Performance Analysis and SW optimization for HPC on Intel® Core™ i7, Xeon™ 5500 and 5600 family Processors*

Presenter: David Levinthal
Principal Engineer

Business Group, Division: DPD, SSG

Version 1.1.2
July 28, 2010

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

-

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.

Performance Analysis Methodology for HPC

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

Performance Analysis Methodology for HPC

- Two possible objectives
  - Influence future silicon design
    - Intel personnel do lots of this
  - Modify build and/or source to improve performance
    - The sole focus of this presentation
- The central objective is to identify performance bottlenecks and estimate the potential gain for fixing them
  - Without an accurate estimate of the gain a great deal of effort can be wasted

Structure of this presentation

- What would the author do with:
  - A brand new machine
  - A tar ball of 100 million source lines
  - Documented, working build procedure
  - Data set and instructions to run the app
  - And one commandment:

Make Go Fast

but get the same answer

Presentation Agenda

- Optimization workflow overview
- Event based sampling
  - Why so complicated
  - How the nuts and bolts work
- HPC/Scientific computing overview
- Compiler problems/tuning compiler usage
- Identifying and removing stalls
- Identifying and removing resource saturation
- Identifying and removing non scaling
- PTU features and data interpretation
- Glossary in backup

Performance Analysis Methodology

- The steps
  - 1. make sure the platform is correct
    - It should be – some thought went into the specifications
    - 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), Intel® CILK™ Plus, explicit threading
- Iterate on 3 and 4

*Vtune and CILK are trademarks 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
    - **Make sure DIMMs are in identical sets of 6 for DP machines**
      - 3 channel memory controller
      - Best performance with completely uniform dimms
  - **Make sure SATA Bios setting is AHCI, not IDE setting**
    - **Use RAID or SSD if disk speed is critical**
  - **Prefetcher BIOS Settings correct for the app: ON**
    - **Intel® 11.0 compiler can generate SW prefetch**
  - **NUMA BIOS setting correct: ON**
  - **Intel® Hyper-Threading Technology BIOS option set correctly for the application**
    - **HT does not always help HPC**
      - Probably makes little difference

Disable C states to ensure machine stability when using event based sampling on Corei7/Xeon 5500

* 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
  - Accurate estimate of potential gain is critical!

- Documentation for Intel® Core™ i7 processor Performance Monitoring Unit (PMU) is available

Parallelization for HPC : Step 4

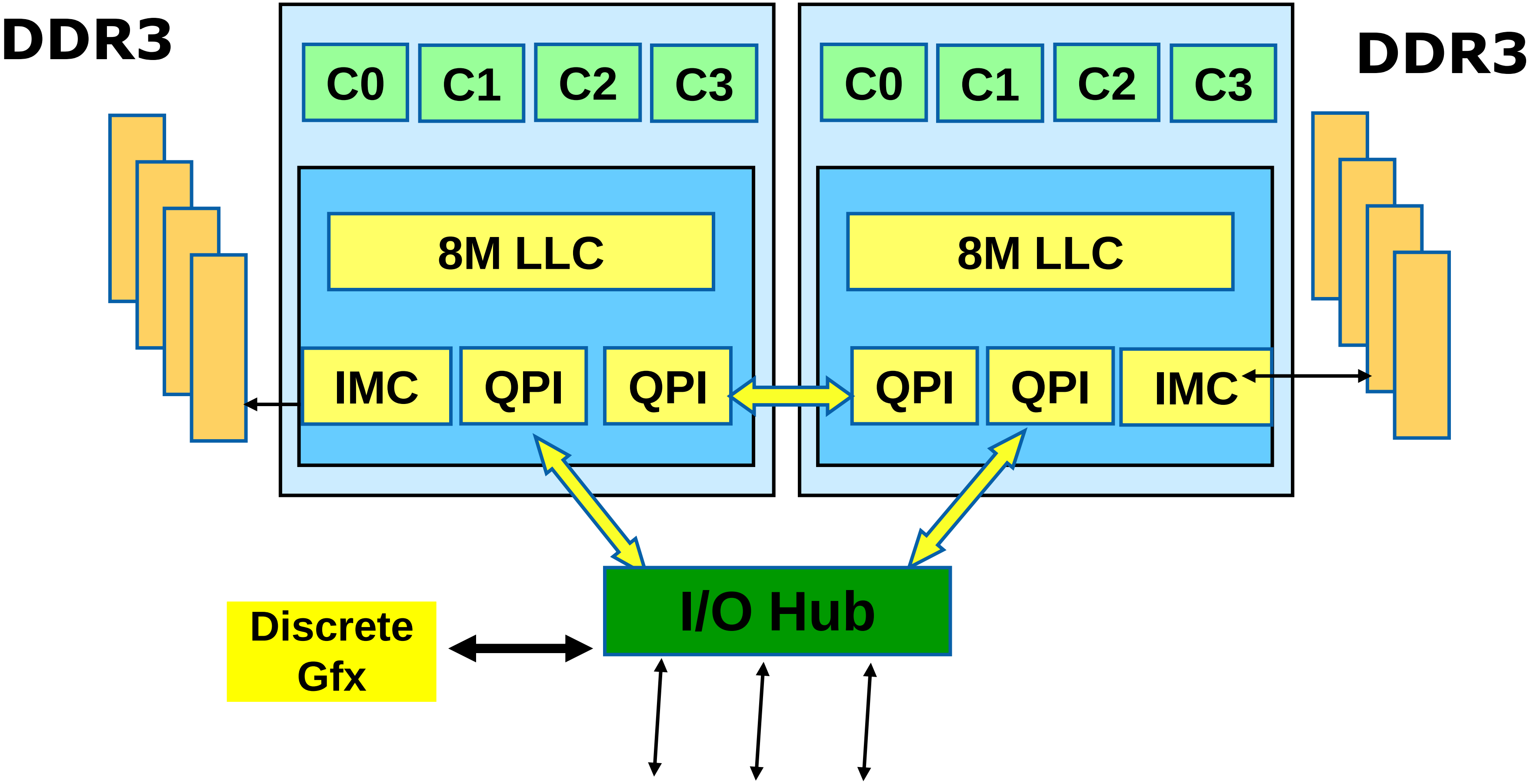
- 4. Use as many cores and machines as possible
  - Parallel processing by batch queue is OK
    - Trivial parallelism
    - Hard to beat the throughput

Parallelization for HPC : 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*, Intel® TBB, CILK, explicit threading for shared memory
    - Can reduce all to all MPI API costs

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

DP Platform


DDR3
C0
C1
C2
C3
8M LLC
IMC
QPI
QPI
C0
C1
C2
C3
8M LLC
QPI
QPI
IMC
DDR3
Discrete Gfx
I/O Hub

Event Based Sampling Analysis

- Code profiling with performance events can identify where the interaction of the code and data with the microarchitecture is sub optimal
  - Ex: What code execution results in load driven cache misses?
  - Event_count*Penalty ~ potential gain
    - A well defined penalty is <u>essential</u>
- Such profiling also provides an execution weighted display of the generated instructions
  - Vectorized code was generated but is it being executed?

**But There are THOUSANDS of Events,
Which Ones Matter?**

Which Events you need depends on what problem you wish to study and what you want to accomplish
Example: Last Level Cache Misses

- What you mean by an LLC miss depends on the exact nature of the question you are asking
- Are you asking about Bandwidth consumption?
  - Due to reads?, RFOs?, HW Prefetch, NT stores? Total?, Code?, SW prefetch?, Cacheable Writebacks?
  - Location of the bandwidth consumption?
  - Source of the data provided?
- Or about Latency/Pipeline stalls
  - Different architectures stall on different things
    - Intel® IA-32/Intel64 Processors' memory access stalls are mostly due to **loads**

Events needed to measure bandwidth and memory stalls are COMPLETELY different

Intel® Xeon™ 5500 load Penalties

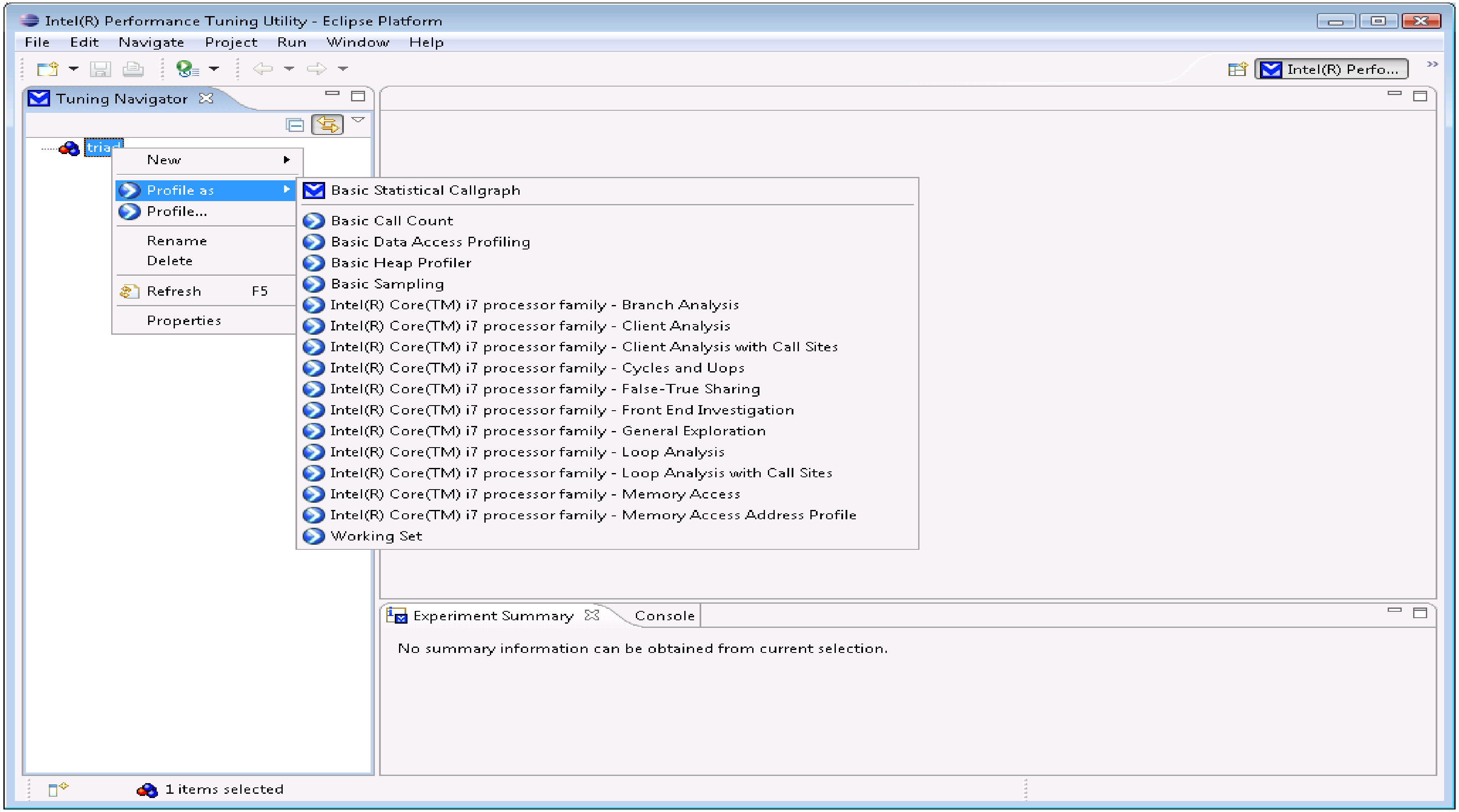
	L1D_HIT	Secondary Miss	L2 Hit	LLC Hit No Snoop	LLC Hit Clean Snoop	LLC Hit Snoop =HITM	Local Dram	Remote Dram	Remote Cache local home Fwd	Remote Cache Remote Home FWD	Remote Cache Local Home HITM	Remote Cache Remote home HITM
Mem_load_retired .L1d_hit	0 (By Def)											
Mem_load_retired .Hit_LFB		0->Max Val										
Mem_load_retired .L2_hit			6									
Mem_load_retired .LLC_Unshared_hit				~35								
Mem_load_retired .other_core_l2 _hit_hitm					~60	~75						
Mem_load_retired .LLC_Miss							~200	~350	~180	~180	~225 -250	~370
Mem_uncore_retired .Other_core_l2_hitm						~75						
Mem_uncore_retired .Local_Dram							~200				~225 -250	
Mem_uncore_retired .Remote_dram								~350				~370
Mem_uncore_retired. Remote_cache_ local_home_hit									~180			

Depend on frequency, dimms, bios, etc

Note: All latencies and memory access penalties shown are merely illustrative. Actual latencies will depend on (among other things) processor model, core and uncore frequencies, type, number and positioning of DIMMS, platform model, bios version and settings. Consult the platform manufacturer for optimal setting for any individual system. Then measure the actual properties of that system by running well established benchmarks.

The Important Penalties Vary by a Factor of TEN

Intel® PTU uses profiles to manage complexity


Intel(R) Performance Tuning Utility - Eclipse Platform
File Edit Navigate Project Run Window Help
Intel(R) Perfo...
Tuning Navigator
triad
New
Profile as
Profile...
Rename
Delete
Refresh F5
Properties
Basic Statistical Callgraph
Basic Call Count
Basic Data Access Profiling
Basic Heap Profiler
Basic Sampling
Intel(R) Core(TM) i7 processor family - Branch Analysis
Intel(R) Core(TM) i7 processor family - Client Analysis
Intel(R) Core(TM) i7 processor family - Client Analysis with Call Sites
Intel(R) Core(TM) i7 processor family - Cycles and Uops
Intel(R) Core(TM) i7 processor family - False-True Sharing
Intel(R) Core(TM) i7 processor family - Front End Investigation
Intel(R) Core(TM) i7 processor family - General Exploration
Intel(R) Core(TM) i7 processor family - Loop Analysis
Intel(R) Core(TM) i7 processor family - Loop Analysis with Call Sites
Intel(R) Core(TM) i7 processor family - Memory Access
Intel(R) Core(TM) i7 processor family - Memory Access Address Profile
Working Set
Experiment Summary
Console

No summary information can be obtained from current selection.

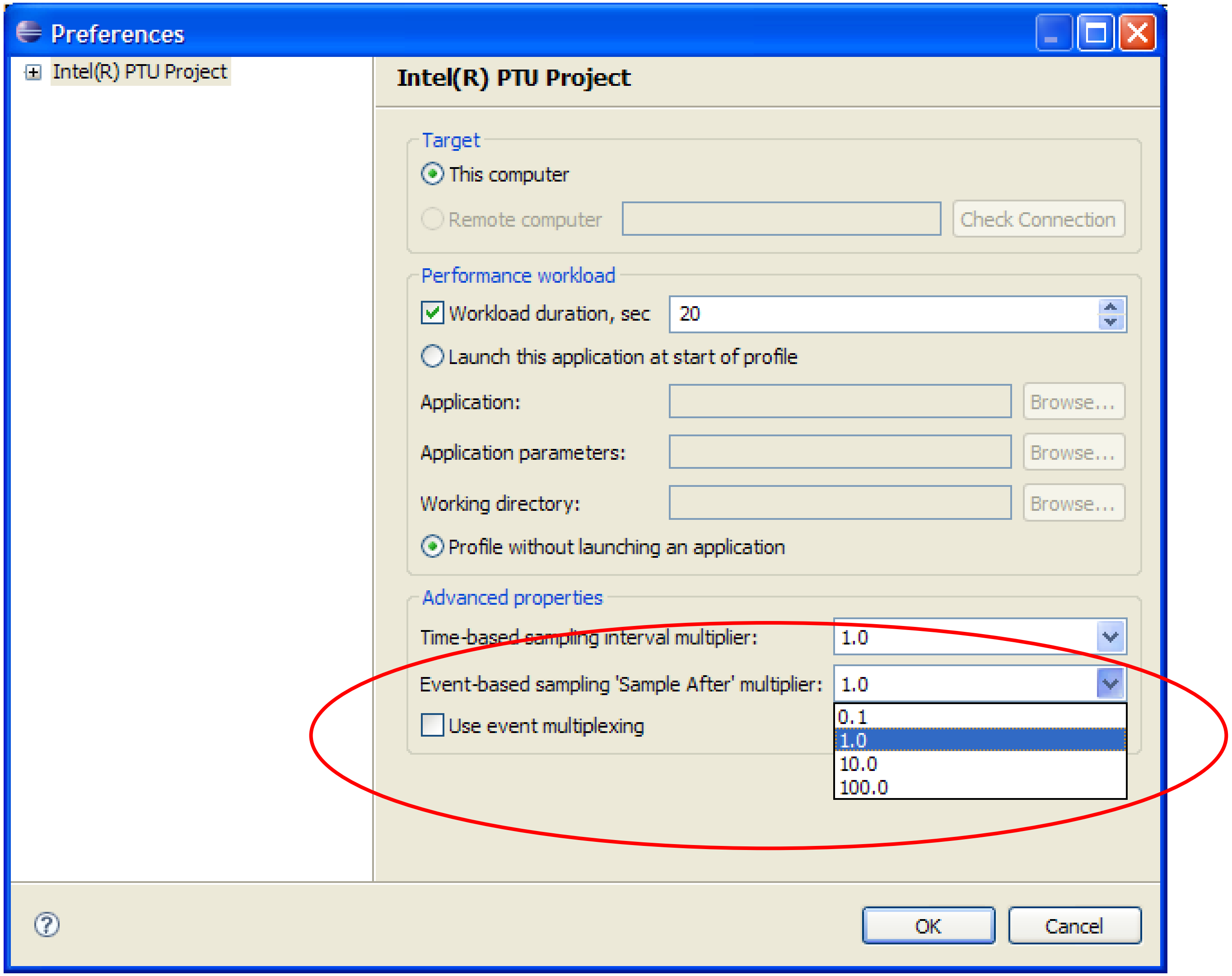
1 items selected

Intel® PTU predefined collections

- 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, instructions, 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

Many Possible Issues -> Many Different Events

Controlling collection



Preferences

Intel(R) PTU Project

Intel(R) PTU Project

Target

● This computer

○ Remote computer   Check Connection

Performance workload

☑ Workload duration, sec 20

○ Launch this application at start of profile

Application: Browse...

Application parameters: Browse...

Working directory: Browse...

● Profile without launching an application

Advanced properties

Time-based sampling interval multiplier: 1.0

Event-based sampling 'Sample After' multiplier: 1.0

0.1
1.0
10.0
100.0

☐ Use event multiplexing

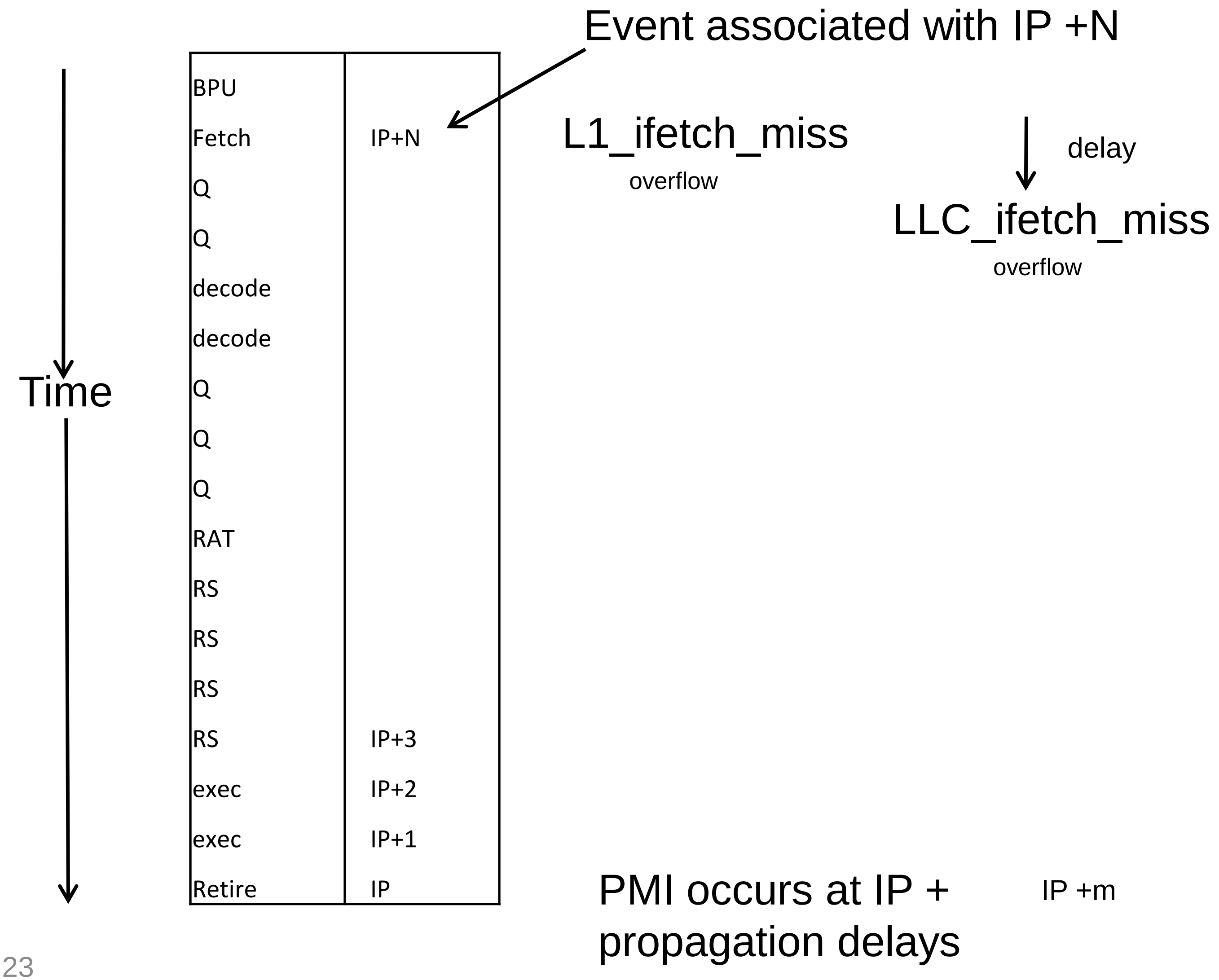
Performance Monitoring Unit

- **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**
  - **Particularly useful for studying the architecture’s strengths and weaknesses running an application**

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 events occur in the program**
    - **Application profiling by event**

SKID:
IP of causal instruction vs IP of PMI



Event associated with IP +N

BPU	
Fetch	IP+N
Q	
Q	
decode	
decode	
Q	
Q	
Q	
RAT	
RS	
RS	
RS	
RS	IP+3
exec	IP+2
exec	IP+1
Retire	IP

L1_ifetch_miss
overflow

delay

LLC_ifetch_miss
overflow

PMI occurs at IP + propagation delays

IP +m

Analyzing HPC Applications

- Overview
- Loop analysis
  - Tripcounts
  - Vectorization
- Memory access dominated
  - Latency dominated
  - Bandwidth dominated
- Execution dominated

Overview

- **Performance Breakdown/cycle accounting can be applied to any scale of a program**
  - **Multiple interacting applications-> single apps-> single modules-> source files/functions-> basic blocks**
- **Methodology does not change**
  - **But can inherit conclusions from higher levels based on importance/cycle cost**
- **At all stages in the process look for poorly written, actively executing code that can be improved**

HPC Applications

- Dominated by loops
- Rarely have pipeline front end problems
  - Except for very large binaries (ifetch latency)
- Large data sets
  - Not cache resident
  - Ex: Weather simulation, Oil Reservoir
  - Frequently DRAM bandwidth limited
  - Or DRAM Latency limited
- Occasionally HPC apps are uop flow limited
  - Data blocked
  - Ex: oil exploration, FFTs

What matters when optimizing a loop?

1. **The Trip Count**
2. **The Trip Count**
3. **The TRIP COUNT!**
4. **Variations in the tripcount**
5. **And some other things**

   BUT..what you do about them depends on

   THE TRIP COUNT

And of course there are virtually no tools to assist you in determining this..other than printf

(you can use PIN..)

This Will be Discussed Later

HPC Loops and Memory Access

- Calculations require data as input and the most severe limitations in a computer are on data access
  - CPU speed and efficiency have increased much faster than memory speeds and bandwidth.
- Load operations are almost always scheduled almost immediately before consumption (adds, multiplies etc)
- Lack of availability will quickly lead to execution stalls
  - OOO execution can buy only a few cycles.

Event Classes: High Level View

1. Execution flow events
   - Cycles, Branches, stalls, uops/inst_retired
   - Guide compiler usage
2. Penalty events
   - Ex: load requiring access to dram
   - Modify code/build to reduce penalties
3. Resource saturation events
   - Bandwidth, load/store buffers, dispatch ports
   - No well defined cost
   - Change data layout/access patterns
4. Architectural characterization
   - Cache accesses, MESI states, snoops
   - Used to improve silicon design, not application performance
5. Instruction mix
   - Do not measure what you think, extremely difficult to validate

Event Classes

1. **Execution flow events: Guide Compiler Usage**
   - **Cycles, Branches, stalls, uops/inst_retired**
2. Penalty events
   - Ex: load requiring access to dram
3. Resource saturation events
   - Bandwidth, load/store buffers, dispatch ports
   - No well defined cost
4. Architectural characterization
   - Cache accesses, MESI states, snoops
5. Instruction mix

Cycles: Multiple time domains

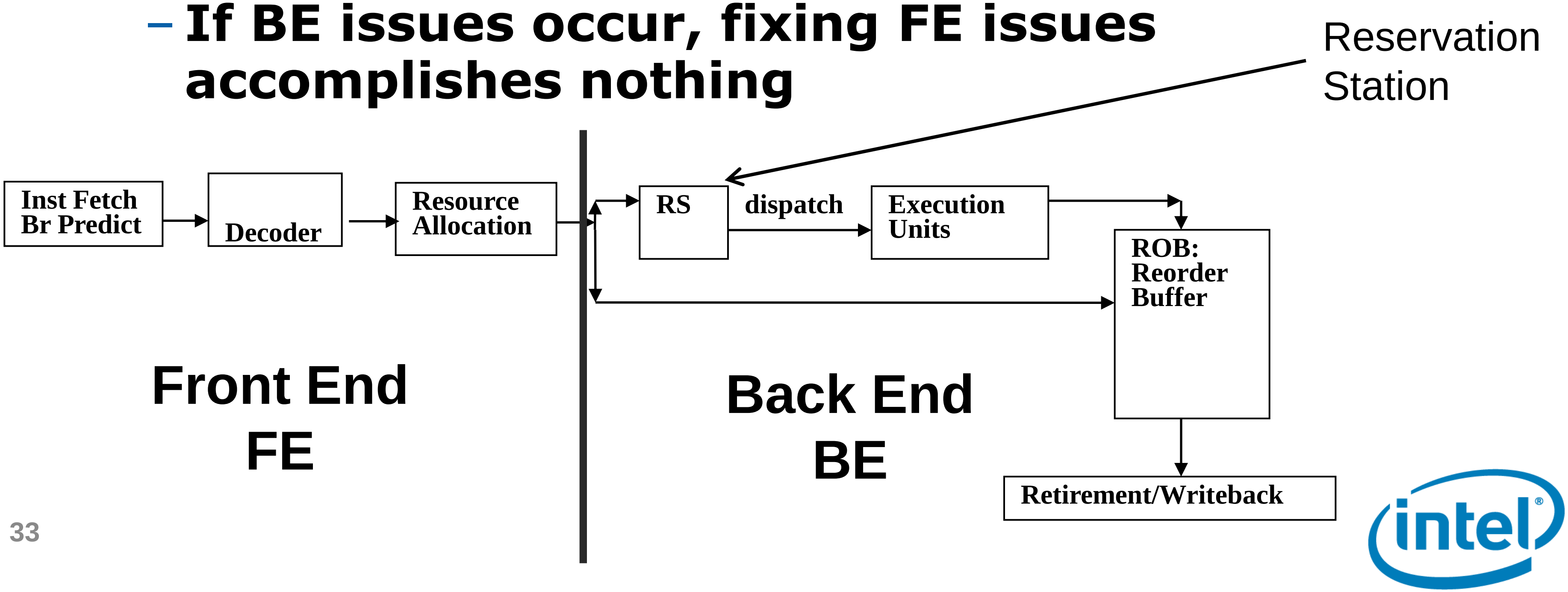
- There are actually 4 cycle events on a modern microprocessor
  - Core unhalted cycles
  - Reference frequency unhalted cycles
  - Core halted cycles
  - Reference Frequency halted cycles
- Core frequency needed for perf issues entirely in the core
  - Penalties (ie pipeline stalls) in core cycles
- Reference frequency needed for:
  - Evaluation of variable frequency effects (Turbo/Power Management)
  - Wall clock time utilization
    - Ex: Network server applications
  - Bandwidth/memory latency
- Unhalted events are required for counting modes to work at all
- Halted.ref = TSC change – cpu_clk_unhalted.ref

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
    - Uops_issued.any – uops_retired.slots
- Optimization Reduces Total Cycles by
  - Reducing stalls
  - Reducing retired uops (better code generation)
  - Reducing non retired uops (reducing mispredictions)

