

Performance Analysis and Software Enabling for Intel® Core™ i7 Processors*

Presenter: David Levinthal
Principal Engineer

Business Group, Division: DPD, SSG

* Intel, the Intel logo, Intel Core and Core Inside are trademarks of Intel Corporation in the U.S. and other countries.



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.

- All products, computer systems, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice.
- Customers, licensees, and other third parties are not authorized by Intel to use Intel code names in advertising, promotion or marketing of any product or service.
- Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, visit Intel [Performance Benchmark Limitations](#)
- Copyright © 2009, Intel Corporation. All rights reserved.



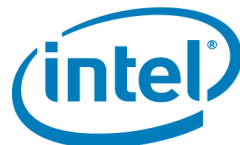
Risk Factors

The above statements and any others in this document that refer to plans and expectations for the first quarter, the year and the future are forward-looking statements that involve a number of risks and uncertainties. Many factors could affect Intel's actual results, and variances from Intel's current expectations regarding such factors could cause actual results to differ materially from those expressed in these forward-looking statements. Intel presently considers the following to be the important factors that could cause actual results to differ materially from the corporation's expectations. Current uncertainty in global economic conditions pose a risk to the overall economy as consumers and businesses may defer purchases in response to tighter credit and negative financial news, which could negatively affect product demand and other related matters. Consequently, demand could be different from Intel's expectations due to factors including changes in business and economic conditions, including conditions in the credit market that could affect consumer confidence; customer acceptance of Intel's and competitors' products; changes in customer order patterns including order cancellations; and changes in the level of inventory at customers. Intel operates in intensely competitive industries that are characterized by a high percentage of costs that are fixed or difficult to reduce in the short term and product demand that is highly variable and difficult to forecast. Revenue and the gross margin percentage are affected by the timing of new Intel product introductions and the demand for and market acceptance of Intel's products; actions taken by Intel's competitors, including product offerings and introductions, marketing programs and pricing pressures and Intel's response to such actions; Intel's ability to respond quickly to technological developments and to incorporate new features into its products; and the availability of sufficient supply of components from suppliers to meet demand. The gross margin percentage could vary significantly from expectations based on changes in revenue levels; capacity utilization; excess or obsolete inventory; product mix and pricing; variations in inventory valuation, including variations related to the timing of qualifying products for sale; manufacturing yields; changes in unit costs; impairments of long-lived assets, including manufacturing, assembly/test and intangible assets; and the timing and execution of the manufacturing ramp and associated costs, including start-up costs. Expenses, particularly certain marketing and compensation expenses, as well as restructuring and asset impairment charges, vary depending on the level of demand for Intel's products and the level of revenue and profits. The recent financial crisis affecting the banking system and financial markets and the going concern threats to investment banks and other financial institutions have resulted in a tightening in the credit markets, a reduced level of liquidity in many financial markets, and extreme volatility in fixed income, credit and equity markets. There could be a number of follow-on effects from the credit crisis on Intel's business, including insolvency of key suppliers resulting in product delays; inability of customers to obtain credit to finance purchases of our products and/or customer insolvencies; counterparty failures negatively impacting our treasury operations; increased expense or inability to obtain short-term financing of Intel's operations from the issuance of commercial paper; and increased impairments from the inability of investee companies to obtain financing. Intel's results could be impacted by adverse economic, social, political and physical/infrastructure conditions in the countries in which Intel, its customers or its suppliers operate, including military conflict and other security risks, natural disasters, infrastructure disruptions, health concerns and fluctuations in currency exchange rates. Intel's results could be affected by adverse effects associated with product defects and errata (deviations from published specifications), and by litigation or regulatory matters involving intellectual property, stockholder, consumer, antitrust and other issues, such as the litigation and regulatory matters described in Intel's SEC reports.



Agenda

- **Performance Analysis and Software Enabling**
- **Processor Overview**
- **Core Pipeline Cycle Accounting**
 - **Non-Intel® Hyper-Threading (Intel® HT) Technology (much easier)**
 - **Intel® HT Technology**
- **Memory Access**
- **Branch Events**
- **Summary**
- **Why are there so many events?**



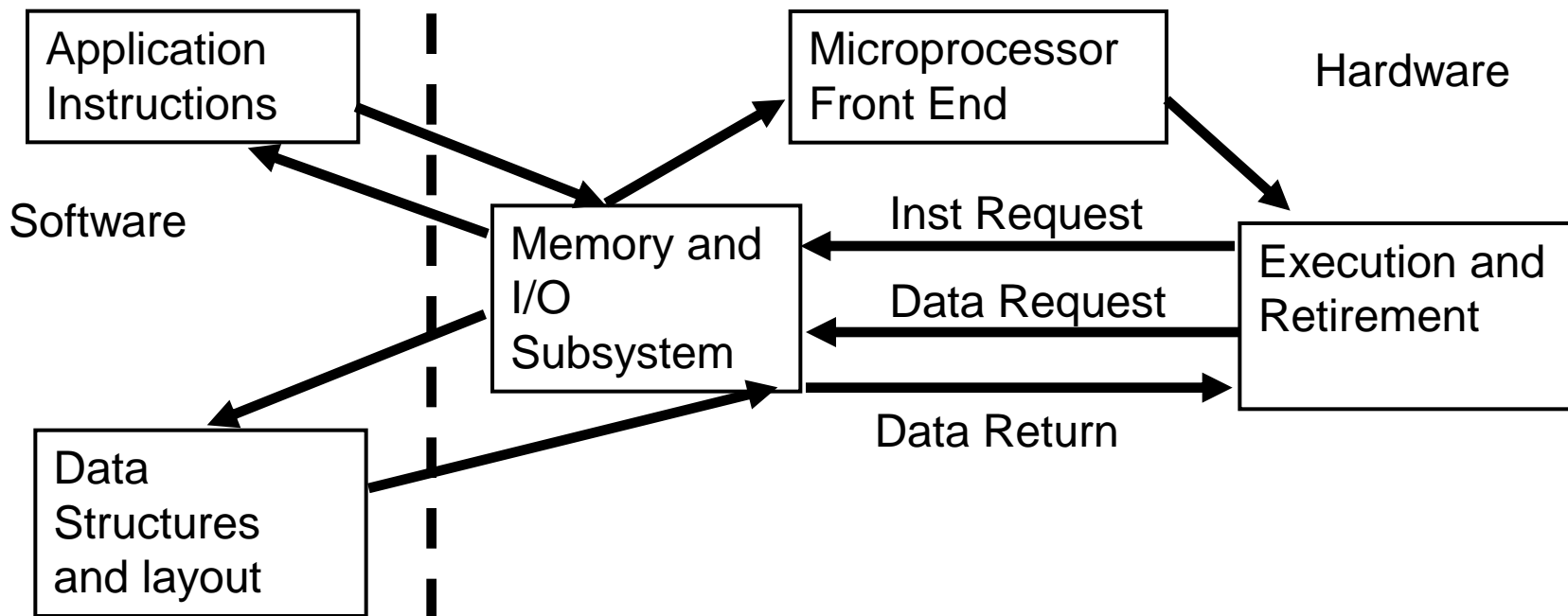
Performance Analysis and Software Enabling

- **In most cases application performance improves with new processors**
 - **If this is the case for your apps**
 - **You can sleep through this talk**
 - **Or, you can see if there is even more performance on the table**
- **Less than inspiring out of the box performance does not mean disaster**
 - **More likely: intrinsic SW design issue became manifest on new architecture**



What is Performance Analysis?

- **Performance Analysis = measurement of the interaction of the software and data structures with the platform and microarchitecture**



Performance Analysis Methodology

- **Measure application performance**
 - Time or rate of work
 - Compare to other platforms
- **Analyze the contributions to performance bottlenecks methodically**
 - Top Down



Performance Analysis Methodology

- **The steps**
 - **1. make sure the platform is correct**
 - It should be -- it was ordered for the customer
 - But don't take this for granted
 - **2. Use the correct compiler (Intel® Compiler)**
 - And invoke it correctly
 - This should also have already been done...but..
 - **3. Analyze interaction of SW and micro architecture and tune code/compiler usage**
 - Intel® VTune™ Analyzer* or better, Intel® Performance Tuning Utility (PTU)
 - Iterative process
 - **4. Parallelize the execution as appropriate**
 - Batch queue / Intel® MPI Library
 - OpenMP** product, Intel® Threading Building Blocks (Intel® TBB), explicit threading
- **Iterate on 3 and 4**

*Vtune is a trademark of Intel Corporation in the U.S. and other countries.

**Other names and brands may be claimed as the property of others.



Platform Optimization: Step 1

- **1. Make sure the platform is correct**
 - **Enough memory**
 - **Page faults (Perfmon*, vmstat*)**
 - rates of >100 sec is cause for investigation
 - **Get rid of old disk drives**
 - **This Had better not be a problem for Intel® Core™ i7 processor based systems!!!**
 - **Make sure disks are in AHCI, not IDE setting**
 - **Prefetcher BIOS Settings correct for the app**
 - **on**
 - **Intel® 11.0 compiler can generate SW prefetch**
 - **NUMA BIOS setting correct (on)**
 - **Intel® HT BIOS setting correct for the application**
 - **HPC off, Enterprise Server on, Desktop on**
 - **Probably makes little difference**

* Other names and brands may be claimed as the property of others.



Compiler Usage Optimization: Step 2

- **2. Optimize the time consuming functions**
 - **Profile functions, and check compiler options**
 - **Intel® VTune™ Analyzer and Intel® PTU have source file granularities**
 - **Data grouped per source file to identify hot files**
 - **Do not assume this has been done**
 - **Build environments are complex**



Micro architectural Optimization: Step 3

- **3. Identify & Optimize the time-consuming functions**
- **Use performance events methodically to identify performance limitations**
 - Intel® PTU, Intel® VTune™ Analyzer, etc.
- **Confirm that compiler really did produce good code (visual inspection of ASM)**
 - For the components of the code using the cycles
- **Go after largest, easy things first**
- **Documentation for Intel® Core™ i7 processor Performance Monitoring Unit (PMU) is available**



Parallelization : Step 4

- **4. Use as many cores and machines as possible**
 - **Parallel processing by batch queue is OK**
 - **Trivial parallelism**
 - **Hard to beat the throughput**



Parallelization : Step 4

- **4. Use as many cores and machines as possible**
 - **Figure out clean data decomposition**
 - **Intel® MPI Library for process parallel execution**
 - **Minimal shared elements**
 - **Maximal address separation**
 - **OpenMP* product, Intel® TBB, explicit threading for shared memory**
 - **Intel® Thread Checker/ Intel® Thread Profiler**

* Other names and brands may be claimed as the property of others.



Intel® Core™ i7 Processor Specific Steps

- **Micro architectural Analysis and Optimization**
- **Interaction of SW with HW**
 - Where the rubber hits the road
- **Microprocessor performance events are the observables of the SW/HW interaction**
 - Systematic usage identifies location and nature of performance bottlenecks
- **Intel® performance analysis tools greatly simplify the task**



Performance Monitoring Unit

- **The Performance Monitoring Unit (PMU) consists of a set of counters that can be programmed to count user-selected signals of microprocessor activity**
 - **Cpu_clk_unhalted, inst_retired, LLC_miss, etc..**
- **Counting the number of events that occur in a fixed time period allows workload characterization**
 - **Using a spectrum of events allows a decomposition of the applications activity with respect to the microarchitecture components**



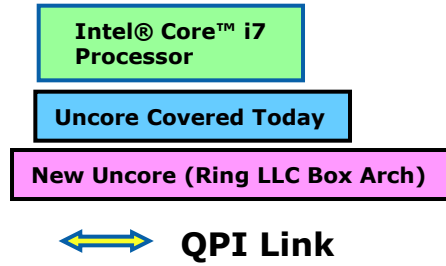
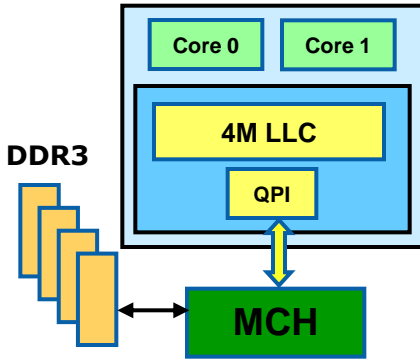
Performance Monitoring Unit

- **The PMU can be programmed to generate interrupts on counter overflow**
 - **Allows periodic sampling of program counter for any user-chosen event**
 - **Initialize count to (overflow – periodic rate)**
 - **Interrupt Vector Table is programmed with the address of the interrupt handler**
 - **Intel® VTune™ Analyzer driver is invoked by HW on counter overflows and given a program counter where the interrupt (i.e. counter overflow) happened**
 - **Identify statistically where in program events occur**
 - **Application profiling by event**

