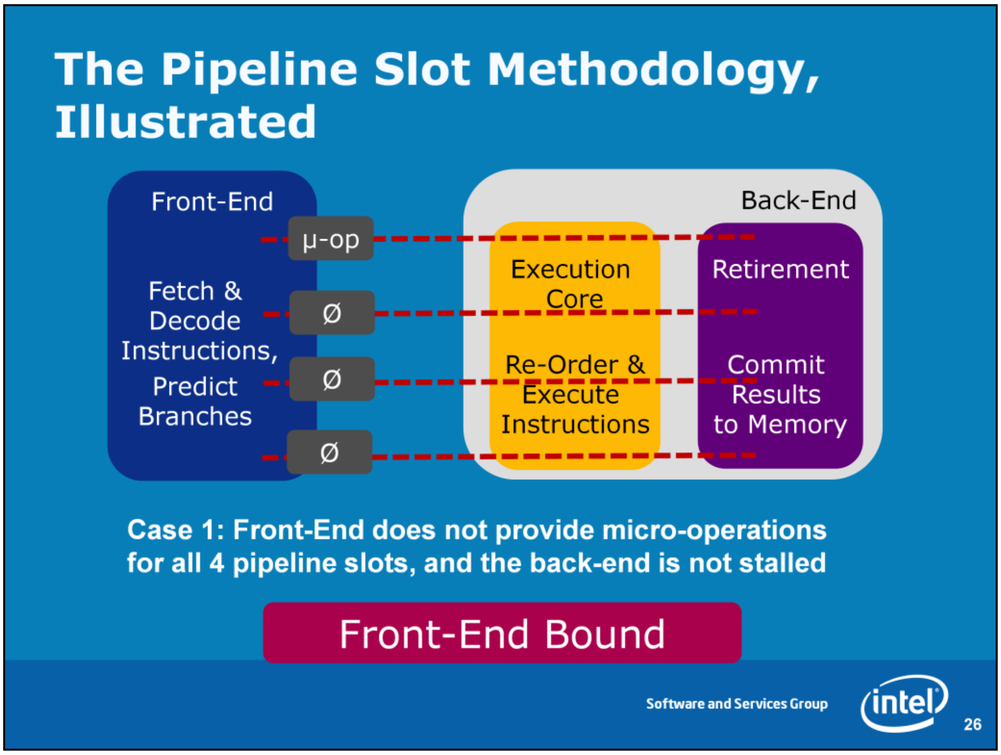
- The General Exploration View for Intel® Microarchitecture Codename Sandy Bridge uses the methodology outlined in the Optimization Reference Manual to analyze performance by studying “Slots” in a core’s execution pipeline
- Performance issues are classified according to what happened for each possible slot in the pipeline, per cycle:

Micro-op Issued?
No
Yes
Allocation Stall?
No
Yes
Front-End Bound
Back-End Bound
Did Micro-op Retire?
No
Yes
Bad Speculation
Retiring

Software and Services Group
25

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://www.youtube.com/watch?v=n8ALDNh4Zes.



The Pipeline Slot Methodology, Illustrated

Front-End

Fetch & Decode Instructions, Predict Branches

μ-op

Ø

Ø

Ø

Back-End

Execution Core

Re-Order & Execute Instructions

Retirement

Commit Results to Memory

Case 1: Front-End does not provide micro-operations for all 4 pipeline slots, and the back-end is not stalled

Front-End Bound

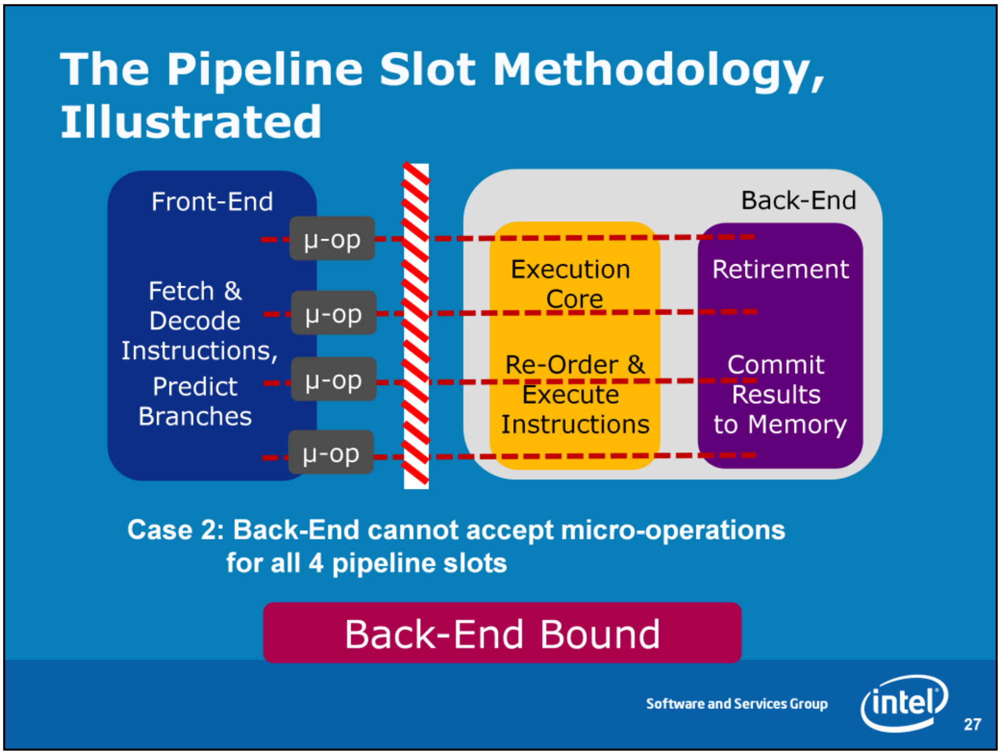
Software and Services Group

intel

26

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 Pipeline Slot Methodology, Illustrated

Front-End

Fetch & Decode Instructions, Predict Branches

μ-op

μ-op

μ-op

μ-op

Back-End

Execution Core

Re-Order & Execute Instructions

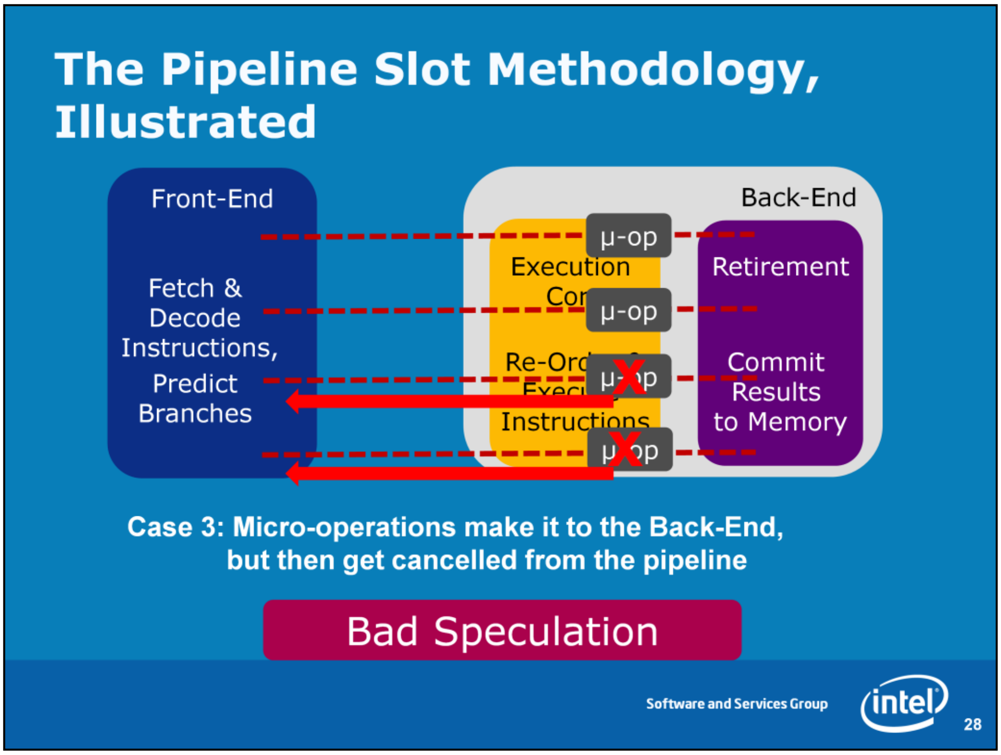
Retirement

Commit Results to Memory

Case 2: Back-End cannot accept micro-operations for all 4 pipeline slots

Back-End Bound

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.