(Simplified) Execution in an OOO Engine

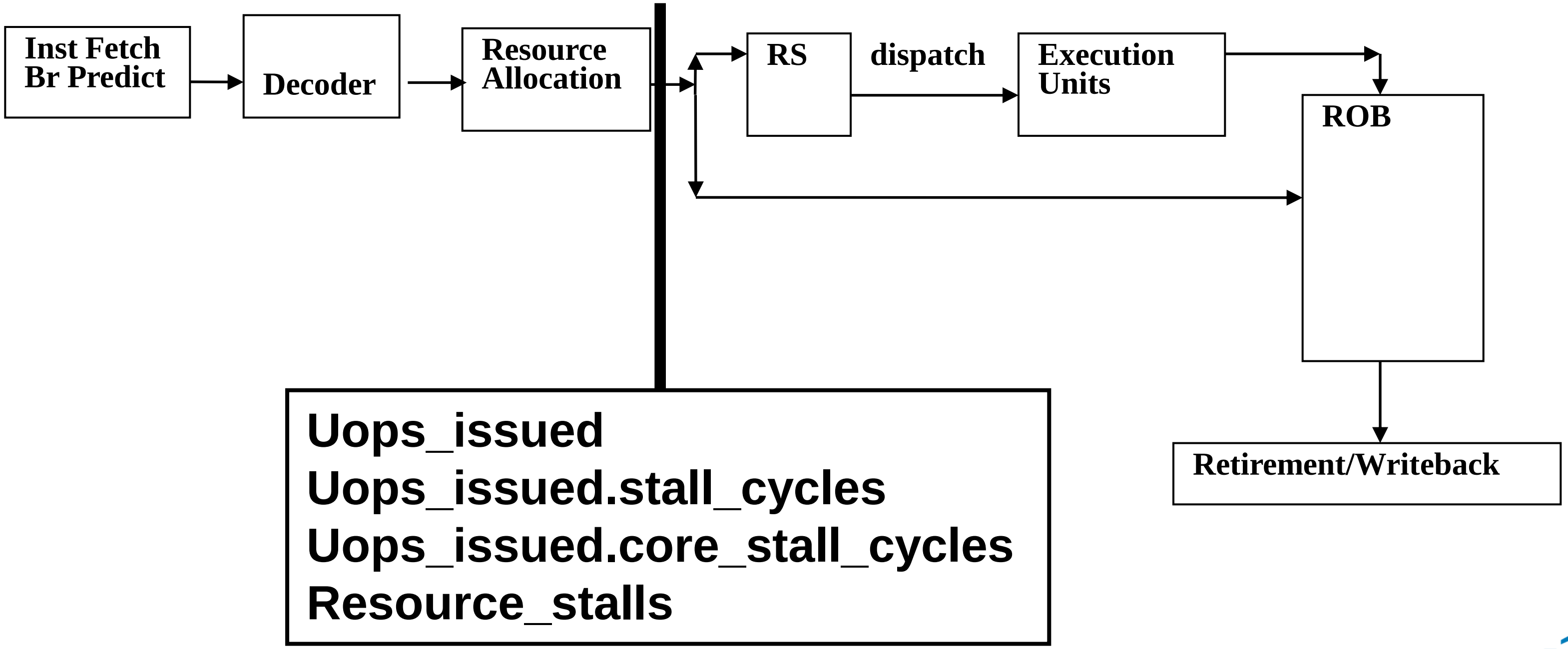
- **Two asynchronous components connected by buffering**
  - **Front End provides instructions**
  - **Back End gets data and executes instructions**
  - **Back End trumps Front End**
    - **If BE issues occur, fixing FE issues accomplishes nothing**



Identifying Front End Stalls

- Uop issue
  - Uops have been allocated resources
  - No downstream blockage (resource_stalls)
  - FE Stalls = an instruction delivery problem

    = Uops_issued.stall_cycles – Resource_stalls



Inst Fetch
Br Predict

Decoder

Resource
Allocation

RS

dispatch

Execution
Units

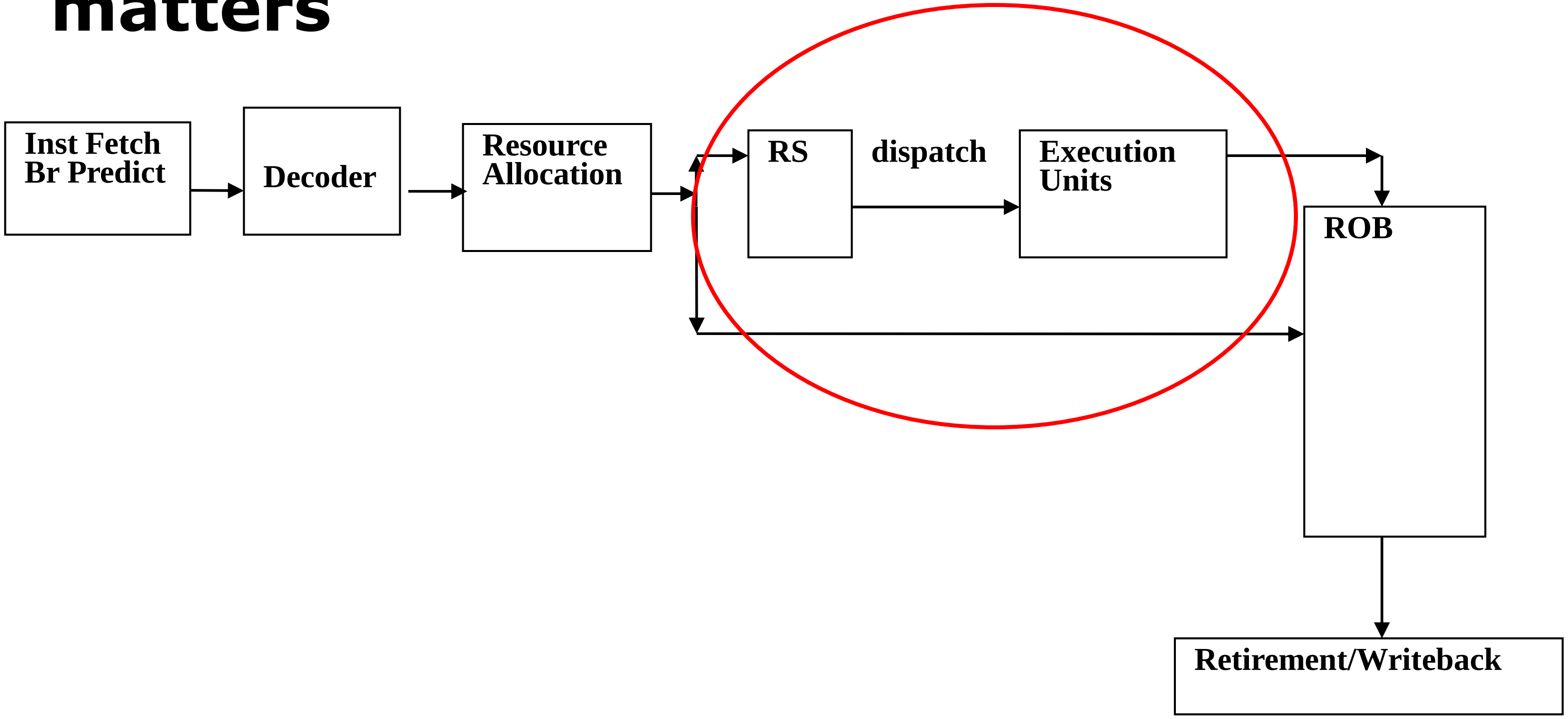
ROB

Retirement/Writeback

Uops_issued
Uops_issued.stall_cycles
Uops_issued.core_stall_cycles
Resource_stalls

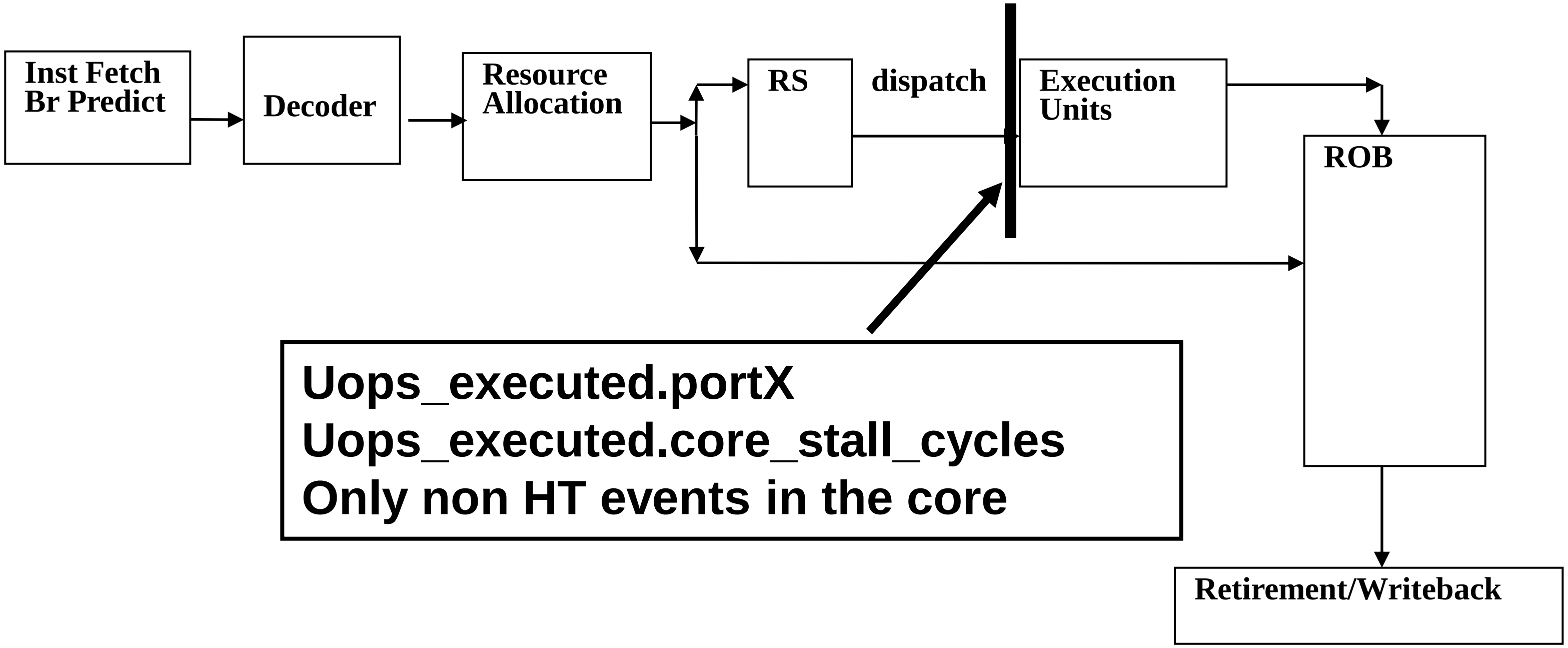
(Simplified) Execution in an OOO Engine

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



Uop Flow Monitors Execution

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



Inst Fetch
Br Predict

Decoder

Resource
Allocation

RS

dispatch

Execution
Units

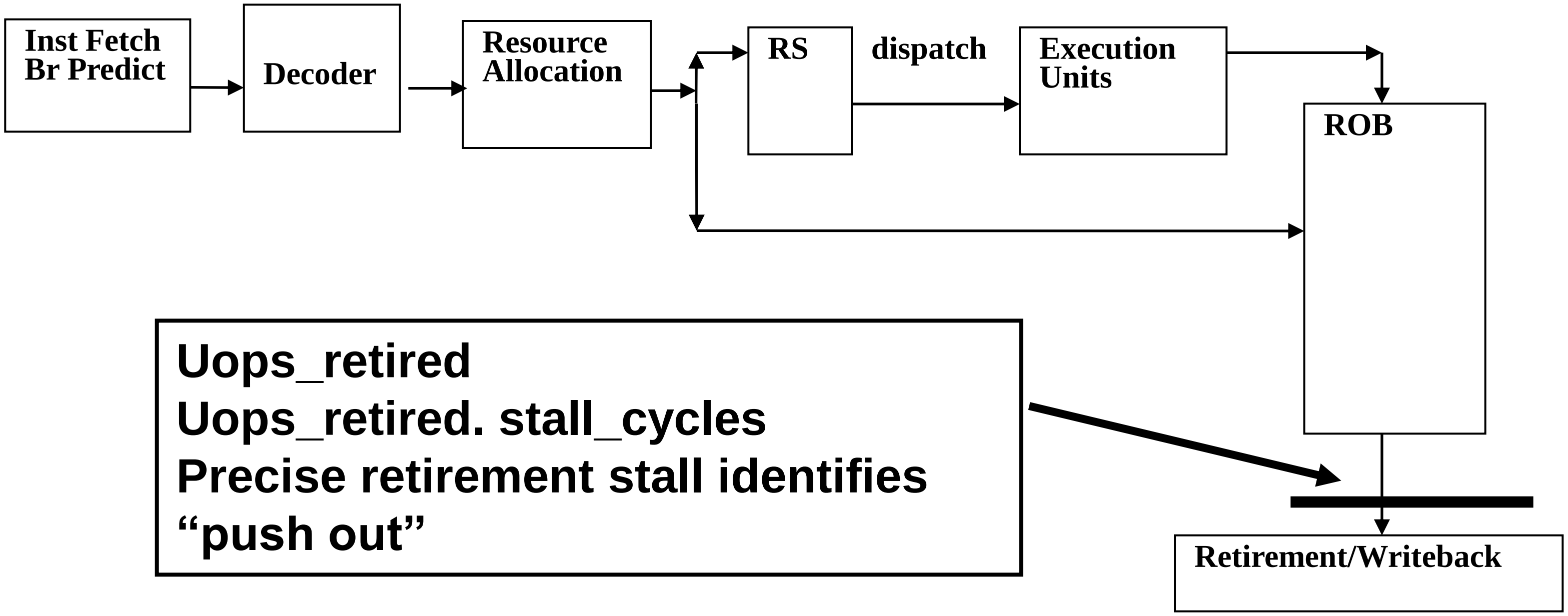
ROB

Retirement/Writeback

Uops_executed.portX
Uops_executed.core_stall_cycles
Only non HT events in the core

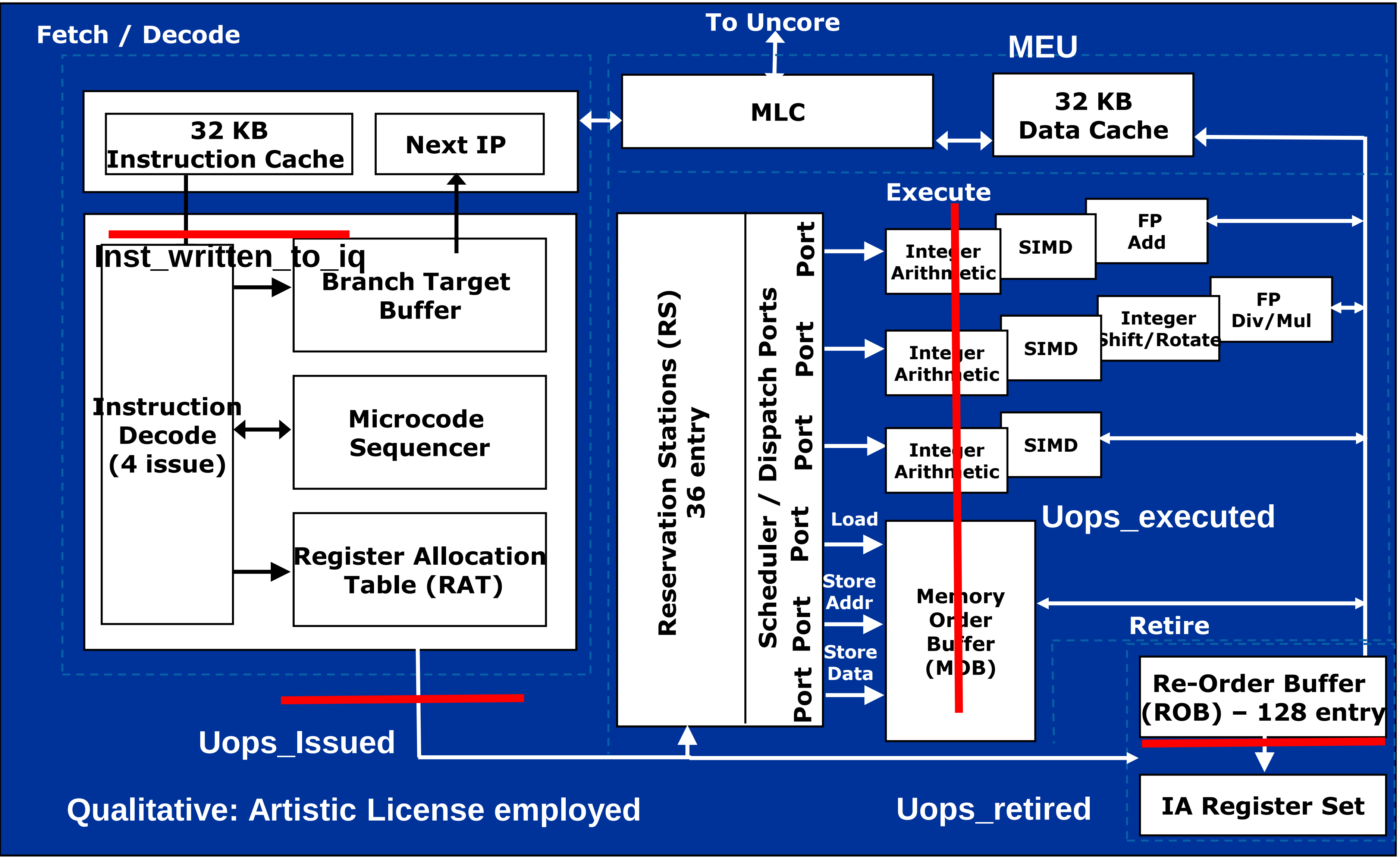
Uop Flow Monitors Execution

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



Uops_retired
Uops_retired. stall_cycles
Precise retirement stall identifies
“push out”

Uop Flow



Qualitative: Artistic License employed

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**

instr_retired
itlb_miss_retired
uops_retired
br_instr_retired
mem_instr_retired.loads
mem_instr_retired.stores

For memory events, EIP (+1) is always next instruction

Branch Events

- Measure Control flow through the program
- Can be used for
  - loop trip counts
  - Reconstructing (multi function) execution paths
  - Driving inlining, IPO, PGO compilations
- Used in conjunction with Last Branch Record (LBR) even more can be done
  - Basic block execution counts
  - Instruction mix
  - Call counts per source
  - etc

Basic Branch Analysis

- **Vastly improved precise branch monitoring capabilities**
  - **Branches retired**
  - **16 deep LBR**
    - LBR can be filtered by branch type and privilege level
  - **One per SMT**
    - Not merged when SMT disabled
  - **Only taken branches are captured**
- **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.* on Corei7/Xeon 5500
- Br_misp_retired.* on Xeon 5600 (PEBS)

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

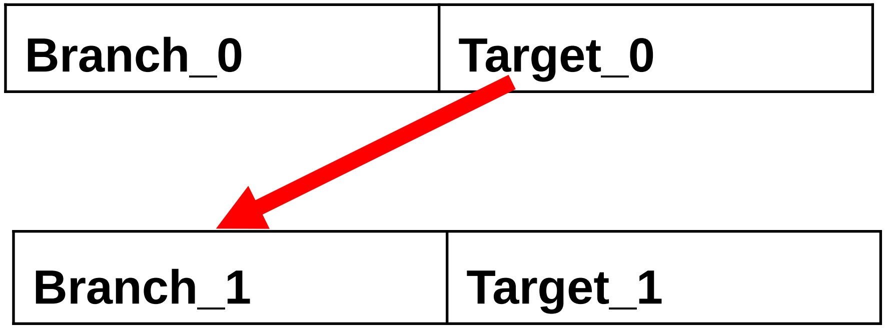
- Counted loops that actually use the induction variable will frequently keep the tripcount in a register for the termination test
  - E.g. heavily optimized triad with the Intel compiler has

    ```
    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” (<6?) 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

Processing LBRs



Branch_0 | Target_0

Branch_1 | Target_1

- All instructions between Target_0 and Branch_1 are retired 1 time
- All Basic Blocks between Target_0 and Branch_1 are executed 1 time
- All Branch Instructions between Target_0 and Branch_1 are not taken

So it would all Seem Very Straight Forward

Shadowing and Precise Data Collection

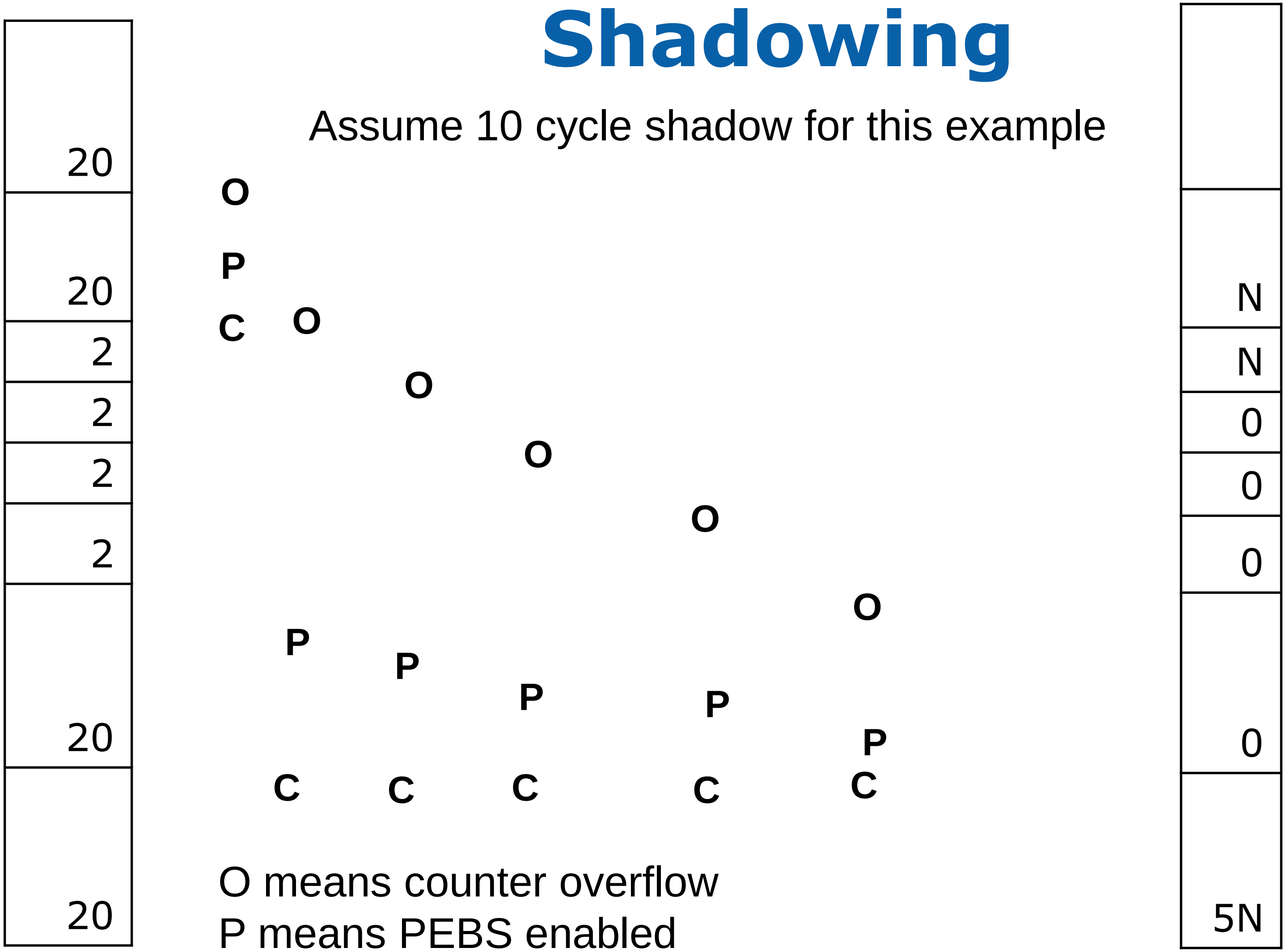
- The time between the counter overflow and the PEBS arming creates a "shadow", during which events cannot be collected

  ~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

Assume 10 cycle shadow for this example



O means counter overflow
P means PEBS enabled
C means interupt occurs

Reducing Shadowing Impact

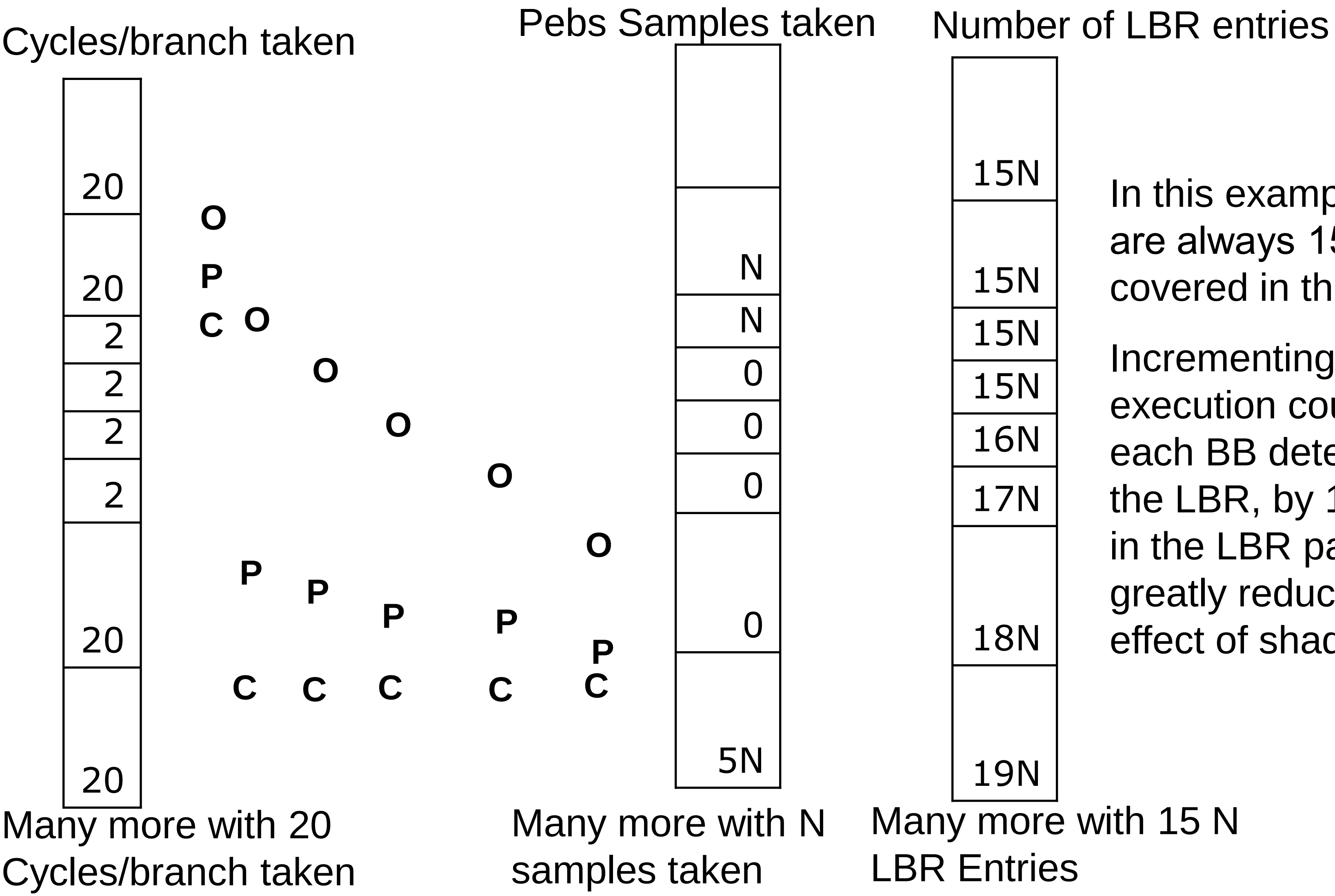
- Some “events” will never occur!
  - Falling into shadowed window
- Use LBR to extend range of the single sample
- Count the number of objects in LBR and increment count for all of them by 1/15
  - Since you have only one sample

Minimizing Shadowing Impact on BB Execution Count

Cycles/branch taken

Pebs Samples taken

Number of LBR entries



20
20
2
2
2
2
20
20

N
N
0
0
0
0
5N

15N
15N
15N
15N
16N
17N
18N
19N

Many more with 20 Cycles/branch taken

Many more with N samples taken

Many more with 15 N LBR Entries

In this example there are always 15 BB's covered in the LBR.