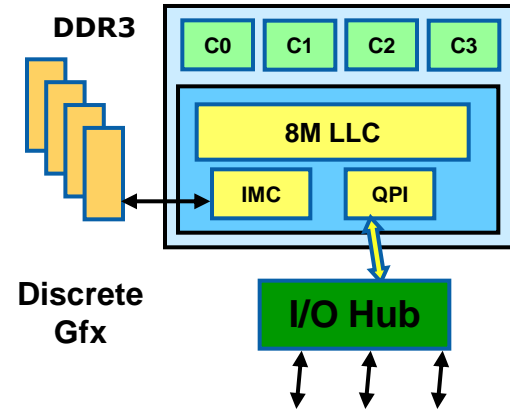


Platforms

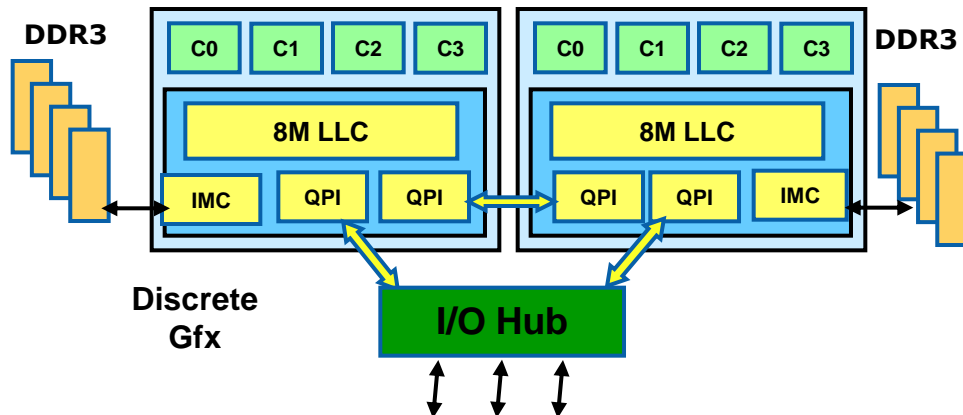
Mobile/Desktop



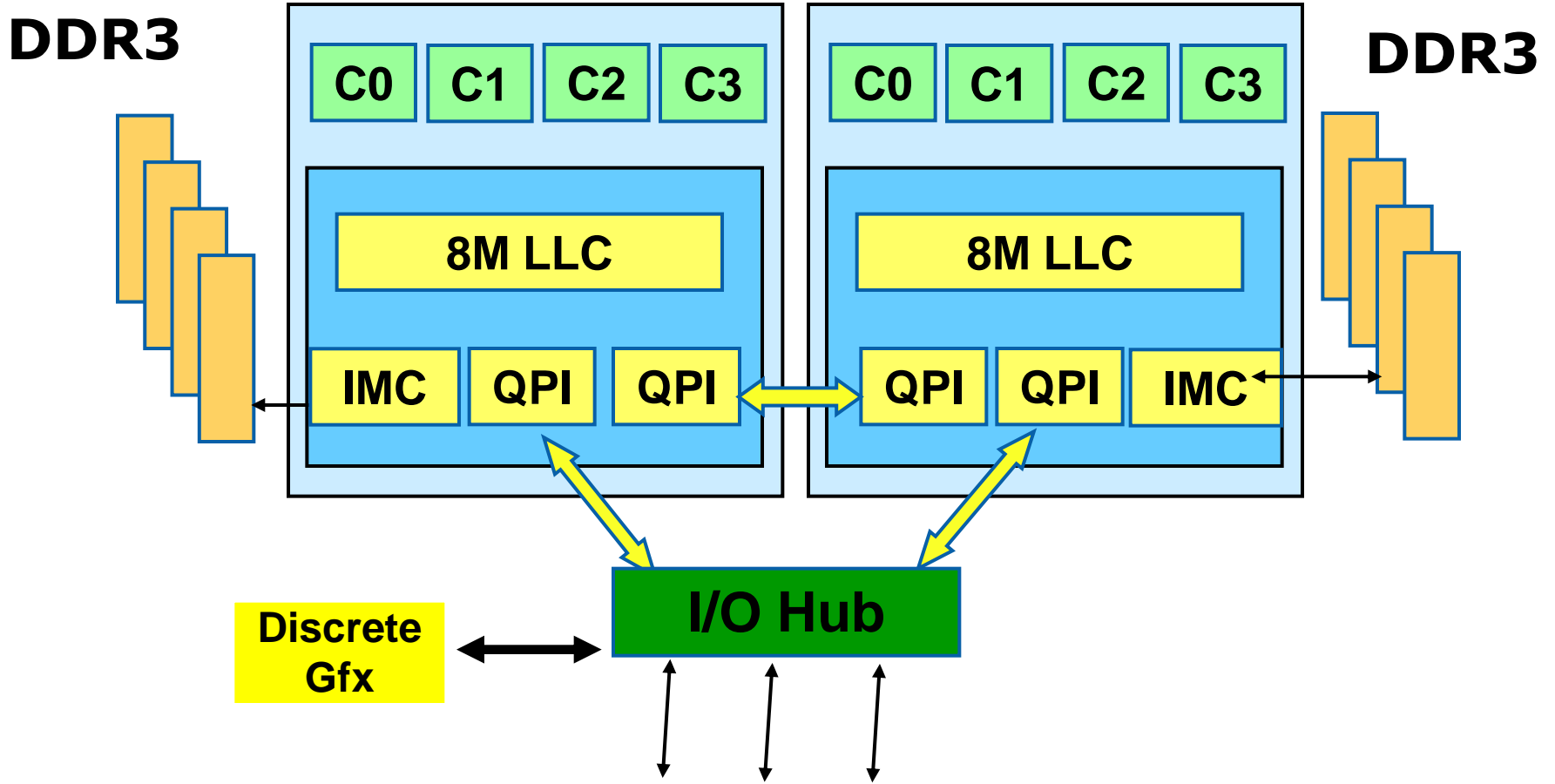
HE Desktop /mobile UP server & WS



DP Servers & WS

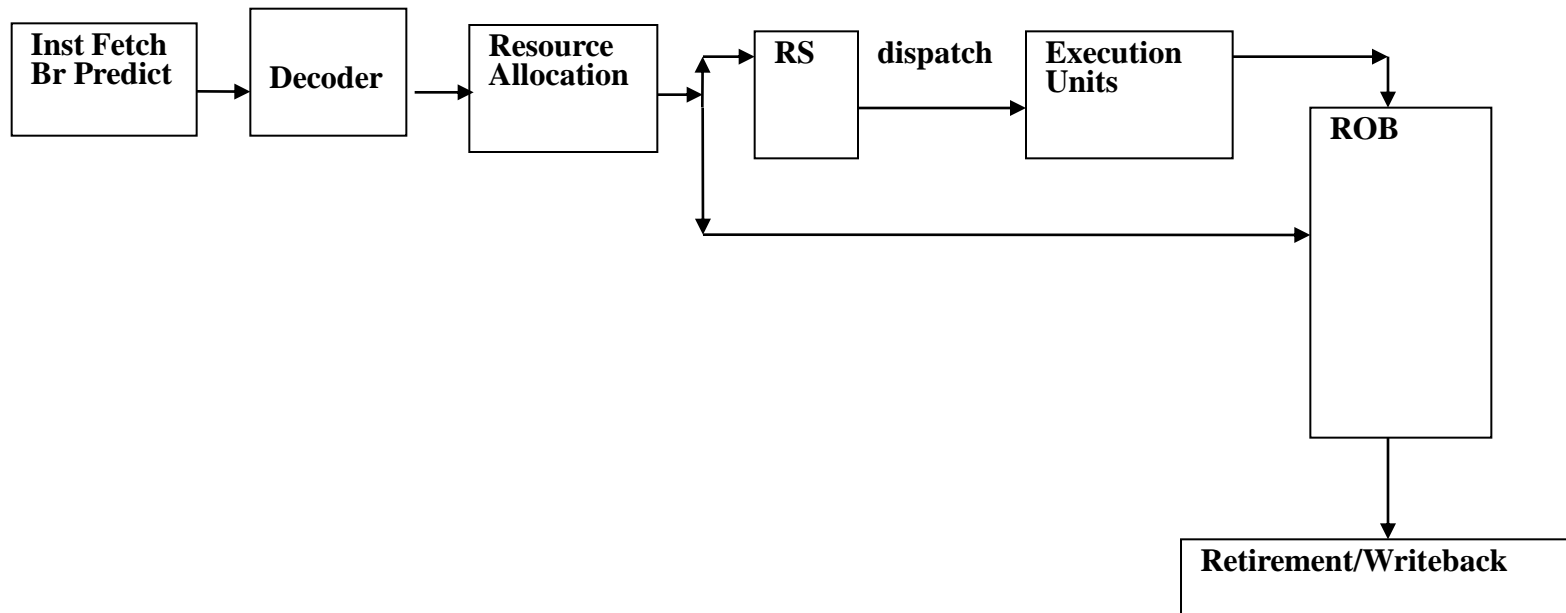


DP Platform



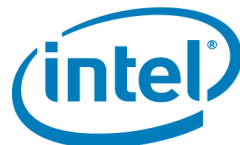
(Simplified) Execution in an OOO Engine

- Design optimizes Dispatch to Execution
 - Uops wait in RS until inputs are available
 - Keeping the Execution Units occupied matters



Cycle Accounting and Uop Flow

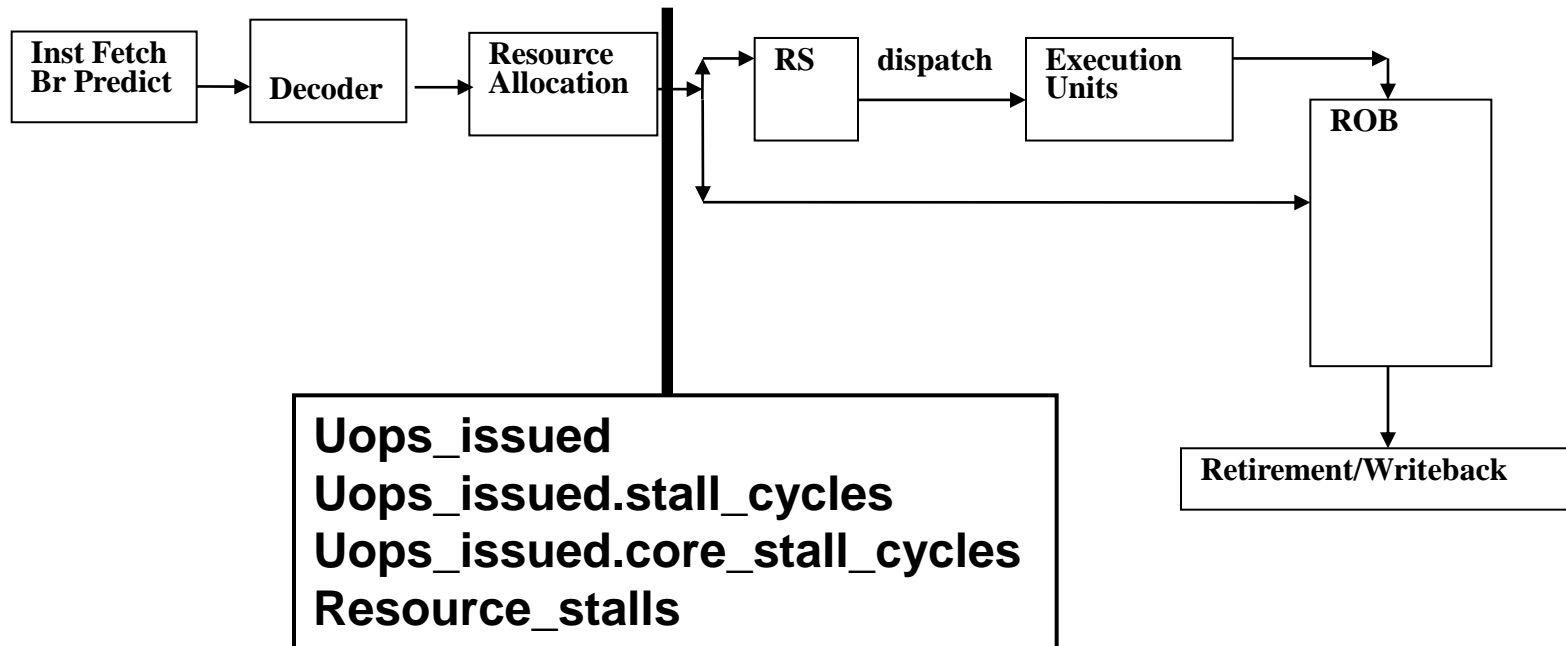
- **Cycles =**
 - Cycles dispatching to execution units +**
 - Cycles not dispatching (stalls)**
 - A trivial truism
- **Uops dispatched = uops retired +**
 - speculative uops that are not retired**
 - Non-retired uops due to mispredicted branches
- **Optimization Reduces Total Cycles by**
 - Reducing stalls
 - Reducing retired uops (better code generation)
 - Reducing non retired uops (reducing mispredictions)



Uop Flow Monitors Execution

- **Uop issue**

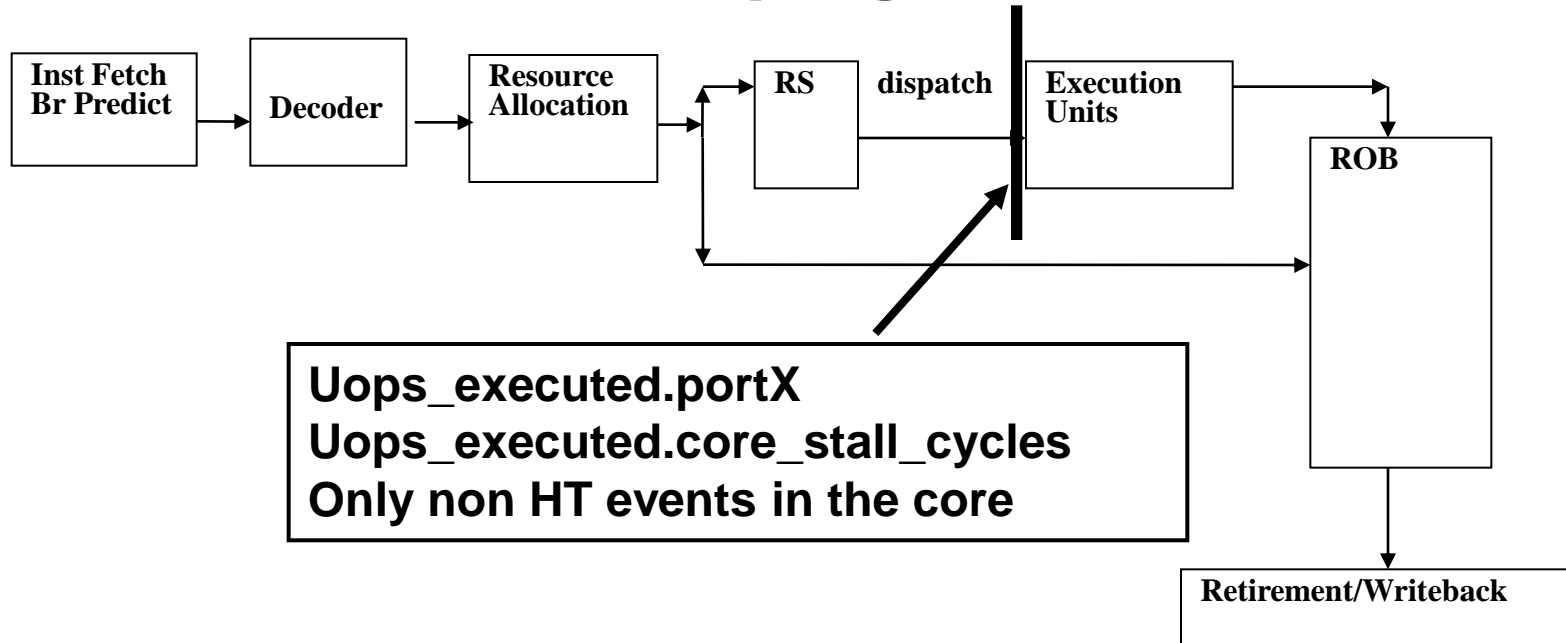
- **Uops have been allocated resources**
- **No downstream blockage (resource_stalls)**
- **FE Stalls = an instruction delivery problem**
= Uops_issued.stall_cycles - Resource_stalls



Uop Flow Monitors Execution

- **Uop Execute**

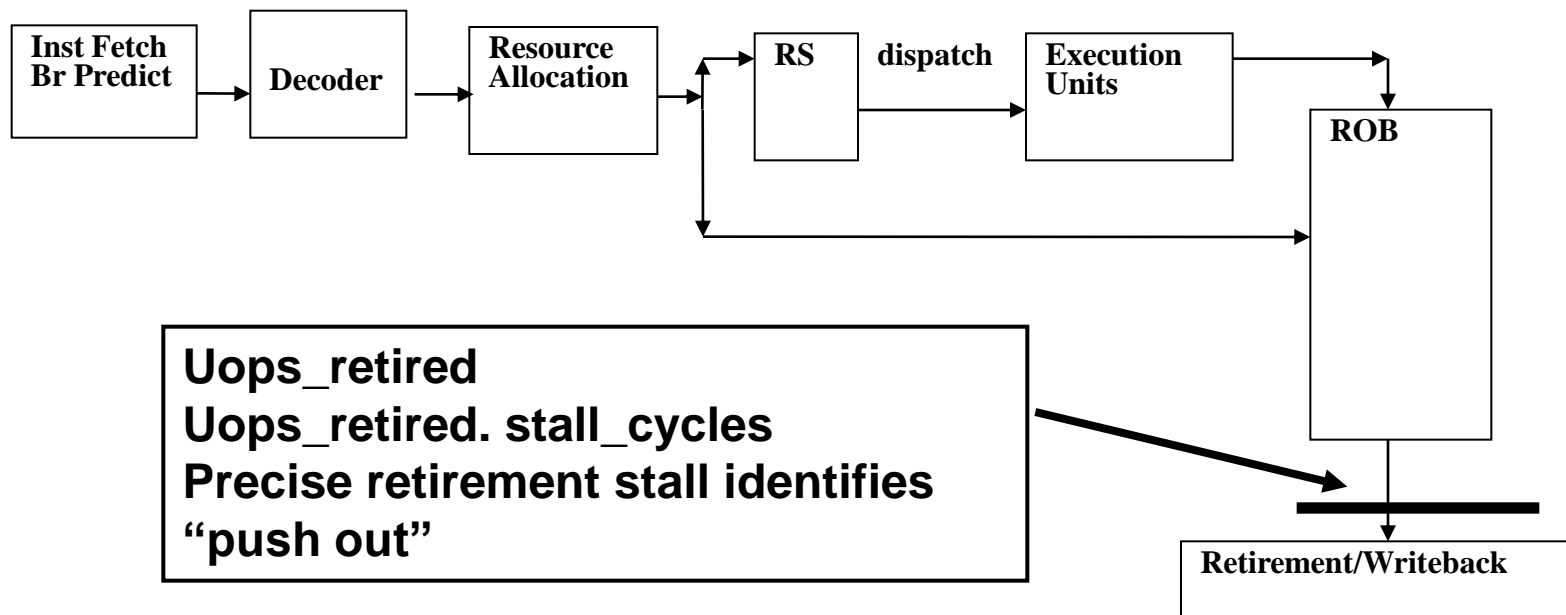
- Uops have inputs ?
- No downstream blockage (DIV/SQRT)
- No execution = no progress



Uop Flow Monitors Execution

- **Uop Retire**

- All older instructions retired ?
- No retirement = ? (out of order execution)