The Pipeline Slot Methodology, Illustrated

Front-End

Fetch & Decode Instructions, Predict Branches

Back-End

Execution Cor

Re-Ord

Execu

Instructions

μ-op

μ-op

μ-op

μ-op

Retirement

Commit Results to Memory

Case 3: Micro-operations make it to the Back-End, but then get cancelled from the pipeline

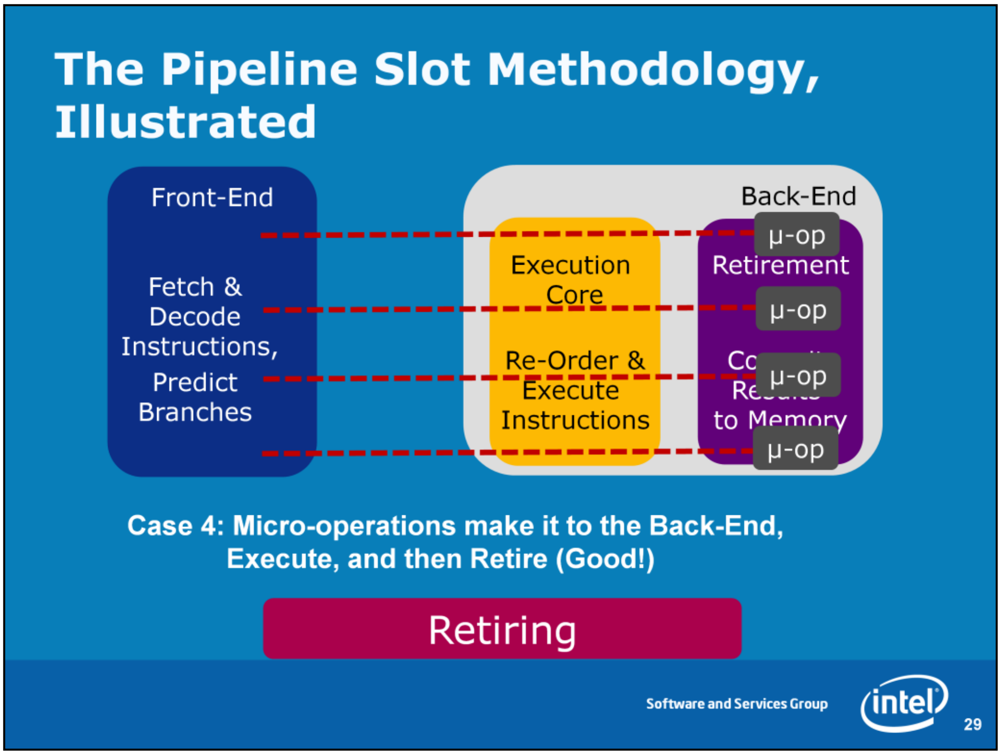
Bad Speculation

Software and Services Group

intel

28

Micro-operations that are removed from the Back-End 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.



The Pipeline Slot Methodology, Illustrated

Front-End

Fetch & Decode Instructions, Predict Branches

Back-End

Execution Core

Re-Order & Execute Instructions

Retirement

Co Re to Memory

µ-op

µ-op

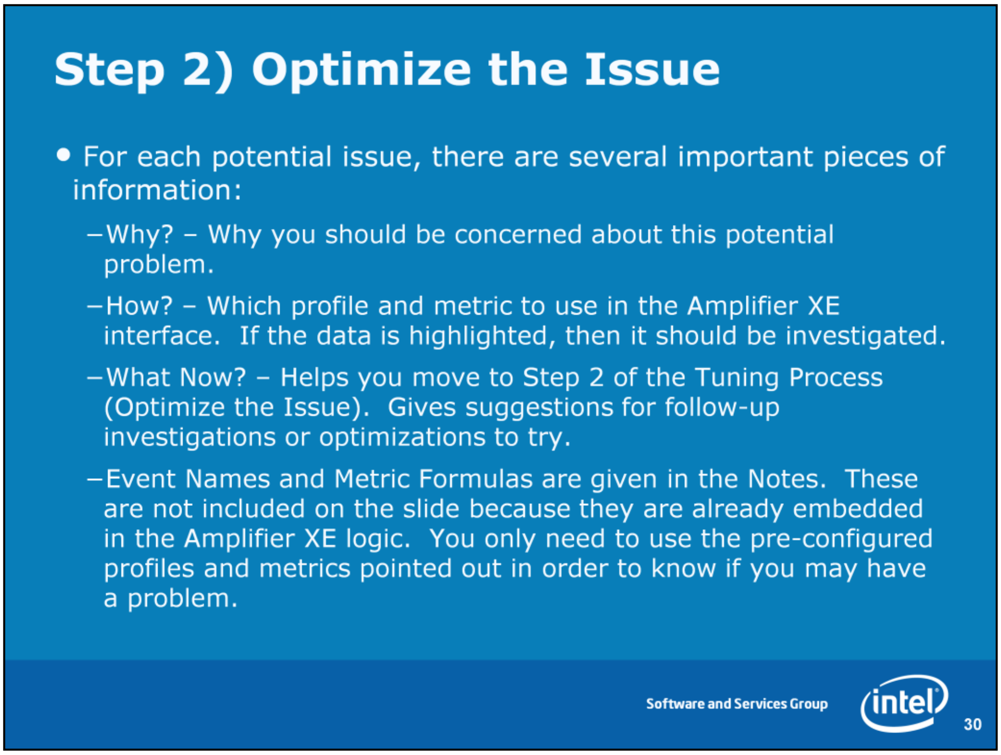
µ-op

µ-op

Case 4: Micro-operations make it to the Back-End, Execute, and then Retire (Good!)

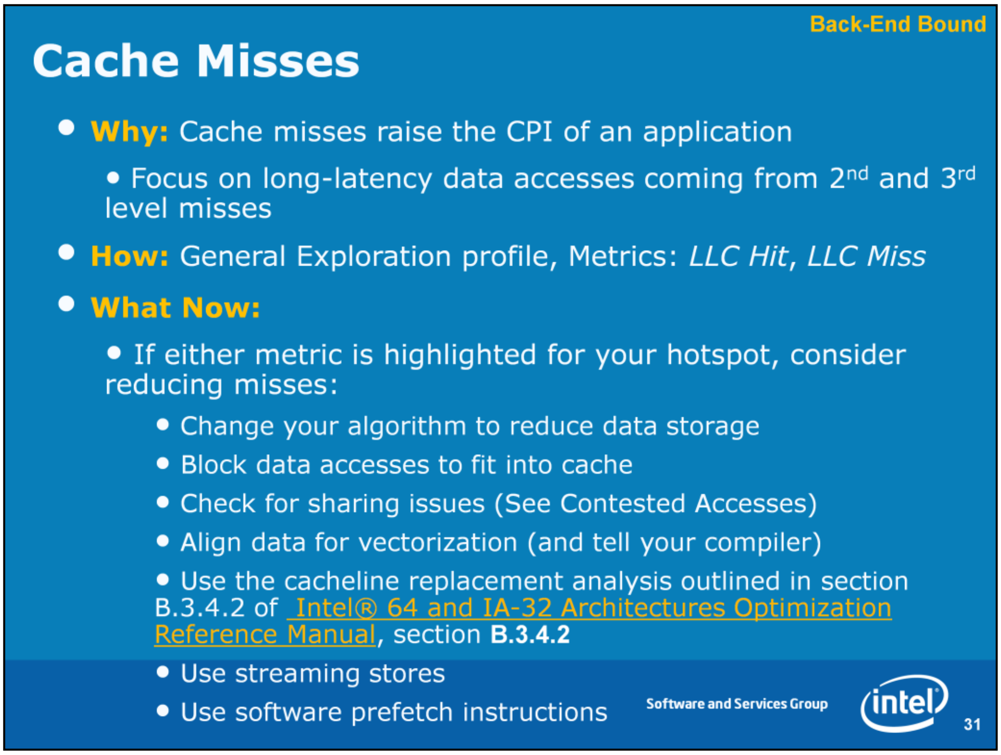
Retiring

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.