Incrementing the BB execution count for each BB detected in the LBR, by 1/15 seen in the LBR path will greatly reduce the effect of shadowing

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

Precise cycles can be constructed from any PEBS event

- Allow profiling code sections screened with STI/CLI semantics
  - Ring 0 OS critical sections
- PEBS sampling mechanism may loose interrupts during halted state
  - Instruction retirement required to generate performance monitoring interrupts (PMI)

    Counts will not occur without PEBS being invoked

Using cycles to optimize the optimizations

- **Profile the application for cycle usage and uop flow.**
  - **Identify hot functions**
  - **Check asm of FP intensive code for correct instruction mix**
    - **X87 is slower than SSE**
    - **Intel® Compiler has FP-model flags and many pragmas**
- **Vectorize long tripcount loops**
  - **-SSE4.2 uses unaligned loads more aggressively**
    - **Align data whenever possible**
  - **Check loop tripcounts with br events and register values (described later)**
    - **Interchange loop orders to get long loops as inner loop**
      - **Change multi dimensional array layout as needed**
    - **Completely unroll short tripcount (<~7) inner loops**
    - **Split/merge loops depending on code size**
    - **Predicate hoist constant condition if’s out of loops**
    - **Etc, etc , etc…I could write a book**

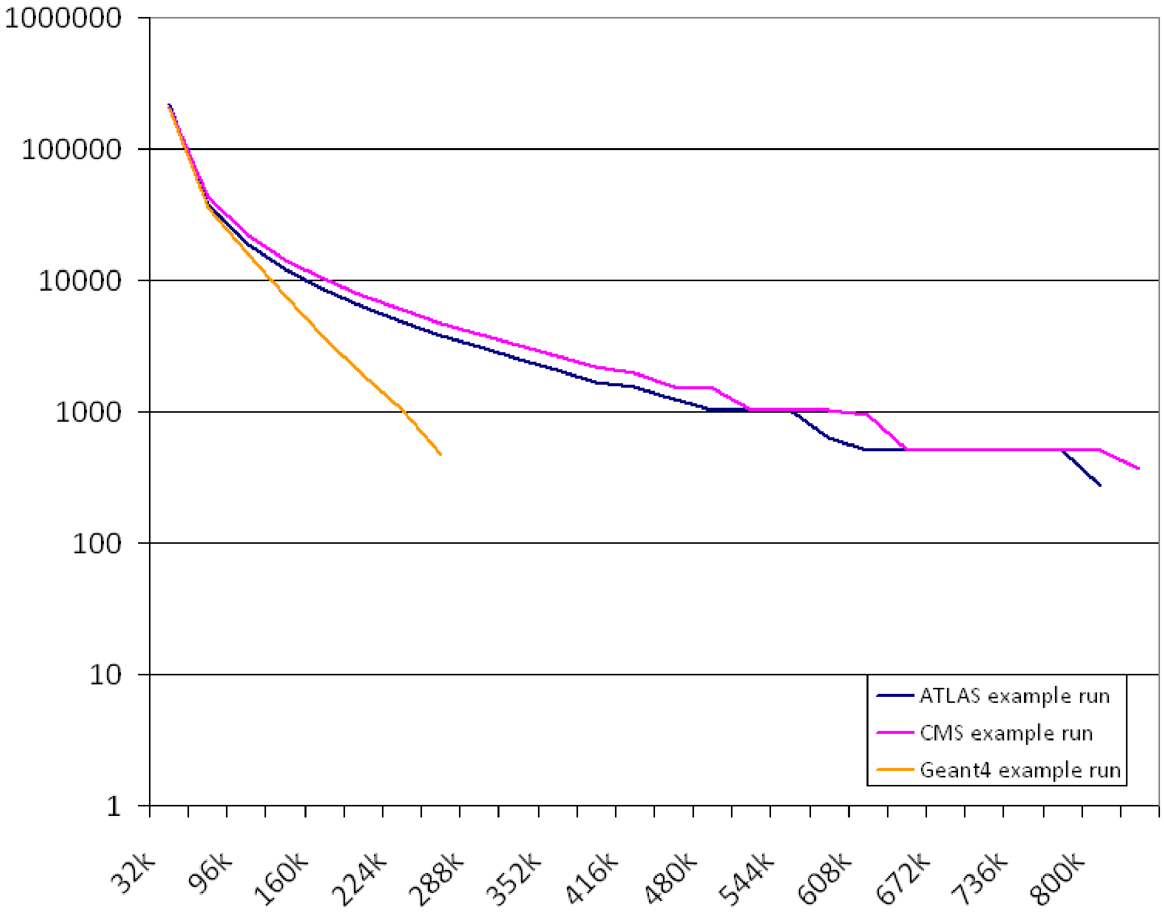
Using cycles to optimize the optimizations

- C++ and large binaries: Only optimize what uses cycles
  - Use call counts to drive compiler inlining
    - Compiler needs to evaluate a large enough scope to do its best work
    - Particularly functions/methods invoked inside loops
  - Size vs Speed
    - Extremely large binaries need to minimize size
      - -Os (linux) –O1 (windows)
  - Conditional Branch Mispredictions
    - HW prediction is shockingly good
      - Cost is unretired uop flow (uops_issued.any – uops_retired.slots)
      - Optimize case statement order, lowers uops_retired
- Use Intel Compiler LIBM,MKL, tbbmalloc, tbbmalloc_proxy
  - Intel linker with LD_PRELOAD env variable
  - -L/path/to/intel/libs –limf etc
  - http://software.intel.com/en-us/articles/optimizing-without-breaking-a-sweat/

Thoughts on optimizing large OOP code bases

- **Classic OOP will result in code bases of small functions integrated together to invoke the algorithm**
- **Signatures**
  - **Low instruction_retired/call_retired**
  - **High call_retired/branch_retired**
  - **High indirect_call/call_retired**
  - **High uops_issued.core_stall_cycles – resource_stalls.any**
  - **High $\sum$latency(source)*ifetch_miss(source)**

How big are the CERN programs



Cacheline access frequency evaluated by sorting cachelines by their accesses

Thus a binary working set size measurement

Optimizing large Object Oriented Code

- Inlining is the advice of choice but things are more complicated.
- Inlining increases binary size and can make ifetch misses more costly and code slows down
  - Even if fewer in overall number
- Ifetch miss events have among the largest IP skids of all events
  - They can show up in the wrong function
- Large codes built of many small methods can result in flat cycle profiles
  - It can take thousands of functions to account for 80% of the clock cycle samples
  - Thus thousands of functions must be optimized to achieve a significant performance improvement

Optimizing large Object Oriented Code

- The author knows of no proven methodology to correct the cost of excessive taken branches and the resulting flat cycle profile.
  - Need fewer calls,
    - instructions required for calling conventions
  - Larger functions to allow the compiler to see the whole calculation and do a better job
  - Larger shared objects to allow greater effect from IPO
    - Create shared objects using just the hot methods to avoid excessive inlining
- This has to be applied to enough methods to account for 80->95% of the cycles

Mostly this is about reducing the total instruction count

Thoughts on optimizing large OOP code bases

- Function calls result in added instructions
  - Call and return
  - Trampolines required for position independent code/ shared object cross invocations
    - Indirect branches can be more costly
  - Freeing & restoring registers for local use
    - Mostly an ia32 issue
  - Setting and reading function arguments
    - Larger on ia32 due to required use of stack
- Virtual function calls (function pointers) increase indirect call instructions and associated pointer loads

Thoughts on optimizing large OOP code bases

- Does a call graph help?
  - Unlikely
  - Provides the direct path back to main
    - Usually sampled on time
    - Does not provide call counts in most cases
  - Does not identify clusters of active (excessive) call activity

Thoughts on optimizing large OOP code bases

- **A modest proposal:**
- **Use LBRs and static analysis to evaluate frequency and cost of function calls**
  - **the call count**
  - **count taken branches between call and arrival in function**
  - **Get count of indirect branches invoked**
  - **Add cost for function arguments**
  - **Add a cost for push/pop of registers**

Thoughts on optimizing large OOP code bases

- **A modest proposal:**
- **Use social network analysis/network theory to identify clusters of active, costly function call activity**
  - **Web search on Social networking/social networking analysis**
- **Order clusters by total time and/or total "cost"**
  - **Split time of functions shared between clusters by call counts**
  - **Calls have a direction**
    - **Utility functions must not be viewed as bridges**

Thoughts on optimizing large OOP code bases

- A modest proposal:
- Manually reduce function count in hot clusters by explicit code inlining
  - Prioritize work by call overhead cost to be gained
  - Duplicate code as needed
  - Reduce cross shared object call counts

Using cycles to optimize the optimizations

- PEBS near call event + LBRs to get call counts/source
  - Selecting source files to compile with enhanced inlining
    - IPO can be enahnced when used with PGO
- PEBS near call event + registers (em64T) to get function arguments
  - Fix memset/memcpy calls with short lengths
  - Excessive calls to malloc/free due to constructor/destructor?
    - Identify small malloc’s/free’s
    - Let the compiler allocate small structures statically rather than malloc and free them excessively

Using cycles to optimize the optimizations

- Optimize only functions that use significant cycles
  - Reduces build time
  - Minimize fighting the compiler
    - Changing optimizations or compilers in large builds can be problematic
- Move gcc/icc and create script called gcc/icc

```
#!/bin/sh
if echo $@ | grep -f /tmp/sourcefilelist.txt > /dev/null ;
  then /opt/intel/Compiler/11.0/083/bin/intel64/icc.ori –g -fast $@;
  else gcc.ori -g -O2 $@;
fi
```

Using cycles to optimize the optimizations

- PTU sometimes shows *.h files as source
- Generate a list of c/cpp files as follows:
  - Export list of functions from Intel® PTU
  - Create script grepf.sh to grep for defined symbols:
    ```
    #!/bin/sh
    if nm --defined-only --demangle $1 | grep -f $2 >
    /dev/null ; then echo `basename $1 .o`.cpp; fi
    ```
  - Find hot object files and remember cpp files:
    ```
    find -name "*.o" -exec grepf.sh '{}'
    /tmp/functionlist.txt \; > /tmp/sourcefilelist.txt
    ```
- This will produce sourcefilelist that only includes targets of compiler

Event Classes

1. Execution flow events
   - Cycles, Branches, stalls, uops/inst_retired
2. **Penalty events**
Change code to remove the penalty
   - **Ex: load requiring access to dram**
3. Resource saturation events
   - Bandwidth, load/store buffers, dispatch ports
   - No well defined cost
4. Architectural characterization
   - Cache accesses, MESI states, snoops
5. Instruction mix

Memory Access

- Load instruction uses virtual address to access memory space
- HW translates that to physical address to access caches
  - DTLB does this
- Access is hierarchical
  - Check L1D first
  - If (miss) check if Line Fill Buffer (LFB) allocated
  - If(LFB miss) allocate LFB, escalate miss to L2
  - If(miss L2) get Super Queue (SQ) slot, escalate to uncore

Memory Access Penalties

- **Load misses cause execution stalls**
  - **In most cases store misses will not stall execution**
    - **Data to be stored is held in store buffer until desired line is in L1d, thus execution continues**
- **Loads that hit LFBs overlap in time with original line request**
  - **If the original request was a load, the original miss accounts for the entire penalty**
  - **If there are multiple load request to the LFB the least costly would be the penalty**
  - **Not all load misses are equally costly**

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 at execution
  - SMT will make Σevents$_i$*penalties$_i$ > 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: Σevents$_i$*penalties$_i$ The Elephants

- LLC, L2, and DTLB misses are the large penalty, common events
- LLC activity must be measured at L2 for it to have core, PID, TID context
  - Uncore has no ability to track core, PID or ThreadID
  - Uncore event collection not yet supported
- Figure of merit: Events*Penalty/cycles
  - Samples_ev*SAV(ev)*Penalty(ev)/ (Samples_cyc*SAV(cyc))
  - If SAV(ev) = SAV(cyc)/Penalty(ev)
  - FOM = Samples_ev/Samples_cyc
  - This is ~ how the default SAVs are set
  - Minimizes required screen area in the data display

Stall Decomposition: Σevents$_i$*penalties$_i$ The Elephants

- Figure of merit: Events*Penalty/cycles
  - Overcounts when there are temporally overlapping penalties
  - Compilers can hoist loads. So make sure there are stalls as well
    - PEBS event uops_retired.stall_cycles should pile up very close to instructions suffering large penalties
  - The combination provides the answer to the critical question:

Is the fix worth the effort?

Penalty Events: Memory Access

- Intel® Core™ i7 processor memory access events are "per source"
  - How many times cacheline came from "here"
- Unique sources have unique Penalties
  - 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.

Offcore Response Latencies

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

Note: All latencies and memory access penalties shown are merely illustrative. Actual latencies will depend on (among other things) processor model, core and uncore frequencies, type, number and positioning of DIMMS, platform model, bios version and settings. Consult the platform manufacturer for optimal setting for any individual system. Then measure the actual properties of that system by running well established benchmarks.

Memory Access PEBS Events

Identify LLC and DTLB load miss

- Precise load events do not include DCU prefetch/ L2 prefetch

Name	Penalty	Umask	Umask_name
mem_load_retired	0	0x1	L1D_HIT
	6	0x2	L2_HIT
	~35	0x4	LLC_HIT_UNSHARED*
	~75	0x8	OTHER_CORE_L2_HIT_HITM*
	depends	0x10	LLC_MISS
	depends	0x40	HIT_LFB
		0x80	DTLB_MISS*

LLC_HIT_UNSHARED should be LLC_HIT_NO_SNOOP
OTHER_CORE_L2_HIT_HITM should be LLC_HIT_SNOOP
DTLB_MISS counts primary and secondary DTLB misses on CoreI7
Only counts primary on Xeon™ 5600 Family Processors
Penalty for DTLB miss is not a constant
Also use Dtlb_load_misses.walk_cycles on Xeon™ 5600 Family Processors

Note: All latencies and memory access penalties shown are merely illustrative. Actual latencies will depend on (among other things) processor model, core and uncore frequencies, type, number and positioning of DIMMS, platform model, bios version and settings. Consult the platform manufacturer for optimal setting for any individual system. Then measure the actual properties of that system by running well established benchmarks.

Precise Uncore Response Xeon™ 5500 Family Processors

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

Name	Penalty	Umask	Umask_name
mem_uncore_retired	~85	0x2	OTHER_CORE_L2_HITM
	~185	0x8	REMOTE_CACHE_ LOCAL_HOME_HIT
	~200	0x20	LOCAL_DRAM
	~350	0x40	REMOTE_DRAM
		0x80	IO

OTHER_CORE_L2_HITM should be LOCAL_HITM

Note: All latencies and memory access penalties shown are merely illustrative. Actual latencies will depend on (among other things) processor model, core and uncore frequencies, type, number and positioning of DIMMS, platform model, bios version and settings. Consult the platform manufacturer for optimal setting for any individual system. Then measure the actual properties of that system by running well established benchmarks.

Precise Uncore Response Xeon™ 5600 Family Processors

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

Name	Penalty	Umask	Umask_name
mem_uncore_retired	~85	0x2	LOCAL_HITM
	~375	0x4	REMOTE_HITM
	~220	0x8	LOCAL_DRAM_AND_ REMOTE_CACHE_HIT
	~375	0x10	REMOTE_DRAM
		0x80	UNCACHEABLE

Note: All latencies and memory access penalties shown are merely illustrative. Actual latencies will depend on (among other things) processor model, core and uncore frequencies, type, number and positioning of DIMMS, platform model, bios version and settings. Consult the platform manufacturer for optimal setting for any individual system. Then measure the actual properties of that system by running well established benchmarks.

Precise Store DTLB miss

Name	Event	Umask	Umask_name
mem_store_retired	0x0c	0x1	DTLB_MISS*
		0x2	dropped events

DTLB_MISS counts primary and secondary DTLB misses on CoreI7
Only counts primary on Xeon™ 5600 Family Processors

Overlapping Memory access penalties Xeon 5600 family: Offcore_request_outstanding

Event Name	umask	cmask, inv
OFFCORE_REQUESTS_OUTSTANDING.ANY.READ	0x8	
OFFCORE_REQUESTS_OUTSTANDING.ANY.READ_NOT_EMPTY	0x8	1,0
OFFCORE_REQUESTS_OUTSTANDING.DEMAND.READ_CODE	0x2	
OFFCORE_REQUESTS_OUTSTANDING.DEMAND.READ_CODE_NOT_EMPTY	0x2	1,0
OFFCORE_REQUESTS_OUTSTANDING.DEMAND.READ_DATA	0x1	
OFFCORE_REQUESTS_OUTSTANDING.DEMAND.READ_DATA_NOT_EMPTY	0x1	1,0
OFFCORE_REQUESTS_OUTSTANDING.DEMAND.RFO	0x4	
OFFCORE_REQUESTS_OUTSTANDING.DEMAND.RFO_NOT_EMPTY	0x4	1,0

Offcore_requests_outstanding.demand.read_data_not_empty = cycles there is at least one request from L1d that had to be satisfied by escalation to uncore Includes L1d HW prefetch, loads and SW_prefetch

Defines upper limit of memory access penalties due to L2 miss

So what do you do?

- Load driven misses resulting in pipeline stalls can be fixed by
  - Use longest tripcount loop to drive strategy
  - Change loop order/data layout to give HW prefetcher a chance
    - Divide large structures by usage (See MILC)
    - Structures of arrays rather than arrays of structures
  - Make sure buffer initialization is consistent with usage
    - Make remote_dram misses local dram misses & cut latency in half
- DTLB misses: use large pages

So what do you do?

- Load driven misses resulting in pipeline stalls can be fixed by
- SW prefetch _mm_prefetch(addr, hint) <ia32intrin.h>
  - Use LOAD_HIT_PRE to identify when prefetch distance is too small
    - Min prefetch dist (iter) ~ 200/(uops_per_iteration/3)
      - For local dram
      - Will change as latency changes
  - long inner loop-> prefetch ahead in inner loop
  - Short inner loop-> prefetch 1,2 iterations ahead on outer
  - Reused linked list -> create indirect address array
  - #pragma openmp for (guided) will cause havoc
  - Volume 2 of that book
  - SW prefetches will not help a BW limited application

Other Penalties

- Divides and SQRT (Arith.Cycles_div_active)
  - Vectorize
  - Save reciprocals that are reused
  - Merge with bandwidth limited loops
- Store Forwarding (Load_Block.overlap_store)
  - Event only on Xeon™ 5600
  - Use Intel Compiler
  - Be careful with data type sizes (keep consistent)
- FP exceptions (uops_decoded.ms)
  - Use Intel compiler (no x87, FTZ)
  - Uninitialized values in simd registers
- No ability to measure stalls associated with chained long latency instructions
  - Sum = a+b+c+d+e…evaluated left to right

Instruction Starvation

- Lots of calls to small functions can lead to starving the pipeline of instructions
  - Only L2 prefetchers prefetch instructions
- Uops_issued.core_stall_cycles – resource_stalls.any = cycles BE wants instructions, but does not get them
  - This is more accurate with HT off
- Can be cross checked on Xeon™ 5600 processor with offcore_requests_outstanding.demand.read_code_not_empty (for L2 miss)

Decomposing instruction starvation

Event	Penalty
l2_rqsts.ifetch_hit	~6
offcore_response_0.demand_ifetch.local_cache	~35
offcore_response_0.demand_ifetch.local_dram	~200
offcore_response_0.demand_ifetch.remote_dram	~350

Ifetch miss events have among the largest IP skids of all performance events. The IP can easily have been on in a previously executing function at the time the ifetch miss occurred. See slide 23

Uncertainties are also larger, due to the many buffers in the pipeline

Instruction starvation does not occur unless the buffers drain

Note: All latencies and memory access penalties shown are merely illustrative. Actual latencies will depend on (among other things) processor model, core and uncore frequencies, type, number and positioning of DIMMS, platform model, bios version and settings. Consult the platform manufacturer for optimal setting for any individual system. Then measure the actual properties of that system by running well established benchmarks.

Instruction Access Penalties

- **Demand Ifetch: offcore_response.demand_ifetch.***
  - **Usually associated with function calls followed by taken branches in LARGE binaries**
  - **IPO, force inlining**
  - **PGO to reduce taken branches**
  - **shrink sizes of other functions**
  - **Change order of link command**
  - **Offcore_response.demand_ifetch.local_dram**
    - **Sw_prefetch(&foo(),1);   ?????**
  - **Offcore_response.demand_ifetch.remote_dram**
    - **Run 1 copy of binary per socket**
      - **Must have two complete copies on the disk**
  - **Offcore_response.demand_ifetch.llc_hit_no_other_core**
    - **Sw prefetch?, PGO, IPO**
- **ITLB misses: use large Itlb pages**

Reducing calls and *.so

- Use linker and a control list to identify internal and external functions in *.so to reduce the use of trampolines
  - icpc -Wl,-z,defs -L/External -L/Linker -Wl,-version-script,export.tmp

```
$ cat export.tmp
{
  global:
           _Foo1;
           _Foo2;
  local:
            _Bar1;
            _Bar2;
   };
```

Reducing calls and *.so

- Identifying the internal functions is not simple
- Use LBRs, and sfdump5 (see backup) to identify call chains between *.so
- Merge source files into fewer *.so
  - This will improve effectiveness of PGO/IPO
- Use global/local file of previous slide to reduce trampolines

NOTE: Author has never personally done this, so he does not know if it really works, or if the syntax is really correct.

Event Classes

1. Execution flow events
   - Cycles, Branches, stalls, uops/inst_retired
2. Penalty events
   - Ex: load requiring access to dram
3. **Resource saturation events**
   - **Bandwidth, ld/st buffers, dispatch ports**
   - **No well defined cost**
4. Architectural characterization
   - Cache accesses, MESI states, snoops
5. Instruction mix

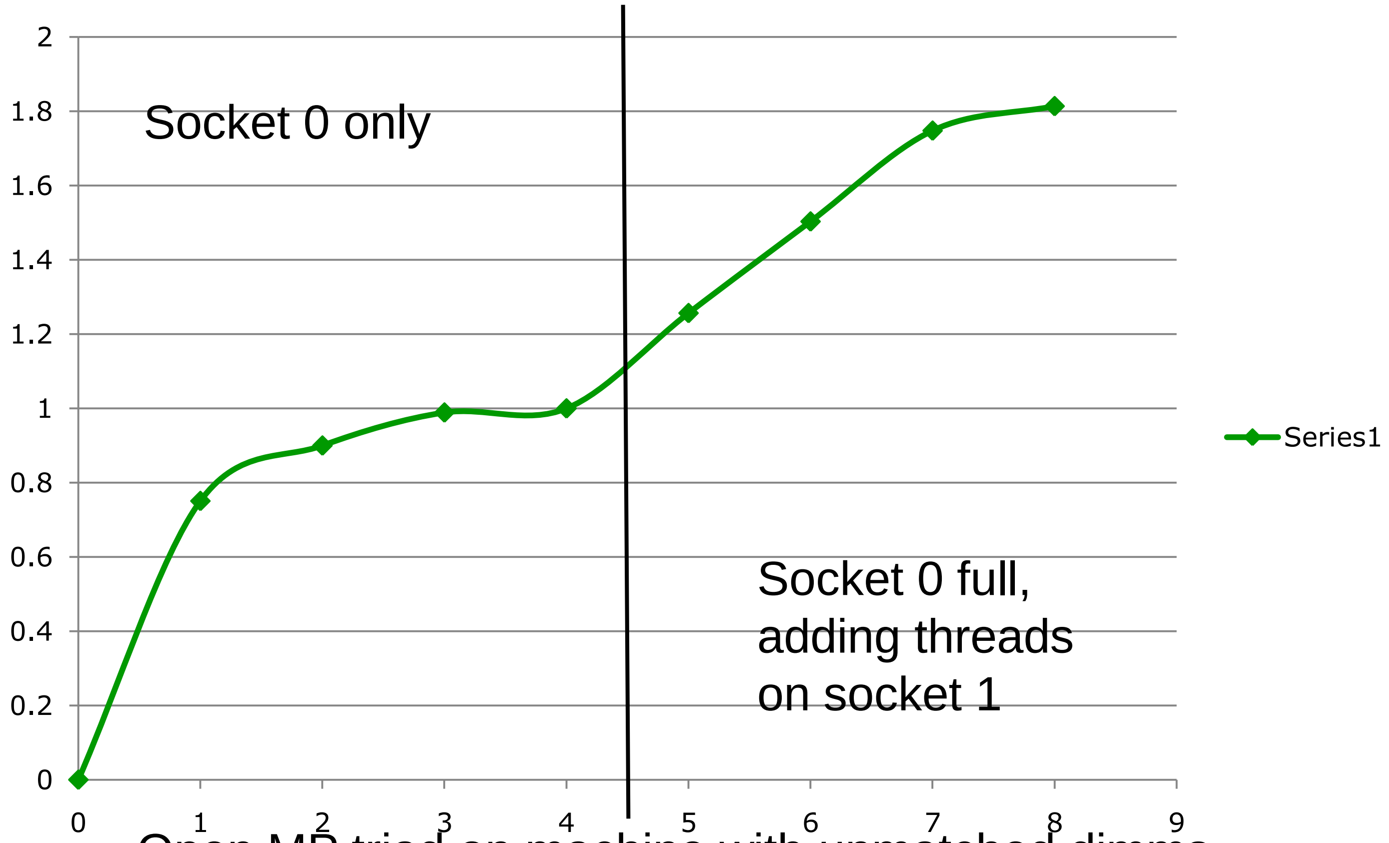
Resource Limitation Events

- Resource limitation is usually only a problem when the resource is saturated
  - There is ~no cost* for bandwidth until the bandwidth is close to saturated
    - *Latency depends weakly on BW on Corei7
- Lost cycles due to resource saturation can be hard to measure
- Only way to determine bandwidth limit is to measure it
  - Count cachelines transferred/cycle for triad
    - (w/wo SSE NT stores)
  - Depends on the number of triad threads
- Resource saturation results in no gain from HT

Resource Limitation: Memory Bandwidth

- Usually needs HW (or SW) prefetch
  - Load latencies will restrict execution otherwise
    - Exception: for(i=0;i<len;i++)a[i] = b[addr[i]];
- Limit depends on
  - number and location of concurrent threads consuming large numbers of lines
    - For asynchronous execution this becomes ~impossible to know
  - core and uncore frequencies
  - type, number, size, location of dimms
  - bios version and settings
  - Motherboard
- Measured in cycles/cacheline transferred
  - Triad with/wo RFO result in ~ same limit!
  - All "BW" events discussed here count cachelines transferred

Triad bandwidth vs thread count



Socket 0 only

Socket 0 full,
adding threads
on socket 1

Open MP triad on machine with unmatched dimms

NUMA = ON

Note: All latencies and memory access penalties shown are merely illustrative. Actual latencies will depend on (among other things) processor model, core and uncore frequencies, type, number and positioning of DIMMS, platform model, bios version and settings. Consult the platform manufacturer for optimal setting for any individual system. Then measure the actual properties of that system by running well established benchmarks.

Latency stalls vs Bandwith saturation

- A latency stalled program has a small number of outstanding data cachelines in flight simultaneously

```
i=0;
While(mystruc->next !=0){
        mystruc=mystruc->next;
        a[i] = mystruc->b_val;
        i++;
}
```

Only one (possibly 2) loads in flight at a time

- Clearly a triad with prefetchers enabled in BW limited

Gather, OOO execution and Bandwidth saturation

Consider:

```
   For(i=0;i<len;i++)A[i] = B[ADDR[i]];
```

A data collection might show something like 1000 cycle samples, 200 instruction retired samples and 5000 mem_uncore_retired.local_dram samples

The mem_uncore SAV is 10K, the cycle SAV is 2 million
This absorbs the 200 cycle penalty..so the ratio of the samples is the ratio of the cycles…

Clearly, there are more cycles in dram access than cycles executed.

Gather, OOO execution and Bandwidth saturation

In a gather loop the RS acts as a prefetcher.
There are 6 uops/iteration -> ~5 iterations in the RS?
except the loads go out immediately..
there is no dependency so the 2 loads can be executed,
the incr, cmp and branch can execute, again as there are no dependencies
so only the stores pile up
This would suggest ~30 iterations in flight at a time

the number of load buffers might be what blocks FE uop issue
there are 48 and 2/iteration are needed

The loads of ADDR[i] are sequential and thus HW prefetched.
All the stalls are on the load of B[ADDR[I]]
Thus the events fall on the next instruction.