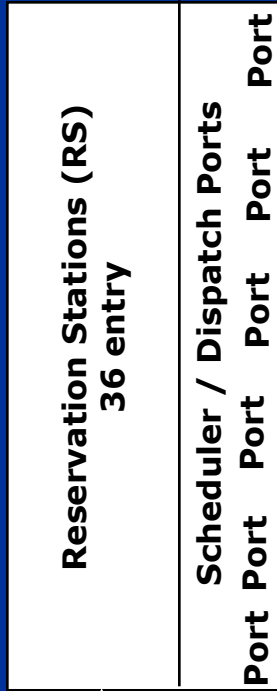
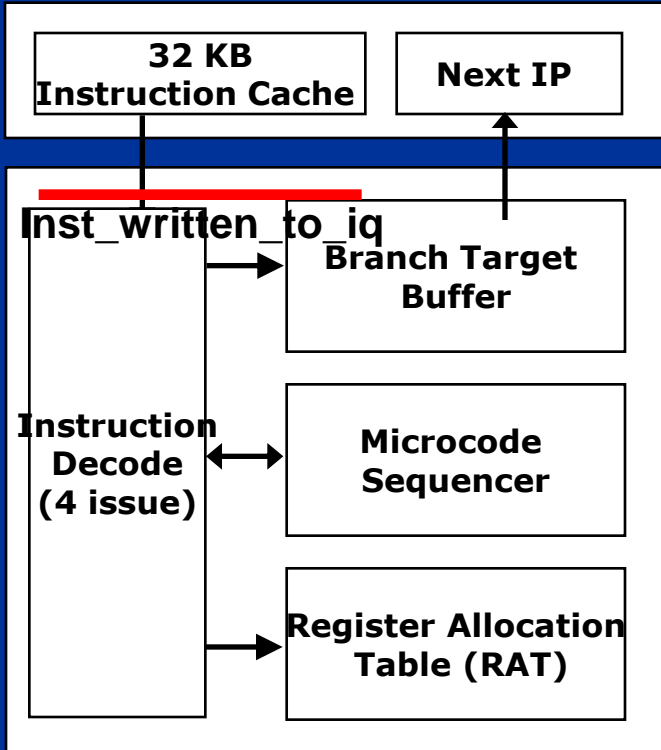


Uop Flow

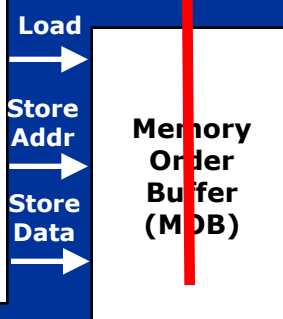
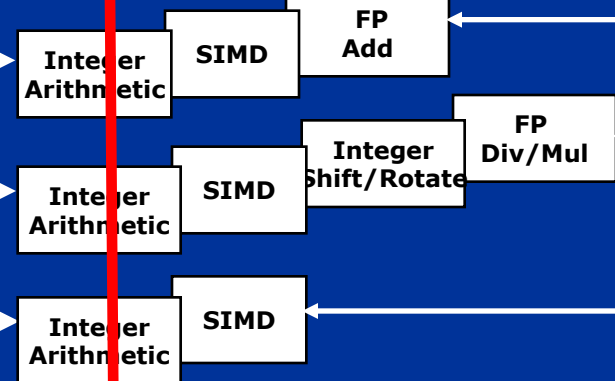
Fetch / Decode

To Uncore

MEU

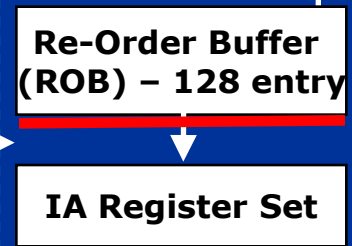


Execute



Uops_executed

Retire



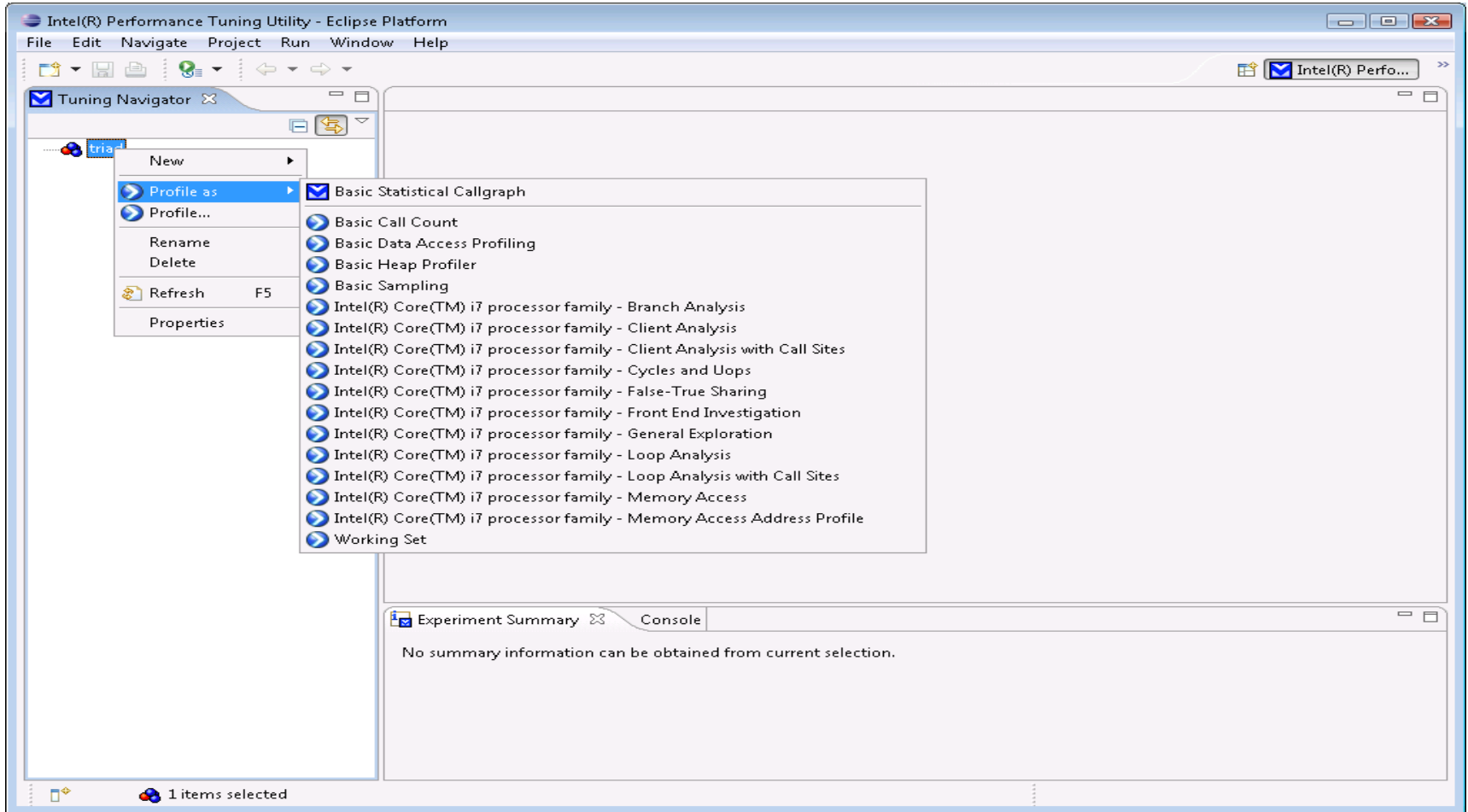
Uops_Issued

Uops_retired

Qualitative: Artistic License employed



Intel® PTU uses profiles to manage complexity



Intel® PTU predefined event lists

- **Cycles and Uops**
 - Cycle usage and uop flow through the pipeline
- **Branch Analysis**
 - Branch execution analysis for loop tripcounts and call counts
- **General Exploration**
 - Cycles, inst, stalls, branches, basic memory access
- **Memory Access**
 - Detailed breakdown of off-core memory access (w/wo address profiling)
- **Working Set**
 - Precise loads and stores enabling address space analysis
- **FrontEnd (FE) Investigation**
 - Detailed instruction starvation analysis
- **Contested lines**
 - Precise HITM and Store events
- **Loop Analysis**
 - 32 events for HPC type codes, w/wo call sites , i.e. including LBR capture
- **Client Analysis**
 - 54 events for client type codes, w/wo call sites , i.e. including LBR capture
- **And others...**



Intel® PTU uses predefined event lists to manage the complexity

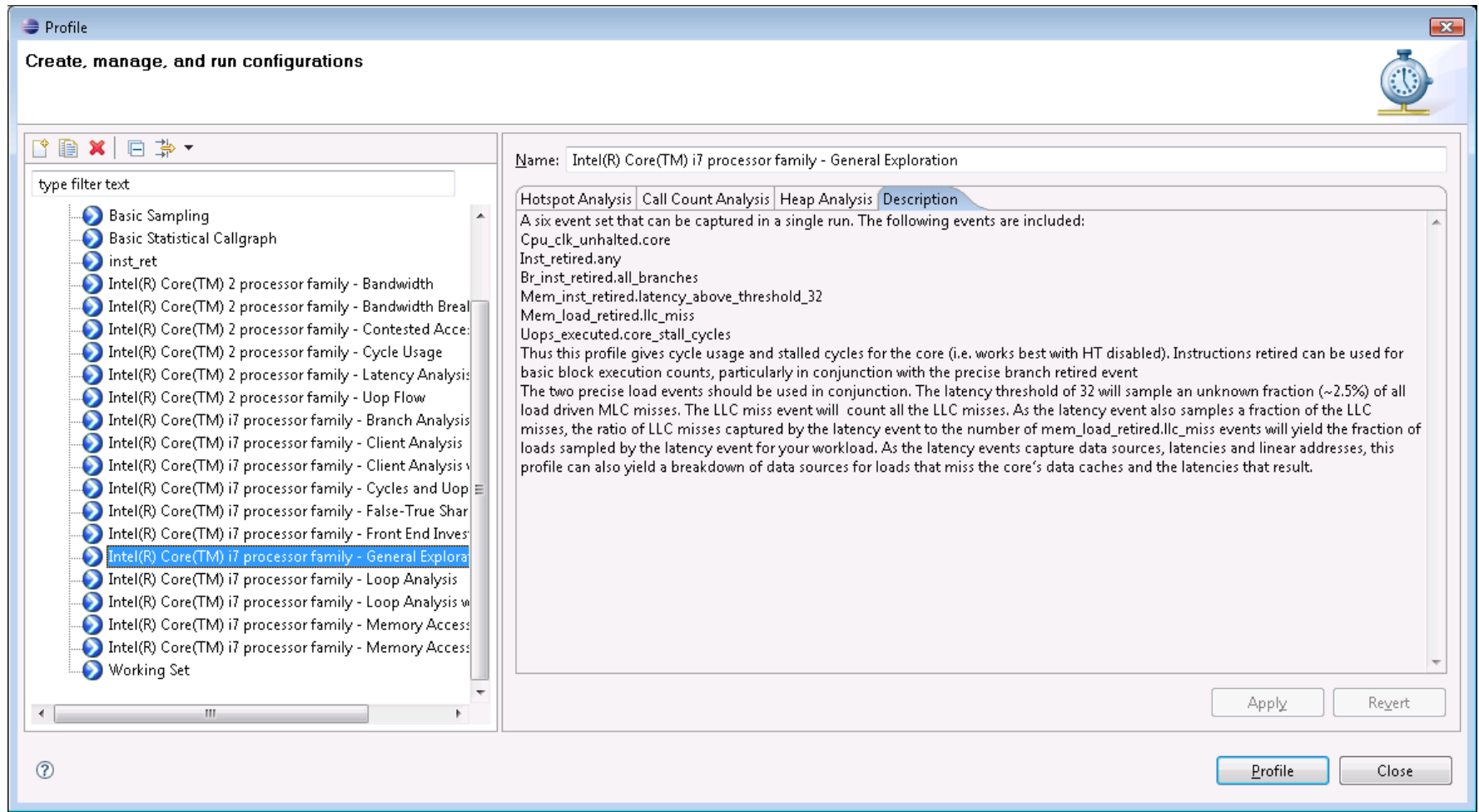
- **General Exploration**

- Cpu_clk_unhalted.thread
- Inst_retired.any
- Br_inst_retired.all_branches
- Mem_inst_retired.latency_above_threshold_32
- Mem_Load_retired.llc_miss
- Uops_executed.core_stall_cycles

Code profiles with respect to cycles, stalls, instructions and longer latency data sources



Intel® PTU uses predefined event lists to manage the complexity



The screenshot shows the Intel PTU Profile configuration window. The title bar reads "Profile" and the subtitle is "Create, manage, and run configurations". On the left, there is a tree view of predefined event sets, with "Intel(R) Core(TM) i7 processor family - General Exploration" selected. The right pane shows the configuration for this profile, including a list of events and a detailed description.

Name: Intel(R) Core(TM) i7 processor family - General Exploration

Hotspot Analysis	Call Count Analysis	Heap Analysis	Description
			A six event set that can be captured in a single run. The following events are included: Cpu_clk_unhalted.core Inst_retired.any Br_inst_retired.all_branches Mem_inst_retired.latency_above_threshold_32 Mem_load_retired.llc_miss Uops_executed.core_stall_cycles

Thus this profile gives cycle usage and stalled cycles for the core (i.e. works best with HT disabled). Instructions retired can be used for basic block execution counts, particularly in conjunction with the precise branch retired event

The two precise load events should be used in conjunction. The latency threshold of 32 will sample an unknown fraction (~2.5%) of all load driven LLC misses. The LLC miss event will count all the LLC misses. As the latency event also samples a fraction of the LLC misses, the ratio of LLC misses captured by the latency event to the number of mem_load_retired.llc_miss events will yield the fraction of loads sampled by the latency event for your workload. As the latency events capture data sources, latencies and linear addresses, this profile can also yield a breakdown of data sources for loads that miss the core's data caches and the latencies that result.

Buttons: Apply, Revert, Profile, Close



Memory Access

- **Intel® Core™ i7 processor memory access events are “per source”**
 - How many times cacheline came from “here”
 - DP system has ~10 sources outside a core
 - Large number of performance events
- **Memory access events are precise**
 - HW captures IP and register values
 - Sample + Disassembly => Reconstruct Address
- **Latency Event captures IP, load latency, data source and address**
 - Similar to Itanium® Processor Family* Data Ear

* Itanium is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.



Data source / home node identification

Unknown source >>	On Core >>	Local LLC >>	Remote LLC >>	Local DRAM >>	Remote DRAM >>	Unknown >>	Local Home >>	Remote Home >>
121,000 (78.6%)	82,206,000 (99.7%)	90,000 (90.5%)	0 (0.0%)	1,558,000 (97.4%)	2,943,000 (98.3%)	0 (N/A)	85,930,000 (99.7%)	1,288,000 (89.8%)
7,000 (4.5%)	187,000 (0.2%)	26,000 (6.0%)	5,000 (100.0%)	30,000 (1.9%)	32,000 (1.1%)	0 (N/A)	167,000 (0.2%)	120,000 (8.4%)
25,000 (16.2%)	1,000 (0.0%)	1,000 (0.2%)	0 (0.0%)	2,000 (0.1%)	2,000 (0.1%)	0 (N/A)	31,000 (0.0%)	0 (0.0%)
0 (0.0%)	15,000 (0.0%)	1,000 (0.2%)	0 (0.0%)	4,000 (0.3%)	8,000 (0.3%)	0 (N/A)	18,000 (0.0%)	10,000 (0.7%)
1,000 (0.6%)	9,000 (0.0%)	1,000 (0.2%)	0 (0.0%)	2,000 (0.1%)	5,000 (0.2%)	0 (N/A)	16,000 (0.0%)	2,000 (0.1%)
0 (0.0%)	1,000 (0.0%)	11,000 (2.6%)	0 (0.0%)	1,000 (0.1%)	0 (0.0%)	0 (N/A)	0 (0.0%)	13,000 (0.9%)
0 (0.0%)	1,000 (0.0%)	1,000 (0.2%)	0 (0.0%)	1,000 (0.1%)	4,000 (0.1%)	0 (N/A)	6,000 (0.0%)	1,000 (0.1%)
0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	1,000 (0.1%)	0 (0.0%)	0 (N/A)	1,000 (0.0%)	0 (0.0%)