Back-End Bound

Cache Misses

- **Why:** Cache misses raise the CPI of an application
  - Focus on long-latency data accesses coming from 2nd and 3rd level misses
- **How:** General Exploration profile, Metrics: *LLC Hit*, *LLC Miss*
- **What Now:**
  - If either metric is highlighted for your hotspot, consider reducing misses:
    - Change your algorithm to reduce data storage
    - Block data accesses to fit into cache
    - Check for sharing issues (See Contested Accesses)
    - Align data for vectorization (and tell your compiler)
    - Use the cacheline replacement analysis outlined in section B.3.4.2 of Intel® 64 and IA-32 Architectures Optimization Reference Manual, section **B.3.4.2**
    - Use streaming stores
    - Use software prefetch instructions

Formulas:

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

(MEM_LOAD_UOPS_MISC_RETIRED.LLC_MISS_PS * 210) / CPU_CLK_UNHALTED.THREAD

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

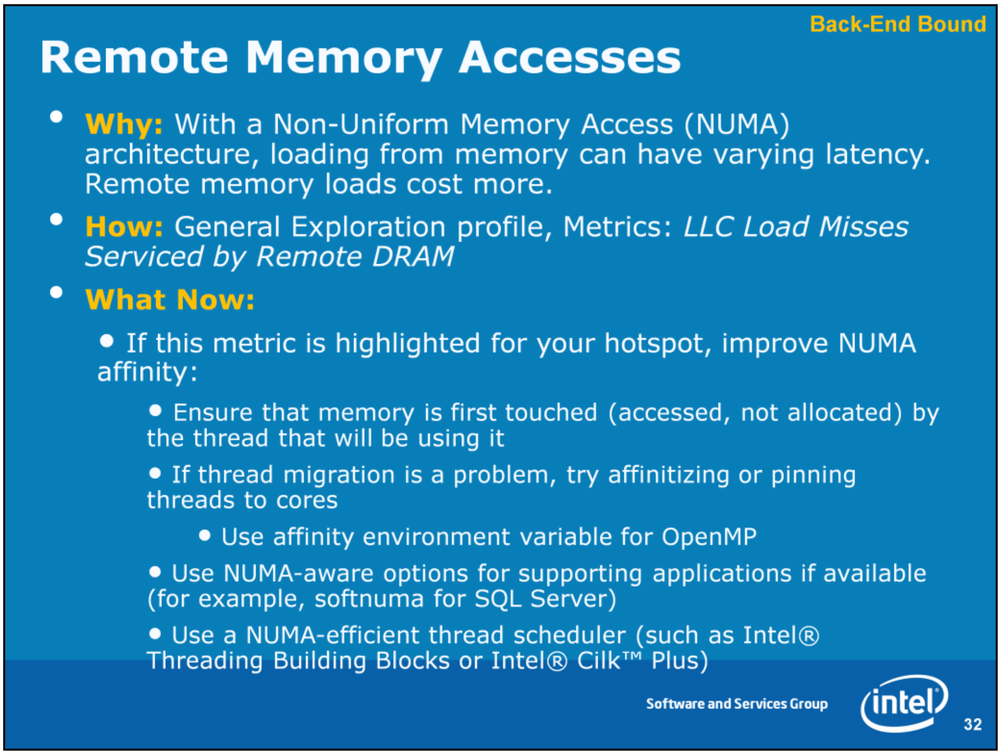
((MEM_LOAD_RETIRED.L3_HIT_PS * 40) + (MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HIT_PS * 88) +
(MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HITM_PS * 99)) / CPU_CLK_UNHALTED.THREAD

Thresholds: Investigate if –

% cycles for LLC miss ≥ .2,

% cycles for LLC Hit ≥ .2

LLC stands for "last level cache", and is the L3 cache on the Xeon processor E5 family. All accesses that miss the L3 cache are counted here as an LLC miss, regardless of whether the data was found in local memory, remote memory, or a remote cache. For the LLC hit formula, it includes standard hits to the L3 as well as hits that required snoops of local L2 caches.



Back-End Bound

Remote Memory Accesses

- **Why:** With a Non-Uniform Memory Access (NUMA) architecture, loading from memory can have varying latency. Remote memory loads cost more.
- **How:** General Exploration profile, Metrics: *LLC Load Misses Serviced by Remote DRAM*
- **What Now:**
  - If this metric is highlighted for your hotspot, improve NUMA affinity:
    - Ensure that memory is first touched (accessed, not allocated) by the thread that will be using it
    - If thread migration is a problem, try affinitizing or pinning threads to cores
      - Use affinity environment variable for OpenMP
    - Use NUMA-aware options for supporting applications if available (for example, softnuma for SQL Server)
    - Use a NUMA-efficient thread scheduler (such as Intel® Threading Building Blocks or Intel® Cilk™ Plus)

Software and Services Group

32

Formulas:

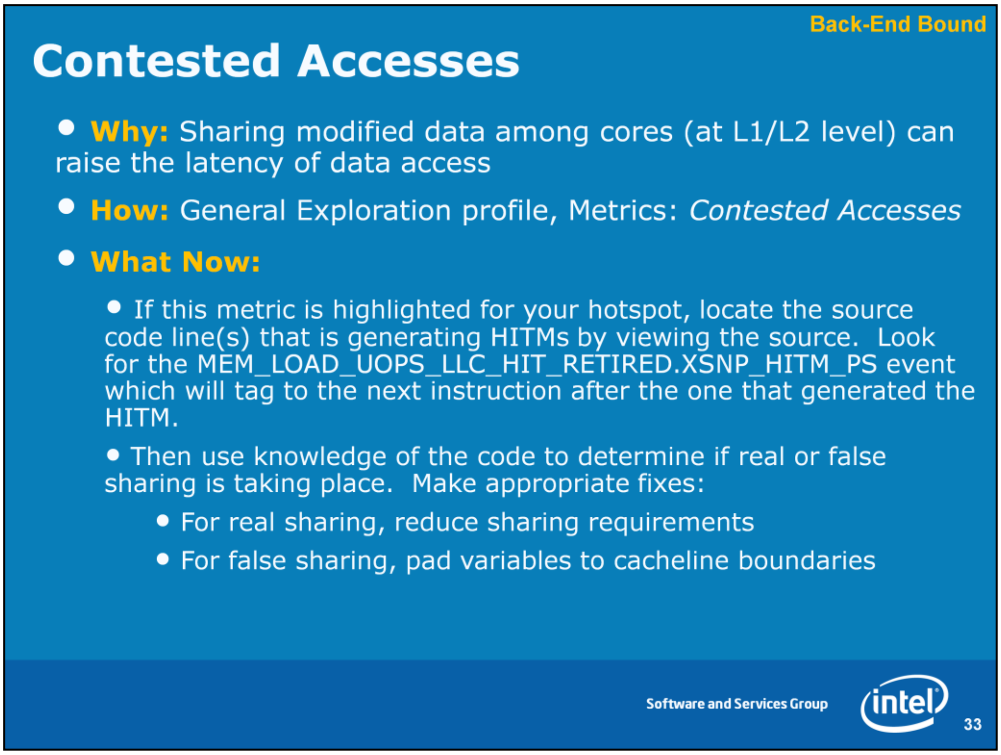
% of cycles spent on memory access (LLC misses):

108 * (OFFCORE_RESPONSE.ALL_DEMAND_MLC_PREF_READS.LLC_MISS.ANY_RESPONSE_1 - OFFCORE_RESPONSE.ALL_DEMAND_MLC_PREF_READS.LLC_MISS.LOCAL_DRAM_0 - OFFCORE_RESPONSE.ALL_DEMAND_MLC_PREF_READS.LLC_MISS.REMOTE_HITM_HIT_FORWARD_1) / CPU_CLK_UNHALTED.THREAD

Thresholds: Investigate if –

% cycles for remote access ≥ .2,

Malloc() or VirtualAlloc() is not touching memory. The o/s only reserves a virtual address for the request. Physical memory is not allocated until the address is accessed. Each 4K page will be physically allocated on the node where the thread makes the first reference. Note that this metric measures only remote *memory* (DRAM) accesses, and does not include data found in cache in the remote socket.



Contested Accesses

- **Why:** Sharing modified data among cores (at L1/L2 level) can raise the latency of data access
- **How:** General Exploration profile, Metrics: *Contested Accesses*
- **What Now:**
  - If this metric is highlighted for your hotspot, locate the source code line(s) that is generating HITMs by viewing the source. Look for the MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HITM_PS event which will tag to the next instruction after the one that generated the HITM.
  - Then use knowledge of the code to determine if real or false sharing is taking place. Make appropriate fixes:
    - For real sharing, reduce sharing requirements
    - For false sharing, pad variables to cacheline boundaries

Formula:

% of cycles spent accessing data modified by another core:

(MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HITM_PS * 60) / CPU_CLK_UNHALTED.THREAD