The mem_uncore_retired.local_dram events are all overlapped..
Thus events*penalties overcounts by a huge factor

Bandwidth per core

- **Much more complicated than on Intel® Core™2 processors**
  - **Bandwidth limit depends on number of threads using maximum BW and core position of those threads**
    - **CAN ONLY BE MEASURED**
  - **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: 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 programmed in 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
  - Two versions on XEON 5600 processor family
    - Programming a little different

	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

A bit different on Xeon 5600 Processor Family

Total 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)

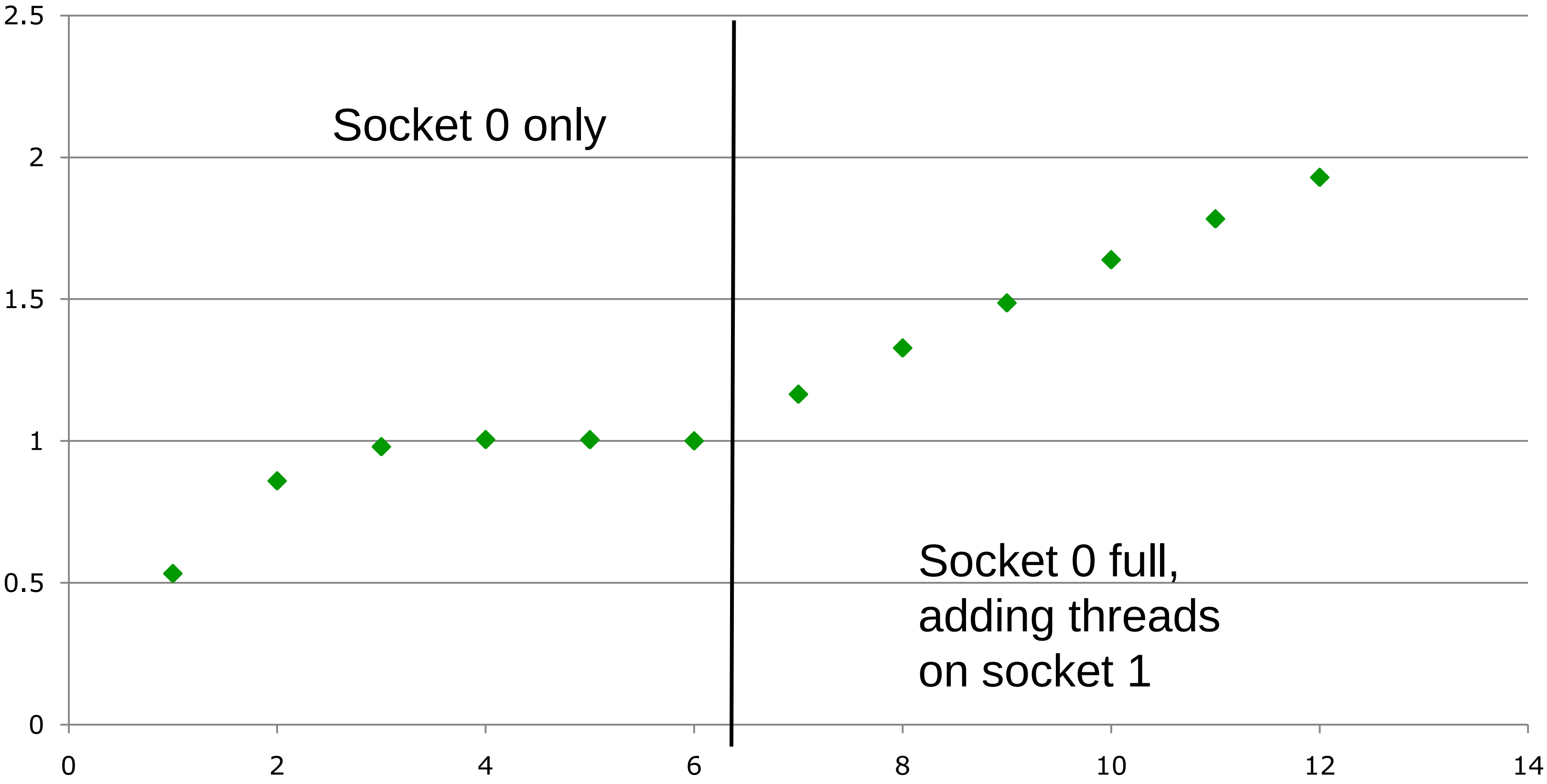
A few particularly useful events for measuring BW

- Offcore_response.data_in.local_dram
  - Read BW (per core) from local dram
- Offcore_response.data_in.remote_dram
  - Read BW (per core) from remote dram
    - Indicates NUMA locality problem
- Uncore events get totals but only in counting mode with no data/core
  - Unc_imc_normal_reads.any
    - Total read cachelines from this mem controller
  - Unc_imc_writes.full.any
    - Total written cachelines to this mem controller

Latency vs Bandwidth

- On Xeon™ 5600 processors the average occupancy of the super queue can be evaluated as
  offcore_requests_outstanding.any.reads/
  cpu_clk_unhalted.thread
- If this is large then the loop is likely BW limited
- If it is small and the event counts indicate a memory access problem due to loads then it is likely to be a latency issue

Triad bandwidth vs thread count

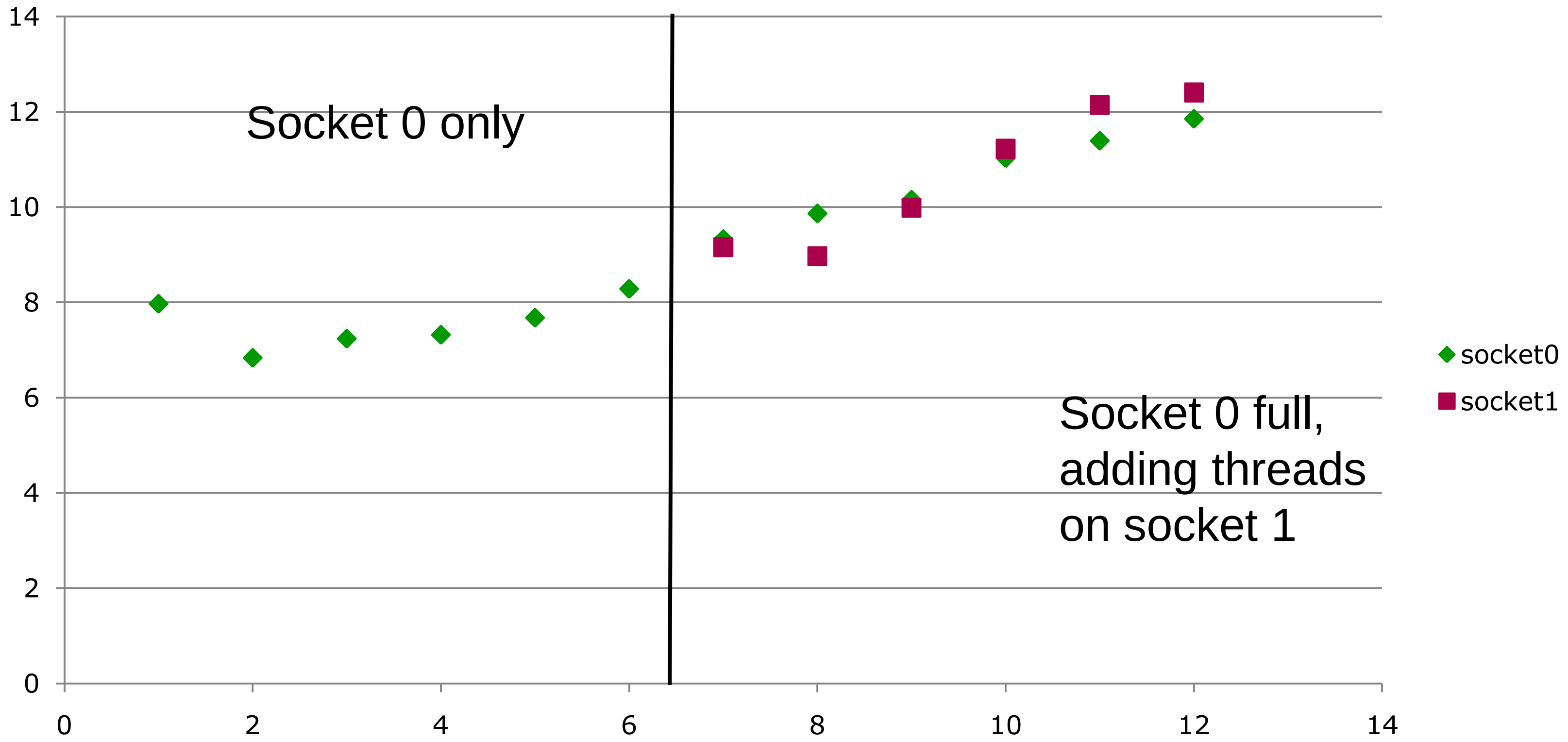


Open MP triad on machine with matched dimms

NUMA = ON

Note: All latencies and memory access penalties shown are merely illustrative. Actual latencies will depend on (among other things) processor model, core and uncore frequencies, type, number and positioning of DIMMS, platform model, bios version and settings. Consult the platform manufacturer for optimal setting for any individual system. Then measure the actual properties of that system by running well established benchmarks.

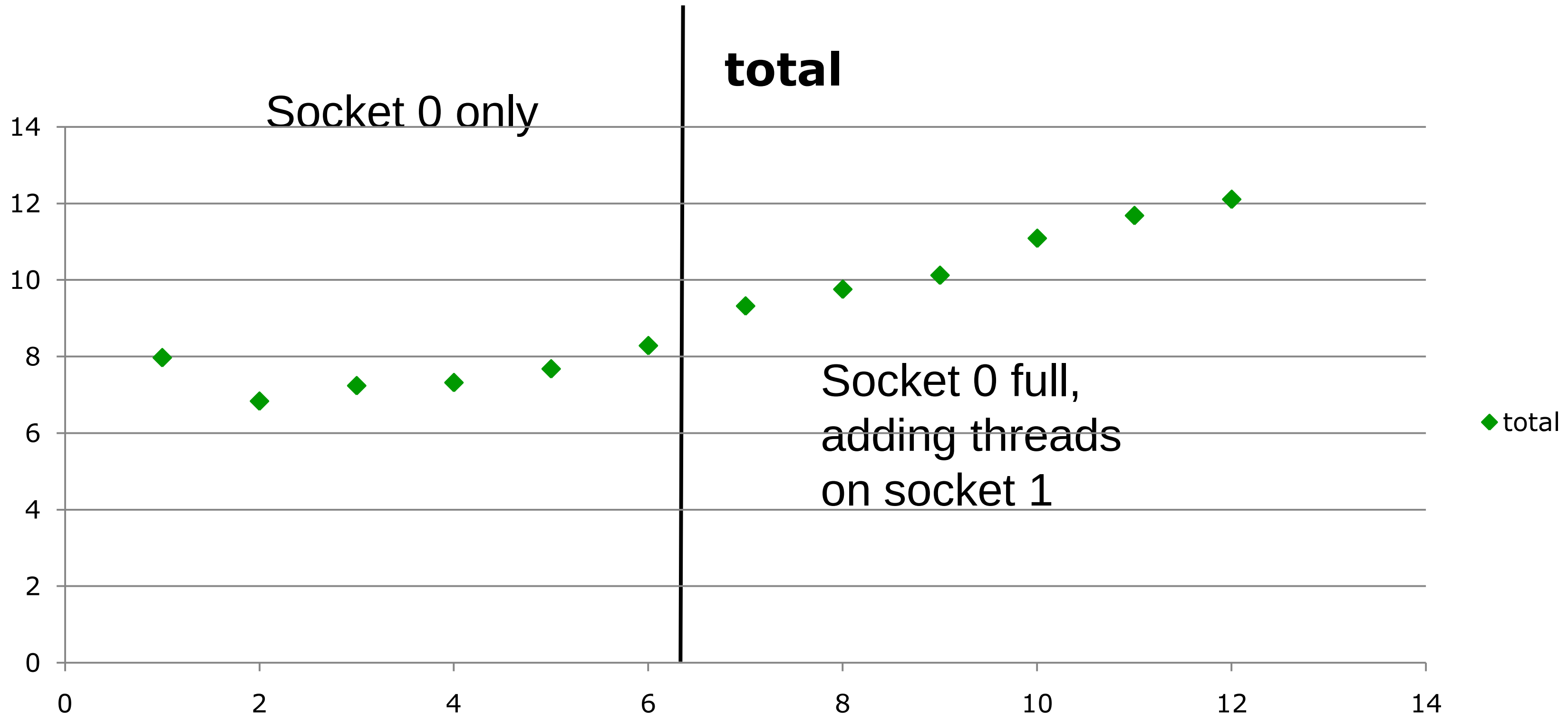
Average super queue occupancy



Variation per core is ~ 20%
Many temporally overlapping requests identifies bandwidth saturation

Note: All latencies and memory access penalties shown are merely illustrative. Actual latencies will depend on (among other things) processor model, core and uncore frequencies, type, number and positioning of DIMMS, platform model, bios version and settings. Consult the platform manufacturer for optimal setting for any individual system. Then measure the actual properties of that system by running well established benchmarks.

Average super queue occupancy



Evaluated with no knowledge of thread count

Note: All latencies and memory access penalties shown are merely illustrative. Actual latencies will depend on (among other things) processor model, core and uncore frequencies, type, number and positioning of DIMMS, platform model, bios version and settings. Consult the platform manufacturer for optimal setting for any individual system. Then measure the actual properties of that system by running well established benchmarks.

Identifying bandwidth saturation

- Identifying BW saturation by measuring bytes/time is complicated by the BW limit changing with the number of threads consuming BW (slide 90)
- Non concurrent execution, with some threads consuming large BW, while others consume little, can make identifying saturation extremely difficult

Identifying bandwidth saturation

- Average SQ occupancy limit varies less with thread count/concurrency
- It does not distinguish between LLC hits and LLC misses
- Recipe:
  - Identify problematic functions with <SQ occup>
  - Use offcore_response events to determine the fraction associated with LLC hits vs misses

But what is the potential gain?

- <u>**None of this measures what is needed!**</u>
  - **It does not tell us if the fix is worth the effort!**
- **The fix is to reduce the number of lines transferred**
  - **Consume more data per line transferred**
- **Gain**
  - **BW_time = total_lines/BW_limit**
  - **Exec_time = time to execute instructions**
    - **Memory latency of ~0**
  - **Time = MAX(BW_time, Exec_time)**
  - **Completely BW limited ~ change_in_total_lines/BW_limit**

Problem: cannot measure exec time,
BW limit is absurdly complex in general
(must assume synchronous execution)

An example

```
Double *a, *b;
For(i=0; i<len; i+=8)a[i] = sqrt(b[i]);
```

We might be able to compress a and b to transfer fewer lines

```
Double *ap, *bp;
For(i=0; i<len/8; i++)ap[i] = sqrt(bp[i]);
```

But would it actually go any faster?

No, The SQRT latency ~ matches the BW limit

Estimating the gain

- Exec time ~ uops_retired.slots/`*3*’+ arith.cycles_div_active
  - Undercounts cycles associated with chained long latency uops
- Optimized BW time = Adjusted_lines/Max_bw
- Gain ~ Cpu_clk_unhalted.thread – MAX(Optimized BW time, Exec Time)
- Many Uncertainties, but better than nothing
  - Assumptions about concurrency of high BW usage
  - Assumptions about cycles associated with chained long latency uops
  - Is uops/3 realistic?

What do you do about Bandwidth?

- Data layout change is usually best
  - Fix buffer initialization to make remote_dram small
  - Fix order of structure elements (big to small)
  - Eliminate unused structure elements
    - Divide structures into parallel structures by use
  - Measure data consumed/cacheline in
    - Sum load/store in loops (ignore stack pointer, +=)
    - Multiply by total tripcount & divide by 64*offcore_response.data_in.local_dram
  - Fix nested loop order
- Measure data_in with prefetchers on & off
  - If difference is large
    - Change data layout to help HW prefetcher or
    - Consider sw prefetching everything and disabling HW prefetchers

OOO resource Saturation

- Load buffer saturation (resource_stalls.ld)
  - In HPC, frequently due to bandwidth saturation
- Store buffer saturation (resource_stalls.st)
  - This will cause stores to stop the pipeline
  - Usually associated with stores missing l1d/l2 etc
    - SW prefetch, change layout to help HW prefetch
- Port saturation (uops_executed.portX/cycles)
  - Most common for load port (2)
    - Avoid loop distribution (F90)
    - Merge loops to reuse data while available
    - Align data and vectorize

Less than ideal multi core scaling

- Perfect scaling results in the number of perf events (summed over cores) being constant
- Difference of event counts can identify locality using cycles and some reasons for non scaling behavior
  - Cacheline access contention can cause non scaling
    - Load-hitm and store address analysis identifies this
- Most non scaling due to resource saturation and evaluated as a ratio: events/wall_cycles
    - Wall_cycles ~ cycles/active cores
      or Cpu_clk_unhalted.thread max(ICPU)
  - **Cannot be seen in difference display**

Sources/signatures of non scaling

- Turbo
  - Having this on results in large drop from 1->2
- Smaller share of LLC
  - Decrease in LLC hits, increase in LLC miss
- Increase in page faults
  - More threads require more memory
- Asymmetry associated with core 0
  - OS induced imbalance
- Context switching
  - OS’s love to move things around, being the boss!
  - Don’t know about logical cores & double up on one physical core, while other phys cores are idle

Sources/signatures of non scaling

- Saturating a resource
  - Ex: Bandwidth
  - Code optimization increases resource saturation
- Shared memory application specific
  - Serial execution
  - Overly contested lock access
  - False sharing (non overlapping access to a line)
- NUMA based non scaling
  - Increase in *.remote_dram
- HT can be viewed as a way to recover scaling

More sources of non scaling

- Load imbalance
  - Increase in halted cycles
- MPI global operations
  - increase in time associates with MPI global APIs
    - Ex: allreduce
- Synchronous message passing
  - “Intrinsically” non scaling

Resolving non scaling issues

- Disable turbo while doing measurements
- Disable HT while doing measurements
- Pin all affinities
  - OS’s love to move things
  - Old OS’s will schedule 2 threads on a physical core while leaving other physical cores idle. This increases with thread count
- Make sure there is enough memory
  - /proc/meminfo->Active (?)
- Do 1 thread baseline on a core other than 0
- Increased LLC miss
  - Usual approaches to fixing these, see previous

Resolving non scaling issues

- Bandwidth issues
  - Check data decomposition for sepparation
  - Improve data layout to reduce cacheline usage
  - See previous section on BW issues
- Excessive lock contention
  - Use finer grained locking
  - Use faster locking APIs
  - Make sure the global update is really needed
    - Can you continue working with local copy
- False sharing
  - Put 64 bytes between data elements

Resolving non scaling issues

- NUMA related non scaling
  - Remote dram data access
    - Improve buffer initialization for local access
    - Make multiple copies for each socket
  - Remote dram ifetch access
    - Make two binaries on the disk and affinity pin per socket
- MPI global operations
  - Use openMP within a box to reduce MPI nodes
  - Use good MPI library

Resolving non scaling issues

- Load imbalance
  - Seen as halted cycles
      - TSC difference for successive cpu_clk_unhalted.ref != SAV
  - Work queue approach dynamically restores balance
    - At a cost
      - NUMA locality can be lost
      - SW prefetching can become unpredictable within a thread
  - Estimate work during data decomposition to create balanced work rather than balanced iteration count
  - Save some iterations for final work queue balancing

Graphical tool needed to organize data viewing

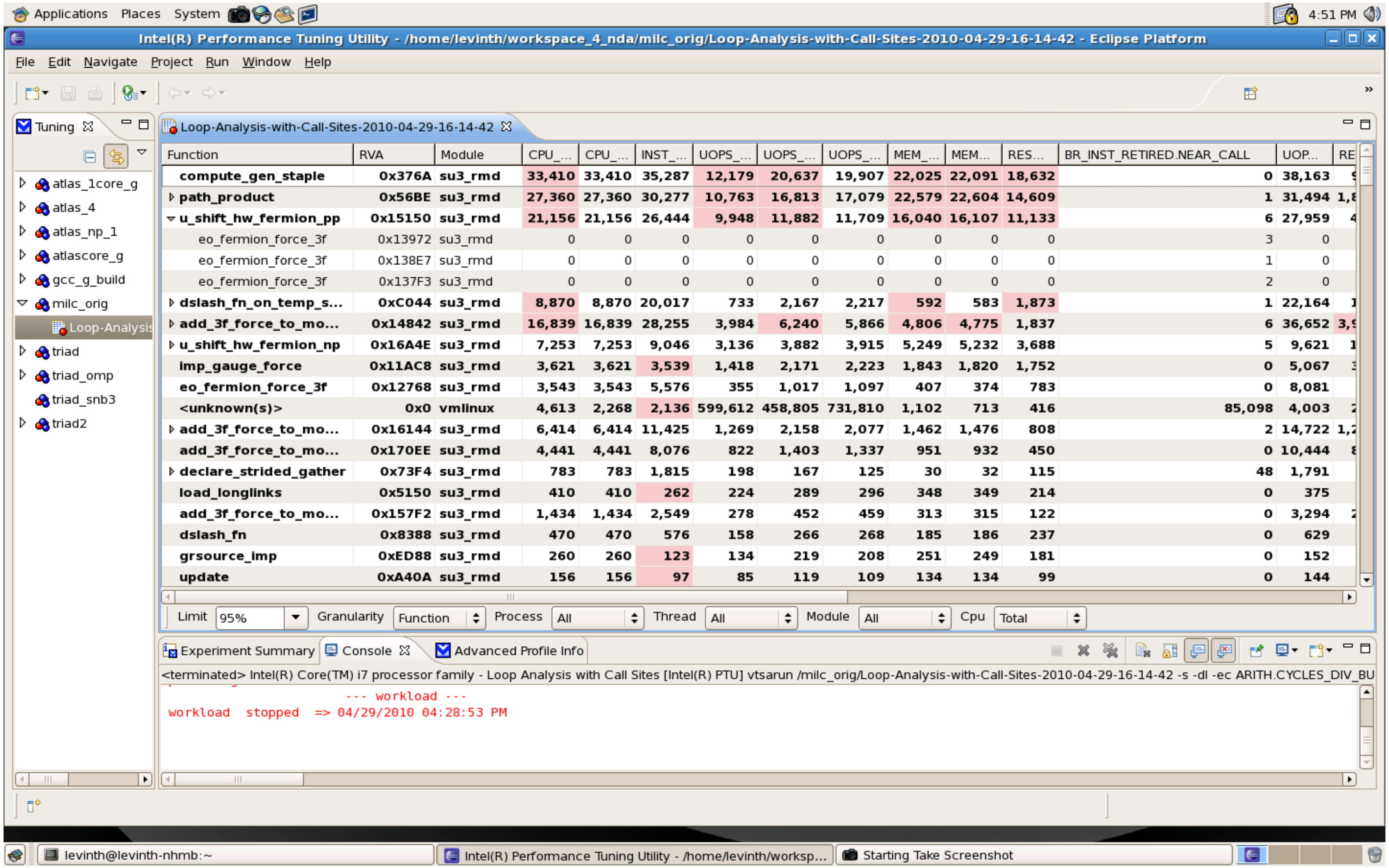
- Workflow of event based performance analysis is extremely complicated
  - Requires an enormous number of features/options to enable all possible tasks
  - Automation is very difficult
- To do a lot of things requires a lot of options
  - Many docking windows, menus, buttons
  - Easier to make a tool for a knowledgable user
- The data collection is the easy part

Interpreting the data and determining the correct action is the hard part

Tool Requirements

- Maximize data density
  - Required quantity of data is enormous
- Integrated source/asm display
- Ability to restart sessions later
- Difference utility to monitor changes
- Minimize mouse clicks
- Predefined event lists
- Predefined penalty file
  - Cycle accounting
  - dynamic column layout

Primary display shows offending events and even call counts

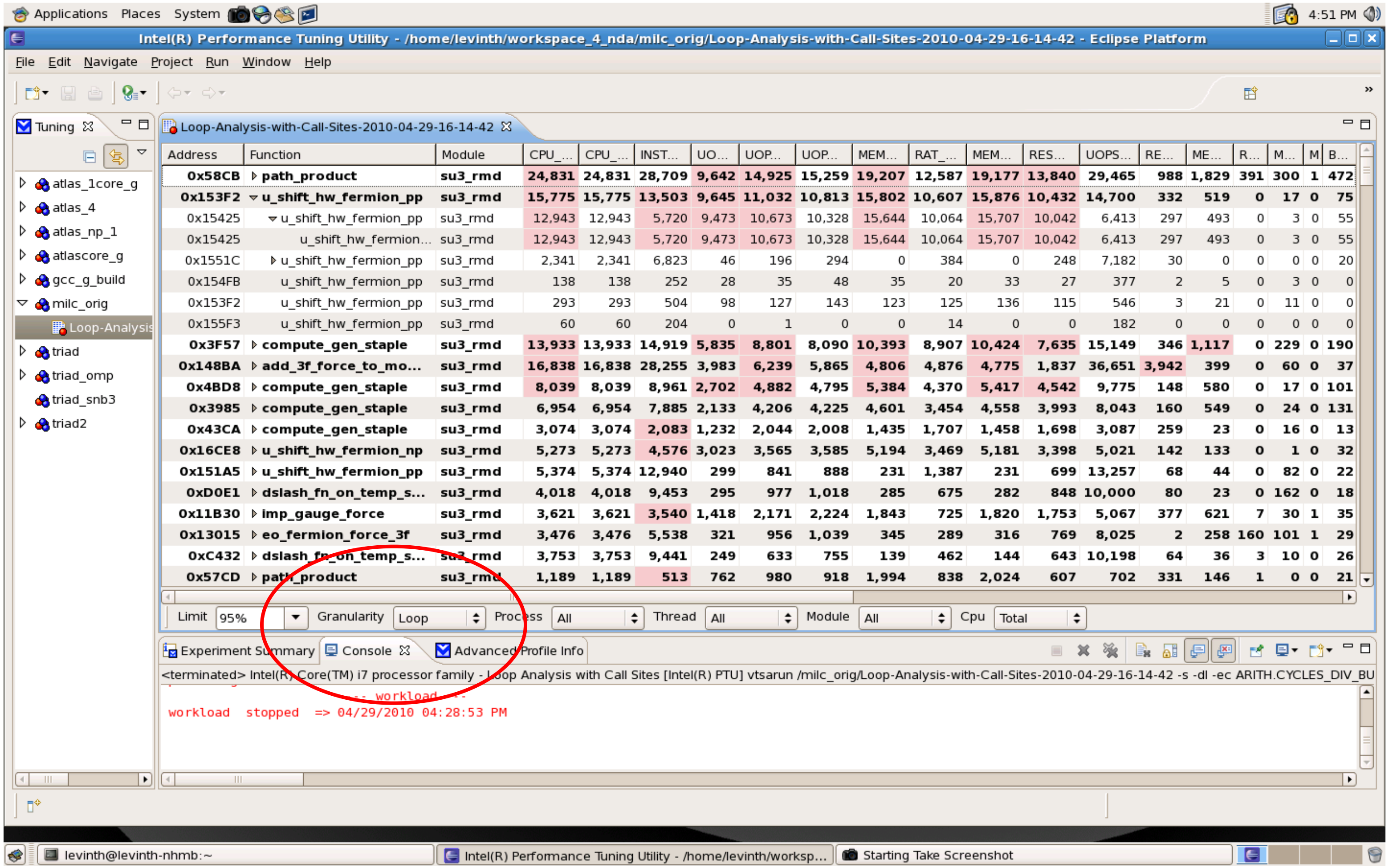

Applications Places System
4:51 PM
Intel(R) Performance Tuning Utility - /home/levinth/workspace_4_nda/milc_orig/Loop-Analysis-with-Call-Sites-2010-04-29-16-14-42 - Eclipse Platform
File Edit Navigate Project Run Window Help
Tuning
atlas_1core_g
atlas_4
atlas_np_1
atlascore_g
gcc_g_build
milc_orig
Loop-Analysis
triad
triad_omp
triad_snb3
triad2
Loop-Analysis-with-Call-Sites-2010-04-29-16-14-42
Experiment Summary
Console
Advanced Profile Info
levinth@levinth-nhmb:~
Intel(R) Performance Tuning Utility - /home/levinth/worksp...
Starting Take Screenshot

Function	RVA	Module	CPU_...	CPU_...	INST_...	UOPS_...	UOPS_...	UOPS_...	MEM_...	MEM...	RES...	BR_INST_RETIRED.NEAR_CALL	UOP...	RE
compute_gen_staple	0x376A	su3_rmd	33,410	33,410	35,287	12,179	20,637	19,907	22,025	22,091	18,632	0	38,163	9
▷ path_product	0x56BE	su3_rmd	27,360	27,360	30,277	10,763	16,813	17,079	22,579	22,604	14,609	1	31,494	1,8
▽ u_shift_hw_fermion_pp	0x15150	su3_rmd	21,156	21,156	26,444	9,948	11,882	11,709	16,040	16,107	11,133	6	27,959	4
eo_fermion_force_3f	0x13972	su3_rmd	0	0	0	0	0	0	0	0	0	3	0	
eo_fermion_force_3f	0x138E7	su3_rmd	0	0	0	0	0	0	0	0	0	1	0	
eo_fermion_force_3f	0x137F3	su3_rmd	0	0	0	0	0	0	0	0	0	2	0	
▷ dslash_fn_on_temp_s...	0xC044	su3_rmd	8,870	8,870	20,017	733	2,167	2,217	592	583	1,873	1	22,164	1
▷ add_3f_force_to_mo...	0x14842	su3_rmd	16,839	16,839	28,255	3,984	6,240	5,866	4,806	4,775	1,837	6	36,652	3,9
▷ u_shift_hw_fermion_np	0x16A4E	su3_rmd	7,253	7,253	9,046	3,136	3,882	3,915	5,249	5,232	3,688	5	9,621	1
imp_gauge_force	0x11AC8	su3_rmd	3,621	3,621	3,539	1,418	2,171	2,223	1,843	1,820	1,752	0	5,067	3
eo_fermion_force_3f	0x12768	su3_rmd	3,543	3,543	5,576	355	1,017	1,097	407	374	783	0	8,081	
<unknown(s)>	0x0	vmlinux	4,613	2,268	2,136	599,612	458,805	731,810	1,102	713	416	85,098	4,003	2
▷ add_3f_force_to_mo...	0x16144	su3_rmd	6,414	6,414	11,425	1,269	2,158	2,077	1,462	1,476	808	2	14,722	1,2
add_3f_force_to_mo...	0x170EE	su3_rmd	4,441	4,441	8,076	822	1,403	1,337	951	932	450	0	10,444	8
▷ declare_strided_gather	0x73F4	su3_rmd	783	783	1,815	198	167	125	30	32	115	48	1,791	
load_longlinks	0x5150	su3_rmd	410	410	262	224	289	296	348	349	214	0	375	
add_3f_force_to_mo...	0x157F2	su3_rmd	1,434	1,434	2,549	278	452	459	313	315	122	0	3,294	2
dslash_fn	0x8388	su3_rmd	470	470	576	158	266	268	185	186	237	0	629	
grsource_imp	0xED88	su3_rmd	260	260	123	134	219	208	251	249	181	0	152	
update	0xA40A	su3_rmd	156	156	97	85	119	109	134	134	99	0	144	

Limit 95% Granularity Function Process All Thread All Module All Cpu Total

```
<terminated> Intel(R) Core(TM) i7 processor family - Loop Analysis with Call Sites [Intel(R) PTU] vtsarun /milc_orig/Loop-Analysis-with-Call-Sites-2010-04-29-16-14-42 -s -dl -ec ARITH.CYCLES_DIV_BU
                                  --- workload ---
 workload  stopped  => 04/29/2010 04:28:53 PM
```

Set the Granularity to LOOPS

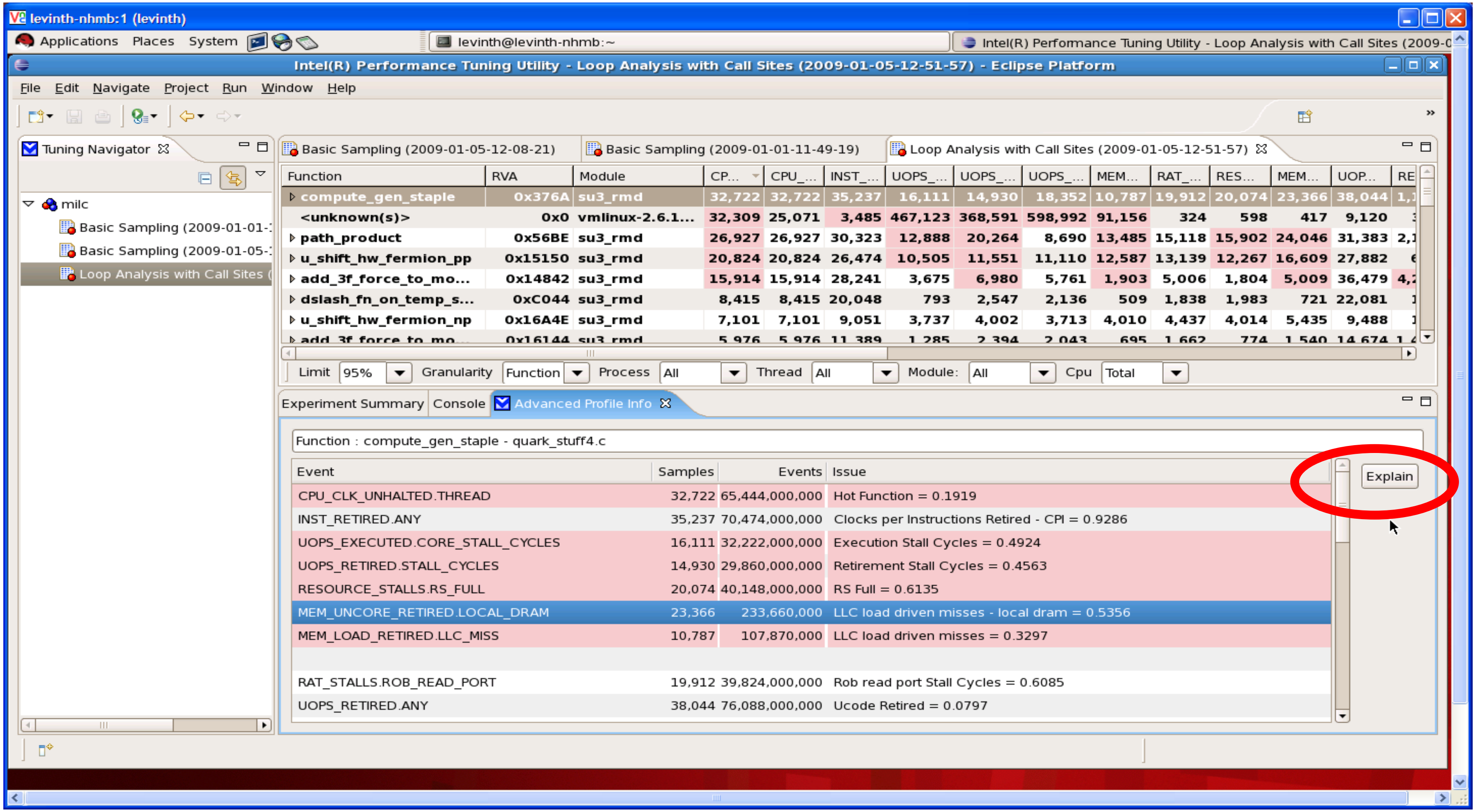

Applications Places System
4:51 PM
Intel(R) Performance Tuning Utility - /home/levinth/workspace_4_nda/milc_orig/Loop-Analysis-with-Call-Sites-2010-04-29-16-14-42 - Eclipse Platform
File Edit Navigate Project Run Window Help
Tuning
atlas_1core_g
atlas_4
atlas_np_1
atlascore_g
gcc_g_build
milc_orig
Loop-Analysis
triad
triad_omp
triad_snb3
triad2
Loop-Analysis-with-Call-Sites-2010-04-29-16-14-42
Experiment Summary
Console
Advanced Profile Info
levinth@levinth-nhmb:~
Intel(R) Performance Tuning Utility - /home/levinth/worksp...
Starting Take Screenshot