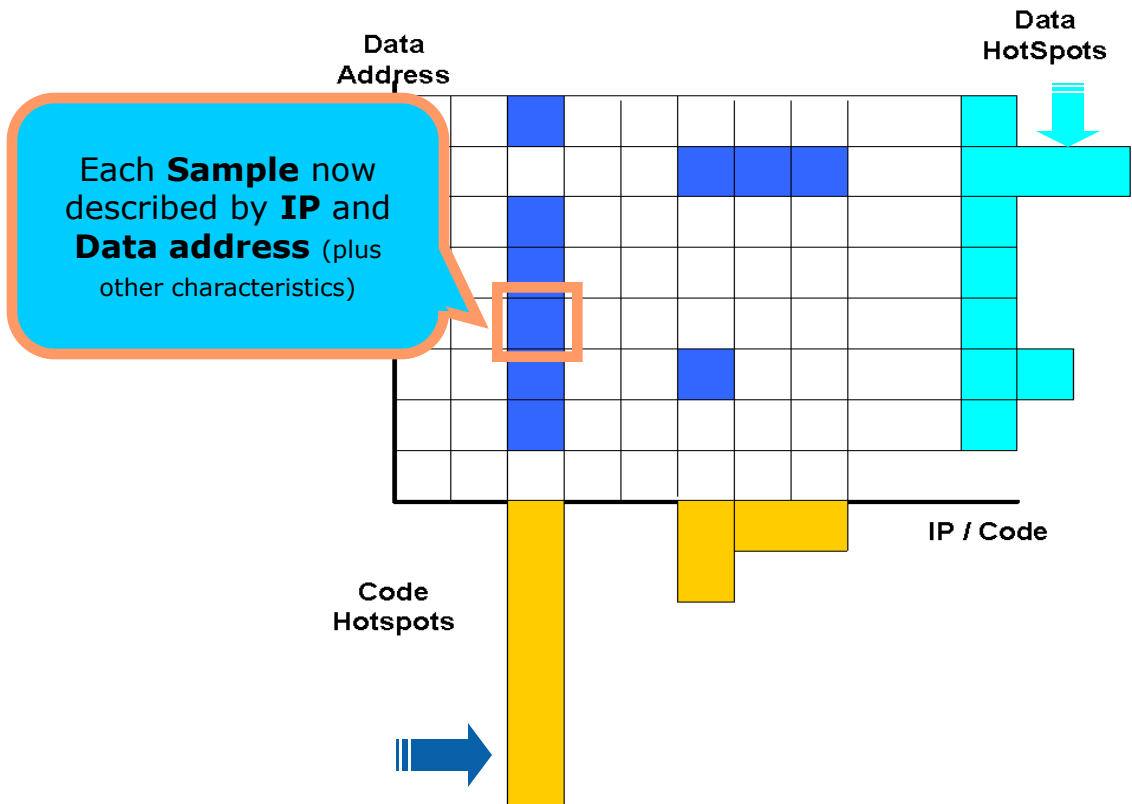
Unknown source >>	On Core <<			Local LLC >>	Remote LLC >>	Local DRAM >>	Remote DRAM >>
	MEM_INST_...ESHOLD_16	MEM_INST_...ESHOLD_64	MEM_LOAD_RETIRED.L2_HIT				
158,000 (96.3%)	437,450,000 (32.5%)	87,056,000 (27.1%)	1,616,400,000 (42.8%)	122,358,000 (67.6%)	52,000 (25.0%)	466,372,000 (65.3%)	19,908,000 (58.9%)
0 (0.0%)	160,680,000 (11.9%)	15,010,000 (4.5%)	1,255,400,000 (33.3%)	35,388,000 (19.5%)	40,000 (19.2%)	136,442,000 (19.1%)	7,728,000 (22.8%)
0 (0.0%)	458,860,000 (34.1%)	138,020,000 (42.9%)	63,800,000 (1.7%)	2,424,000 (1.3%)	10,000 (4.8%)	37,768,000 (5.3%)	30,000 (0.1%)
0 (0.0%)	128,250,000 (9.5%)	42,338,000 (13.2%)	25,200,000 (0.7%)	884,000 (0.5%)	0 (0.0%)	14,086,000 (2.0%)	0 (0.0%)
0 (0.0%)	7,600,000 (0.6%)	1,114,000 (0.3%)	198,000,000 (5.2%)	9,424,000 (5.2%)	0 (0.0%)	29,232,000 (4.1%)	3,120,000 (9.2%)
0 (0.0%)	7,660,000 (0.6%)	1,170,000 (0.4%)	171,200,000 (4.5%)	7,642,000 (4.2%)	0 (0.0%)	24,696,000 (3.5%)	2,280,000 (6.7%)
6,000 (3.7%)	1,480,000 (0.1%)	298,000 (0.1%)	2,800,000 (0.1%)	2,438,000 (1.3%)	86,000 (41.3%)	3,980,000 (0.6%)	726,000 (2.1%)
0 (0.0%)	19,360,000 (1.4%)	484,000 (0.2%)	0 (0.0%)	64,000 (0.0%)	0 (0.0%)	604,000 (0.1%)	22,000 (0.1%)
0 (0.0%)	800,000 (0.1%)	238,000 (0.1%)	0 (0.0%)	64,000 (0.0%)	0 (0.0%)	734,000 (0.1%)	0 (0.0%)

Simplifies Data Source Hierarchy



Data Address Profiling and False Sharing Detection

Data Mining in 2 Dimensional Model



- **Sorting** – repositioning segments of the axes
- **Applying granularity** – changing scale of the axis
- **Filtering** - projecting slices onto another dimension

Filtering by cachelines marked as “falsely-shared” isolate the causing instructions And the data objects

Data Address Profiling and False Sharing Detection

Sampling during app execution

Symbolization & Data Address reconstruction

Aggregation

Precise Event Sampling:
events associated with memory operations, e.g. MEM_INST_RETIRED.LOADS, MEM_INST_RETIRED.STORES...

Pin threads affinity

Iterate over Samples and PEBS records in ebs.tb5

Sample: IP, data address, threadID..
To aggregate addresses into cachelines:

Binary

...		
0x7F5D	mulpd	xmm1, xmm2
0x7F61	movaps	xmm2, XMMWORD PTR [rsp+0230h]
0x7F69	mulpd	xmm7, xmm2
0x7F6D	subpd	xmm3, xmm1
0x7F71	movsd	xmm1, MMWORD PTR [rcx+rbx+rho_i.0+018h]
0x7F7A	movhpd	xmm1, MMWORD PTR [rcx+rbx+rho_i.0+020h]
0x7F83	mulpd	xmm1, xmm8
...		

Using the binary, identify the instruction that overflowed event counter -> IP-1

IP-1

```
00000000055CC9900
00000000055CC9908
00000000055CC9910
00000000055CC9916
00000000055CC9928
00000000055CC9938
```

&&...FFFC0

Sample record:
IP, process, module, threadID..

PEBS record:
IP, rax, rbx, rcx



Same cacheline accessed by different threads at different offsets
True and False Sharing
Next foils Illustrate GUI Navigation

Cacheline Address / Offset / Thread ...	Contributors	MEM...L1D_MISS
▼ 0x00000000055cc900	Offsets: 5 Threads: 3	31 (0.0%)
▼ Offset:0x00(0)	Threads: 1	21 (0.0%)
▶ Thread:00003fbb(0014)	Functions: 3	21 (0.0%)
▼ Offset:0x38(56)	Threads: 1	5 (0.0%)
▶ Thread:00003fbd(0009)	Functions: 3	5 (0.0%)
▼ Offset:0x08(8)	Threads: 2	1 (0.0%)
▶ Thread:00003fbb(0014)	Functions: 3	0 (0.0%)
▶ Thread:00003fbc(0015)	Functions: 1	1 (0.0%)
▼ Offset:0x10(16)	Threads: 1	3 (0.0%)
▶ Thread:00003fbc(0015)	Functions: 2	3 (0.0%)
▼ Offset:0x28(40)	Threads: 1	1 (0.0%)
▶ Thread:00003fbc(0015)	Functions: 1	1 (0.0%)



Use Cacheline Access Count to Measure Working Set Size

The screenshot displays the Intel VTune Performance Analyzer interface. At the top, a table lists functions and their cache access statistics. Below this, the 'Cachelines View' is active, showing a list of cache lines with their addresses, access counts, and contributors. A red circle highlights a specific cache line address in the 'Cachelines View' table. To the right, the 'Working Set' utility is open, showing a graph of memory usage over time. A yellow tooltip indicates the current memory size and reference counts.

Function	Module	BR_INST...ANCHES	BR...ALL	MEM_INS...LOADS	MEM_INS...STORES
OpenMPUpdateStress	test_seismic_static_r100.exe	267,944 (49.1%)	35 (38.9%)	3,614,571 (49.9%)	1,036,316 (66.5%)
OpenMPUpdateVelocity	test_seismic_static_r100.exe	267,687 (49.0%)	51 (56.7%)	3,626,685 (50.0%)	518,283 (33.2%)
<unknown(s)>	vtune_drv	7,794 (1.4%)	0 (0.0%)	3,136 (0.0%)	2,105 (0.1%)
<unknown(s)>	vmlinux-2.6.18-53.el5	1,508 (0.3%)	3 (3.3%)	2,683 (0.0%)	1,664 (0.1%)
_kmp_fork_call	libguide.so	129 (0.0%)	0 (0.0%)	262 (0.0%)	206 (0.0%)

Cacheline Address / Offset...	B..	B..	MEM...ADS	MEM...RES	Contributors
0x000000000d7ffc0	0..	0..	233 (0.0%)	53 (0.0%)	Offsets: 9 Threads: 1
0x000000000d02fc0	0..	0..	239 (0.0%)	37 (0.0%)	Offsets: 9 Threads: 1
0x000000000d56000	0..	0..	239 (0.0%)	24 (0.0%)	Offsets: 8 Threads: 1
0x000000000d7e080	0..	0..	180 (0.0%)	72 (0.0%)	Offsets: 9 Threads: 1
0x000000000c08fc0	0..	0..	205 (0.0%)	38 (0.0%)	Offsets: 9 Threads: 1
0x000000000d7c140	0..	0..	188 (0.0%)	54 (0.0%)	Offsets: 10 Threads: 1
0x000000000d55040	0..	0..	181 (0.0%)	57 (0.0%)	Offsets: 9 Threads: 1
0x000000000c05140	0..	0..	176 (0.0%)	58 (0.0%)	Offsets: 10 Threads: 1
0x000000000da9fc0	0..	0..	192 (0.0%)	41 (0.0%)	Offsets: 9 Threads: 1
0x000000000c2e180	0..	0..	175 (0.0%)	58 (0.0%)	Offsets: 10 Threads: 1
0x000000000d2c000	0..	0..	187 (0.0%)	43 (0.0%)	Offsets: 9 Threads: 1
0x000000000cddb00	0..	0..	173 (0.0%)	57 (0.0%)	Offsets: 9 Threads: 1
0x000000000cf9380	0..	0..	176 (0.0%)	53 (0.0%)	Offsets: 10 Threads: 1
0x000000000ccc4c0	0..	0..	166 (0.0%)	63 (0.0%)	Offsets: 9 Threads: 1
0x000000000cca580	0..	0..	171 (0.0%)	57 (0.0%)	Offsets: 9 Threads: 1
0x000000000ca9240	0..	0..	170 (0.0%)	55 (0.0%)	Offsets: 10 Threads: 1

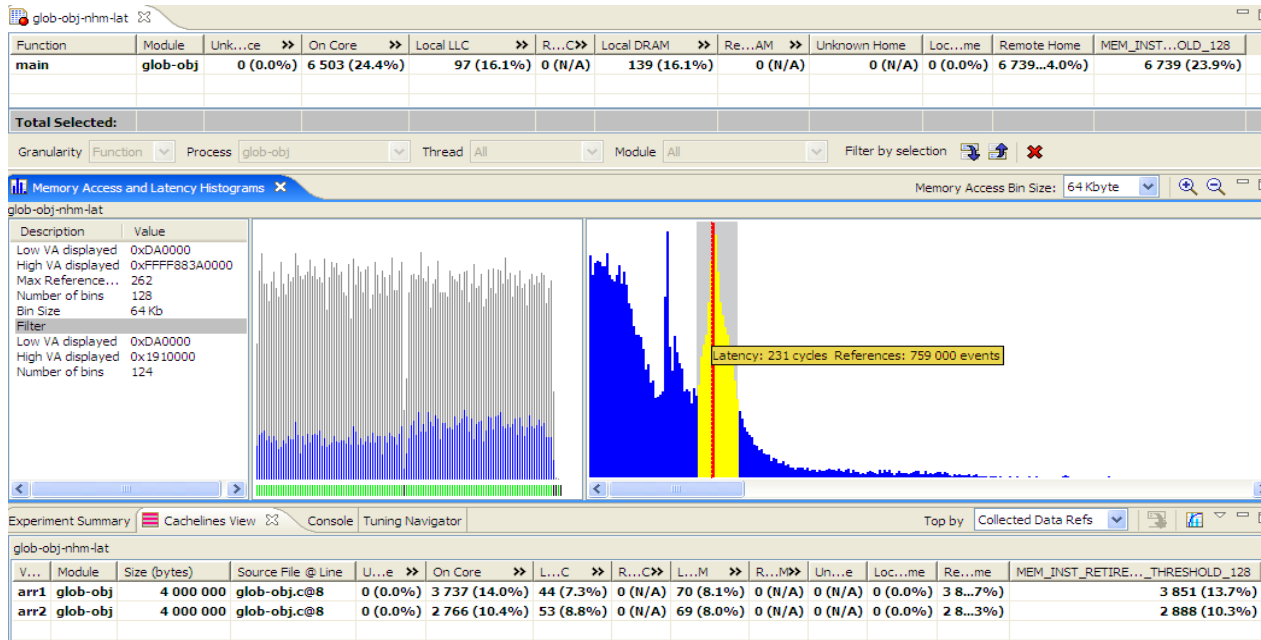
Utility: 2008-05-06-14-53-00 (2) : Working Set

Address Range: [] Stride: 100

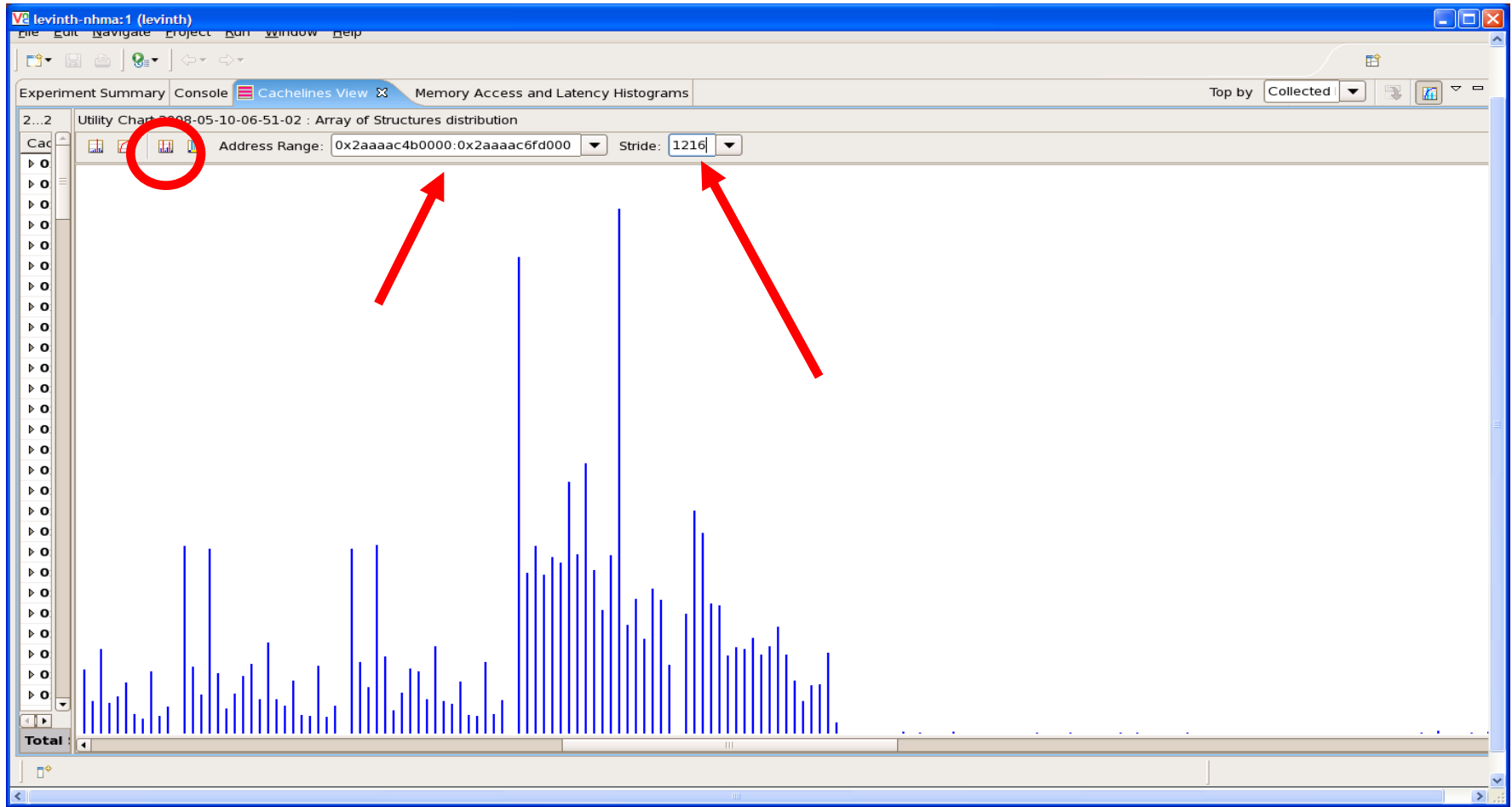
Memory Size: 5 Mb 688 Kb (Cumulative: 49.1%) Refs: 43710 (Cumulative: 80.8%)

NEW – Exact latency / Latency Histogram

- Exact latency in CPU cycles for loads collected with Latency events
- Intel® PTU offers a latency histogram
 - Can be filtered by selected hotspots
 - IP and address spreadsheets, and memory histogram can be filtered by latency region (shown below)



Array of Structures (address-base)% struct_size Most structure elements never accessed



Filtering to a Single Thread Displays the Data Decomposition

The screenshot shows the Intel Performance Tuning Advisor (PTA) interface. The top window displays a table of memory access statistics for the process `gather_fma16_omp` on thread `p00039bb(0003)`. The table includes columns for Function, Module, and various memory access metrics.