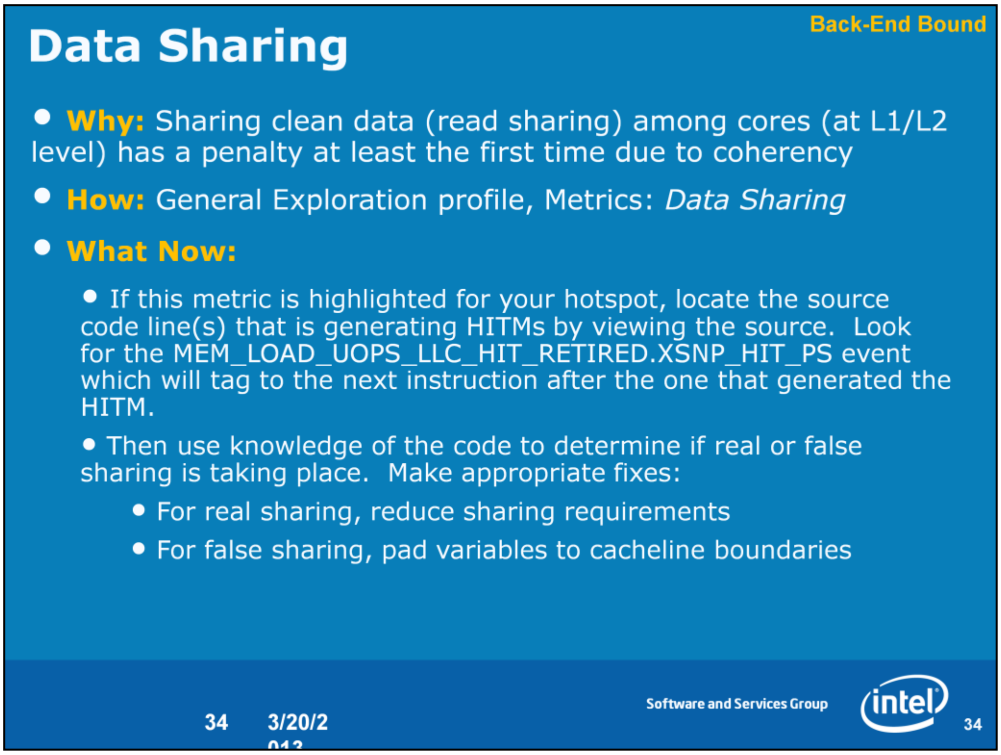
Thresholds: Investigate if –

% 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. This metric measures write sharing at the L2 level only – that is, within one socket. If write sharing is observed at this level it is possible it is occurring across sockets as well.

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).



Back-End Bound

Data Sharing

- **Why:** Sharing clean data (read sharing) among cores (at L1/L2 level) has a penalty at least the first time due to coherency
- **How:** General Exploration profile, Metrics: *Data Sharing*
- **What Now:**
  - If this metric is highlighted for your hotspot, locate the source code line(s) that is generating HITMs by viewing the source. Look for the MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HIT_PS event which will tag to the next instruction after the one that generated the HITM.
  - Then use knowledge of the code to determine if real or false sharing is taking place. Make appropriate fixes:
    - For real sharing, reduce sharing requirements
    - For false sharing, pad variables to cacheline boundaries

Formula:

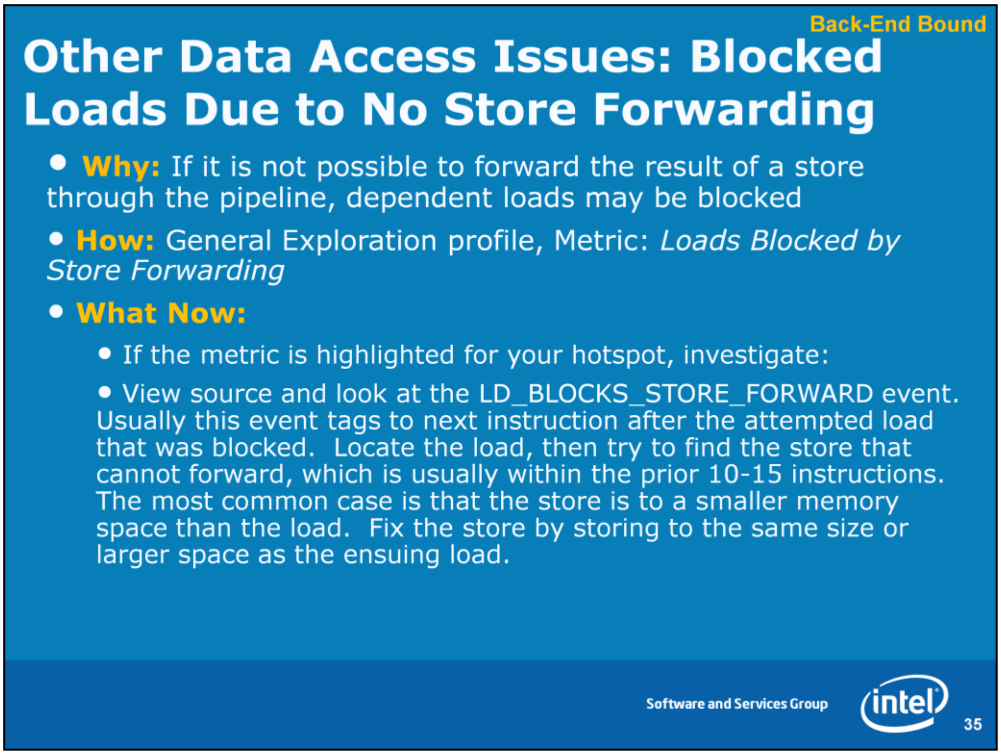
% of cycles spent accessing data modified by another core:

(MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HITM_PS * 43) / CPU_CLK_UNHALTED.THREAD

Thresholds: Investigate if –

% cycles accessing clean shared data > .05

This metric measures read sharing, or sharing of "clean" data, across L2 caches within 1 CPU socket. The L3 cache has a set of "core valid" bits that indicate whether each cacheline could be found in any L2 caches on the same socket, and if so, which ones. The first time a line is brought into the L3 cache, it will have core valid bits set to 1 for whichever L2 cache it went into. If that line is then read by a different core, then it will be fetched from L3, where the core valid bits will indicate it is present in one other core. The other L2 will have to be snooped, resulting in a longer latency access for that line. This metric measures the impact of that additional access time, when the cacheline in question is only being read-shared. In the case of read-sharing, the line can co-exist in multiple L2 caches in shared state, and for future accesses more than one core valid bit will be set. Then when other cores request the line, no L2 caches will need to be snooped, because the presence of 2 or more core valid bits tells the LLC that the line is shared (for reading) and ok to serve. Thus the impact of this only happens the first time a cacheline is requested for reading by a second L2 after it has already been placed in the L3 cache. The impact of sharing modified data across L2s is different and is measured with the "Contested Accesses" metric.



Back-End Bound

Other Data Access Issues: Blocked Loads Due to No Store Forwarding

- **Why:** If it is not possible to forward the result of a store through the pipeline, dependent loads may be blocked
- **How:** General Exploration profile, Metric: *Loads Blocked by Store Forwarding*
- **What Now:**
  - If the metric is highlighted for your hotspot, investigate:
  - View source and look at the LD_BLOCKS_STORE_FORWARD event. Usually this event tags to next instruction after the attempted load that was blocked. Locate the load, then try to find the store that cannot forward, which is usually within the prior 10-15 instructions. The most common case is that the store is to a smaller memory space than the load. Fix the store by storing to the same size or larger space as the ensuing load.

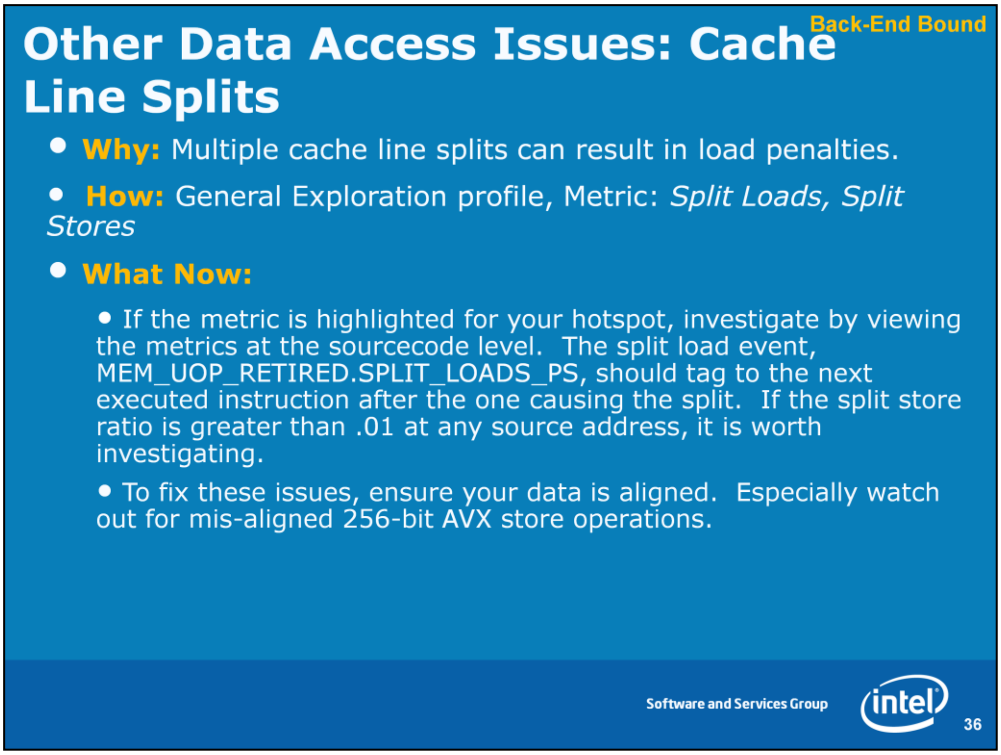
Formula:

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. For more information on what types of stores can forward and which can't, see the Intel® 64 and IA-32 Architectures Optimization Reference Manual, section 2.3.5.2.