Address	Function	Module	CPU_...	CPU_...	INST...	UO...	UOP...	UOP...	MEM...	RAT_...	MEM...	RES...	UOPS...	RE...	ME...	R...	M...	M	B...
0x58CB	▷ **path_product**	**su3_rmd**	**24,831**	**24,831**	**28,709**	**9,642**	**14,925**	**15,259**	**19,207**	**12,587**	**19,177**	**13,840**	**29,465**	**988**	**1,829**	**391**	**300**	**1**	**472**
0x153F2	▽ **u_shift_hw_fermion_pp**	**su3_rmd**	**15,775**	**15,775**	**13,503**	**9,645**	**11,032**	**10,813**	**15,802**	**10,607**	**15,876**	**10,432**	**14,700**	**332**	**519**	**0**	**17**	**0**	**75**
0x15425	▽ u_shift_hw_fermion_pp	su3_rmd	12,943	12,943	5,720	9,473	10,673	10,328	15,644	10,064	15,707	10,042	6,413	297	493	0	3	0	55
0x15425	u_shift_hw_fermion...	su3_rmd	12,943	12,943	5,720	9,473	10,673	10,328	15,644	10,064	15,707	10,042	6,413	297	493	0	3	0	55
0x1551C	▷ u_shift_hw_fermion_pp	su3_rmd	2,341	2,341	6,823	46	196	294	0	384	0	248	7,182	30	0	0	0	0	20
0x154FB	u_shift_hw_fermion_pp	su3_rmd	138	138	252	28	35	48	35	20	33	27	377	2	5	0	3	0	0
0x153F2	u_shift_hw_fermion_pp	su3_rmd	293	293	504	98	127	143	123	125	136	115	546	3	21	0	11	0	0
0x155F3	u_shift_hw_fermion_pp	su3_rmd	60	60	204	0	1	0	0	14	0	0	182	0	0	0	0	0	0
0x3F57	▷ **compute_gen_staple**	**su3_rmd**	**13,933**	**13,933**	**14,919**	**5,835**	**8,801**	**8,090**	**10,393**	**8,907**	**10,424**	**7,635**	**15,149**	**346**	**1,117**	**0**	**229**	**0**	**190**
0x148BA	▷ **add_3f_force_to_mo...**	**su3_rmd**	**16,838**	**16,838**	**28,255**	**3,983**	**6,239**	**5,865**	**4,806**	**4,876**	**4,775**	**1,837**	**36,651**	**3,942**	**399**	**0**	**60**	**0**	**37**
0x4BD8	▷ **compute_gen_staple**	**su3_rmd**	**8,039**	**8,039**	**8,961**	**2,702**	**4,882**	**4,795**	**5,384**	**4,370**	**5,417**	**4,542**	**9,775**	**148**	**580**	**0**	**17**	**0**	**101**
0x3985	▷ **compute_gen_staple**	**su3_rmd**	**6,954**	**6,954**	**7,885**	**2,133**	**4,206**	**4,225**	**4,601**	**3,454**	**4,558**	**3,993**	**8,043**	**160**	**549**	**0**	**24**	**0**	**131**
0x43CA	▷ **compute_gen_staple**	**su3_rmd**	**3,074**	**3,074**	**2,083**	**1,232**	**2,044**	**2,008**	**1,435**	**1,707**	**1,458**	**1,698**	**3,087**	**259**	**23**	**0**	**16**	**0**	**13**
0x16CE8	▷ **u_shift_hw_fermion_np**	**su3_rmd**	**5,273**	**5,273**	**4,576**	**3,023**	**3,565**	**3,585**	**5,194**	**3,469**	**5,181**	**3,398**	**5,021**	**142**	**133**	**0**	**1**	**0**	**32**
0x151A5	▷ **u_shift_hw_fermion_pp**	**su3_rmd**	**5,374**	**5,374**	**12,940**	**299**	**841**	**888**	**231**	**1,387**	**231**	**699**	**13,257**	**68**	**44**	**0**	**82**	**0**	**22**
0xD0E1	▷ **dslash_fn_on_temp_s...**	**su3_rmd**	**4,018**	**4,018**	**9,453**	**295**	**977**	**1,018**	**285**	**675**	**282**	**848**	**10,000**	**80**	**23**	**0**	**162**	**0**	**18**
0x11B30	▷ **imp_gauge_force**	**su3_rmd**	**3,621**	**3,621**	**3,540**	**1,418**	**2,171**	**2,224**	**1,843**	**725**	**1,820**	**1,753**	**5,067**	**377**	**621**	**7**	**30**	**1**	**35**
0x13015	▷ **eo_fermion_force_3f**	**su3_rmd**	**3,476**	**3,476**	**5,538**	**321**	**956**	**1,039**	**345**	**289**	**316**	**769**	**8,025**	**2**	**258**	**160**	**101**	**1**	**29**
0xC432	▷ **dslash_fn_on_temp_s...**	**su3_rmd**	**3,753**	**3,753**	**9,441**	**249**	**633**	**755**	**139**	**462**	**144**	**643**	**10,198**	**64**	**36**	**3**	**10**	**0**	**26**
0x57CD	▷ **path_product**	**su3_rmd**	**1,189**	**1,189**	**513**	**762**	**980**	**918**	**1,994**	**838**	**2,024**	**607**	**702**	**331**	**146**	**1**	**0**	**0**	**21**

Limit: 95% Granularity: Loop Process: All Thread: All Module: All Cpu: Total

```
<terminated> Intel(R) Core(TM) i7 processor family - Loop Analysis with Call Sites [Intel(R) PTU] vtsarun /milc_orig/Loop-Analysis-with-Call-Sites-2010-04-29-16-14-42 -s -dl -ec ARITH.CYCLES_DIV_BU
                          --- workload ---
 workload  stopped  => 04/29/2010 04:28:53 PM
```

Get Tuning Advice for the Selected Event/Ratio: Highlighting the Event Row Enables Explanation



levinth-nhmb:1 (levinth)

Applications Places System

levinth@levinth-nhmb:~

Intel(R) Performance Tuning Utility - Loop Analysis with Call Sites (2009-01-05-12-51-57) - Eclipse Platform

Tuning Navigator

- milc
  - Basic Sampling (2009-01-01-1
  - Basic Sampling (2009-01-05-1
  - Loop Analysis with Call Sites (

Basic Sampling (2009-01-05-12-08-21) | Basic Sampling (2009-01-01-11-49-19) | Loop Analysis with Call Sites (2009-01-05-12-51-57)

Function	RVA	Module	CP...	CPU_...	INST_...	UOPS_...	UOPS_...	UOPS_...	MEM...	RAT_...	RES...	MEM...	UOP...	RE
compute_gen_staple	0x376A	su3_rmd	32,722	32,722	35,237	16,111	14,930	18,352	10,787	19,912	20,074	23,366	38,044	1,1
<unknown(s)>	0x0	vmlinux-2.6.1...	32,309	25,071	3,485	467,123	368,591	598,992	91,156	324	598	417	9,120	3
path_product	0x56BE	su3_rmd	26,927	26,927	30,323	12,888	20,264	8,690	13,485	15,118	15,902	24,046	31,383	2,1
u_shift_hw_fermion_pp	0x15150	su3_rmd	20,824	20,824	26,474	10,505	11,551	11,110	12,587	13,139	12,267	16,609	27,882	6
add_3f_force_to_mo...	0x14842	su3_rmd	15,914	15,914	28,241	3,675	6,980	5,761	1,903	5,006	1,804	5,009	36,479	4,2
dslash_fn_on_temp_s...	0xC044	su3_rmd	8,415	8,415	20,048	793	2,547	2,136	509	1,838	1,983	721	22,081	1
u_shift_hw_fermion_np	0x16A4E	su3_rmd	7,101	7,101	9,051	3,737	4,002	3,713	4,010	4,437	4,014	5,435	9,488	1
add_3f_force_to_mo...	0x16144	su3_rmd	5,976	5,976	11,389	1,285	2,394	2,043	695	1,662	774	1,540	14,674	1,4

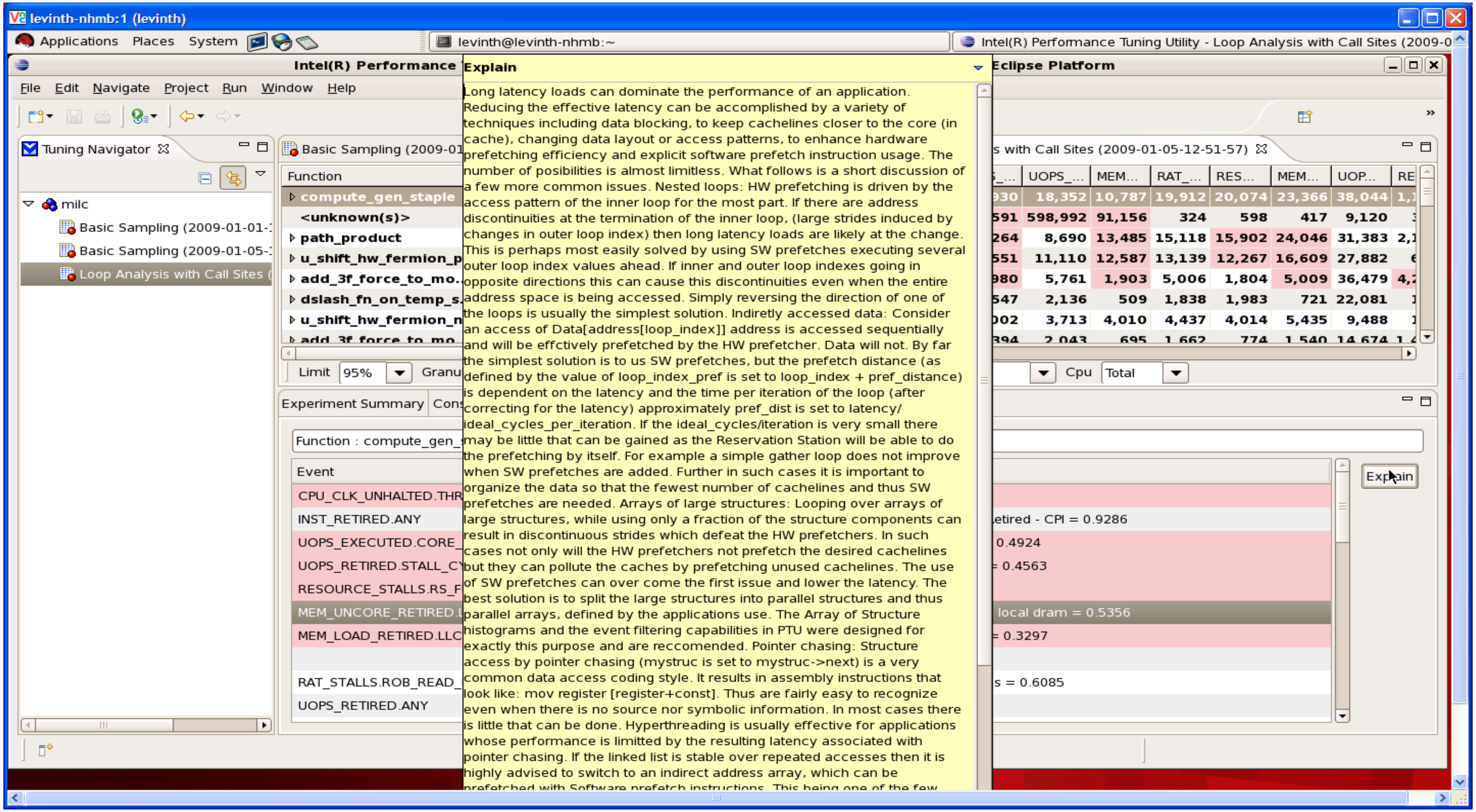
Limit: 95% Granularity: Function Process: All Thread: All Module: All Cpu: Total

Experiment Summary | Console | Advanced Profile Info

Function : compute_gen_staple - quark_stuff4.c

Event	Samples	Events	Issue
CPU_CLK_UNHALTED.THREAD	32,722	65,444,000,000	Hot Function = 0.1919
INST_RETIRED.ANY	35,237	70,474,000,000	Clocks per Instructions Retired - CPI = 0.9286
UOPS_EXECUTED.CORE_STALL_CYCLES	16,111	32,222,000,000	Execution Stall Cycles = 0.4924
UOPS_RETIRED.STALL_CYCLES	14,930	29,860,000,000	Retirement Stall Cycles = 0.4563
RESOURCE_STALLS.RS_FULL	20,074	40,148,000,000	RS Full = 0.6135
MEM_UNCORE_RETIRED.LOCAL_DRAM	23,366	233,660,000	LLC load driven misses - local dram = 0.5356
MEM_LOAD_RETIRED.LLC_MISS	10,787	107,870,000	LLC load driven misses = 0.3297
RAT_STALLS.ROB_READ_PORT	19,912	39,824,000,000	Rob read port Stall Cycles = 0.6085
UOPS_RETIRED.ANY	38,044	76,088,000,000	Ucode Retired = 0.0797

Get Tuning Advice for the Selected Event/Ratio: Highlighting the Event Row Enables Explanation



Function	…_…	UOPS_…	MEM…	RAT_…	RES…	MEM…	UOP…	RE
compute_gen_staple	930	18,352	10,787	19,912	20,074	23,366	38,044	1,1
<unknown(s)>	591	598,992	91,156	324	598	417	9,120	3
path_product	264	8,690	13,485	15,118	15,902	24,046	31,383	2,1
u_shift_hw_fermion_p	551	11,110	12,587	13,139	12,267	16,609	27,882	6
add_3f_force_to_mo..	980	5,761	1,903	5,006	1,804	5,009	36,479	4,2
dslash_fn_on_temp_s.	547	2,136	509	1,838	1,983	721	22,081	1
u_shift_hw_fermion_n	002	3,713	4,010	4,437	4,014	5,435	9,488	1
add_3f_force_to_mo	394	2,043	695	1,662	774	1,540	14,674	1,4

Function : compute_gen_

Event	
CPU_CLK_UNHALTED.THR	
INST_RETIRED.ANY	etired - CPI = 0.9286
UOPS_EXECUTED.CORE_	0.4924
UOPS_RETIRED.STALL_CY	= 0.4563
RESOURCE_STALLS.RS_F	
MEM_UNCORE_RETIRED.L	local dram = 0.5356
MEM_LOAD_RETIRED.LLC	= 0.3297
RAT_STALLS.ROB_READ_	s = 0.6085
UOPS_RETIRED.ANY	

Explain

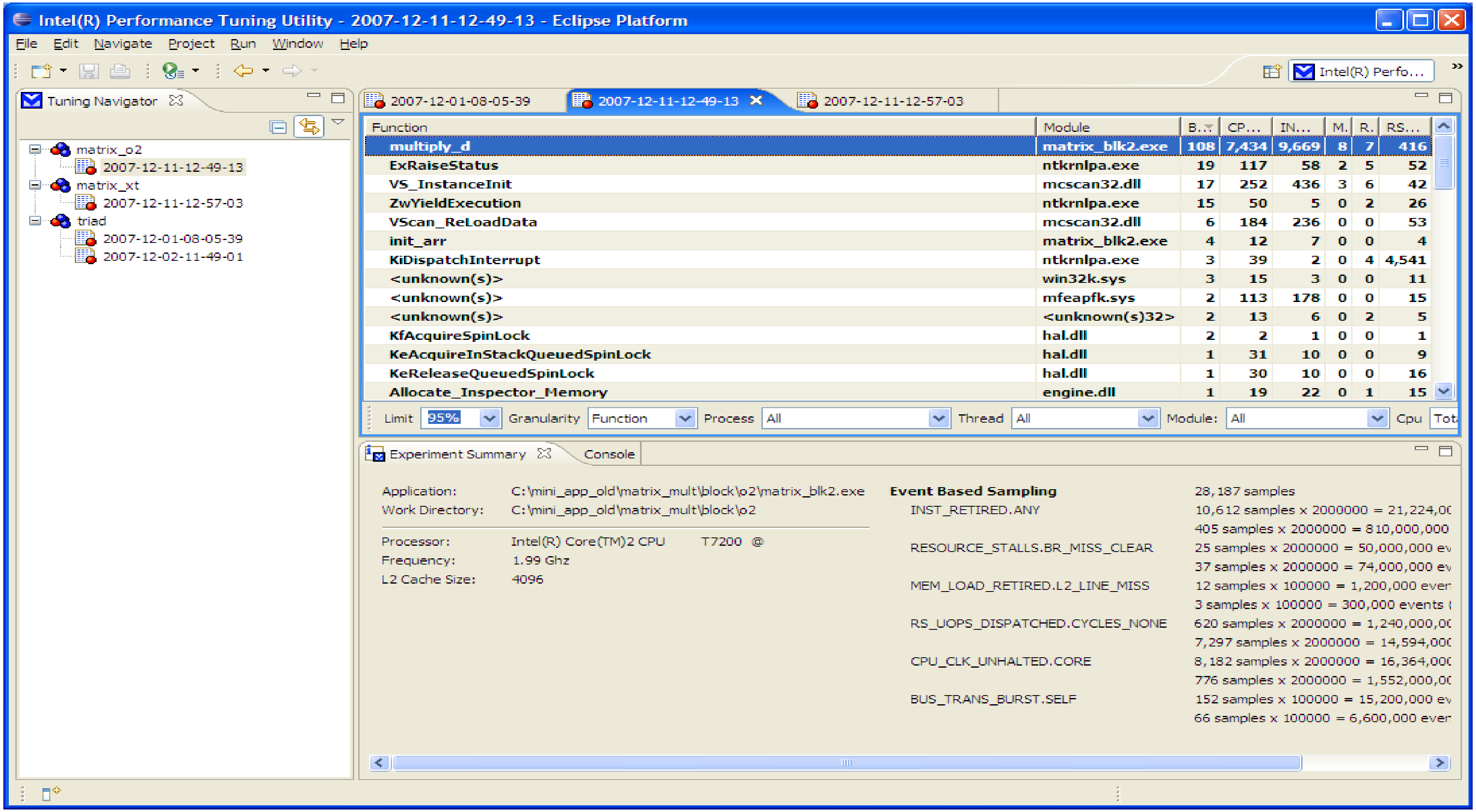
Long latency loads can dominate the performance of an application. Reducing the effective latency can be accomplished by a variety of techniques including data blocking, to keep cachelines closer to the core (in cache), changing data layout or access patterns, to enhance hardware prefetching efficiency and explicit software prefetch instruction usage. The number of posibilities is almost limitless. What follows is a short discussion of a few more common issues. Nested loops: HW prefetching is driven by the access pattern of the inner loop for the most part. If there are address discontinuities at the termination of the inner loop, (large strides induced by changes in outer loop index) then long latency loads are likely at the change. This is perhaps most easily solved by using SW prefetches executing several outer loop index values ahead. If inner and outer loop indexes going in opposite directions this can cause this discontinuities even when the entire address space is being accessed. Simply reversing the direction of one of the loops is usually the simplest solution. Indiretly accessed data: Consider an access of Data[address[loop_index]] address is accessed sequentially and will be effctively prefetched by the HW prefetcher. Data will not. By far the simplest solution is to us SW prefetches, but the prefetch distance (as defined by the value of loop_index_pref is set to loop_index + pref_distance) is dependent on the latency and the time per iteration of the loop (after correcting for the latency) approximately pref_dist is set to latency/ideal_cycles_per_iteration. If the ideal_cycles/iteration is very small there may be little that can be gained as the Reservation Station will be able to do the prefetching by itself. For example a simple gather loop does not improve when SW prefetches are added. Further in such cases it is important to organize the data so that the fewest number of cachelines and thus SW prefetches are needed. Arrays of large structures: Looping over arrays of large structures, while using only a fraction of the structure components can result in discontinuous strides which defeat the HW prefetchers. In such cases not only will the HW prefetchers not prefetch the desired cachelines but they can pollute the caches by prefetching unused cachelines. The use of SW prefetches can over come the first issue and lower the latency. The best solution is to split the large structures into parallel structures and thus parallel arrays, defined by the applications use. The Array of Structure histograms and the event filtering capabilities in PTU were designed for exactly this purpose and are reccomended. Pointer chasing: Structure access by pointer chasing (mystruc is set to mystruc->next) is a very common data access coding style. It results in assembly instructions that look like: mov register [register+const]. Thus are fairly easy to recognize even when there is no source nor symbolic information. In most cases there is little that can be done. Hyperthreading is usually effective for applications whose performance is limitted by the resulting latency associated with pointer chasing. If the linked list is stable over repeated accesses then it is highly advised to switch to an indirect address array, which can be prefetched with Software prefetch instructions. This being one of the few

Differences of EBS Measurments

- Intel® PTU supports an analysis of differences of experiments
- This requires
  - Event names must be the ~same
  - Load Modules have the same names
    - They can be the same, with data taken on different machines
    - They can be different but built from the same source
      - Allowing differences to be analyzed down to source view
    - They can be completely different (sources and binaries)
      - PTU will compare functions with the same names for modules with the same names
- Identify compiler differences/regressions
- Multi core scaling

**For perfect scaling and identical work,
total event counts, summed over cores,
will be equal**

Data blocked 2X2 unrolled Matrix Multiply compiled at -O2   (Binary = o2\matrix_blk2.exe) Cycle_Usage Profile

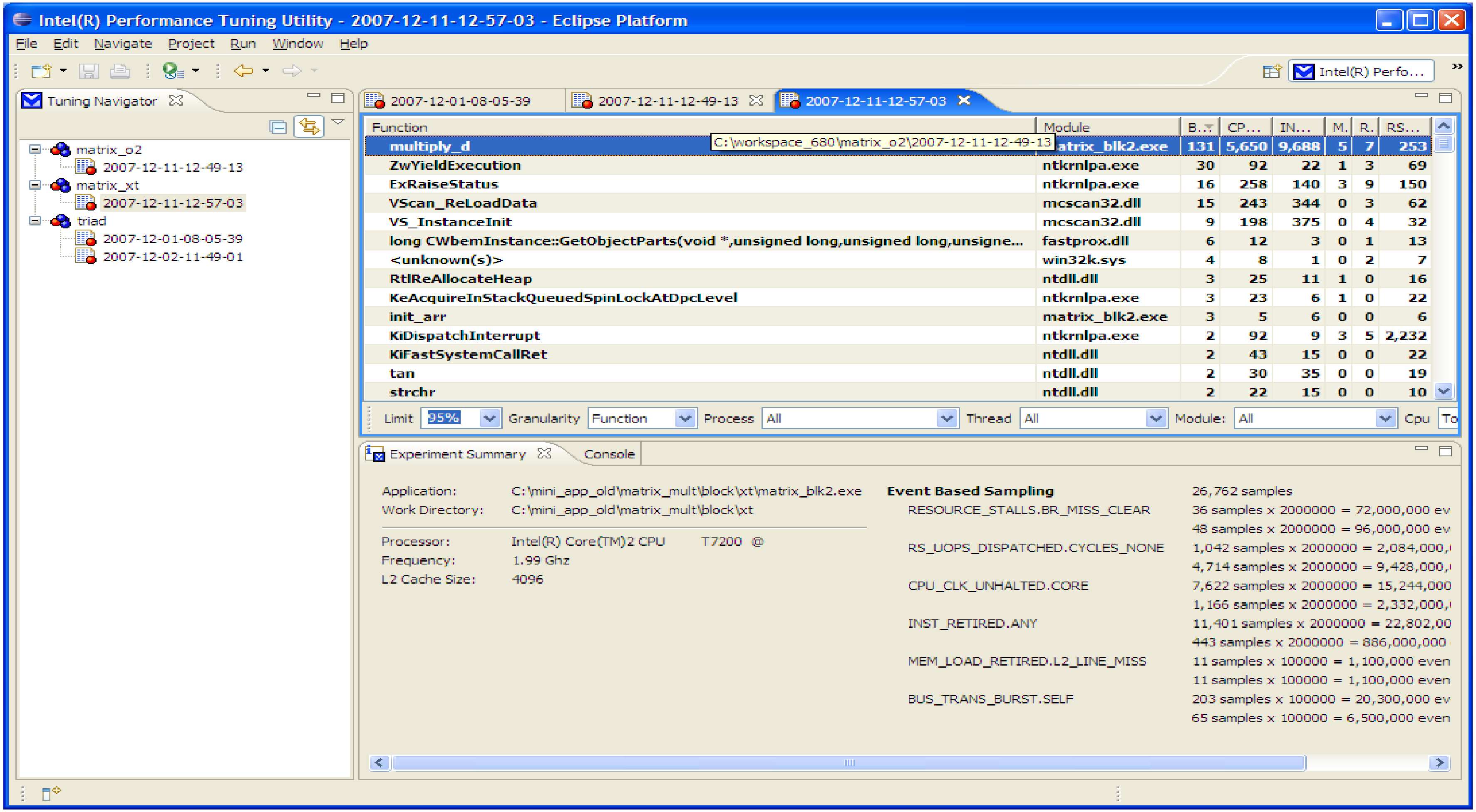


Function	Module	B..	CP...	IN...	M.	R.	RS...
multiply_d	matrix_blk2.exe	108	7,434	9,669	8	7	416
ExRaiseStatus	ntkrnlpa.exe	19	117	58	2	5	52
VS_InstanceInit	mcscan32.dll	17	252	436	3	6	42
ZwYieldExecution	ntkrnlpa.exe	15	50	5	0	2	26
VScan_ReLoadData	mcscan32.dll	6	184	236	0	0	53
init_arr	matrix_blk2.exe	4	12	7	0	0	4
KiDispatchInterrupt	ntkrnlpa.exe	3	39	2	0	4	4,541
<unknown(s)>	win32k.sys	3	15	3	0	0	11
<unknown(s)>	mfeapfk.sys	2	113	178	0	0	15
<unknown(s)>	<unknown(s)32>	2	13	6	0	2	5
KfAcquireSpinLock	hal.dll	2	2	1	0	0	1
KeAcquireInStackQueuedSpinLock	hal.dll	1	31	10	0	0	9
KeReleaseQueuedSpinLock	hal.dll	1	30	10	0	0	16
Allocate_Inspector_Memory	engine.dll	1	19	22	0	1	15

Limit 95% Granularity Function Process All Thread All Module: All Cpu Tot

```
Application:      C:\mini_app_old\matrix_mult\block\o2\matrix_blk2.exe   Event Based Sampling                   28,187 samples
Work Directory:   C:\mini_app_old\matrix_mult\block\o2                     INST_RETIRED.ANY                     10,612 samples x 2000000 = 21,224,0(
                                                                                                                405 samples x 2000000 = 810,000,000
Processor:        Intel(R) Core(TM)2 CPU      T7200  @                     RESOURCE_STALLS.BR_MISS_CLEAR        25 samples x 2000000 = 50,000,000 ev
Frequency:        1.99 Ghz                                                                                      37 samples x 2000000 = 74,000,000 ev
L2 Cache Size:    4096                                                     MEM_LOAD_RETIRED.L2_LINE_MISS        12 samples x 100000 = 1,200,000 ever
                                                                                                                3 samples x 100000 = 300,000 events (
                                                                           RS_UOPS_DISPATCHED.CYCLES_NONE       620 samples x 2000000 = 1,240,000,0(
                                                                                                                7,297 samples x 2000000 = 14,594,00(
                                                                           CPU_CLK_UNHALTED.CORE                8,182 samples x 2000000 = 16,364,00(
                                                                                                                776 samples x 2000000 = 1,552,000,0(
                                                                           BUS_TRANS_BURST.SELF                 152 samples x 100000 = 15,200,000 ev
                                                                                                                66 samples x 100000 = 6,600,000 ever
```

Data blocked 2X2 unrolled Matrix Multiply compiled at -O3 –QxT (Binary = xt\matrix_blk2.exe) Cycle_Usage Profile