Function	Module	U...e	O...e	L...C	R...C	Local DRAM	Rem...RAM	MEM_U...HITM	MEM_U...HITM
TRIAD	gather_fma16_omp	0 (N/A)	0 (N/A)	0 (N/A)	0 (N/A)	2,359 (99.8%)	2,368 (98.2%)	2,359 (99.8%)	2,368 (98.2%)
<unknown(s)>	vmlinux-2.6.18-53.el5	0 (N/A)	0 (N/A)	0 (N/A)	0 (N/A)	5 (0.2%)	44 (1.8%)	5 (0.2%)	44 (1.8%)

The bottom window shows a memory access histogram for the same thread. The histogram displays a series of blue vertical bars representing memory access patterns. A red vertical line indicates a specific memory address, `0x2AAB18400000`, with a tooltip showing `0x2AAB18400000 Samples:53`. The histogram is filtered to show only the data for the selected thread.

A Different Thread

The screenshot displays the Intel Performance Tuning Center interface. The main window shows a table of memory access statistics for the process `gather_fma16_omp` on thread `p00039b9(0008)`. The table includes columns for Function, Module, and various memory access metrics. Below the table, a histogram titled "Memory Access and Latency Histograms" shows the distribution of memory accesses over time, with several prominent blue vertical bars indicating high-frequency access events.

Function	Module	U...e >>	O...e >>	L...C >>	R...C >>	Local DRAM >>	Rem...RAM >>	MEM_U...HITM	MEM_U...HITM
TRIAD	gather_fma16_omp	0 (N/A)	0 (N/A)	0 (N/A)	0 (N/A)	2,363 (99.9%)	2,369 (98.2%)	2,363 (99.9%)	2,369 (98.2%)
<unknown(s)>	vmlinux-2.6.18-53.el5	0 (N/A)	0 (N/A)	0 (N/A)	0 (N/A)	2 (0.1%)	43 (1.8%)	2 (0.1%)	43 (1.8%)

Granularity: Function | Process: gather_fma16_omp | Thread: p00039b9(0008) | Module: All | Filter by selection

Experiment Summary | Console | Cachelines View | **Memory Access and Latency Histograms** | Memory Access Bin Size: 4 MByte

2008-04-28-11-08-58 (2)

Description: Low VA displa, High VA displa, Max Referenc, Number of bir, Bin Size, Filter: Low VA displa, High VA displa, Number of bir

Example: False Sharing

What is it and why is it a Problem

- Cache coherency protocols require that all cores use the most current version of every cacheline**
- Shared lines can be modified by any thread**
 - Causing lines to be renewed regularly, if any thread writes to any byte in the line**
 - (replace an invalid state copy with new valid copy)**
 - Line renewal can cause a cache miss by other threads**
 - and a 40-300 cycle execution stall**
 - Depending on cacheline location**
- False sharing is when different threads access non-overlapping regions of a cacheline**

False Sharing Causes Avoidable 40-300 Cycle Stalls For Every Read Following a Write by Another Thread



Synthetic Example: Heavy Contention on this Line -- Multiple Threads Accessing Different Offsets Indicate False Sharing (Identified by Rose Highlighting)

Intel(R) Performance Tuning Utility - 2007-12-15-08-33-27 - Eclipse Platform

File Edit Navigate Project Run Window Help

2007-12-15-08-22-51 2007-12-15-08-33-27

Function	Module	Collected Data Refs (%Total)	LLC Misses (%Total)	Avg. Latency	Total Latency (%Total)	Cachelines #	Pages # (%Total)	MEM_LOAD_RETIRED.L2_MISS (%Total)
sort	main_share.exe	8,594,000,000 (100.0%)	400,000 (100.0%)	3	26,186,000,000 (100.0%)	1,029	24 (85.7%)	400,000 (100.0)

Total Selected:

Granularity: Function Process: main_share.exe Thread: All Module: All Filter by selection

Experiment Summary Console Cachelines View Top by: Collected Data Refs

Cacheline Address / Offset / Thread / Function	Collected Data ...	LLC Misses (%T...	Avg. Latency	Total Latency (...	Contention (%...	MEM_LOAD_RE...	MEM_LOAD_RE...	INST_RETIRED...	Contributors
0x0042a3c0	1,959,600,000 ...	400,000 (100...	3	6,252,000,000 ...	909,100,000 (...	400,000 (100...	39,200,000 (8...	1,920,000,000 ...	Offsets: 2 Threads: 2
0x0064ff40	836,000,000 (...	0 (0.0%)	3	2,508,000,000 ...	0 (N/A)	0 (0.0%)	0 (0.0%)	836,000,000 (...	Offsets: 1 Threads: 1
0x0054ff40	764,000,000 (...	0 (0.0%)	3	2,292,000,000 ...	0 (N/A)	0 (0.0%)	0 (0.0%)	764,000,000 (...	Offsets: 1 Threads: 1
0x0054ff80	366,000,000 (...	0 (0.0%)	3	1,098,000,000 ...	0 (N/A)	0 (0.0%)	0 (0.0%)	366,000,000 (...	Offsets: 2 Threads: 1
0x0064ff80	276,000,000 (...	0 (0.0%)	3	828,000,000 (...	0 (N/A)	0 (0.0%)	0 (0.0%)	276,000,000 (...	Offsets: 2 Threads: 1
0x004369c0	14,000,000 (0...	0 (0.0%)	3	42,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	14,000,000 (0...	Offsets: 7 Threads: 1
0x0042e580	14,000,000 (0...	0 (0.0%)	3	42,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	14,000,000 (0...	Offsets: 6 Threads: 1
0x0042f380	14,000,000 (0...	0 (0.0%)	3	42,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	14,000,000 (0...	Offsets: 6 Threads: 1
0x004327c0	12,000,000 (0...	0 (0.0%)	3	36,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	12,000,000 (0...	Offsets: 4 Threads: 1
0x00440900	12,000,000 (0...	0 (0.0%)	3	36,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	12,000,000 (0...	Offsets: 5 Threads: 1
0x0042e9c0	12,000,000 (0...	0 (0.0%)	3	36,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	12,000,000 (0...	Offsets: 5 Threads: 1
0x004396c0	12,000,000 (0...	0 (0.0%)	3	36,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	12,000,000 (0...	Offsets: 5 Threads: 1
0x004399c0	12,000,000 (0...	0 (0.0%)	3	36,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	12,000,000 (0...	Offsets: 5 Threads: 1
0x00440dc0	12,000,000 (0...	0 (0.0%)	3	36,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	12,000,000 (0...	Offsets: 5 Threads: 1
0x00430280	10,000,000 (0...	0 (0.0%)	3	30,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	10,000,000 (0...	Offsets: 5 Threads: 1
0x0042f9c0	10,000,000 (0...	0 (0.0%)	3	30,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	10,000,000 (0...	Offsets: 5 Threads: 1

Total Selected:

2007-12-15-08-33-27 (Basic Data Access Profiling)



Expanding the "arrow" we see the 2 threads access the line at Different Offsets...This is False Sharing

The screenshot displays the Intel(R) Performance Tuning Utility interface. The top table shows the 'sort' function in 'main_share.exe' with 8,594,000,000 collected data references, 400,000 LLC misses, and a total latency of 26,186,000,000. Below this, the 'Cachelines View' is expanded to show details for cache line 0x0042a3c0. This line is accessed by 3 threads, with 6,252,000,000 total latency and 909,100,000 contention. The 'Contributors' column for this line is circled in red and shows 'Offsets: 2 Threads: 2', indicating that two threads are accessing the cache line at different offsets, which is a sign of false sharing.

Function	Module	Collected Data Refs (%Total)	LLC Misses (%Total)	Avg. Latency	Total Latency (%Total)	Cachelines #	Pages # (%Total)	MEM_LOAD_RETIRED.L2_MISS (%Total)
sort	main_share.exe	8,594,000,000 (100.0%)	400,000 (100.0%)	3	26,186,000,000 (100.0%)	1,029	24 (85.7%)	400,000 (100.0)

Cacheline Address / Offset / Thread / Function	Collected Data ...	LLC Misses (%T...	Avg. Latency	Total Latency (...	Contention (%...	MEM_LOAD_RE...	MEM_LOAD_RE...	INST_RETIRED...	Contributors
0x0042a3c0	1,959,600,000 ...	400,000 (100...		3 6,252,000,000 ...	909,100,000 (...	400,000 (100...	39,200,000 (8...	1,920,000,000 ...	Offsets: 2 Threads: 2
▶ Offset:0x04(4)	1,050,500,000 (...	100,000 (25.0%)		3 3,319,000,000 (...	0 (N/A)	100,000 (25.0%)	20,400,000 (46...	1,030,000,000 (...	Threads: 1
▶ Offset:0x00(0)	909,100,000 (1...	300,000 (75.0%)		3 2,933,000,000 (...	0 (N/A)	300,000 (75.0%)	18,800,000 (42...	890,000,000 (...	Threads: 1
▶ 0x0064ff40	836,000,000 (...	0 (0.0%)		3 2,508,000,000 ...	0 (N/A)	0 (0.0%)	0 (0.0%)	836,000,000 (...	Offsets: 1 Threads: 1
▶ 0x0054ff40	764,000,000 (...	0 (0.0%)		3 2,292,000,000 ...	0 (N/A)	0 (0.0%)	0 (0.0%)	764,000,000 (...	Offsets: 1 Threads: 1
▶ 0x0054ff80	366,000,000 (...	0 (0.0%)		3 1,098,000,000 ...	0 (N/A)	0 (0.0%)	0 (0.0%)	366,000,000 (...	Offsets: 2 Threads: 1
▶ 0x0064ff80	276,000,000 (...	0 (0.0%)		3 828,000,000 (...	0 (N/A)	0 (0.0%)	0 (0.0%)	276,000,000 (...	Offsets: 2 Threads: 1
▶ 0x004369c0	14,000,000 (0...	0 (0.0%)		3 42,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	14,000,000 (0...	Offsets: 7 Threads: 1
▶ 0x0042e580	14,000,000 (0...	0 (0.0%)		3 42,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	14,000,000 (0...	Offsets: 6 Threads: 1
▶ 0x0042f380	14,000,000 (0...	0 (0.0%)		3 42,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	14,000,000 (0...	Offsets: 6 Threads: 1
▶ 0x004327c0	12,000,000 (0...	0 (0.0%)		3 36,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	12,000,000 (0...	Offsets: 4 Threads: 1
▶ 0x00440900	12,000,000 (0...	0 (0.0%)		3 36,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	12,000,000 (0...	Offsets: 5 Threads: 1
▶ 0x0042e9c0	12,000,000 (0...	0 (0.0%)		3 36,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	12,000,000 (0...	Offsets: 5 Threads: 1
▶ 0x004396c0	12,000,000 (0...	0 (0.0%)		3 36,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	12,000,000 (0...	Offsets: 5 Threads: 1
▶ 0x004399c0	12,000,000 (0...	0 (0.0%)		3 36,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	12,000,000 (0...	Offsets: 5 Threads: 1
▶ 0x00440dc0	12,000,000 (0...	0 (0.0%)		3 36,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	12,000,000 (0...	Offsets: 5 Threads: 1



Select the falsely shared cacheline (now blue) and Filter the Hotspot view to only Display Accesses to that Line (multiple lines also work)

The screenshot displays the Intel(R) Performance Tuning Utility interface. The top table shows a summary for the 'sort' function in 'main_share.exe', with 8,594,000,000 collected data references, 400,000 LLC misses, and 26,186,000,000 total latency. Below this, the 'Cachelines View' shows a detailed list of cachelines. The first cacheline, '0x0042a3c0', is highlighted in blue and has a red arrow pointing to it from a red circle around a filter button in the interface. The filter button is labeled 'Filter by selection' and has a red 'X' icon next to it. The table below shows various cachelines with their respective metrics, including collected data, LLC misses, latency, and contention.

Function	Module	Collected Data Refs (%Total)	LLC Misses (%Total)	Avg. Latency	Total Latency (%Total)	Cachelines #	Pages # (%Total)	MEM_LOAD_RETIRED.L2_MISS (%Total)
sort	main_share.exe	8,594,000,000 (100.0%)	400,000 (100.0%)	3	26,186,000,000 (100.0%)	1,029	24 (85.7%)	400,000 (100.0)

Cacheline Address / Offset / Thread / Function	Collected Data ...	LLC Misses (%T...	Avg. Latency	Total Latency (...	Contention (%...	MEM_LOAD_RE...	MEM_LOAD_RE...	INST_RETIRED...	Contributors
0x0042a3c0	1,959,600,000 ...	400,000 (100...	3	6,252,000,000 ...	909,100,000 (...	400,000 (100...	39,200,000 (8...	1,920,000,000 ...	Offsets: 2 Threads: 2
▶ Offset:0x04(4)	1,050,500,000 (...	100,000 (25.0%)	3	3,319,000,000 (...	0 (N/A)	100,000 (25.0%)	20,400,000 (46...	1,030,000,000 (...	Threads: 1
▶ Offset:0x00(0)	909,100,000 (1...	300,000 (75.0%)	3	2,933,000,000 (...	0 (N/A)	300,000 (75.0%)	18,800,000 (42...	890,000,000 (1...	Threads: 1
▶ 0x0064ff40	836,000,000 (...	0 (0.0%)	3	2,508,000,000 ...	0 (N/A)	0 (0.0%)	0 (0.0%)	836,000,000 (...	Offsets: 1 Threads: 1
▶ 0x0054ff40	764,000,000 (...	0 (0.0%)	3	2,292,000,000 ...	0 (N/A)	0 (0.0%)	0 (0.0%)	764,000,000 (...	Offsets: 1 Threads: 1
▶ 0x0054ff80	366,000,000 (...	0 (0.0%)	3	1,098,000,000 ...	0 (N/A)	0 (0.0%)	0 (0.0%)	366,000,000 (...	Offsets: 2 Threads: 1
▶ 0x0064ff80	276,000,000 (...	0 (0.0%)	3	828,000,000 (...	0 (N/A)	0 (0.0%)	0 (0.0%)	276,000,000 (...	Offsets: 2 Threads: 1
▶ 0x004369c0	14,000,000 (0...	0 (0.0%)	3	42,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	14,000,000 (0...	Offsets: 7 Threads: 1
▶ 0x0042e580	14,000,000 (0...	0 (0.0%)	3	42,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	14,000,000 (0...	Offsets: 6 Threads: 1
▶ 0x0042f380	14,000,000 (0...	0 (0.0%)	3	42,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	14,000,000 (0...	Offsets: 6 Threads: 1
▶ 0x004327c0	12,000,000 (0...	0 (0.0%)	3	36,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	12,000,000 (0...	Offsets: 4 Threads: 1
▶ 0x00440900	12,000,000 (0...	0 (0.0%)	3	36,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	12,000,000 (0...	Offsets: 5 Threads: 1
▶ 0x0042e9c0	12,000,000 (0...	0 (0.0%)	3	36,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	12,000,000 (0...	Offsets: 5 Threads: 1
▶ 0x004396c0	12,000,000 (0...	0 (0.0%)	3	36,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	12,000,000 (0...	Offsets: 5 Threads: 1
▶ 0x004399c0	12,000,000 (0...	0 (0.0%)	3	36,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	12,000,000 (0...	Offsets: 5 Threads: 1
▶ 0x00440dc0	12,000,000 (0...	0 (0.0%)	3	36,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	12,000,000 (0...	Offsets: 5 Threads: 1

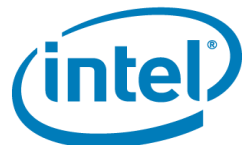


Only Events Referencing the Selected Line(s) are now in the Hotspot View Double Click to reach source/ASM view

The screenshot shows the Intel(R) Performance Tuning Utility interface. The main table displays performance metrics for the 'sort' function in the 'main_share.exe' module. A row is circled, indicating the selected function. Below the main table, the 'Cachelines View' window is open, showing a detailed view of the selected function's performance metrics, including cache line addresses, offsets, and thread counts.