Back-End Bound

Other Data Access Issues: Cache Line Splits

- **Why:** Multiple cache line splits can result in load penalties.
- **How:** General Exploration profile, Metric: *Split Loads, Split Stores*
- **What Now:**
  - If the metric is highlighted for your hotspot, investigate by viewing the metrics at the sourcecode level. The split load event, MEM_UOP_RETIRED.SPLIT_LOADS_PS, should tag to the next executed instruction after the one causing the split. If the split store ratio is greater than .01 at any source address, it is worth investigating.
  - To fix these issues, ensure your data is aligned. Especially watch out for mis-aligned 256-bit AVX store operations.

Software and Services Group

36

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

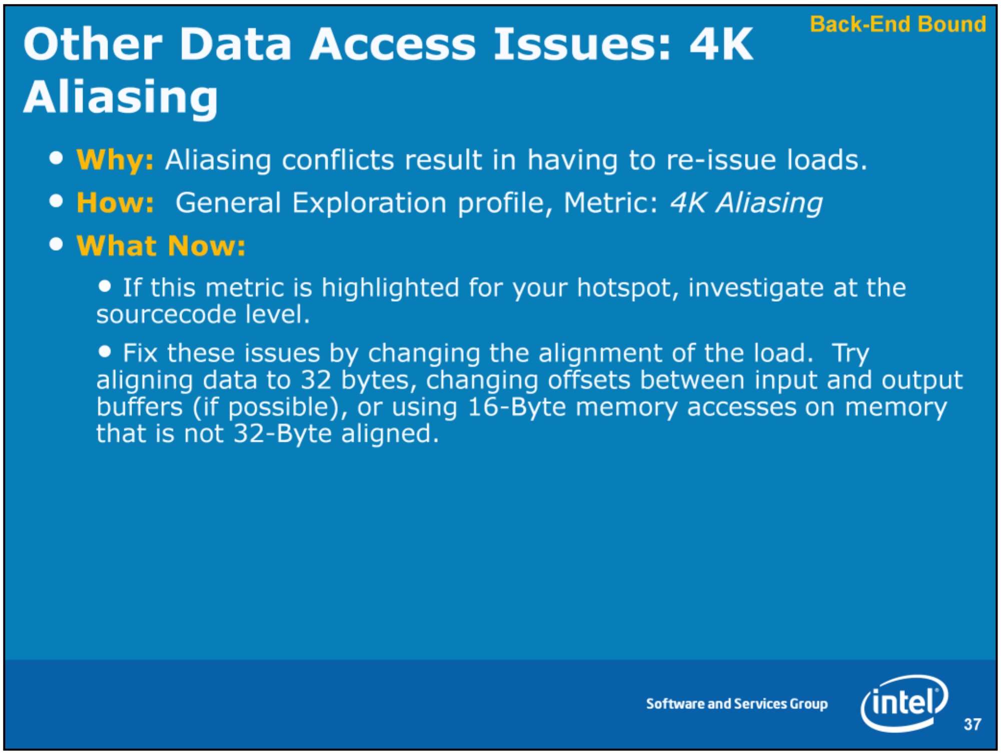
Split Store Ratio = MEM_UOP_RETIRED.SPLIT_STORES_PS / MEM_UOP_RETIRED.ANY_STORES_PS

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.



Back-End Bound

Other Data Access Issues: 4K Aliasing

- **Why:** Aliasing conflicts result in having to re-issue loads.
- **How:** General Exploration profile, Metric: *4K Aliasing*
- **What Now:**
  - If this metric is highlighted for your hotspot, investigate at the sourcecode level.
  - Fix these issues by changing the alignment of the load. Try aligning data to 32 bytes, changing offsets between input and output buffers (if possible), or using 16-Byte memory accesses on memory that is not 32-Byte aligned.

Software and Services Group

37

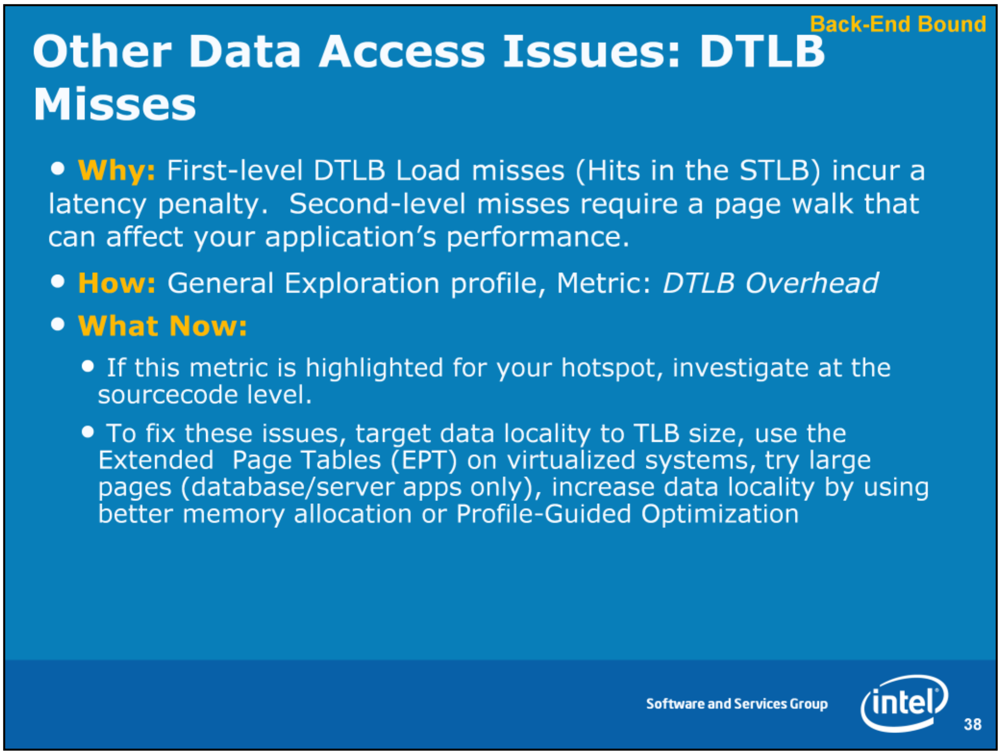
Formula:

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 the load is processed in the pipeline, part of the address is compared to the addresses of other memory operations in flight (the store). Since only part of the address is used for the comparison, the issue of the load will match the previous store. Then the load will be blocked until another check is performed on the full addresses. After the full check determines that the addresses are distinct, the load can proceed. The additional check 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.



Back-End Bound

Other Data Access Issues: DTLB Misses

- **Why:** First-level DTLB Load misses (Hits in the STLB) incur a latency penalty. Second-level misses require a page walk that can affect your application's performance.
- **How:** General Exploration profile, Metric: *DTLB Overhead*
- **What Now:**
  - If this metric is highlighted for your hotspot, investigate at the sourcecode level.
  - To fix these issues, target data locality to TLB size, use the Extended Page Tables (EPT) on virtualized systems, try large pages (database/server apps only), increase data locality by using better memory allocation or Profile-Guided Optimization

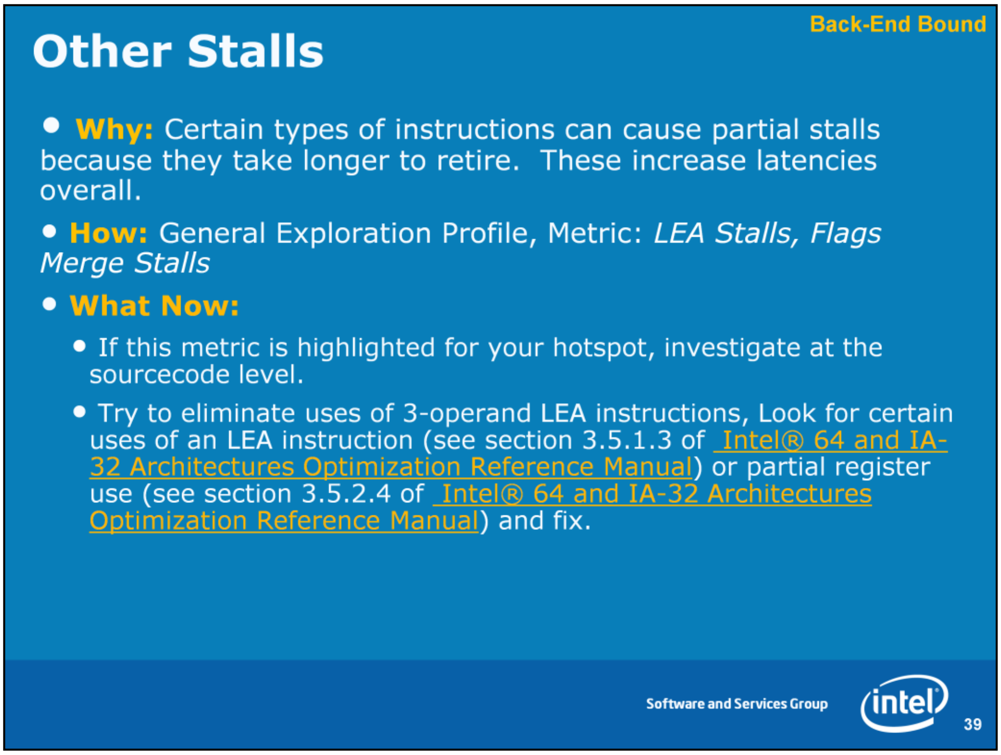
Formula:
DTLB Overhead = ((DTLB_LOAD_MISSES.STLB_HIT * 7) + DTLB_LOAD_MISSES.WALK_DURATION) / CPU_CLK_UNHALTED.THREAD