C:\workspace_680\matrix_o2\2007-12-11-12-49-13

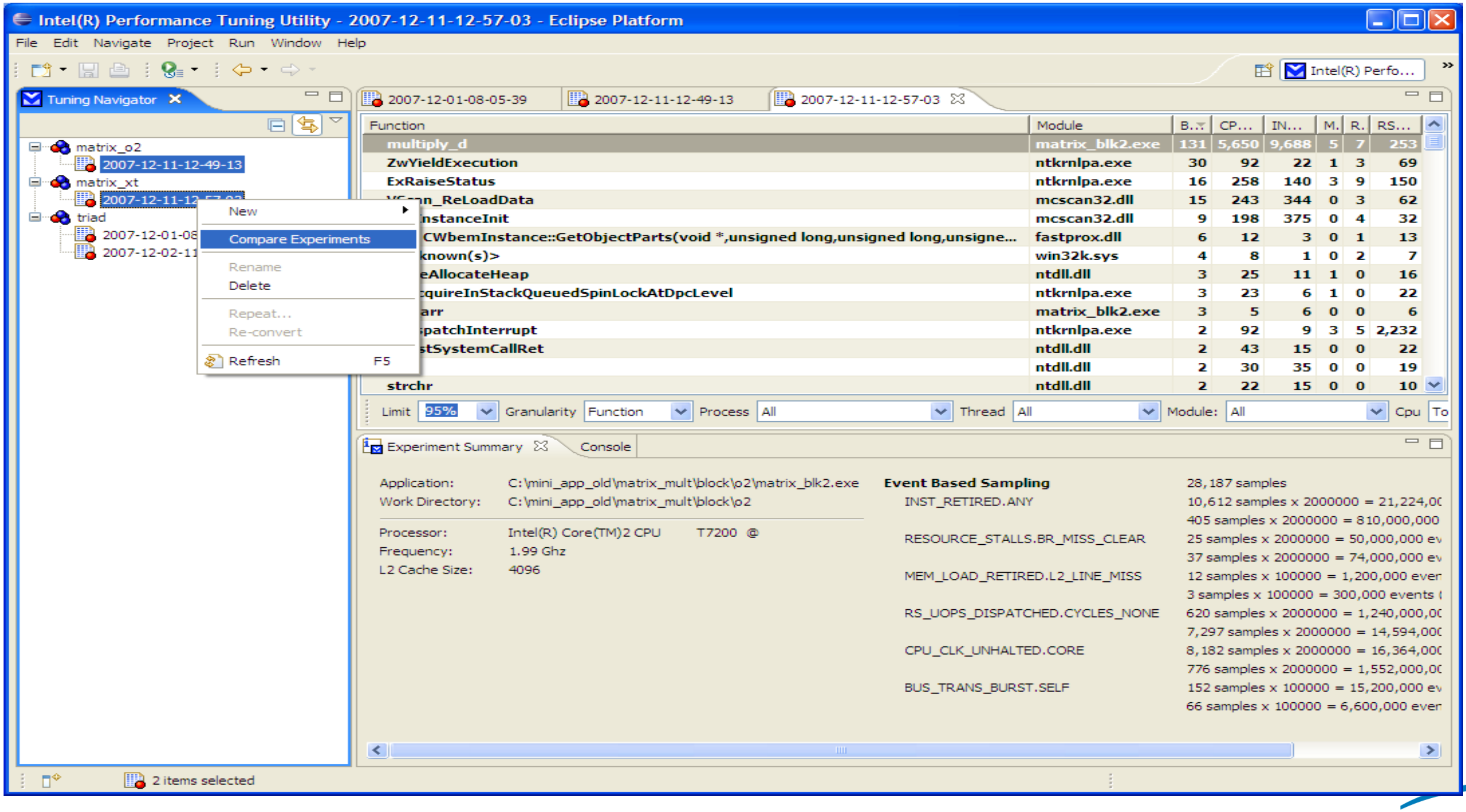
Function	Module	B..	CP...	IN...	M.	R.	RS...
multiply_d	atrix_blk2.exe	131	5,650	9,688	5	7	253
ZwYieldExecution	ntkrnlpa.exe	30	92	22	1	3	69
ExRaiseStatus	ntkrnlpa.exe	16	258	140	3	9	150
VScan_ReLoadData	mcscan32.dll	15	243	344	0	3	62
VS_InstanceInit	mcscan32.dll	9	198	375	0	4	32
long CWbemInstance::GetObjectParts(void *,unsigned long,unsigned long,unsigne...	fastprox.dll	6	12	3	0	1	13
<unknown(s)>	win32k.sys	4	8	1	0	2	7
RtlReAllocateHeap	ntdll.dll	3	25	11	1	0	16
KeAcquireInStackQueuedSpinLockAtDpcLevel	ntkrnlpa.exe	3	23	6	1	0	22
init_arr	matrix_blk2.exe	3	5	6	0	0	6
KiDispatchInterrupt	ntkrnlpa.exe	2	92	9	3	5	2,232
KiFastSystemCallRet	ntdll.dll	2	43	15	0	0	22
tan	ntdll.dll	2	30	35	0	0	19
strchr	ntdll.dll	2	22	15	0	0	10

Limit 95% Granularity Function Process All Thread All Module: All Cpu To

```
Application:       C:\mini_app_old\matrix_mult\block\xt\matrix_blk2.exe    Event Based Sampling                 26,762 samples
Work Directory:    C:\mini_app_old\matrix_mult\block\xt                       RESOURCE_STALLS.BR_MISS_CLEAR       36 samples x 2000000 = 72,000,000 ev
                                                                                                                  48 samples x 2000000 = 96,000,000 ev
Processor:         Intel(R) Core(TM)2 CPU      T7200  @                       RS_UOPS_DISPATCHED.CYCLES_NONE      1,042 samples x 2000000 = 2,084,000,
Frequency:         1.99 Ghz                                                                                       4,714 samples x 2000000 = 9,428,000,
L2 Cache Size:     4096                                                       CPU_CLK_UNHALTED.CORE               7,622 samples x 2000000 = 15,244,000
                                                                                                                  1,166 samples x 2000000 = 2,332,000,
                                                                              INST_RETIRED.ANY                    11,401 samples x 2000000 = 22,802,00
                                                                                                                  443 samples x 2000000 = 886,000,000
                                                                              MEM_LOAD_RETIRED.L2_LINE_MISS       11 samples x 100000 = 1,100,000 even
                                                                                                                  11 samples x 100000 = 1,100,000 even
                                                                              BUS_TRANS_BURST.SELF                203 samples x 100000 = 20,300,000 ev
                                                                                                                  65 samples x 100000 = 6,500,000 even
```

Only Significant Difference is Cycle Count
Create Difference Display

- Control click to select 2 experiments
- Right click to select “Compare Experiments”



Function	Module	B...	CP...	IN...	M.	R.	RS...
multiply_d	matrix_blk2.exe	131	5,650	9,688	5	7	253
ZwYieldExecution	ntkrnlpa.exe	30	92	22	1	3	69
ExRaiseStatus	ntkrnlpa.exe	16	258	140	3	9	150
VScan_ReLoadData	mcscan32.dll	15	243	344	0	3	62
nstanceInit	mcscan32.dll	9	198	375	0	4	32
CWbemInstance::GetObjectParts(void *,unsigned long,unsigned long,unsigne...	fastprox.dll	6	12	3	0	1	13
known(s)>	win32k.sys	4	8	1	0	2	7
eAllocateHeap	ntdll.dll	3	25	11	1	0	16
cquireInStackQueuedSpinLockAtDpcLevel	ntkrnlpa.exe	3	23	6	1	0	22
arr	matrix_blk2.exe	3	5	6	0	0	6
spatchInterrupt	ntkrnlpa.exe	2	92	9	3	5	2,232
stSystemCallRet	ntdll.dll	2	43	15	0	0	22
	ntdll.dll	2	30	35	0	0	19
strchr	ntdll.dll	2	22	15	0	0	10

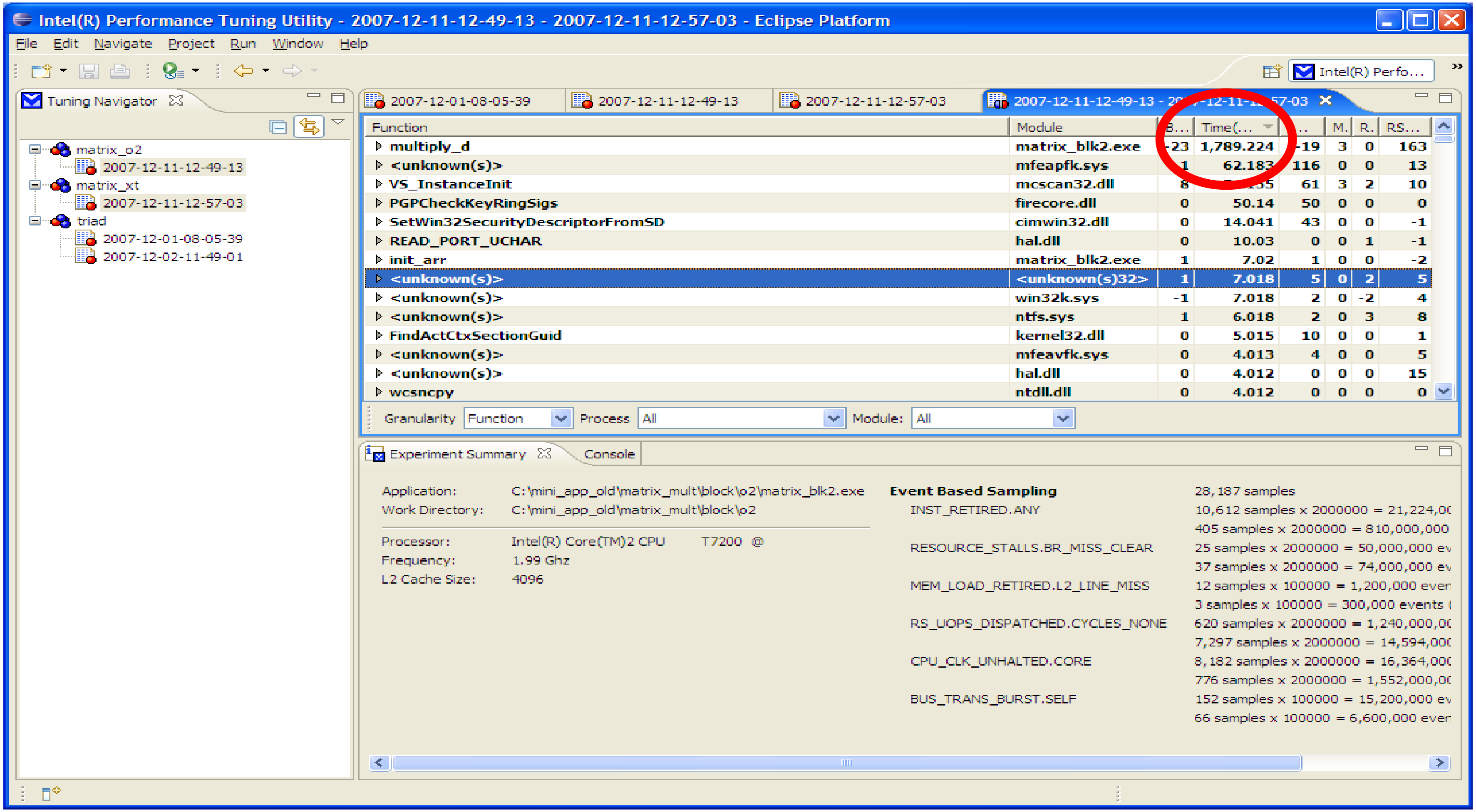
Limit 95% Granularity Function Process All Thread All Module: All Cpu To

Application: C:\mini_app_old\matrix_mult\block\o2\matrix_blk2.exe
Work Directory: C:\mini_app_old\matrix_mult\block\o2

Processor: Intel(R) Core(TM)2 CPU T7200 @
Frequency: 1.99 Ghz
L2 Cache Size: 4096

Event Based Sampling	28,187 samples
INST_RETIRED.ANY	10,612 samples x 2000000 = 21,224,0(
	405 samples x 2000000 = 810,000,000
RESOURCE_STALLS.BR_MISS_CLEAR	25 samples x 2000000 = 50,000,000 ev
	37 samples x 2000000 = 74,000,000 ev
MEM_LOAD_RETIRED.L2_LINE_MISS	12 samples x 100000 = 1,200,000 ever
	3 samples x 100000 = 300,000 events (
RS_UOPS_DISPATCHED.CYCLES_NONE	620 samples x 2000000 = 1,240,000,0(
	7,297 samples x 2000000 = 14,594,00(
CPU_CLK_UNHALTED.CORE	8,182 samples x 2000000 = 16,364,00(
	776 samples x 2000000 = 1,552,000,0(
BUS_TRANS_BURST.SELF	152 samples x 100000 = 15,200,000 ev
	66 samples x 100000 = 6,600,000 ever

Differences of Samples
Differences in Cycles Shown in msec to Correct for Comparison of Machines at Different Frequencies



Function	Module	B...	Time(...	...	M.	R.	RS...
multiply_d	matrix_blk2.exe	-23	1,789.224	-19	3	0	163
<unknown(s)>	mfeapfk.sys	1	62.183	116	0	0	13
VS_InstanceInit	mcscan32.dll	8	[illegible]	61	3	2	10
PGPCheckKeyRingSigs	firecore.dll	0	50.14	50	0	0	0
SetWin32SecurityDescriptorFromSD	cimwin32.dll	0	14.041	43	0	0	-1
READ_PORT_UCHAR	hal.dll	0	10.03	0	0	1	-1
init_arr	matrix_blk2.exe	1	7.02	1	0	0	-2
<unknown(s)>	<unknown(s)32>	1	7.018	5	0	2	5
<unknown(s)>	win32k.sys	-1	7.018	2	0	-2	4
<unknown(s)>	ntfs.sys	1	6.018	2	0	3	8
FindActCtxSectionGuid	kernel32.dll	0	5.015	10	0	0	1
<unknown(s)>	mfeavfk.sys	0	4.013	4	0	0	5
<unknown(s)>	hal.dll	0	4.012	0	0	0	15
wcsncpy	ntdll.dll	0	4.012	0	0	0	0

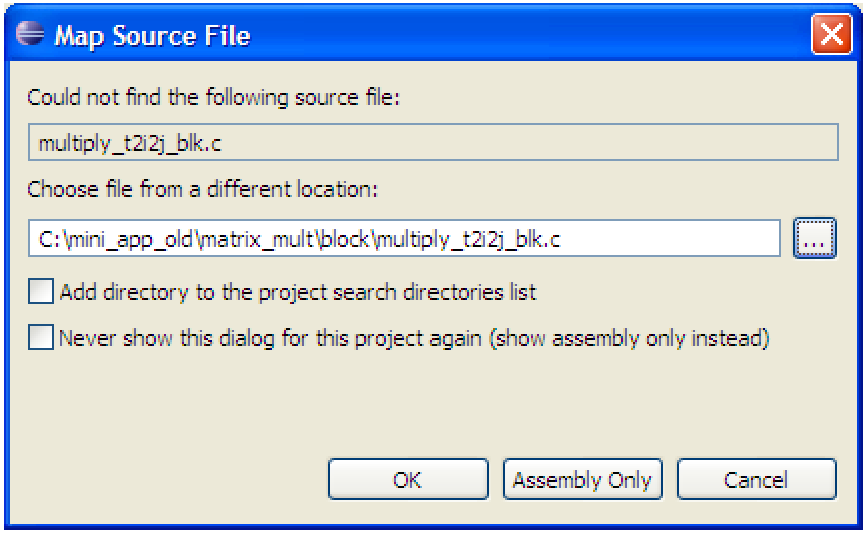
Granularity: Function   Process: All   Module: All

```
Application:      C:\mini_app_old\matrix_mult\block\o2\matrix_blk2.exe     Event Based Sampling                      28,187 samples
Work Directory:   C:\mini_app_old\matrix_mult\block\o2                        INST_RETIRED.ANY                         10,612 samples x 2000000 = 21,224,0(
                                                                                                                        405 samples x 2000000 = 810,000,000
Processor:        Intel(R) Core(TM)2 CPU      T7200  @                        RESOURCE_STALLS.BR_MISS_CLEAR            25 samples x 2000000 = 50,000,000 ev
Frequency:        1.99 Ghz                                                                                              37 samples x 2000000 = 74,000,000 ev
L2 Cache Size:    4096                                                        MEM_LOAD_RETIRED.L2_LINE_MISS            12 samples x 100000 = 1,200,000 ever
                                                                                                                        3 samples x 100000 = 300,000 events (
                                                                              RS_UOPS_DISPATCHED.CYCLES_NONE           620 samples x 2000000 = 1,240,000,0(
                                                                                                                        7,297 samples x 2000000 = 14,594,00(
                                                                              CPU_CLK_UNHALTED.CORE                    8,182 samples x 2000000 = 16,364,00(
                                                                                                                        776 samples x 2000000 = 1,552,000,0(
                                                                              BUS_TRANS_BURST.SELF                     152 samples x 100000 = 15,200,000 ev
                                                                                                                        66 samples x 100000 = 6,600,000 ever
```

Scaling Analysis: Sort by Time and see what causes non scaling

Drill down by Double Click on Function to Source in difference view

- It is likely to ask where to find the source file



Could not find the following source file:

multiply_t2i2j_blk.c

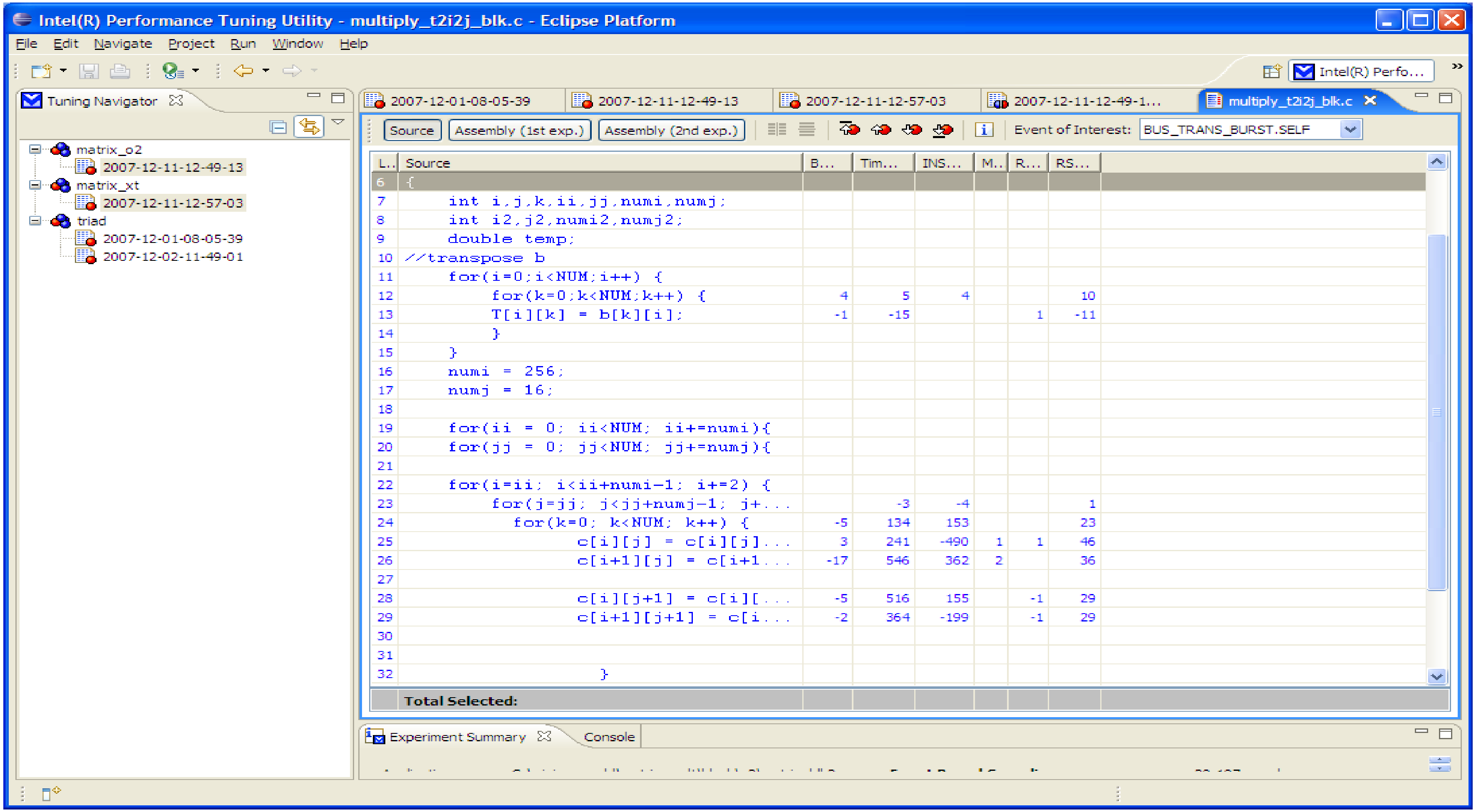
Choose file from a different location:

C:\mini_app_old\matrix_mult\block\multiply_t2i2j_blk.c

□ Add directory to the project search directories list

□ Never show this dialog for this project again (show assembly only instead)

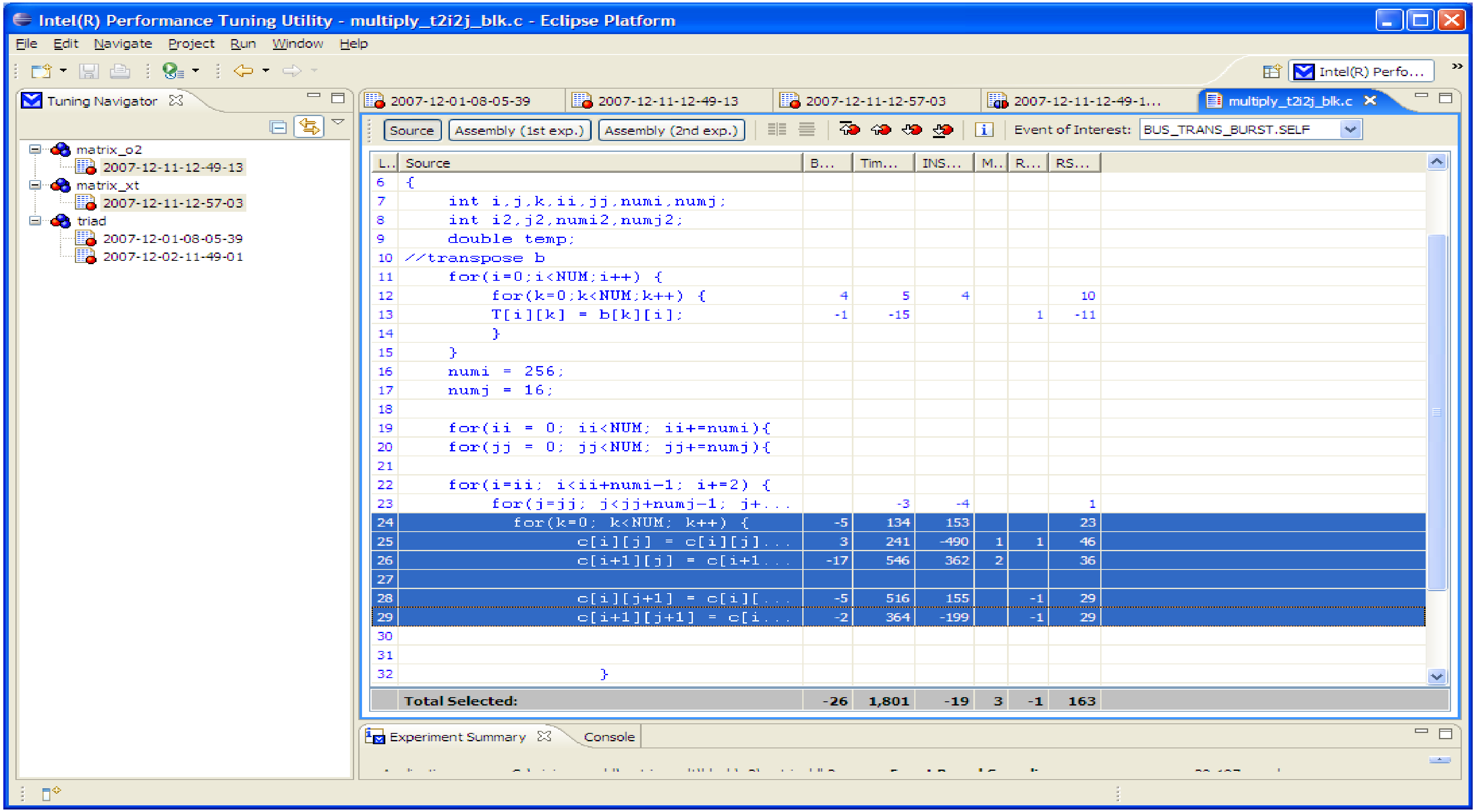
Same Source can Display Difference per Source Line



Event of Interest: BUS_TRANS_BURST.SELF

L...	Source	B...	Tim...	INS...	M...	R...	RS...
6	{						
7	int i,j,k,ii,jj,numi,numj;						
8	int i2,j2,numi2,numj2;						
9	double temp;						
10	//transpose b						
11	for(i=0;i<NUM;i++) {						
12	for(k=0;k<NUM;k++) {	4	5	4			10
13	T[i][k] = b[k][i];	-1	-15			1	-11
14	}						
15	}						
16	numi = 256;						
17	numj = 16;						
18							
19	for(ii = 0; ii<NUM; ii+=numi){						
20	for(jj = 0; jj<NUM; jj+=numj){						
21							
22	for(i=ii; i<ii+numi-1; i+=2) {						
23	for(j=jj; j<jj+numj-1; j+...		-3	-4			1
24	for(k=0; k<NUM; k++) {	-5	134	153			23
25	c[i][j] = c[i][j]...	3	241	-490	1	1	46
26	c[i+1][j] = c[i+1...	-17	546	362	2		36
27							
28	c[i][j+1] = c[i][...	-5	516	155		-1	29
29	c[i+1][j+1] = c[i...	-2	364	-199		-1	29
30							
31							
32	}						
	Total Selected:						

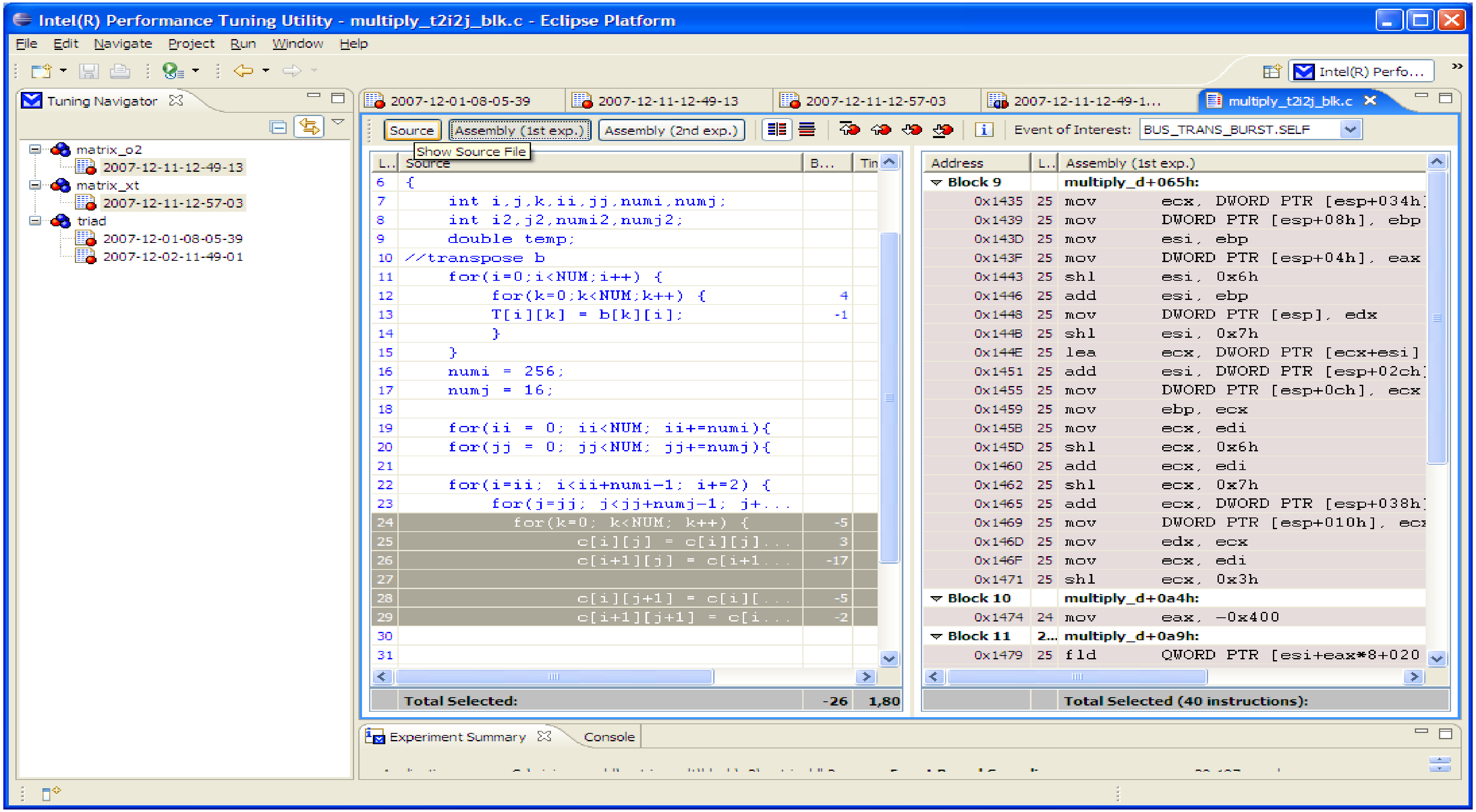
Shift Right click to Highlight a Region and Display Subtotal at the Bottom