Function	Module	Collected Data Refs (%Total)	LLC Misses (%Total)	Avg. Latency	Total Latency (%Total)	Cachelines #	Pages # (%Total)	MEM_LOAD_RETIRED.L2_MISS (%Total)
sort	main_share.exe	1,959,600,000 (22.8%)	400,000 (100.0%)	3	6,252,000,000 (23.9%)	1	1 (3.6%)	400,000 (100.0)

Cacheline Address / Offset / Thread / Function	Collected Data ...	LLC Misses (%T...	Avg. Latency	Total Latency (...	Contention (%...	MEM_LOAD_RE...	MEM_LOAD_RE...	INST_RETIRED...	Contributors
0x0042a3c0	1,959,600,000 ...	400,000 (100...	3	6,252,000,000 ...	909,100,000 (...	400,000 (100...	39,200,000 (8...	1,920,000,000 ...	Offsets: 2 Threads: 2
▶ Offset:0x04(4)	1,050,500,000 (...	100,000 (25.0%)	3	3,319,000,000 (...	0 (N/A)	100,000 (25.0%)	20,400,000 (46...	1,030,000,000 (...	Threads: 1
▶ Offset:0x00(0)	909,100,000 (1...	300,000 (75.0%)	3	2,933,000,000 (...	0 (N/A)	300,000 (75.0%)	18,800,000 (42...	890,000,000 (1...	Threads: 1
▶ 0x0064ff40	836,000,000 (...	0 (0.0%)	3	2,508,000,000 ...	0 (N/A)	0 (0.0%)	0 (0.0%)	836,000,000 (...	Offsets: 1 Threads: 1
▶ 0x0054ff40	764,000,000 (...	0 (0.0%)	3	2,292,000,000 ...	0 (N/A)	0 (0.0%)	0 (0.0%)	764,000,000 (...	Offsets: 1 Threads: 1
▶ 0x0054ff80	366,000,000 (...	0 (0.0%)	3	1,098,000,000 ...	0 (N/A)	0 (0.0%)	0 (0.0%)	366,000,000 (...	Offsets: 2 Threads: 1
▶ 0x0064ff80	276,000,000 (...	0 (0.0%)	3	828,000,000 (...	0 (N/A)	0 (0.0%)	0 (0.0%)	276,000,000 (...	Offsets: 2 Threads: 1
▶ 0x004369c0	14,000,000 (0...	0 (0.0%)	3	42,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	14,000,000 (0...	Offsets: 7 Threads: 1
▶ 0x0042e580	14,000,000 (0...	0 (0.0%)	3	42,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	14,000,000 (0...	Offsets: 6 Threads: 1
▶ 0x0042f380	14,000,000 (0...	0 (0.0%)	3	42,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	14,000,000 (0...	Offsets: 6 Threads: 1
▶ 0x004327c0	12,000,000 (0...	0 (0.0%)	3	36,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	12,000,000 (0...	Offsets: 4 Threads: 1
▶ 0x00440900	12,000,000 (0...	0 (0.0%)	3	36,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	12,000,000 (0...	Offsets: 5 Threads: 1
▶ 0x0042e9c0	12,000,000 (0...	0 (0.0%)	3	36,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	12,000,000 (0...	Offsets: 5 Threads: 1
▶ 0x004396c0	12,000,000 (0...	0 (0.0%)	3	36,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	12,000,000 (0...	Offsets: 5 Threads: 1
▶ 0x004399c0	12,000,000 (0...	0 (0.0%)	3	36,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	12,000,000 (0...	Offsets: 5 Threads: 1
▶ 0x00440dc0	12,000,000 (0...	0 (0.0%)	3	36,000,000 (0...	0 (N/A)	0 (0.0%)	0 (0.0%)	12,000,000 (0...	Offsets: 5 Threads: 1



The Pointer "sum" is Causing the False Sharing

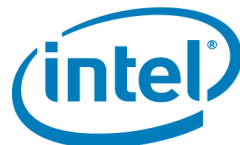
The screenshot displays the Intel(R) Performance Tuning Utility interface for a C program named 'sort.c'. The left pane shows the source code, and the right pane shows the corresponding assembly instructions. The assembly view is filtered to show instructions that update the 'sum' pointer, which is causing false sharing.

L.	Source	Collect...	LLC Mis...	Total ...	MEM_L.
1	int sort(int* data, volatile int* sum, int size...				
2	{				
3					
4	int i;				
5	for(i=0; i<size; i++)*sum += data[i]*data[i];	1,959,6...	400,000	6,252...	400,0
6	return *sum;				
7	}				

Address	L.	Assembly	Collected D...	LLC Mis...	Total La...	MEM
0x1550	2	push ebp				
0x1551	2	mov ebp, esp				
0x1553	2	push ecx				
0x1554	2	push esi				
0x1555	5	mov DWORD PTR [ebp-4], 0x0h				
0x155C	5	jmp sort+017h				
▼ Block 1 sort+00eh:						
0x155E	5	mov eax, DWORD PTR [ebp-4]				
0x1561	5	add eax, 0x1h				
0x1564	5	mov DWORD PTR [ebp-4], eax				
▼ Block 2 sort+017h:						
0x1567	5	mov ecx, DWORD PTR [ebp-4]				
0x156A	5	cmp ecx, DWORD PTR [ebp+010h]				
0x156D	5	jge sort+040h				
▼ Block 3 sort+01fh:						
0x156F	5	mov edx, DWORD PTR [ebp-4]				
0x1572	5	mov eax, DWORD PTR [ebp+08h]				
0x1575	5	mov ecx, DWORD PTR [ebp-4]				
0x1578	5	mov esi, DWORD PTR [ebp+08h]				
0x157B	5	mov edx, DWORD PTR [eax+edx*4]				
0x157E	5	imul edx, DWORD PTR [esi+ecx*4]				
0x1582	5	mov eax, DWORD PTR [ebp+0ch]				
0x1585	5	mov ecx, DWORD PTR [eax]	553,600,000	400,000	2,034,00...	400
0x1587	5	add ecx, edx				
0x1589	5	mov edx, DWORD PTR [ebp+0ch]				
0x158C	5	mov DWORD PTR [edx], ecx	1,406,000,000		4,218,00...	
0x158E	5	jmp sort+00eh				
▼ Block 4 sort+040h:						
0x1590	6	mov eax, DWORD PTR [ebp+0ch]				

Lots more where that came from:

- **HW call graph**
- **Automated disassembly analysis**
- **Predefined event lists**
- **NUMA event hierarchy built into display**
- **Differences of measurements**
 - **Compiler comparison**
- **Great Online Help**
- **Lots of methodology documentation**
- **Etc.**



Intel® Core™ i7 Processor Performance Events

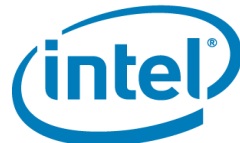
- **We have discussed uop flow and stalls at:**
 - **Issue from RAT**
 - **Execution**
 - **Retirement**
- **Next, decompose stalls**
 - **Check FE/Instruction Starvation issue**
 - **Desktop/server; rarely HPC**
 - **Memory Access**



Stall Decomposition on Intel® Core™ i7 Processors

- **Same basic methodology as on Intel® Core™ 2 processors***
- **Basic strategy is to identify the largest penalty event contributions first**
 - **Work your way down to smaller contributors**
- **FE starvation can now be measured**
 - **And no branch misprediction flush penalty**
- **Only both_threads_stalled can be measured**
 - **SMT will make $\sum \text{events}_i * \text{penalties}_i > \text{both_thread_stalled}$**
 - **ALU_only stalls can be measured per thread**
 - **Ports 0,1 and 5**

* Intel, the Intel logo, Intel Core and Core Inside are trademarks of Intel Corporation in the U.S. and other countries.



Stall Decomposition: Σ events, *penalties, The Elephants

- **LLC, MLC, and DTLB misses are the large penalty, common events**
- **LLC activity must be measured at the MLC for it to have core, PID, TID context**
 - **Uncore has no ability to track core, PID or ThreadID**
 - **Uncore event collection not yet supported**



Offcore Response Latencies

- **LLC Hit that does not need snooping**
 - LLC latency \sim 35-40 cycles
- **LLC Hit requiring snoop, clean response \sim 65**
- **LLC Hit requiring snoop, dirty response \sim 75**
- **LLC Miss from remote LLC \sim 200 cycles**
- **LLC Miss from local Dram \sim 60 ns**
- **LLC Miss from remote Dram \sim 100 ns**



Offcore_Response: Breaking Down Off-core Memory Access

- **Matrix type event**
 - **Request type X Response type**
 - 65025 possible real combinations (65535 – 2 X 255)
 - **Request and Response determined by MSRs**
 - **OR(Request bits true) .AND. OR(Response bits true)**
 - **Ex: all LLC misses = set bits**
0,1,2,3,4,5,6,11,12,13,14
 - 787F
- **Solves problem of averaging over widely differing penalties**
- **Only one version of the event (b7/msr 1a6)**
 - **offcore_response_0**



Memory Access: Off-core Access

- Offcore_Response_0
 - “umasks” set with MSRs 1a6

	Bit position	Description
Request	0	Demand Data Rd = DCU reads (includes partials, DCU Prefetch)
Type	1	Demand RFO = DCU RFOs
	2	Demand Ifetch = IFU Fetches
	3	Writeback = MLC_EVICT/DCUWB
	4	PF Data Rd = MPL Reads
	5	PF RFO = MPL RFOs
	6	PF Ifetch = MPL Fetches
	7	OTHER
Response	8	LLC_HIT_UNCORE_HIT
Type	9	LLC_HIT_OTHER_CORE_HIT_SNP
	10	LLC_HIT_OTHER_CORE_HITM
	11	LLC_MISS_REMOTE_HIT_SCRUB
	12	LLC_MISS_REMOTE_FWD
	13	LLC_MISS_REMOTE_DRAM
	14	LLC_MISS_LOCAL_DRAM
	15	IO_CSR_MMIO



Offcore_response Reasonable Combinations?

Request Type	MSR Encoding
ANY_DATA	xx11
ANY_IFETCH	xx44
ANY_REQUEST	xxFF
ANY_RFO	xx22
COREWB	xx08
DATA_IFETCH	xx77
DATA_IN	xx33
DEMAND_DATA	xx03
DEMAND_DATA_RD	xx01
DEMAND_IFETCH	xx04
DEMAND_RFO	xx02
OTHER	xx80
PF_DATA	xx30
PF_DATA_RD	xx10
PF_IFETCH	xx40
PF_RFO	xx20
PREFETCH	xx70

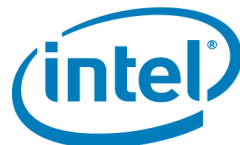
Response Type	MSR Encoding
ANY_CACHE_DRAM	7Fxx
ANY_DRAM	60xx
ANY_LLC_MISS	F8xx
ANY_LOCATION	FFxx
IO_CSR_MMIO	80xx
LLC_HIT_NO_OTHER_CORE	01xx
LLC_OTHER_CORE_HIT	02xx
LLC_OTHER_CORE_HITM	04xx
LCOAL_CACHE	07xx
LOCAL_CACHE_DRAM	47xx
LOCAL_DRAM	40xx
REMOTE_CACHE	18xx
REMOTE_CACHE_DRAM	38xx
REMOTE_CACHE_HIT	10xx
REMOTE_CACHE_HITM	08xx
REMOTE_DRAM	20xx

NT local stores counted by 0200 not 4000



Bandwidth per core

- **Much more complicated than on Intel® Core™2 processors**
 - **No single event counts total cachelines in+out to memory /core**
 - **Cacheable writebacks are written to LLC and written to memory at a later time**
 - **Offcore_response.data_ifetch.all_dram**
 - However, WB ->dram makes no sense
 - **Local vs remote memory**
 - **NT SSE Stored cachelines are problematic**
 - Offcore_response with MSR bit 7
 - offcore_response_0.other.llc_other_core_hit for SSE writes to local dram

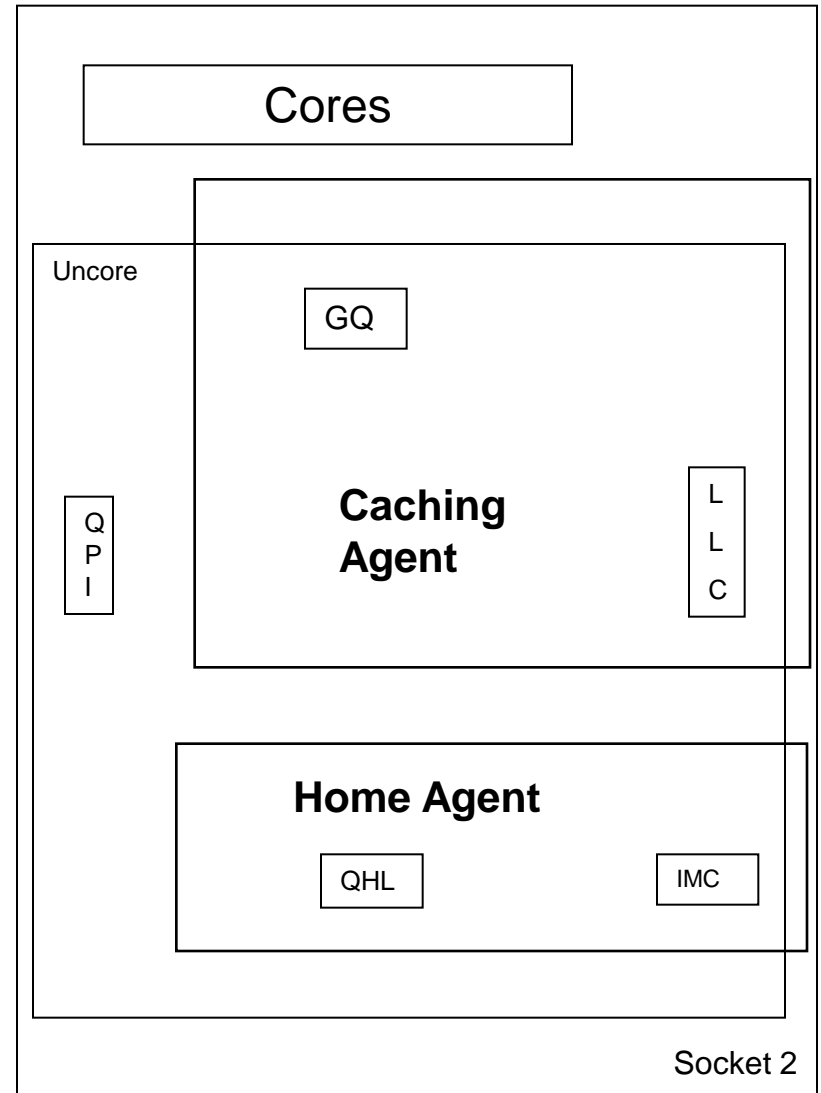
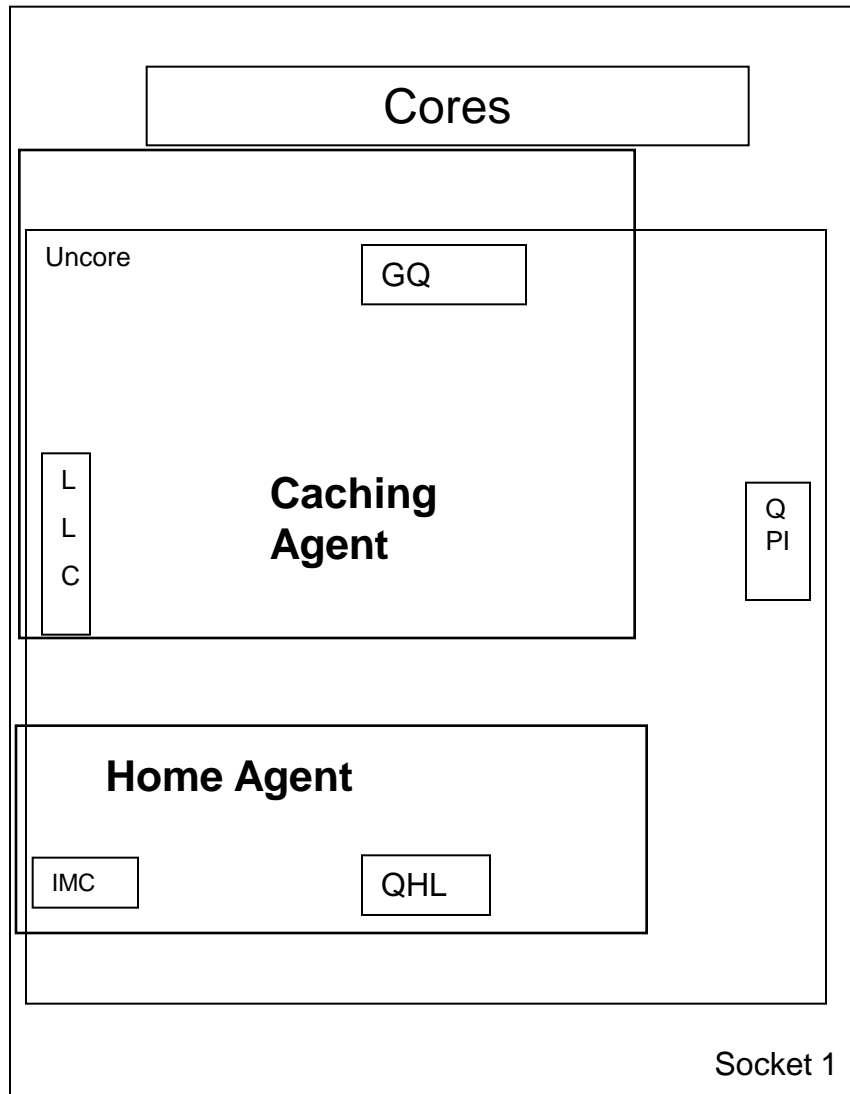