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



Back-End Bound

Other Stalls

- **Why:** Certain types of instructions can cause partial stalls because they take longer to retire. These increase latencies overall.
- **How:** General Exploration Profile, Metric: *LEA Stalls, Flags Merge Stalls*
- **What Now:**
  - If this metric is highlighted for your hotspot, investigate at the sourcecode level.
  - Try to eliminate uses of 3-operand LEA instructions, Look for certain uses of an LEA instruction (see section 3.5.1.3 of Intel® 64 and IA-32 Architectures Optimization Reference Manual) or partial register use (see section 3.5.2.4 of Intel® 64 and IA-32 Architectures Optimization Reference Manual) and fix.

Software and Services Group

39

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

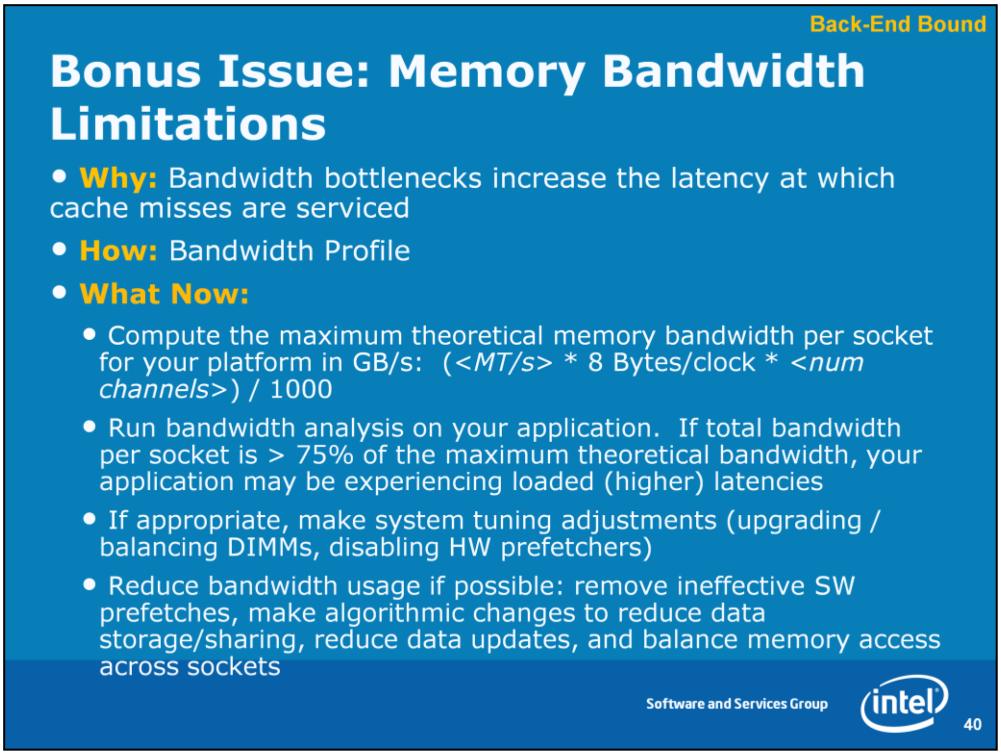
LEA Stalls = PARTIAL_RAT_STALLS.SLOW_LEA_WINDOW/CPU_CLK_UNHALTED.THREAD

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).



Back-End Bound

Bonus Issue: Memory Bandwidth Limitations

- **Why:** Bandwidth bottlenecks increase the latency at which cache misses are serviced
- **How:** Bandwidth Profile
- **What Now:**
  - Compute the maximum theoretical memory bandwidth per socket for your platform in GB/s: (<*MT/s*> * 8 Bytes/clock * <*num channels*>) / 1000
  - Run bandwidth analysis on your application. If total bandwidth per socket is > 75% of the maximum theoretical bandwidth, your application may be experiencing loaded (higher) latencies
  - If appropriate, make system tuning adjustments (upgrading / balancing DIMMs, disabling HW prefetchers)
  - Reduce bandwidth usage if possible: remove ineffective SW prefetches, make algorithmic changes to reduce data storage/sharing, reduce data updates, and balance memory access across sockets

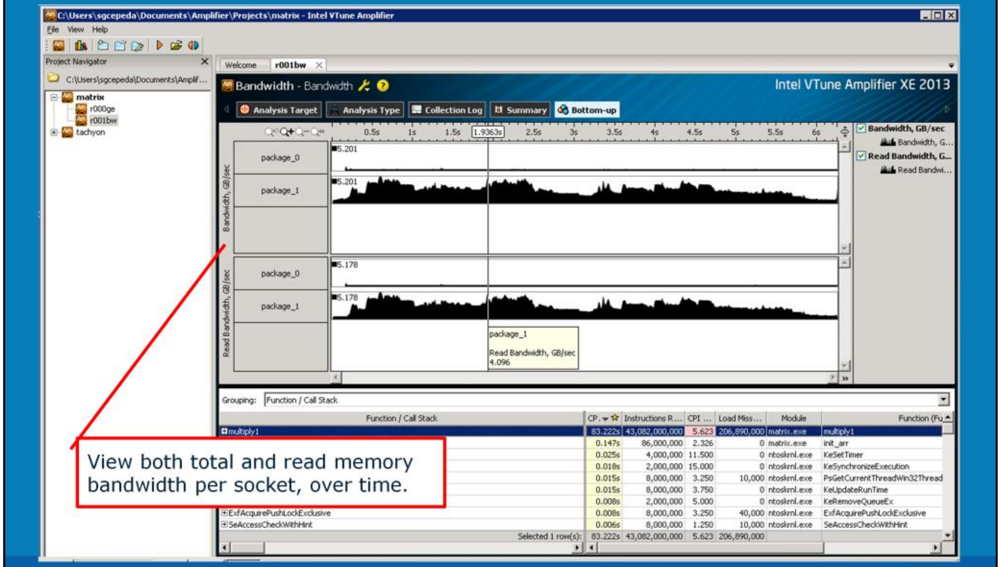
Software and Services Group

40

Max theoretical bandwidth, per socket, for Xeon E5 processor family: 51.2GB/s with 4 memory channels, 38.4 GB/s with 3 memory channels