Event of Interest: BUS_TRANS_BURST.SELF

L..	Source	B...	Tim...	INS...	M..	R...	RS...
6	{						
7	int i,j,k,ii,jj,numi,numj;						
8	int i2,j2,numi2,numj2;						
9	double temp;						
10	//transpose b						
11	for(i=0;i<NUM;i++) {						
12	for(k=0;k<NUM;k++) {	4	5	4			10
13	T[i][k] = b[k][i];	-1	-15			1	-11
14	}						
15	}						
16	numi = 256;						
17	numj = 16;						
18							
19	for(ii = 0; ii<NUM; ii+=numi){						
20	for(jj = 0; jj<NUM; jj+=numj){						
21							
22	for(i=ii; i<ii+numi-1; i+=2) {						
23	for(j=jj; j<jj+numj-1; j+...		-3	-4			1
24	for(k=0; k<NUM; k++) {	-5	134	153			23
25	c[i][j] = c[i][j]...	3	241	-490	1	1	46
26	c[i+1][j] = c[i+1...	-17	546	362	2		36
27							
28	c[i][j+1] = c[i][...	-5	516	155		-1	29
29	c[i+1][j+1] = c[i...	-2	364	-199		-1	29
30							
31							
32	}						
	Total Selected:	**-26**	**1,801**	**-19**	**3**	**-1**	**163**

Select “Assembly (1^{st} Exp.)”
Only Contributing Basic Blocks are Displayed

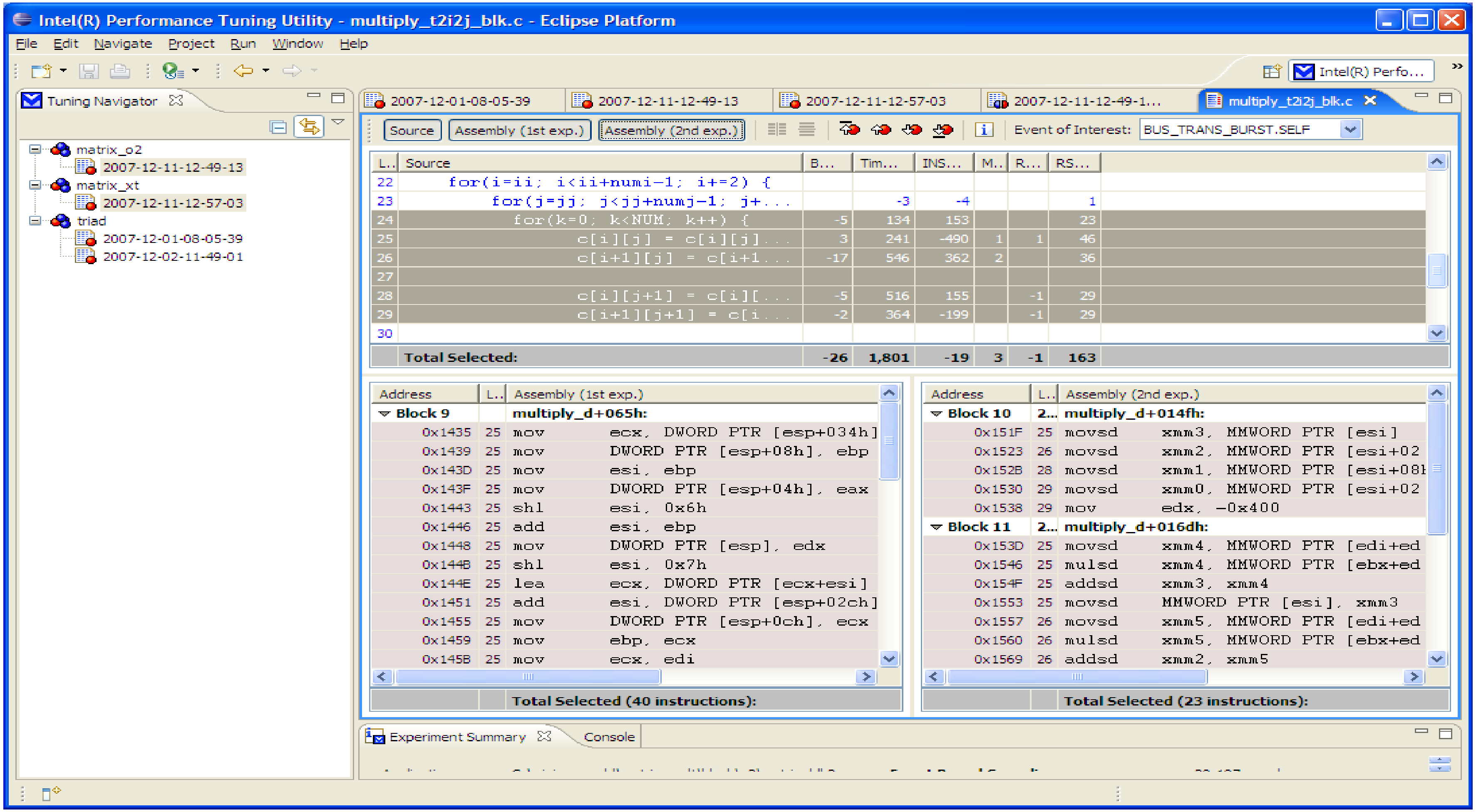


Event of Interest: BUS_TRANS_BURST.SELF

L...	Source	B...	Tin
6	{		
7	int i,j,k,ii,jj,numi,numj;		
8	int i2,j2,numi2,numj2;		
9	double temp;		
10	//transpose b		
11	for(i=0;i<NUM;i++) {		
12	for(k=0;k<NUM;k++) {	4	
13	T[i][k] = b[k][i];	-1	
14	}		
15	}		
16	numi = 256;		
17	numj = 16;		
18			
19	for(ii = 0; ii<NUM; ii+=numi){		
20	for(jj = 0; jj<NUM; jj+=numj){		
21			
22	for(i=ii; i<ii+numi-1; i+=2) {		
23	for(j=jj; j<jj+numj-1; j+...		
24	for(k=0; k<NUM; k++) {	-5	
25	c[i][j] = c[i][j]...	3	
26	c[i+1][j] = c[i+1...	-17	
27			
28	c[i][j+1] = c[i][...	-5	
29	c[i+1][j+1] = c[i...	-2	
30			
31			
	Total Selected:	-26	1,80

Address	L...	Assembly (1st exp.)
Block 9		multiply_d+065h:
0x1435	25	mov ecx, DWORD PTR [esp+034h]
0x1439	25	mov DWORD PTR [esp+08h], ebp
0x143D	25	mov esi, ebp
0x143F	25	mov DWORD PTR [esp+04h], eax
0x1443	25	shl esi, 0x6h
0x1446	25	add esi, ebp
0x1448	25	mov DWORD PTR [esp], edx
0x144B	25	shl esi, 0x7h
0x144E	25	lea ecx, DWORD PTR [ecx+esi]
0x1451	25	add esi, DWORD PTR [esp+02ch]
0x1455	25	mov DWORD PTR [esp+0ch], ecx
0x1459	25	mov ebp, ecx
0x145B	25	mov ecx, edi
0x145D	25	shl ecx, 0x6h
0x1460	25	add ecx, edi
0x1462	25	shl ecx, 0x7h
0x1465	25	add ecx, DWORD PTR [esp+038h]
0x1469	25	mov DWORD PTR [esp+010h], ecx
0x146D	25	mov edx, ecx
0x146F	25	mov ecx, edi
0x1471	25	shl ecx, 0x3h
Block 10		multiply_d+0a4h:
0x1474	24	mov eax, -0x400
Block 11	2...	multiply_d+0a9h:
0x1479	25	fld QWORD PTR [esi+eax*8+020
		Total Selected (40 instructions):

Select “Assembly (2nd Exp.)”
Only Contributing Basic Blocks are Displayed Now for BOTH Binaries



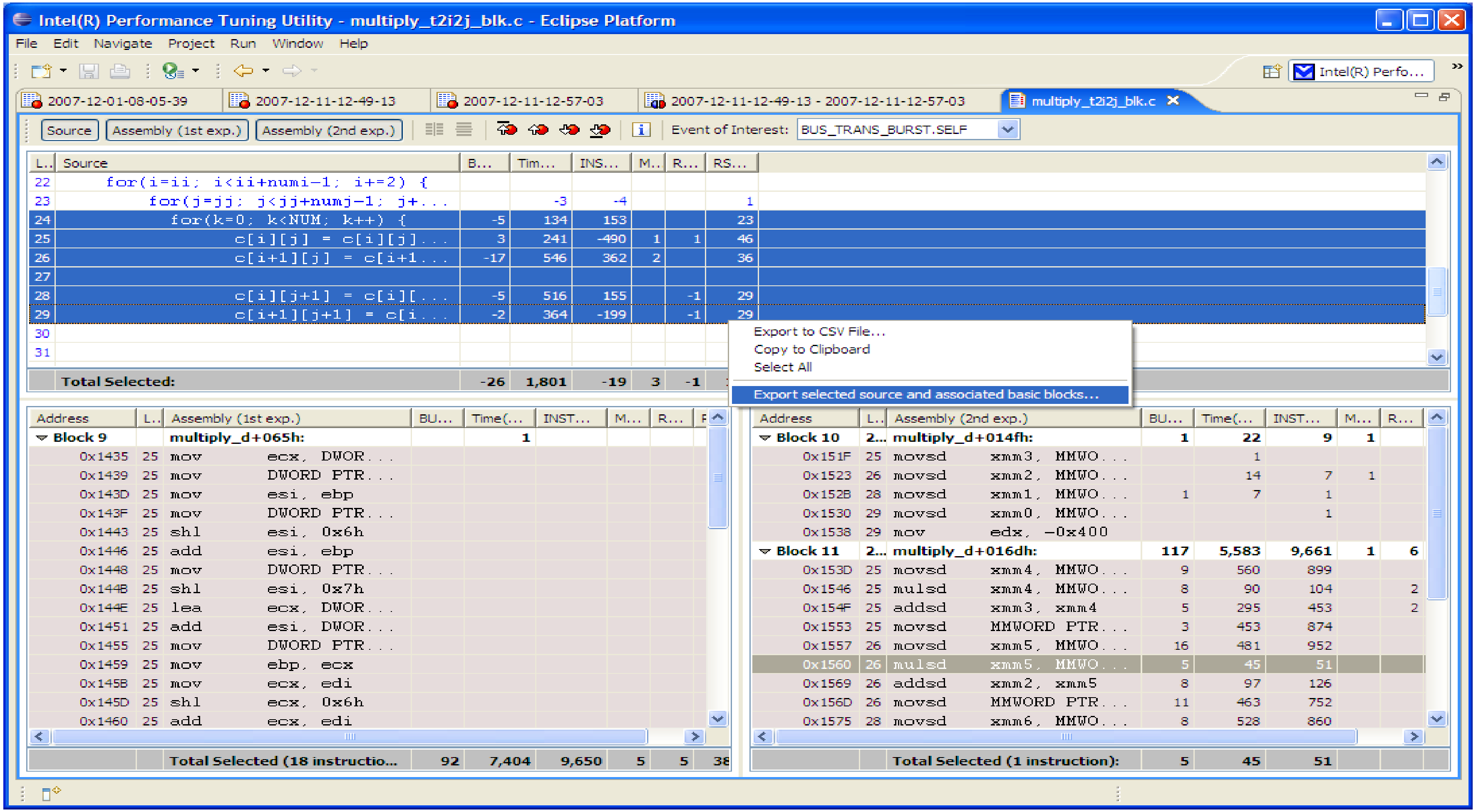
Event of Interest: BUS_TRANS_BURST.SELF

L..	Source	B...	Tim...	INS...	M..	R...	RS...
22	for(i=ii; i<ii+numi-1; i+=2) {						
23	for(j=jj; j<jj+numj-1; j+...		-3	-4			1
24	for(k=0; k<NUM; k++) {	-5	134	153			23
25	c[i][j] = c[i][j]...	3	241	-490	1	1	46
26	c[i+1][j] = c[i+1...	-17	546	362	2		36
27							
28	c[i][j+1] = c[i][...	-5	516	155		-1	29
29	c[i+1][j+1] = c[i...	-2	364	-199		-1	29
30							
	Total Selected:	**-26**	**1,801**	**-19**	**3**	**-1**	**163**

Address	L..	Assembly (1st exp.)
Block 9		**multiply_d+065h:**
0x1435	25	mov ecx, DWORD PTR [esp+034h]
0x1439	25	mov DWORD PTR [esp+08h], ebp
0x143D	25	mov esi, ebp
0x143F	25	mov DWORD PTR [esp+04h], eax
0x1443	25	shl esi, 0x6h
0x1446	25	add esi, ebp
0x1448	25	mov DWORD PTR [esp], edx
0x144B	25	shl esi, 0x7h
0x144E	25	lea ecx, DWORD PTR [ecx+esi]
0x1451	25	add esi, DWORD PTR [esp+02ch]
0x1455	25	mov DWORD PTR [esp+0ch], ecx
0x1459	25	mov ebp, ecx
0x145B	25	mov ecx, edi
		Total Selected (40 instructions):

Address	L..	Assembly (2nd exp.)
Block 10	**2...**	**multiply_d+014fh:**
0x151F	25	movsd xmm3, MMWORD PTR [esi]
0x1523	26	movsd xmm2, MMWORD PTR [esi+02
0x152B	28	movsd xmm1, MMWORD PTR [esi+08h
0x1530	29	movsd xmm0, MMWORD PTR [esi+02
0x1538	29	mov edx, -0x400
Block 11	**2...**	**multiply_d+016dh:**
0x153D	25	movsd xmm4, MMWORD PTR [edi+ed
0x1546	25	mulsd xmm4, MMWORD PTR [ebx+ed
0x154F	25	addsd xmm3, xmm4
0x1553	25	movsd MMWORD PTR [esi], xmm3
0x1557	26	movsd xmm5, MMWORD PTR [edi+ed
0x1560	26	mulsd xmm5, MMWORD PTR [ebx+ed
0x1569	26	addsd xmm2, xmm5
		Total Selected (23 instructions):

Export Selected Source and the Contributing Basic Blocks from Both Binaries to a Single CSV Spread Sheet
Instant Compiler Regression Bug Report



Event of Interest: BUS_TRANS_BURST.SELF

L..	Source	B...	Tim...	INS...	M...	R...	RS...
22	for(i=ii; i<ii+numi-1; i+=2) {						
23	for(j=jj; j<jj+numj-1; j+...		-3	-4			1
24	for(k=0; k<NUM; k++) {	-5	134	153			23
25	c[i][j] = c[i][j]...	3	241	-490	1	1	46
26	c[i+1][j] = c[i+1...	-17	546	362	2		36
27							
28	c[i][j+1] = c[i][...	-5	516	155		-1	29
29	c[i+1][j+1] = c[i...	-2	364	-199		-1	29
30							
31							
	Total Selected:	-26	1,801	-19	3	-1	

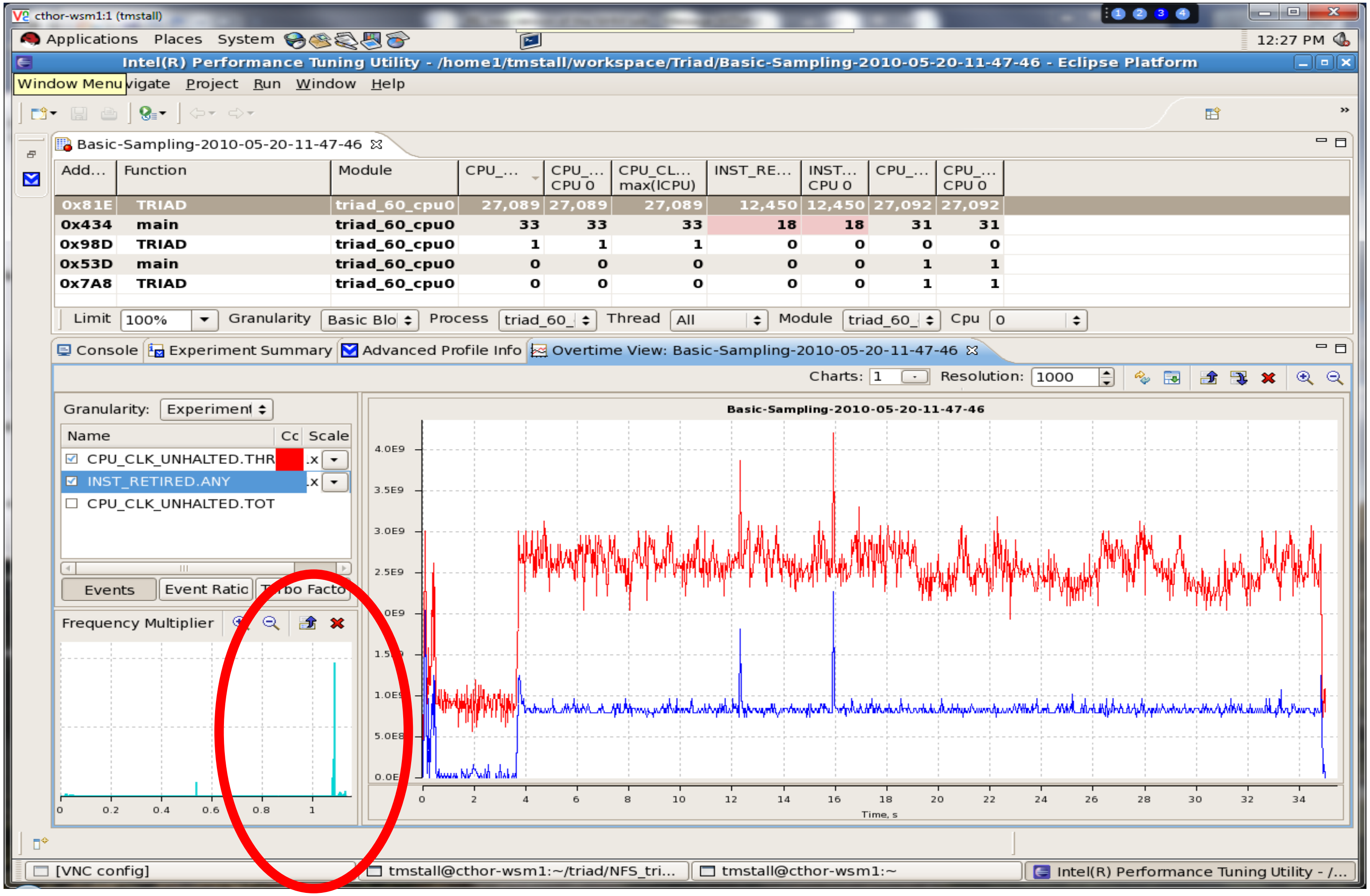
Address	L..	Assembly (1st exp.)		BU...	Time(...	INST...	M...	R...
Block 9		multiply_d+065h:			1			
0x1435	25	mov	ecx, DWOR...					
0x1439	25	mov	DWORD PTR...					
0x143D	25	mov	esi, ebp					
0x143F	25	mov	DWORD PTR...					
0x1443	25	shl	esi, 0x6h					
0x1446	25	add	esi, ebp					
0x1448	25	mov	DWORD PTR...					
0x144B	25	shl	esi, 0x7h					
0x144E	25	lea	ecx, DWOR...					
0x1451	25	add	esi, DWOR...					
0x1455	25	mov	DWORD PTR...					
0x1459	25	mov	ebp, ecx					
0x145B	25	mov	ecx, edi					
0x145D	25	shl	ecx, 0x6h					
0x1460	25	add	ecx, edi					
		Total Selected (18 instructio...		92	7,404	9,650	5	5

Address	L..	Assembly (2nd exp.)		BU...	Time(...	INST...	M...	R...
Block 10	2...	multiply_d+014fh:		1	22	9	1	
0x151F	25	movsd	xmm3, MMWO...		1			
0x1523	26	movsd	xmm2, MMWO...		14	7	1	
0x152B	28	movsd	xmm1, MMWO...	1	7	1		
0x1530	29	movsd	xmm0, MMWO...			1		
0x1538	29	mov	edx, -0x400					
Block 11	2...	multiply_d+016dh:		117	5,583	9,661	1	6
0x153D	25	movsd	xmm4, MMWO...	9	560	899		
0x1546	25	mulsd	xmm4, MMWO...	8	90	104		2
0x154F	25	addsd	xmm3, xmm4	5	295	453		2
0x1553	25	movsd	MMWORD PTR...	3	453	874		
0x1557	26	movsd	xmm5, MMWO...	16	481	952		
0x1560	26	mulsd	xmm5, MMWO...	5	45	51		
0x1569	26	addsd	xmm2, xmm5	8	97	126		
0x156D	26	movsd	MMWORD PTR...	11	463	752		
0x1575	28	movsd	xmm6, MMWO...	8	528	860		
		Total Selected (1 instruction):		5	45	51		

Measuring non parallel execution

- With turbo enabled, non parallel execution will result in a frequency boost to the core executing the serial code
- The serial functions can be identified using the filtering capability of the over time display

Single threaded execution with turbo boost enabled



cthor-wsm1:1 (tmstall)

Intel(R) Performance Tuning Utility - /home1/tmstall/workspace/Triad/Basic-Sampling-2010-05-20-11-47-46 - Eclipse Platform

Basic-Sampling-2010-05-20-11-47-46

Add...	Function	Module	CPU_...	CPU_... CPU 0	CPU_CL... max(ICPU)	INST_RE...	INST... CPU 0	CPU_...	CPU_... CPU 0
0x81E	TRIAD	triad_60_cpu0	27,089	27,089	27,089	12,450	12,450	27,092	27,092
0x434	main	triad_60_cpu0	33	33	33	18	18	31	31
0x98D	TRIAD	triad_60_cpu0	1	1	1	0	0	0	0
0x53D	main	triad_60_cpu0	0	0	0	0	0	1	1
0x7A8	TRIAD	triad_60_cpu0	0	0	0	0	0	1	1

Limit 100% Granularity Basic Blo Process triad_60_ Thread All Module triad_60_ Cpu 0

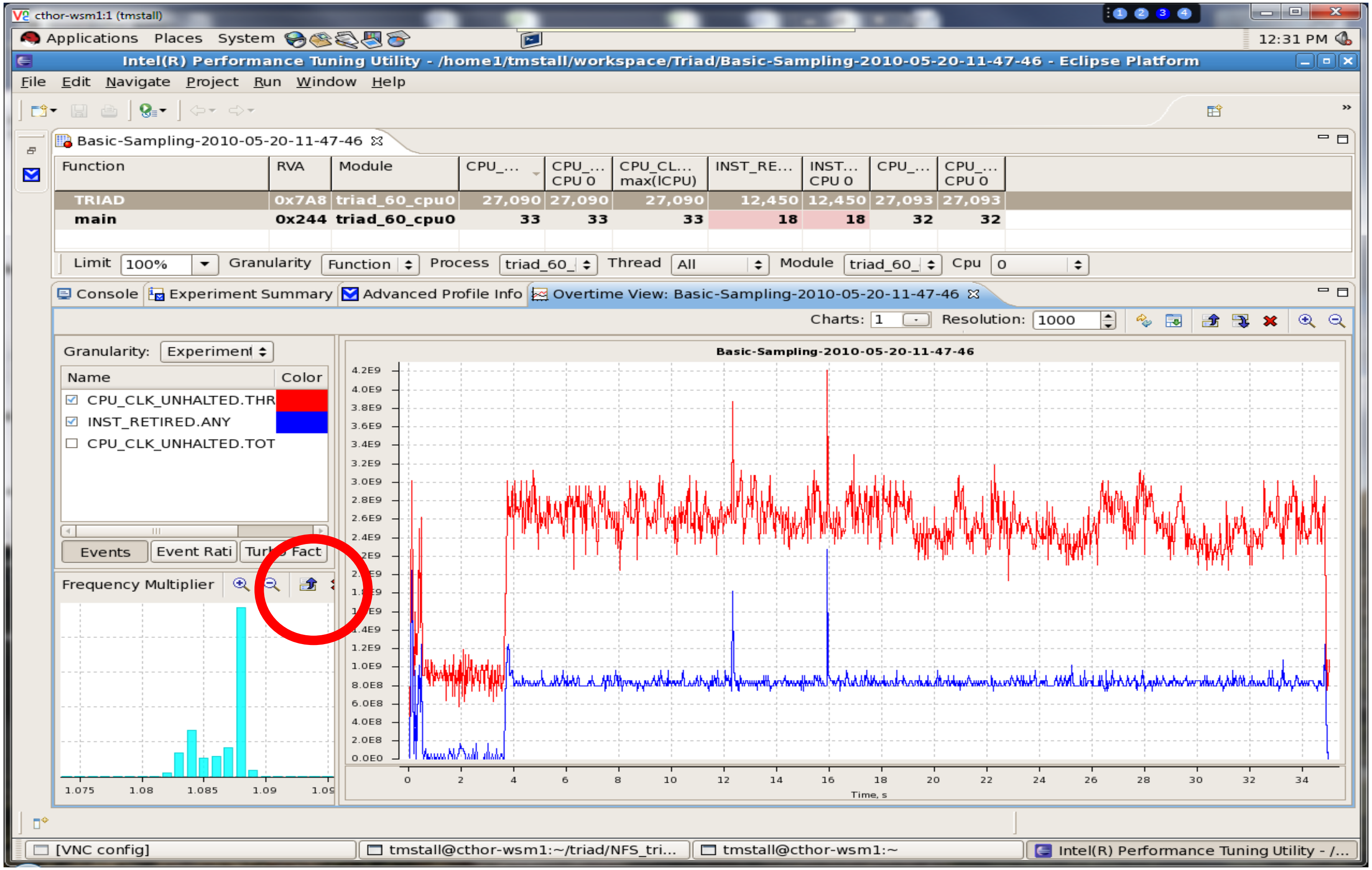
Overtime View: Basic-Sampling-2010-05-20-11-47-46

Charts: 1 Resolution: 1000

Granularity: Experiment

Name	Cc	Scale
☑ CPU_CLK_UNHALTED.THR	🟥	.x
☑ INST_RETIRED.ANY		.x
□ CPU_CLK_UNHALTED.TOT		

Zoom in on frequency multiplier select range and filter up


cthor-wsm1:1 (tmstall)
Applications Places System
12:31 PM
Intel(R) Performance Tuning Utility - /home1/tmstall/workspace/Triad/Basic-Sampling-2010-05-20-11-47-46 - Eclipse Platform
File Edit Navigate Project Run Window Help
Basic-Sampling-2010-05-20-11-47-46

Function	RVA	Module	CPU_...	CPU_... CPU 0	CPU_CL... max(ICPU)	INST_RE...	INST... CPU 0	CPU_...	CPU_... CPU 0
TRIAD	0x7A8	triad_60_cpu0	27,090	27,090	27,090	12,450	12,450	27,093	27,093
main	0x244	triad_60_cpu0	33	33	33	18	18	32	32

Limit 100% Granularity Function Process triad_60_ Thread All Module triad_60_ Cpu 0

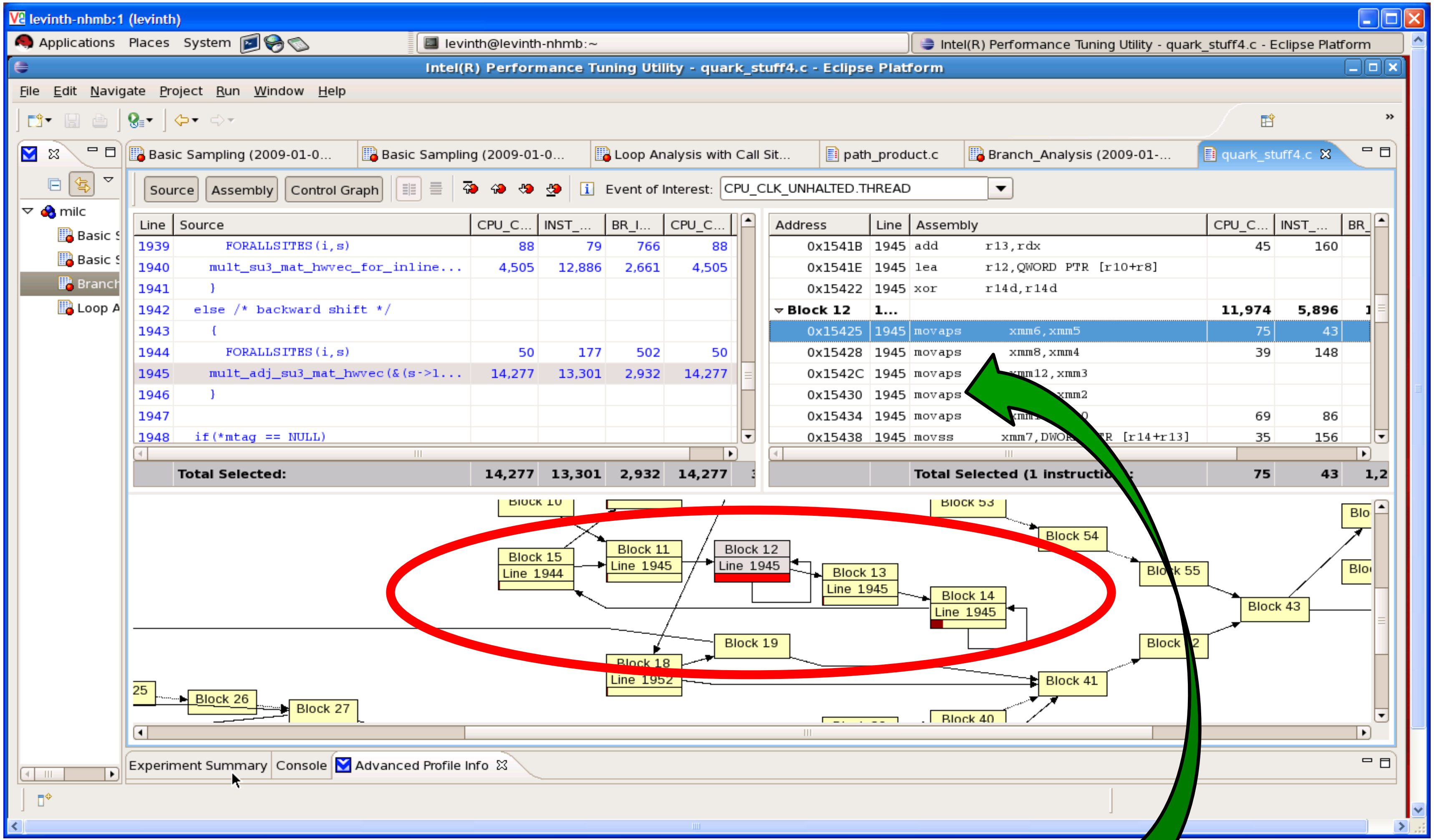
Console
Experiment Summary
Advanced Profile Info
Overtime View: Basic-Sampling-2010-05-20-11-47-46