Memory Bandwidth

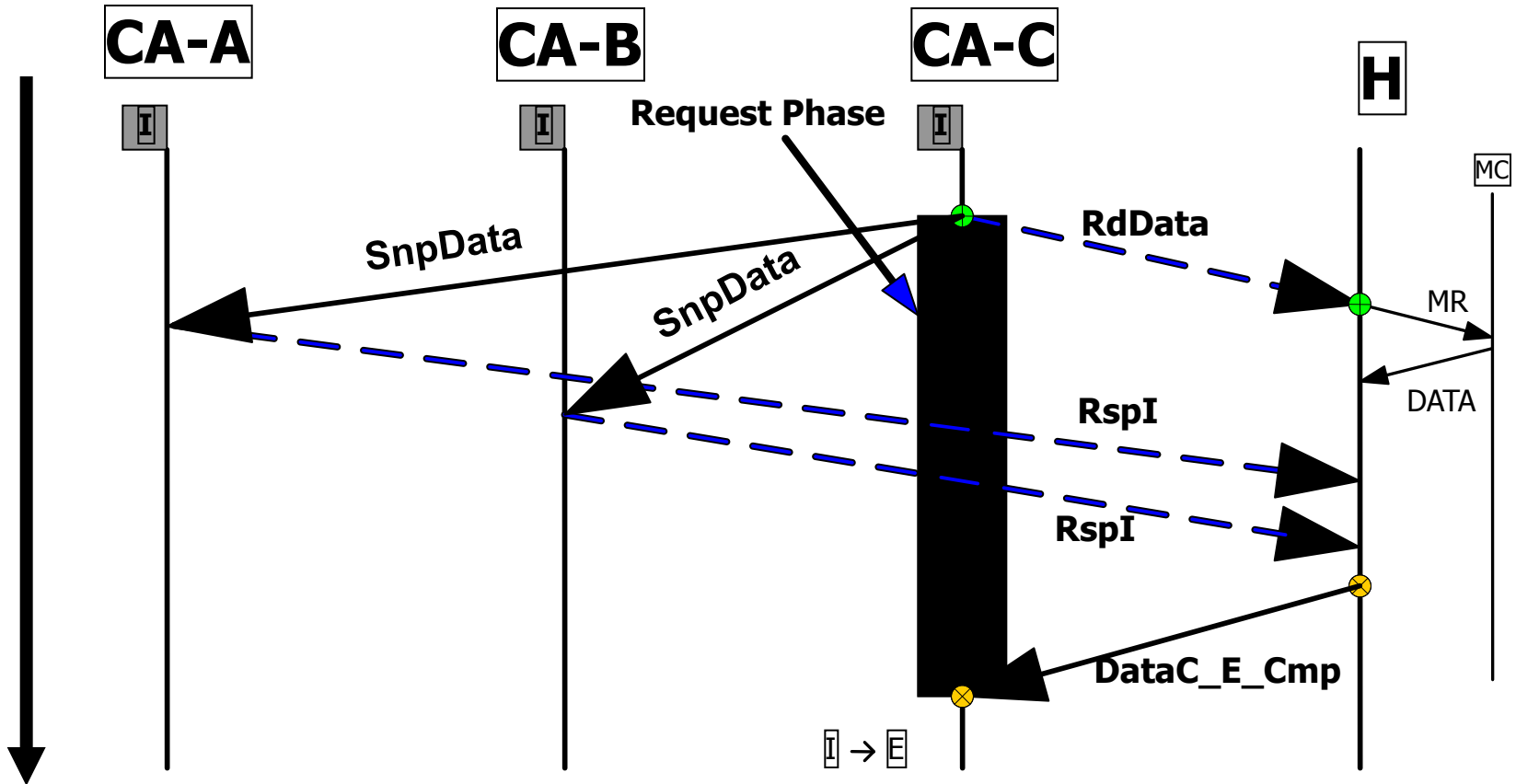
- **Delivered + Speculative Traffic to local memory**
 - **Reads and Writes Per Source**
 - UNC_QHL_REQUESTS.IOH_READS
 - UNC_QHL_REQUESTS.IOH_WRITES
 - UNC_QHL_REQUESTS.REMOTE_READS (includes RFO and NT store)
 - UNC_QHL_REQUESTS.REMOTE_WRITES (includes NT Stores)
 - UNC_QHL_REQUESTS.LOCAL_READS (includes RFO and NT Store)
 - UNC_QHL_REQUESTS.LOCAL_WRITES (no NT stores)
- **Precise totals can be measured in IMC**
 - **But cannot be broken down per source**
 - UNC_IMC_NORMAL_READS.ANY (or by channel, includes RFO)
 - UNC_IMC_WRITES.FULL.ANY (or by channel, includes NT stores)



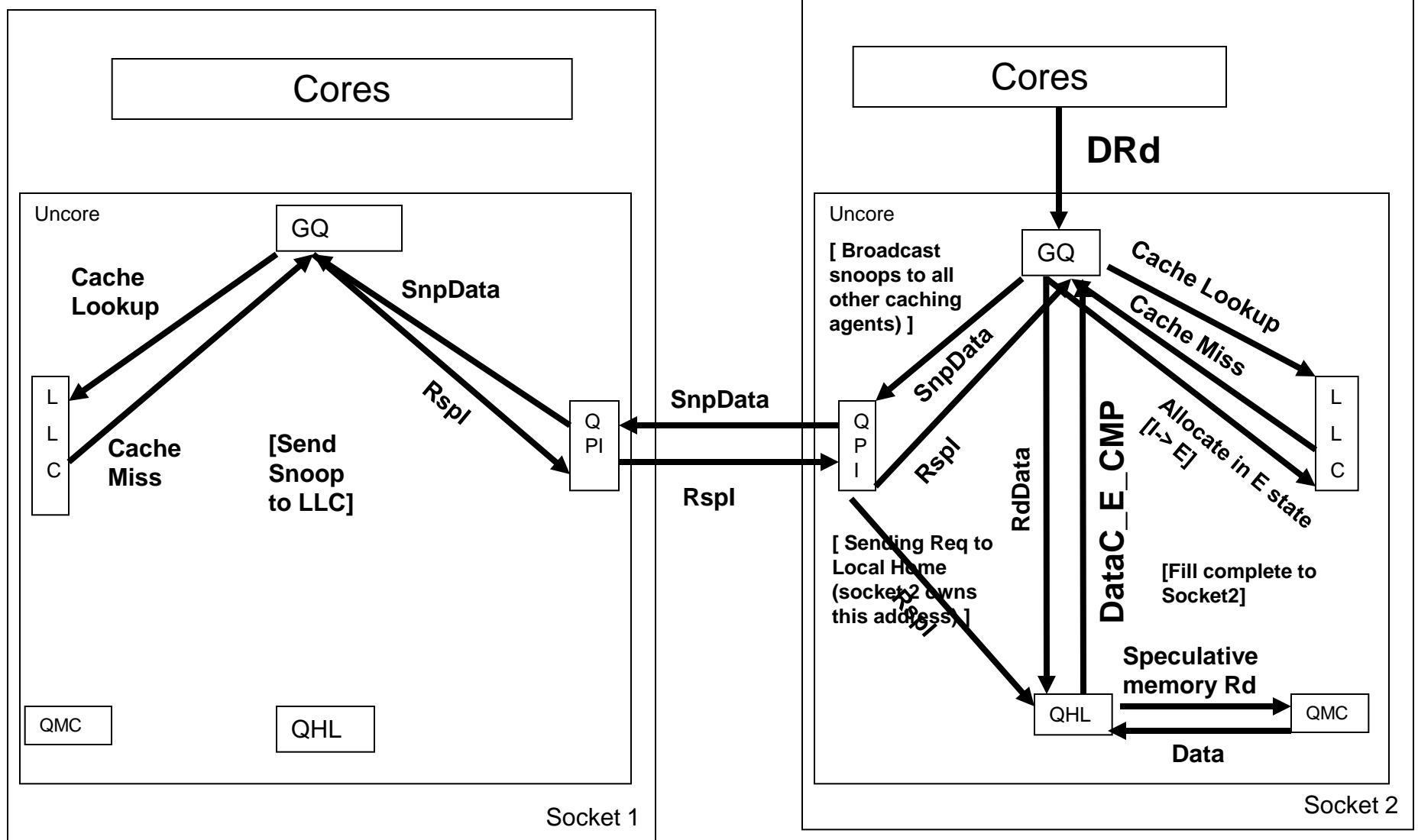
A NHM Socket is a Caching Agent and a Home Agent



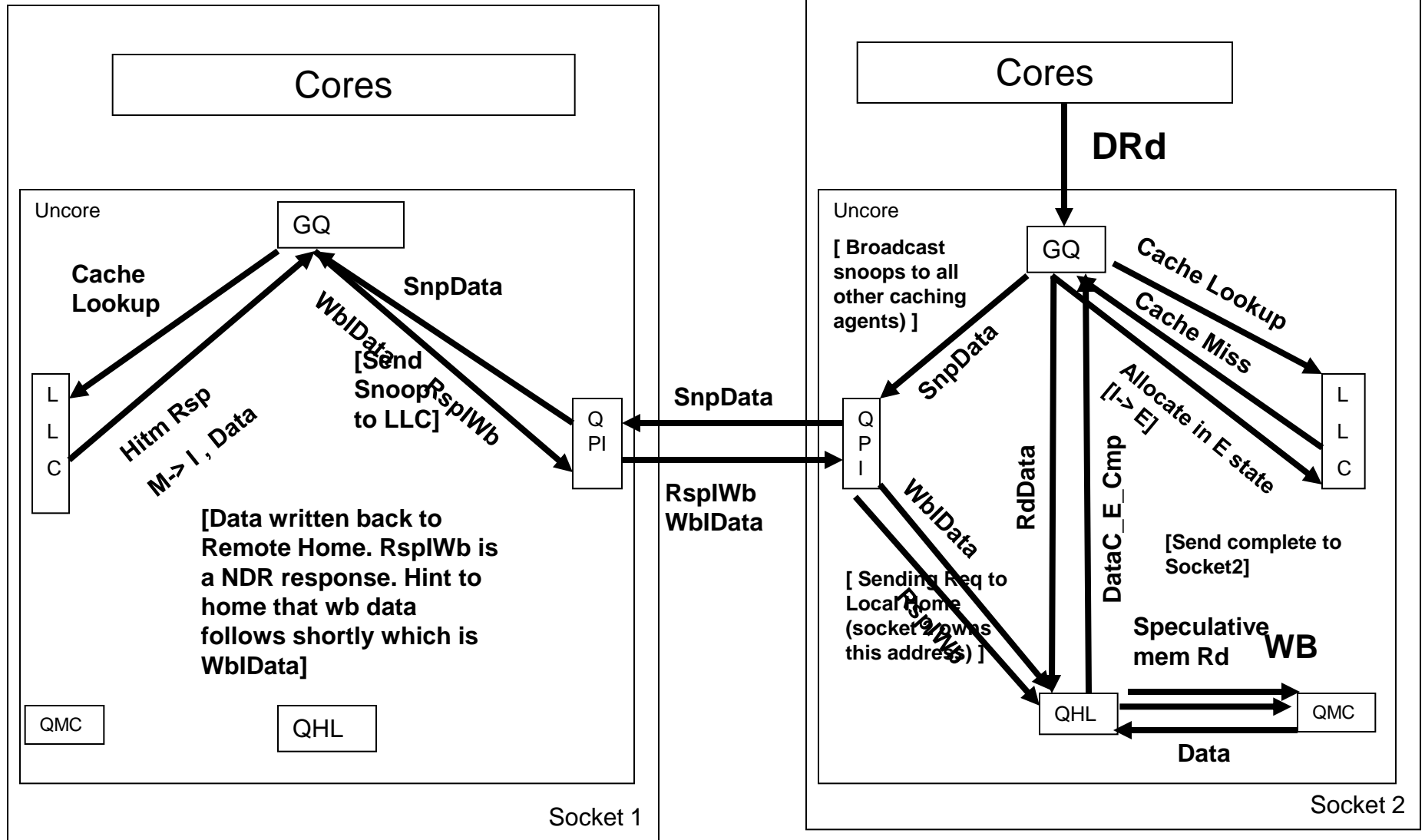
Simple Data Read



RdData request after LLC Miss to Local Home (Clean Rsp)



RdData request after LLC Miss to Local Home (Hitm Response)



Uncore Opcode Match events

- Match address, opcode using an MSR
 - 37 bit address match
 - 8 bit opcode match

Event	Event code	Umask
UNC_ADDR_OPCODE_MATCH.IOH_REQUEST_TRACKER	35	01
UNC_ADDR_OPCODE_MATCH.REMOTE_CORES_REQUEST_TRACKER	35	02
UNC_ADDR_OPCODE_MATCH.LOCAL_CORES_REQUEST_TRACKER	35	04

- Local Home data read, remote LLC hit
 - Ev=35, umask = 2, opcode = RspFwdS = 0001 1010, opcode only
- Local Home data read, remote LLC hitm
 - Ev=35, umask = 2, opcode = RspIWb = 0001 1101, opcode only
- RFO and perhaps other cases also (E->E problematic)



Precise Events

- **Significant expansion of PEBS capability on Intel® Core™ i7 Processors**
 - 4 events simultaneously
 - Latency event = IPF data ear + bit pattern for data source
 - Branches retired by type
 - Calls retired + LBR gives call counts
 - Calls_retired + full PEBS gives function arguments on Intel64



Data Access Analysis and PEBS

- **Data address profiling for loads and stores can be done as it is on Intel® Core™ 2 Processor Family**
 - **Full PEBS buffer + disassembly to identify registers with valid addresses at time of capture**
 - **Mem_inst_retired.load**
 - Cannot deal with `mov rax,[rax]` type instruction
 - **Mem_inst_retired.store**
 - Not subject to constraint of loads
 - **Inst_retired.any**
 - Cannot deal with `EIP+1` = first instr of Basic Block