Back-End Bound

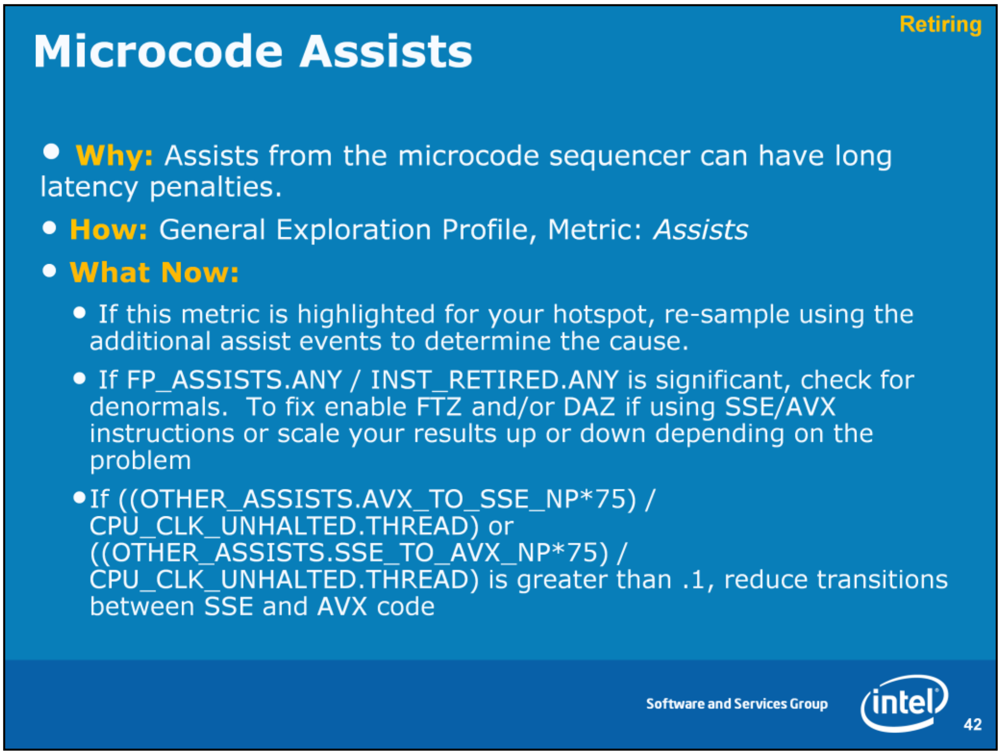
Bandwidth Analysis



Grouping: Function / Call Stack

Function / Call Stack	CP.	Instructions R...	CPI ...	Load Miss...	Module	Function (Fu
⊞multiply1	83.222s	43,082,000,000	5.623	206,890,000	matrix.exe	multiply1
	0.147s	86,000,000	2.326	0	matrix.exe	init_arr
	0.025s	4,000,000	11.500	0	ntoskrnl.exe	KeSetTimer
	0.018s	2,000,000	15.000	0	ntoskrnl.exe	KeSynchronizeExecution
	0.015s	8,000,000	3.250	10,000	ntoskrnl.exe	PsGetCurrentThreadWin32Thread
	0.015s	8,000,000	3.750	0	ntoskrnl.exe	KeUpdateRunTime
	0.008s	2,000,000	5.000	0	ntoskrnl.exe	KeRemoveQueueEx
⊞ExfAcquirePushLockExclusive	0.008s	8,000,000	3.250	40,000	ntoskrnl.exe	ExfAcquirePushLockExclusive
⊞SeAccessCheckWithHint	0.006s	8,000,000	1.250	10,000	ntoskrnl.exe	SeAccessCheckWithHint
Selected 1 row(s):	83.222s	43,082,000,000	5.623	206,890,000		

View both total and read memory bandwidth per socket, over time.



Retiring

Microcode Assists

- **Why:** Assists from the microcode sequencer can have long latency penalties.
- **How:** General Exploration Profile, Metric: *Assists*
- **What Now:**
  - If this metric is highlighted for your hotspot, re-sample using the additional assist events to determine the cause.
  - If FP_ASSISTS.ANY / INST_RETIRED.ANY is significant, check for denormals. To fix enable FTZ and/or DAZ if using SSE/AVX instructions or scale your results up or down depending on the problem
  - If ((OTHER_ASSISTS.AVX_TO_SSE_NP*75) / CPU_CLK_UNHALTED.THREAD) or ((OTHER_ASSISTS.SSE_TO_AVX_NP*75) / CPU_CLK_UNHALTED.THREAD) is greater than .1, reduce transitions between SSE and AVX code

Software and Services Group

42

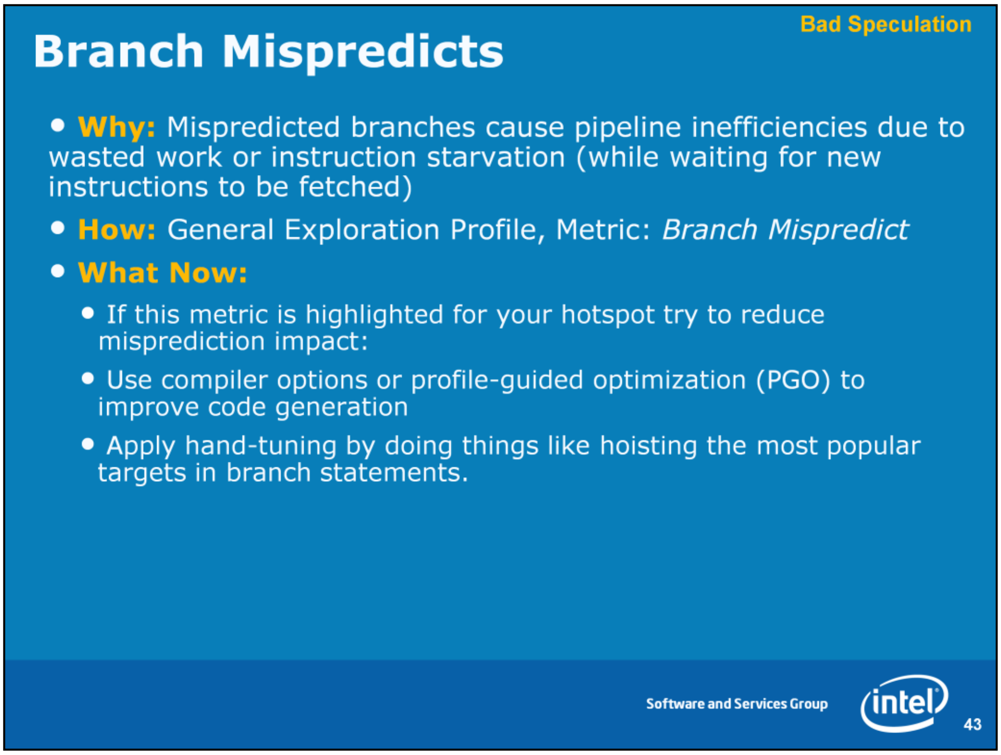
Formula:

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.



Bad Speculation

Branch Mispredicts

- **Why:** Mispredicted branches cause pipeline inefficiencies due to wasted work or instruction starvation (while waiting for new instructions to be fetched)
- **How:** General Exploration Profile, Metric: *Branch Mispredict*
- **What Now:**
  - If this metric is highlighted for your hotspot try to reduce misprediction impact:
  - Use compiler options or profile-guided optimization (PGO) to improve code generation
  - Apply hand-tuning by doing things like hoisting the most popular targets in branch statements.

Formula:

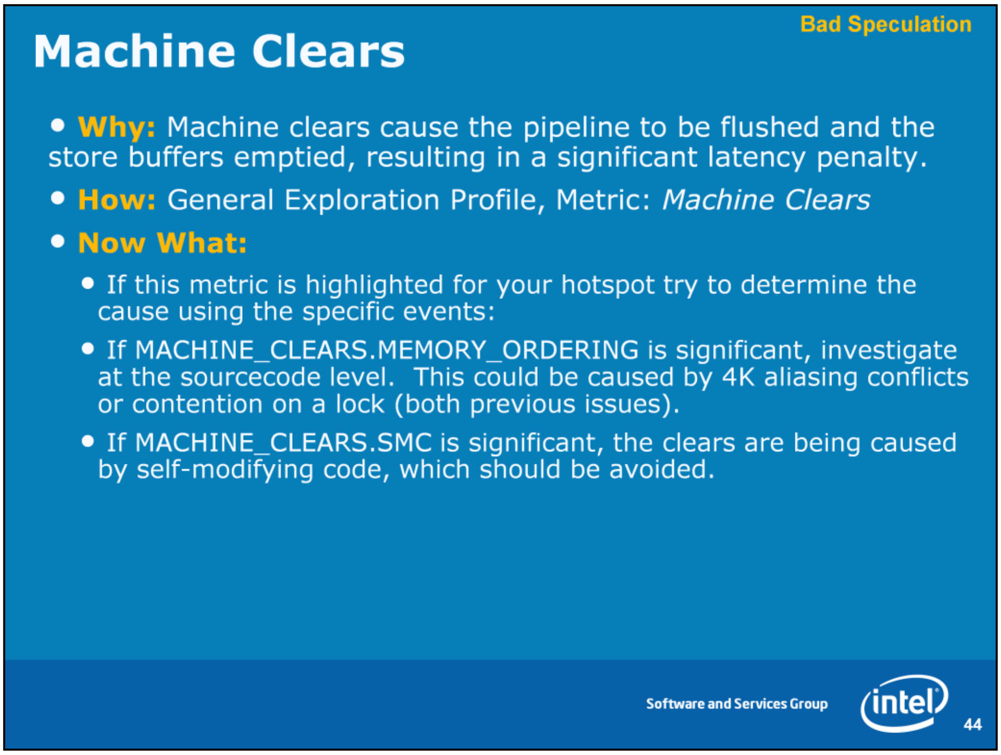
Mispredicted branch cost: (20* BR_MISP_RETIRED.ALL_BRANCHES_PS)/ CPU_CLK_UNHALTED.THREAD

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.