Intel® Core™ i7 Processor PerfMon

PEBS Buffer

63	BTS Buffer Base	0
	BTS Index	
	BTS Absolute Maximum	
	BTS Interrupt Threshold	
	PEBS Buffer Base	
	PEBS Index	
	PEBS Absolute Maximum	
	PEBS Interrupt Threshold	
	PEBS Counter Reset 0	
	PEBS Counter Reset 1	
	PEBS Counter Reset 2	
	PEBS Counter Reset 3	

Merom/Penryn - Format 0000b
Nehalem - Format 0001b

63	RFLAGS	0
	RIP	
	RAX	
	RBX	
	RCX	
	RDX	
	RSI	
	RDI	
	RBP	
	RSP	
	R8	
	R9	
	R10	
	R11	
	R12	
	R13	
	R14	
	R15	
	Global Perf Overflow MSR	
	Data Linear Address	
	Data Source (encodings)	
	Latency (core cycles)	



PEBS Basic Events

- **Mechanism:**
 - counter overflow arms PEBS
 - Next event gets captured and raises PMI
 - PEBS mechanism captures architectural state information at completion of critical instruction
- **Including EIP (+1), even when OS defers PMI**
 - Accurate inst_retired profile

instr_retired
ITLB_MISS_RETIRE
uops_retired
br_instr_retired
ssex_uops_retired
other_assists
fp_assists
mem_instr_retired.loads (0B,umask=0)
mem_instr_retired.stores (0B,umask=1)



Load Latency Threshold Event: Saving the best for last

- **Ability to trigger count on minimum latency**
 - Core cycles from load execute->data availability
- **Linear address in PEBS buffer**
 - Allows driver to collect physical address
 - Only total measurement of local/remote home access
- **Data source captured in bit pattern**
 - Actual NUMA source revealed
- **Only ONE latency event/min thresh can be taken per run**
 - Minimum latency programmed with MSR
 - Global per core
 - 0x3F6 MS_PEBS_LD_LAT_THRESHOLD bits 15:0
 - HW samples loads
 - EX: Sampling fraction for local dram=
mem_inst_retired.latency_gt_128(DS= A or C)
/mem_uncore_retired.local_dram



Memory Access PEBS Events

- **Trigger on data source & capture address**
 - Except for `mov rax [rax];`
- **Identify LLC and DTLB load miss**
 - Precise load events do not include DCU prefetch/ L2 prefetch

Name	Event	Umask	Umask_name
<code>mem_load_retired</code>	<code>0xcb</code>	0	L1D_HIT
		1	L2_HIT
		2	LLC_HIT_UNSHARED
		3	OTHER_CORE_L2_HIT_HITM
		4	LLC_MISS
		6	HIT_LFB
		7	DTLB_MISS*

Precise Uncore Response

- **Load response from LLC, another core, local DRAM, remote socket, remote DRAM and IO**

Name	Event	Umask	Umask_name
mem_uncore_retired	0x0f	2	OTHER_CORE_L2_HITM
		3	REMOTE_CACHE_LOCAL_HOME_HIT
		5	LOCAL_DRAM
		6	REMOTE_DRAM
		7	IO



Precise Events can be Organized as a NUMA/Data Source Hierarchy

Data Source	Source Level Hierarchy		NUMA Hierarchy
L1D_hit	on_core		
L2D_hit			
LLC_hit_simple			
LLC_hit_shared			
LLC_hit_hitm	Local LLC	On Socket	Local/ Remote Home
Local_Dram	Dram		Local Home
Remote_LLC_hit_clean	Remote LLC		Remote/ Local Home
Remote_LLC_hit_hitm			
Remote_dram			Dram



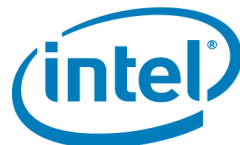
Precise Store DTLB miss

Name	Event	Umask	Umask_name
mem_store_retired	0x0c	0	DTLB_MISS*
		1	dropped events



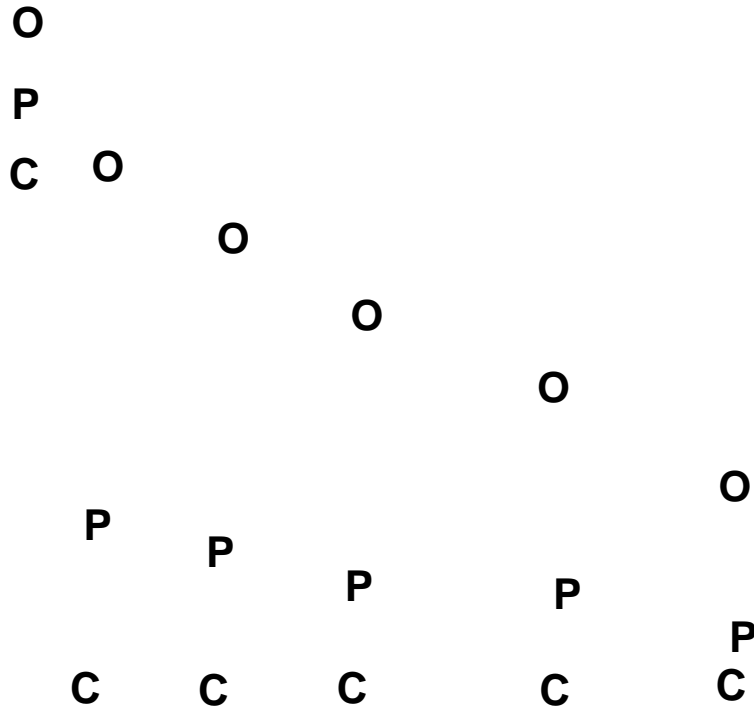
Shadowing and Precise Data Collection

- **The time between the counter overflow and the PEBS arming creates a “shadow”, during which events cannot be collected**
 - ~8 cycles?**
- **Ex: conditional branches retired**
 - **Sequence of short BBs (< 3 cycles in duration)**
 - **If branch into first overflows counter, PEBS event cannot occur until branch at end of 4th BB**
 - **Intervening branches will never be sampled**



Shadowing

20
20
2
2
2
2
20
20



N
N
0
0
0
0
5N

Basic Branch Analysis

- **Vastly improved precise branch monitoring capabilities**
 - Branches retired
 - **16 deep LBR**
 - LBR can be filtered by branch type and privilege level
- **Precise BR retired by branch type**
 - Calls, conditional and all branches
 - **Coupled with LBR capture yields**
 - Call counts
 - “HW call graph”
 - Basic block execution counts



Branch Analysis

- **Precise branch events on NHM enable**
 - **Function call counts**
 - **Function arguments (em64T only)**
 - **Taken fraction/branch**

**Mispredicted Branches must be counted with
Non-PEBS events**

BR_MISP_EXEC.* and BR_INST_EXEC.*



Branch Analysis: Call Counts

- **Call counts require sampling on calls**
 - Sampling on anything else introduces a “trigger bias” that cannot be corrected for
- **Requires PEBS buffer to identify which branch caused the event**
 - EIP+1 results in capturing call target
- **Requires LBR to identify source and target**
 - Matching PEBS EIP with LBR target



Precise Conditional Branch Retired

- **Counted loops that actually use the induction variable will frequently keep the tripcount in a register for the termination test**
 - E.g. heavily optimized triad with the Intel compiler has

```
Addq $0x8, %rcx
Cmpq %rax, %rcx
Jnge triad+0x27
```
- **Average value of RAX is the tripcount**



Branch Analysis: Function Arguments (Intel64 only)

- **Functions with “few” (<4?) arguments use registers for argument values**
- **Capturing full PEBS buffer + LBR on `calls_retired` event allows measurement of distribution of argument values per calling site**
 - **E.g. length of `memcpy`, `memset`**



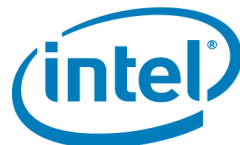
Last Branch Record LBR

- **16 source/target pairs deep**
- **One per SMT**
 - Not merged when SMT disabled
- **Only taken branches are captured**
- **Captured branches can be filtered**



Control Flow Analysis

- **Br_inst_retired.conditional + LBR gives Basic Block Execution Counts**
 - Track back through taken branches incrementing BB exec count by $1/\text{num_bb}$
- **Br_inst_retired.conditional + LBR gives taken fraction**
 - PEBS EIP must be first instruction of basic block
 - Not taken branches identified by PEBS EIP missing from LBR



Branch Filtering

LBR Filter Bit Name	Bit Description	bit
CPL_EQ_0	Exclude ring 0	0
CPL_NEQ_0	Exclude ring3	1
JCC	Exclude taken conditional branches	2
NEAR_REL_CALL	Exclude near relative calls	3
NEAR_INDIRECT_CALL	Exclude near indirect calls	4
NEAR_RET	Exclude near returns	5
NEAR_INDIRECT_JMP	Exclude near unconditional near branches	6
NEAR_REL_JMP	Exclude near unconditional relative branches	7
FAR_BRANCH	Exclude far branches	8



Branch Filtering

- **User near calls only**
 - Tracking back from OS critical sections to user function that caused the problem
 - Lack of returns may be an issue in some cases
 - But not for HPC 😊
 - Use static call analysis to clean up chains
- **User and OS near calls only**
 - Profiling OS call stacks
 - Eliminating leaf functions may be complicated by lack of returns
 - Don't remove returns if this is a problem
 - Use BTS to capture deeper stack
 - **Issue: cannot exclude unconditional jumps without excluding calls**



LBR, BTS and Branch Filtering

- **Filtered LBR can yield user call site in chains to critical event**
 - **Ex: long latency loads in synchronization functions**
- **Branch Filtering should enable accurate HW call graph analysis**
- **50-100 times as much data as current techniques**
 - **For fixed performance impact**



Precise Cycles

- **Allow profiling code sections screened with STI/CLI semantics**
 - Ring 0 OS critical sections
- **PEBS sampling mechanism may lose interrupts during halted state**
 - Instruction retirement required to generate performance monitoring interrupts (PMI)
Counts will not occur without PEBS being invoked



Front End/Decode Analysis

- **Instruction decode BW has lower maximum**
- **Instruction flow interruption at RAT output**
 - **UOPS_ISSUED.STALL_CYCLES – RESOURCE_STALLS.ANY**
 - **HT ON**
 - subtract half the cycles as well
 - Or **UOPS_ISSUED.CORE_STALL_CYCLES-RESOURCE_STALLS.ANY**
- **ILD_STALL.LCP_STALL**



Summary

- **Powerful capabilities = Very complicated**
- **Intel® PTU will simplify this**



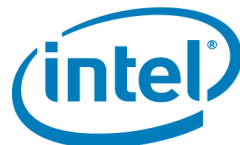
Why are there 50000000 many events?

- **Let's consider LLC misses on Intel® Core™2 Processor Microarchitecture**
 - Simple case
- **First question...what is an LLC miss?**



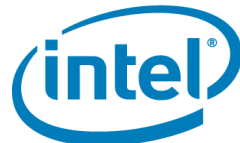
First question... what is an LLC miss?

- For stalled cycle decomposition**
 - LLC miss due to a load**
 - Not a HW prefetch**
 - Not a SW prefetch**
 - Not a secondary miss**
 - Not a RFO**
 - read for ownership; needed for writes**
 - Not a write**
 - 250 cycles stall ensues**
 - Mem_load_retired.l2_line_miss**



First question... what is an LLC miss?

- **For BW**
 - **ANY LLC miss**
 - A HW prefetch
 - A SW prefetch
 - NOT a secondary miss
 - An RFO
 - A cacheable writeback
 - A non-cacheable writeback
 - **BW limit on Intel® Core™2 Architecture Systems**
 - ~ 1 cacheline/30 cycles
 - **BUS_TRANS_BURST.SELF**



First question... what is an LLC miss?

- **For LLC cache thrashing**
 - **ANY LLC line in:**
 - A HW prefetch
 - A SW prefetch
 - NOT a secondary miss
 - A RFO
 - Not necessarily a cacheable writeback
 - Not a non-cacheable writeback
 - **L2_lines_in.any**
 - **but perhaps lines replaced is what you are looking for (L1D)**



First question... what is an LLC miss?

- **For shared line contention**
 - **ANY LLC miss due to loads**
 - A load driven line miss
 - not HW prefetch
 - not SW prefetch
 - A secondary miss
 - Not an RFO
 - Not necessarily a cacheable writeback
 - Not a non-cacheable writeback
 - **Mem_load_retired.L2_miss**

I could keep Going



Why are there 50000000 many events?

- **Trying to abstract counters to generic names is EXTREMELY unlikely to work**
- **It might work for LLC miss...but really?**
 - **Consider Intel® Core™ 2 Processor microarchitecture vs NUMA topology**
 - **Is there really any sense in equating FSB and NUMA architectures?**



Why are there 50000000 many events?

- **NUMA:**
- **An off-core cacheline request in a DP system has 15 possible sources**
 - Local, remote DRAM, or IOH
 - (unshared, shared, modified)*(local, remote home)*(local, remote LLC)
- **Times the request types (8 easily listed)**
- **Plus combinations**
 - Source = any remote socket



Offcore_Response_0

- **Matrix event**
- **(8 request types)*
(8 response sources)**
- **65K possible encodings**
- **We only predefined 270**
 - **You can only collect one event per run due to the complex resources needed for the programming**



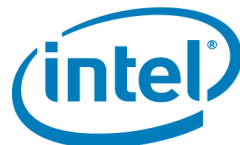
Why are there 50000000 many events?

- **Instruction and uop flow**
- **Multi staged asynchronous pipeline**
 - **Must measure flow at multiple places**
- **Each Stage has its own issues**
 - **FE/decode/branch mispredict are completely different than running out of exec resources for OOO engine**



Why are there 50000000 many events?

- **Uncore**
- **LLC activity**
- **Memory controller activity**
 - “Chipset” is now on the socket
- **NUMA interconnect activity**



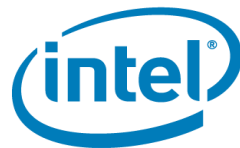
Summary

- **Intel® Core™ i7 Processor is an outstanding processor**
 - Huge Bandwidth increase
 - Improved Latency
 - Extended Resources -> More parallelism
 - Higher Frequencies
- **If the hardware doesn't win outright (unlikely), then it is the SW's fault**
 - And we can fix the SW
 - We have the technology



Call to Action

- **Go out there**
- **And have some fun**



NUMA, Intel® QuickPath Interconnect, and Intel® Core™ i7 Processor DP systems

- **Intel® QuickPath Interconnect (Intel® QPI) will greatly increase memory bandwidth of our platforms**
- **Integrated memory controllers on each socket access DIMMs**
 - Intel® QPI provides cache coherency
 - Bandwidth improves by a lot
- **Bandwidth improvement comes at a price**
 - Non-Uniform Memory Access (NUMA)
 - Latency to DIMMs on remote sockets is $\sim 2X$ larger

Peeling away the Bandwidth layer reveals the NUMA Latency layer



NUMA Modes on DP Systems Controlled in BIOS

- **Non-NUMA**

- Even/Odd lines assigned to sockets 0/1
 - Line interleaving

- **NUMA mode**

- First Half of memory space on socket 0
- Second half of memory space on socket 1



Non-Uniform Memory Access and Parallel Execution

- **Parallel processing is intrinsically NUMA friendly**
 - Affinity pinning maximizes local memory access
 - Message Passing Interface (MPI)
 - Parallel submission to batch queues
 - Standard for HPC
- **Shared memory threading is more problematic**
 - Explicit threading, OpenMP* product, Intel® Threading Building Blocks (Intel® TBB)
 - NUMA friendly data decomposition (page-based) has not been required
 - OS-scheduled thread migration can aggravate situation

*Other names and brands may be claimed as the property of others.



HPC Applications will see Large Performance Gains due to Bandwidth Improvements

- **A remaining performance bottleneck may be due to Non-Uniform Memory Access latency**
- **This next level in the performance onion was not really addressed**
 - **Other performance tools offered little insight**
 - **Default usage of Non-NUMA BIOS settings**
 - **Except for some HPC accounts**
- **Intel® PTU data access profiling feature was designed to address NUMA**
 - **NHM events were designed to provide the required data**



Legal Acknowledgements

- Intel, the Intel logo, Intel Core and Core Inside are trademarks of Intel Corporation in the U.S. and other countries.
- Vtune is a trademark of Intel Corporation in the U.S. and other countries.
- Itanium is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.
- Other names and brands may be claimed as the property of others.