Bad Speculation

Machine Clears

- **Why:** Machine clears cause the pipeline to be flushed and the store buffers emptied, resulting in a significant latency penalty.
- **How:** General Exploration Profile, Metric: *Machine Clears*
- **Now What:**
  - If this metric is highlighted for your hotspot try to determine the cause using the specific events:
  - If MACHINE_CLEARS.MEMORY_ORDERING is significant, investigate at the sourcecode level. This could be caused by 4K aliasing conflicts or contention on a lock (both previous issues).
  - If MACHINE_CLEARS.SMC is significant, the clears are being caused by self-modifying code, which should be avoided.

Software and Services Group

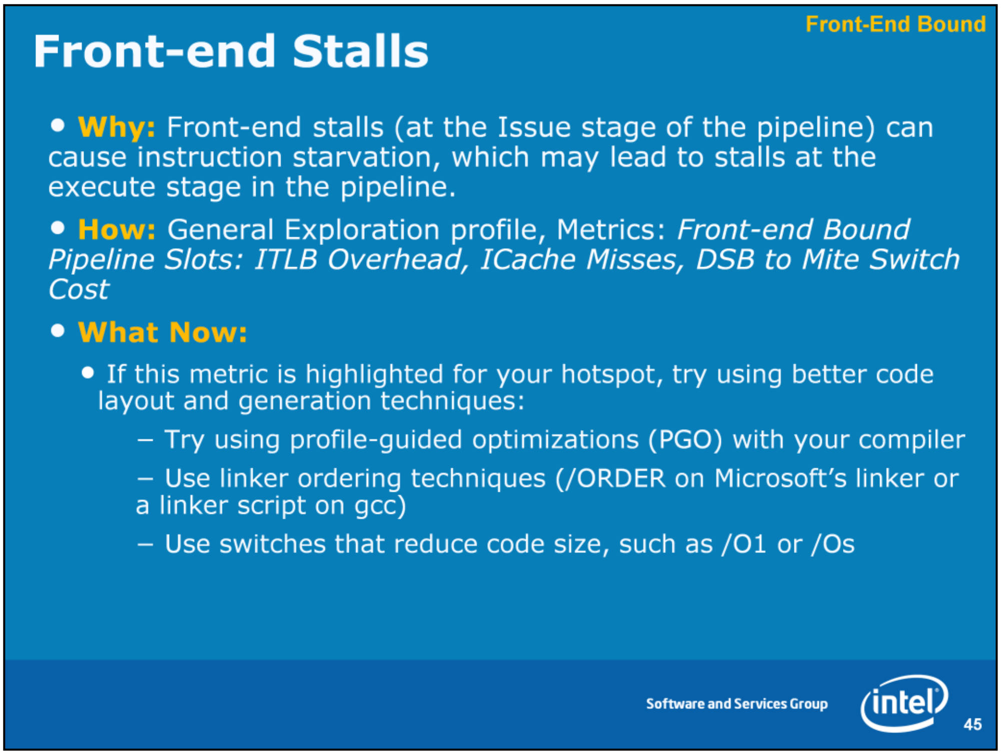
Formula:

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).



Front-End Bound

Front-end Stalls

- **Why:** Front-end stalls (at the Issue stage of the pipeline) can cause instruction starvation, which may lead to stalls at the execute stage in the pipeline.
- **How:** General Exploration profile, Metrics: *Front-end Bound Pipeline Slots: ITLB Overhead, ICache Misses, DSB to Mite Switch Cost*
- **What Now:**
  - If this metric is highlighted for your hotspot, try using better code layout and generation techniques:
    - – Try using profile-guided optimizations (PGO) with your compiler
    - – Use linker ordering techniques (/ORDER on Microsoft’s linker or a linker script on gcc)
    - – Use switches that reduce code size, such as /O1 or /Os

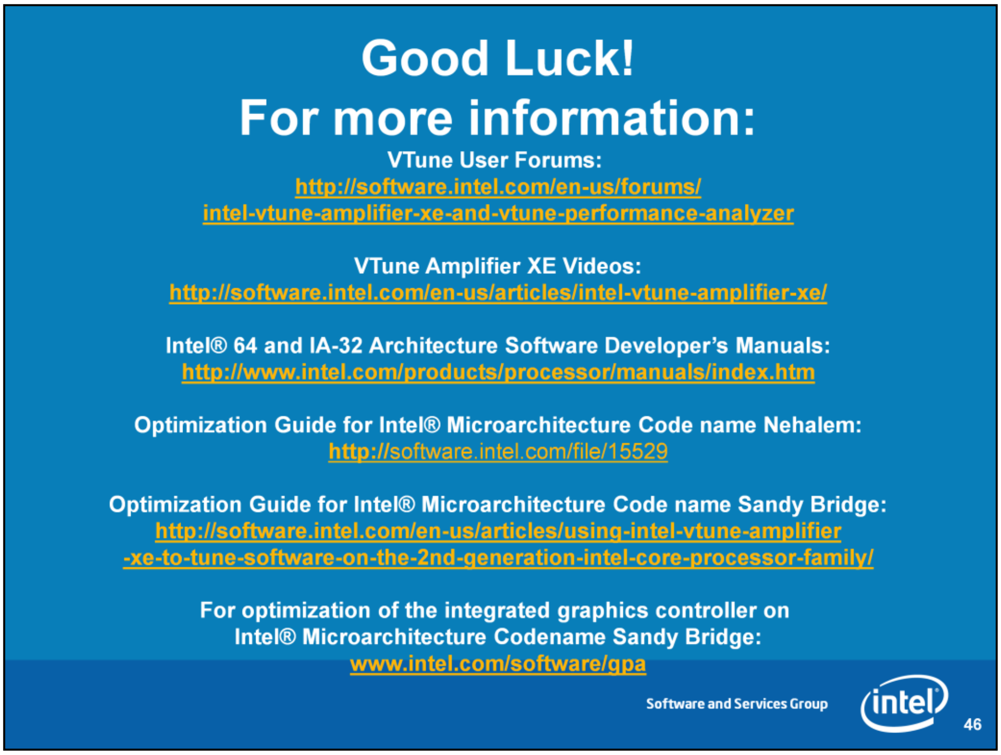
Formula:

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-amplifier-xe-and-vtune-performance-analyzer

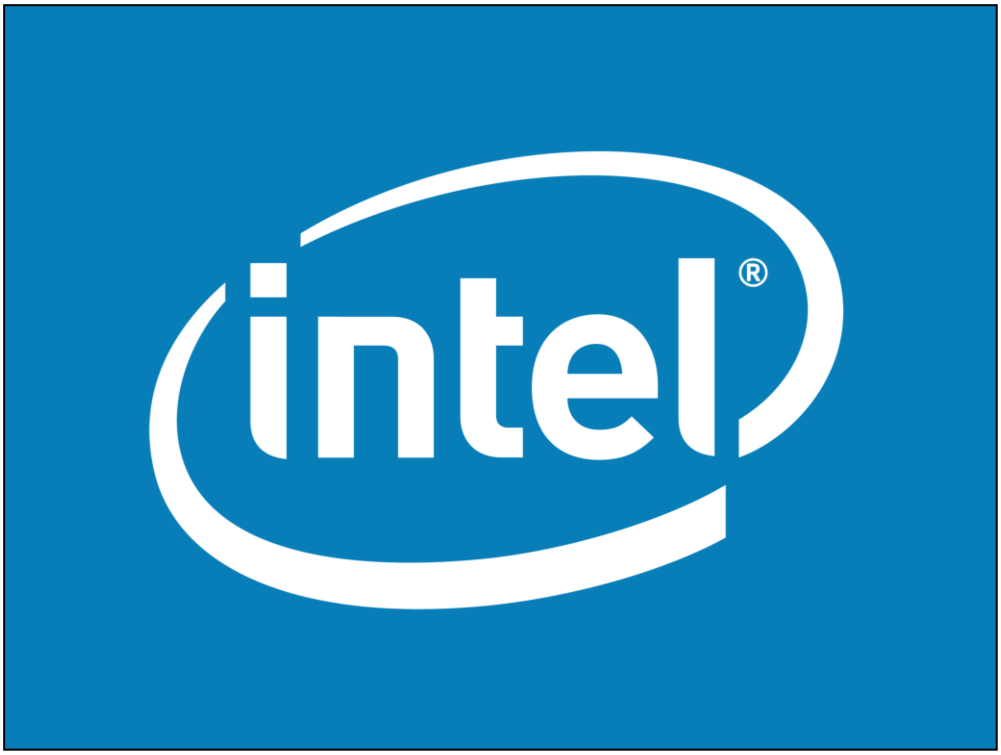
VTune Amplifier XE Videos:
http://software.intel.com/en-us/articles/intel-vtune-amplifier-xe/

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

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

Optimization Guide for Intel® Microarchitecture Code name Sandy Bridge:
http://software.intel.com/en-us/articles/using-intel-vtune-amplifier
-xe-to-tune-software-on-the-2nd-generation-intel-core-processor-family/

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


intel®