Charts: 1 Resolution: 1000

Granularity: Experiment

Name	Color
☑ CPU_CLK_UNHALTED.THR	🟥
☑ INST_RETIRED.ANY	🟦
□ CPU_CLK_UNHALTED.TOT	

Source View Shows what is <u>Executed</u>



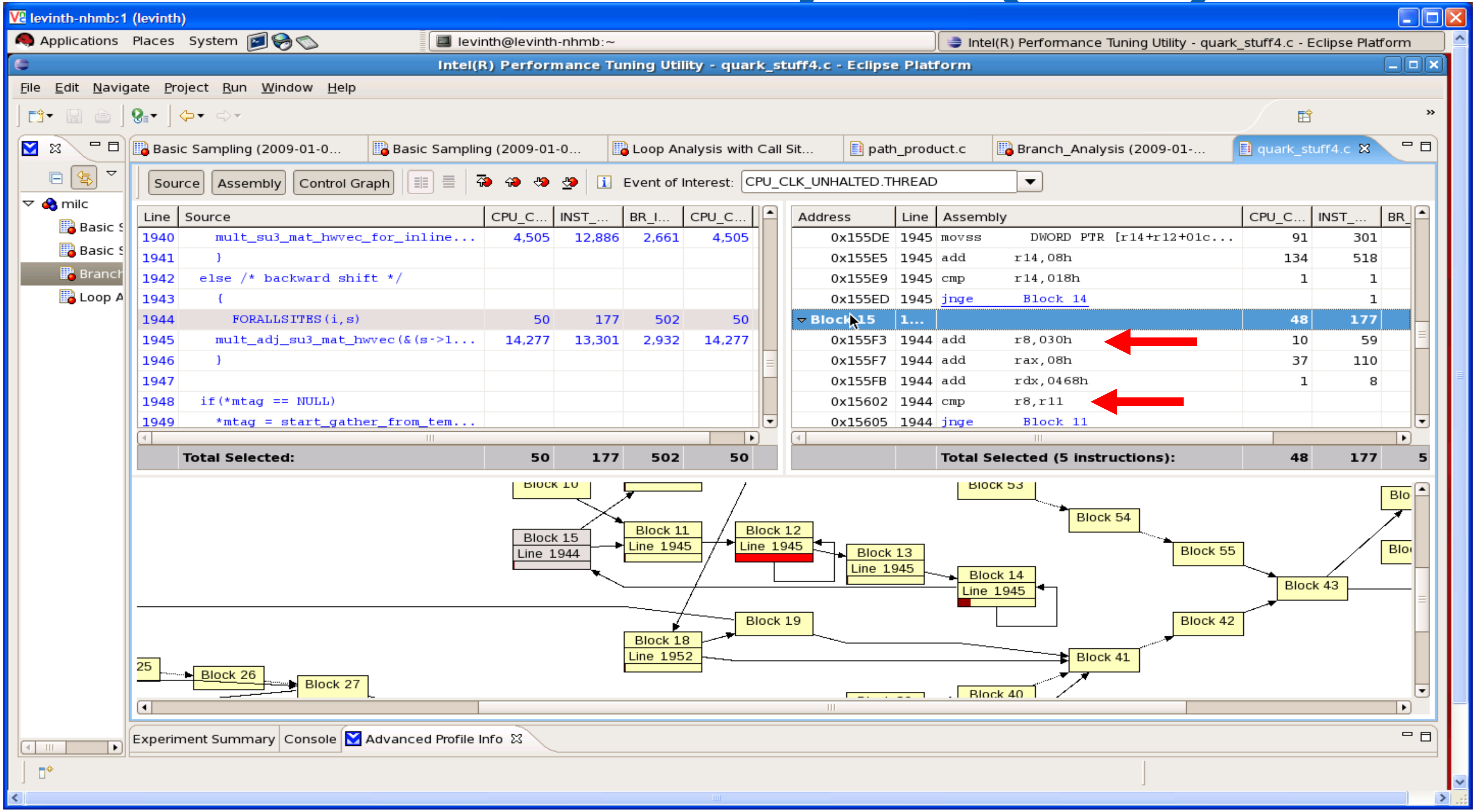
Event of Interest: CPU_CLK_UNHALTED.THREAD

Line	Source	CPU_C...	INST_...	BR_I...	CPU_C...
1939	`FORALLSITES(i,s)`	88	79	766	88
1940	`mult_su3_mat_hwvec_for_inline...`	4,505	12,886	2,661	4,505
1941	`}`				
1942	`else /* backward shift */`				
1943	`{`				
1944	`FORALLSITES(i,s)`	50	177	502	50
1945	`mult_adj_su3_mat_hwvec(&(s->l...`	14,277	13,301	2,932	14,277
1946	`}`				
1947					
1948	`if(*mtag == NULL)`				
	Total Selected:	**14,277**	**13,301**	**2,932**	**14,277**

Address	Line	Assembly	CPU_C...	INST_...	BR_
0x1541B	1945	`add r13,rdx`	45	160	
0x1541E	1945	`lea r12,QWORD PTR [r10+r8]`			
0x15422	1945	`xor r14d,r14d`			
Block 12	**1...**		**11,974**	**5,896**	**1**
0x15425	1945	`movaps xmm6,xmm5`	75	43	
0x15428	1945	`movaps xmm8,xmm4`	39	148	
0x1542C	1945	`movaps xmm12,xmm3`			
0x15430	1945	`movaps [illegible]xmm2`			
0x15434	1945	`movaps xmm1[illegible]0`	69	86	
0x15438	1945	`movss xmm7,DWORD PTR [r14+r13]`	35	156	
		Total Selected (1 instruction):	**75**	**43**	**1,2**

This is Vectorized

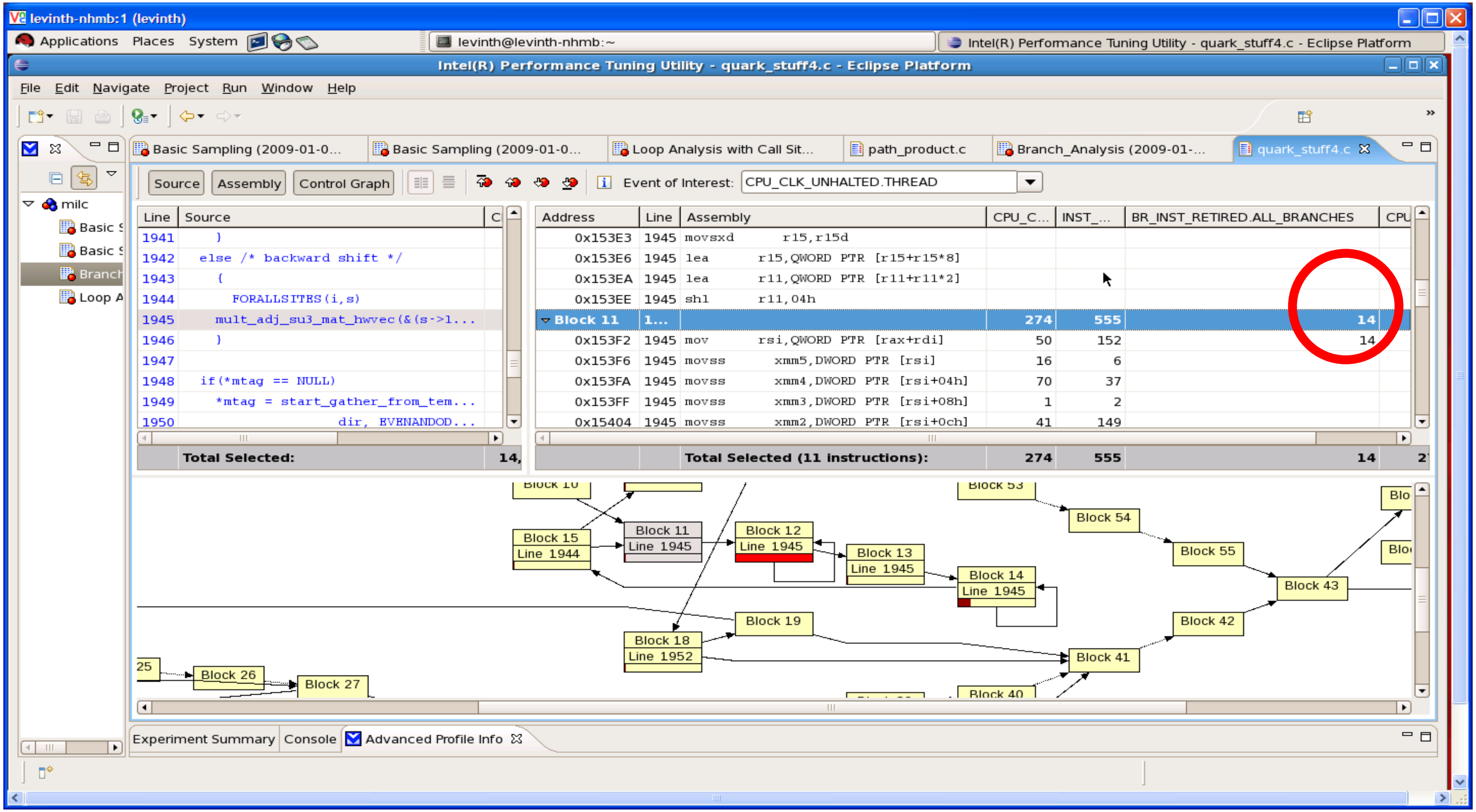
Cmp in Blk 15 Controls Loop, Comparing R8 and R11. R8 increments by 48 (30H)



Line	Source	CPU_C...	INST_...	BR_I...	CPU_C...
1940	mult_su3_mat_hwvec_for_inline...	4,505	12,886	2,661	4,505
1941	}				
1942	else /* backward shift */				
1943	{				
1944	FORALLSITES(i,s)	50	177	502	50
1945	mult_adj_su3_mat_hwvec(&(s->1...	14,277	13,301	2,932	14,277
1946	}				
1947					
1948	if(*mtag == NULL)				
1949	*mtag = start_gather_from_tem...				
	Total Selected:	50	177	502	50

Address	Line	Assembly	CPU_C...	INST_...	BR_
0x155DE	1945	movss DWORD PTR [r14+r12+01c...	91	301	
0x155E5	1945	add r14,08h	134	518	
0x155E9	1945	cmp r14,018h	1	1	
0x155ED	1945	jnge Block 14		1	
Block 15	1...		48	177	
0x155F3	1944	add r8,030h	10	59	
0x155F7	1944	add rax,08h	37	110	
0x155FB	1944	add rdx,0468h	1	8	
0x15602	1944	cmp r8,r11			
0x15605	1944	jnge Block 11			
		Total Selected (5 instructions):	48	177	5

Register Values Collected with Precise Event Br_inst_retired.all_branches in Blk 11 Yield Values for R11 (14 samples)

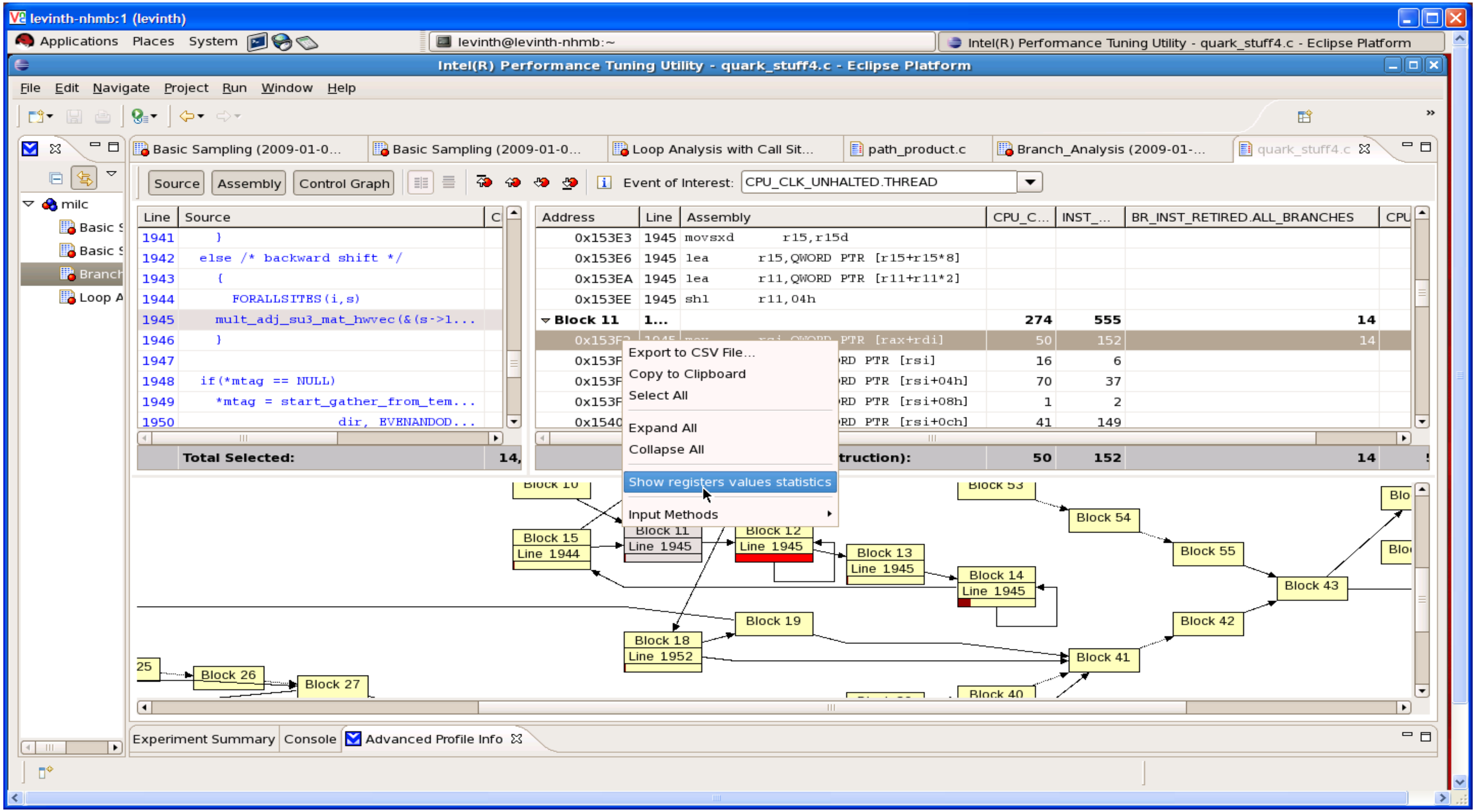


Event of Interest: CPU_CLK_UNHALTED.THREAD

Line	Source	C
1941	}	
1942	else /* backward shift */	
1943	{	
1944	FORALLSITES(i,s)	
1945	mult_adj_su3_mat_hwvec(&(s->l...	
1946	}	
1947		
1948	if(*mtag == NULL)	
1949	*mtag = start_gather_from_tem...	
1950	dir, EVENANDOD...	
	Total Selected:	14,

Address	Line	Assembly	CPU_C...	INST_...	BR_INST_RETIRED.ALL_BRANCHES	CPU
0x153E3	1945	movsxd r15,r15d				
0x153E6	1945	lea r15,QWORD PTR [r15+r15*8]				
0x153EA	1945	lea r11,QWORD PTR [r11+r11*2]				
0x153EE	1945	shl r11,04h				
Block 11	1...		274	555	14	
0x153F2	1945	mov rsi,QWORD PTR [rax+rdi]	50	152	14	
0x153F6	1945	movss xmm5,DWORD PTR [rsi]	16	6		
0x153FA	1945	movss xmm4,DWORD PTR [rsi+04h]	70	37		
0x153FF	1945	movss xmm3,DWORD PTR [rsi+08h]	1	2		
0x15404	1945	movss xmm2,DWORD PTR [rsi+0ch]	41	149		
		Total Selected (11 instructions):	274	555	14	2

Select the Asm Line, Right Click and Show Register Statistics

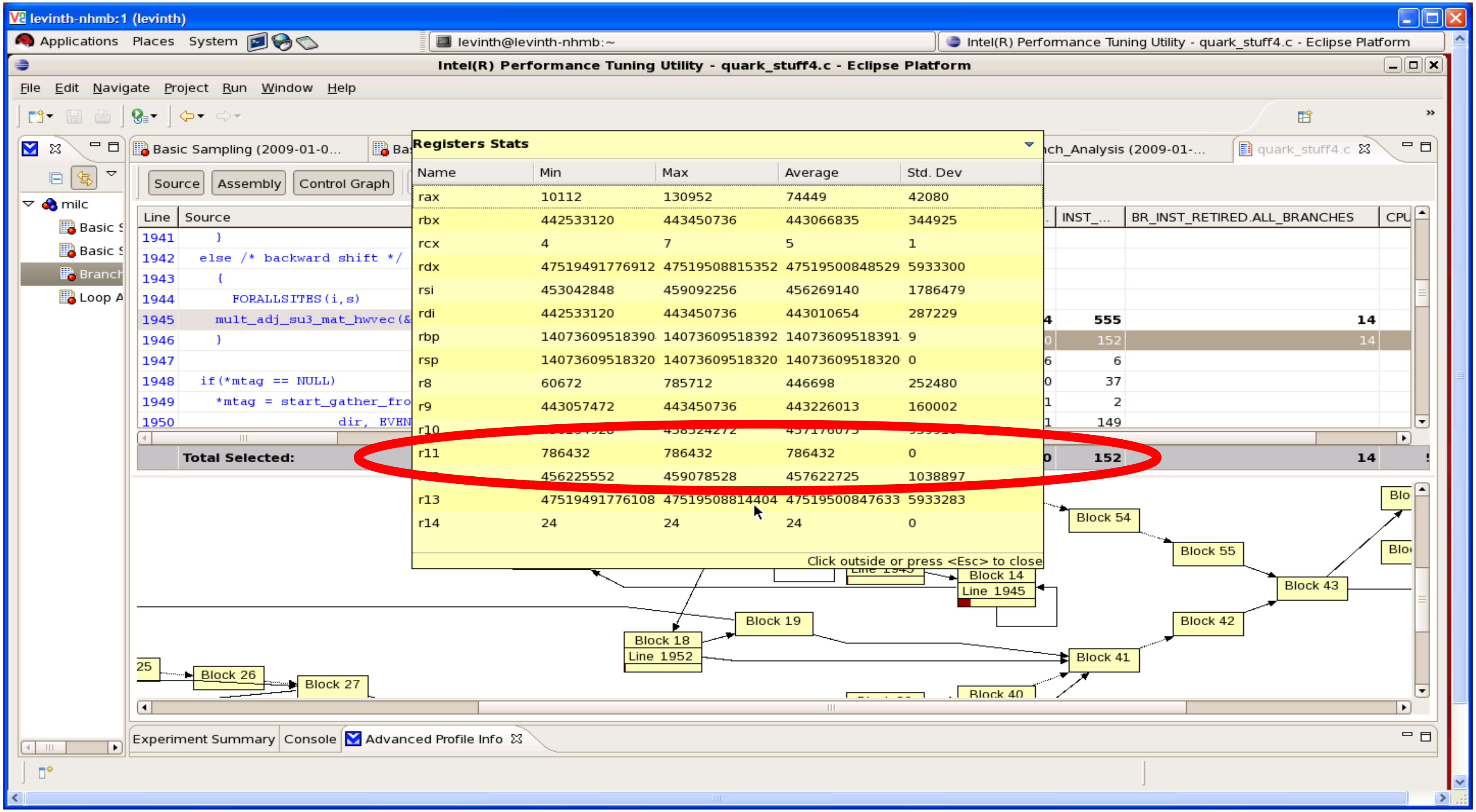


Event of Interest: CPU_CLK_UNHALTED.THREAD

Line	Source	C
1941	}	
1942	else /* backward shift */	
1943	{	
1944	FORALLSITES(i,s)	
1945	mult_adj_su3_mat_hwvec(&(s->1...	
1946	}	
1947		
1948	if(*mtag == NULL)	
1949	*mtag = start_gather_from_tem...	
1950	dir, EVENANDOD...	
	Total Selected:	**14,**

Address	Line	Assembly	CPU_C...	INST_...	BR_INST_RETIRED.ALL_BRANCHES	CPU
0x153E3	1945	movsxd r15,r15d				
0x153E6	1945	lea r15,QWORD PTR [r15+r15*8]				
0x153EA	1945	lea r11,QWORD PTR [r11+r11*2]				
0x153EE	1945	shl r11,04h				
Block 11	**1...**		**274**	**555**	**14**	
0x153F2	1945	mov rsi,QWORD PTR [rax+rdi]	50	152	14	
0x153F		RD PTR [rsi]	16	6		
0x153F		RD PTR [rsi+04h]	70	37		
0x153F		RD PTR [rsi+08h]	1	2		
0x1540		RD PTR [rsi+0ch]	41	149		
		truction):	**50**	**152**	**14**	

Tripcount is constant (min=max=avg, rms=0) and Equals 786432/48 = 16384 Which is the 4-Dim Lattice size for this Problem



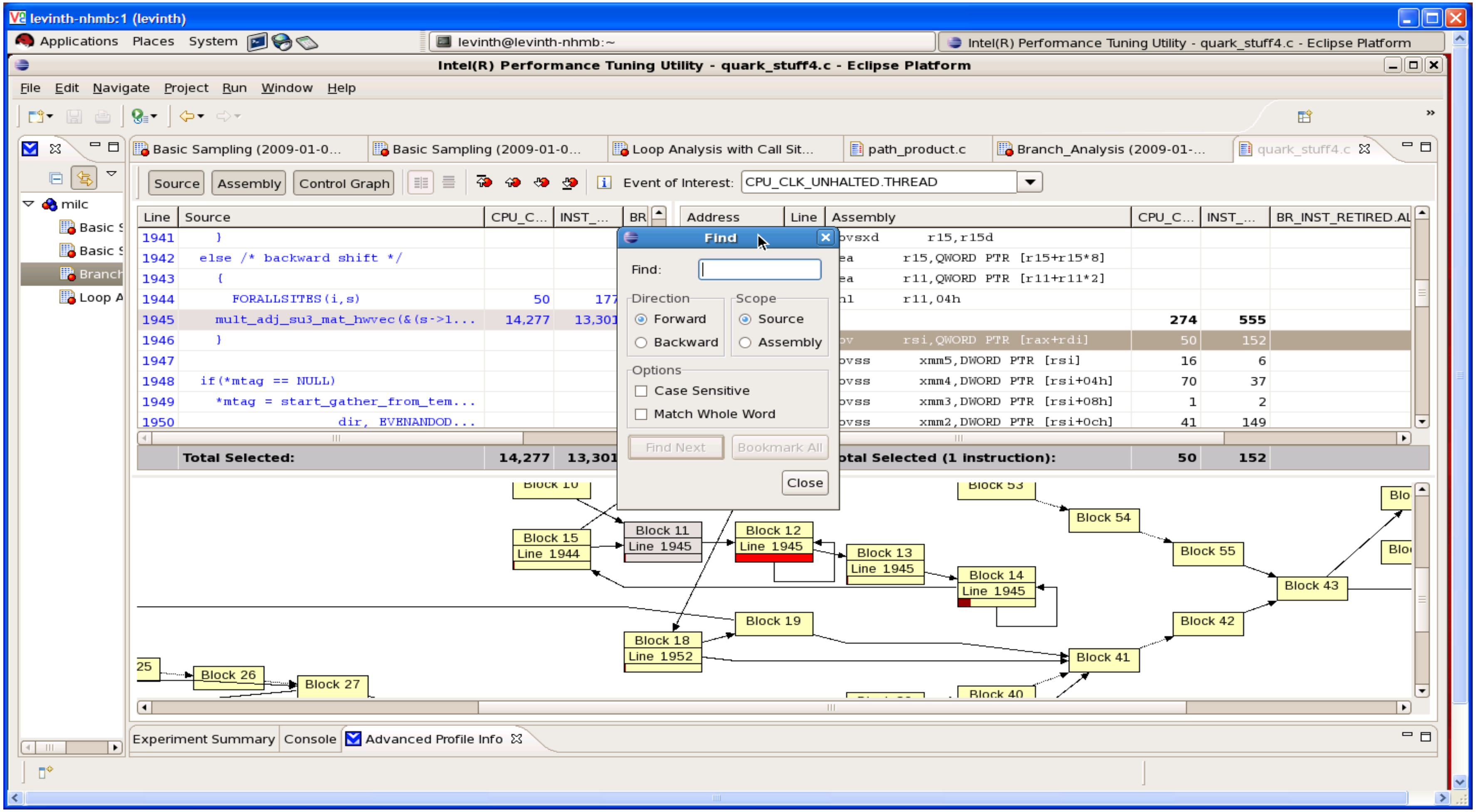
```
1941      }
1942    else /* backward shift */
1943      {
1944        FORALLSITES(i,s)
1945      mult_adj_su3_mat_hwvec(&
1946      }
1947
1948    if(*mtag == NULL)
1949      *mtag = start_gather_fro
1950                    dir, EVEN
```

Registers Stats

Name	Min	Max	Average	Std. Dev
rax	10112	130952	74449	42080
rbx	442533120	443450736	443066835	344925
rcx	4	7	5	1
rdx	47519491776912	47519508815352	47519500848529	5933300
rsi	453042848	459092256	456269140	1786479
rdi	442533120	443450736	443010654	287229
rbp	14073609518390	14073609518392	14073609518391	9
rsp	14073609518320	14073609518320	14073609518320	0
r8	60672	785712	446698	252480
r9	443057472	443450736	443226013	160002
r10	456104928	458324272	457176075	[illegible]
r11	786432	786432	786432	0
r12	456225552	459078528	457622725	1038897
r13	47519491776108	47519508814404	47519500847633	5933283
r14	24	24	24	0

Click outside or press <Esc> to close

Source/Asm View Text Search Utility



Event of Interest: CPU_CLK_UNHALTED.THREAD

Line	Source	CPU_C...	INST_...	BR
1941	}			
1942	else /* backward shift */			
1943	{			
1944	FORALLSITES(i,s)	50	177	
1945	mult_adj_su3_mat_hwvec(&(s->1...	14,277	13,301	
1946	}			
1947				
1948	if(*mtag == NULL)			
1949	*mtag = start_gather_from_tem...			
1950	dir, EVENANDOD...			
	Total Selected:	14,277	13,301	

Address	Line	Assembly	CPU_C...	INST_...	BR_INST_RETIRED.AL
		ovsxd r15,r15d			
		ea r15,QWORD PTR [r15+r15*8]			
		ea r11,QWORD PTR [r11+r11*2]			
		hl r11,04h			
			274	555	
		ov rsi,QWORD PTR [rax+rdi]	50	152	
		ovss xmm5,DWORD PTR [rsi]	16	6	
		ovss xmm4,DWORD PTR [rsi+04h]	70	37	
		ovss xmm3,DWORD PTR [rsi+08h]	1	2	
		ovss xmm2,DWORD PTR [rsi+0ch]	41	149	
		otal Selected (1 instruction):	50	152	

Find

Find:

Direction
● Forward
○ Backward

Scope
● Source
○ Assembly

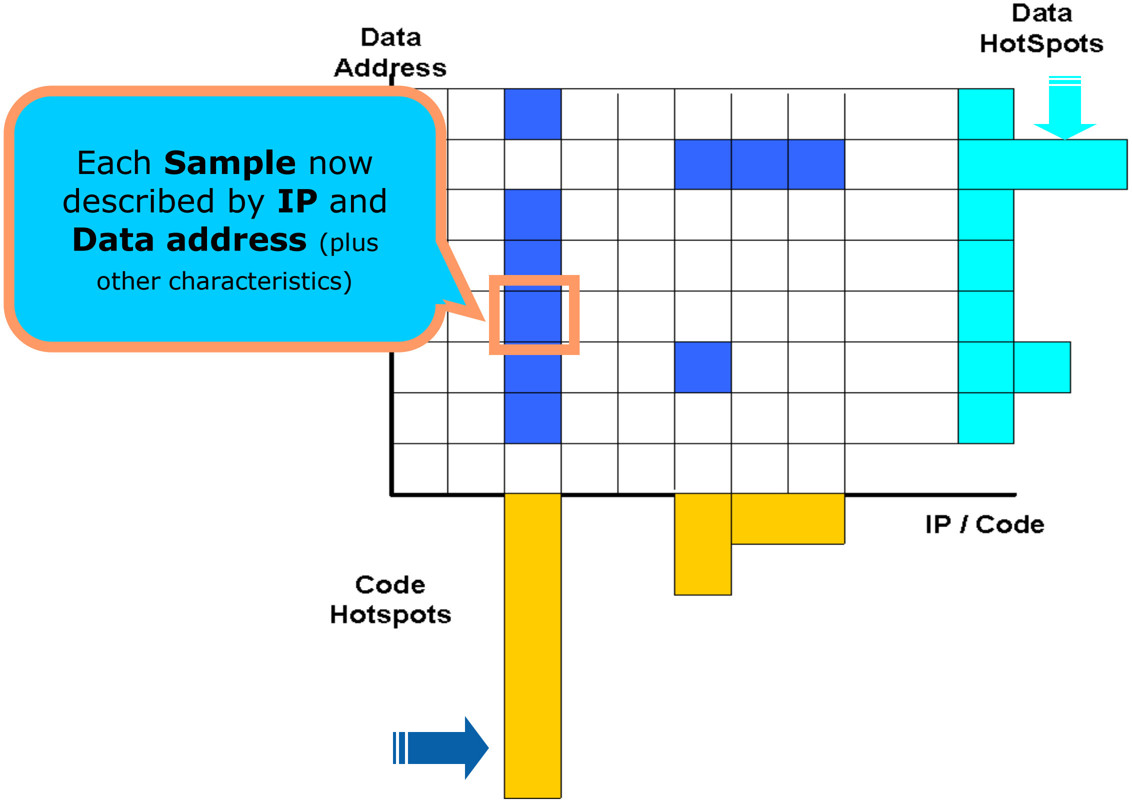
Options
☐ Case Sensitive
☐ Match Whole Word

Find Next | Bookmark All

Close

Data Address Profiling and False Sharing Detection

Data Mining in 2 Dimensional Model



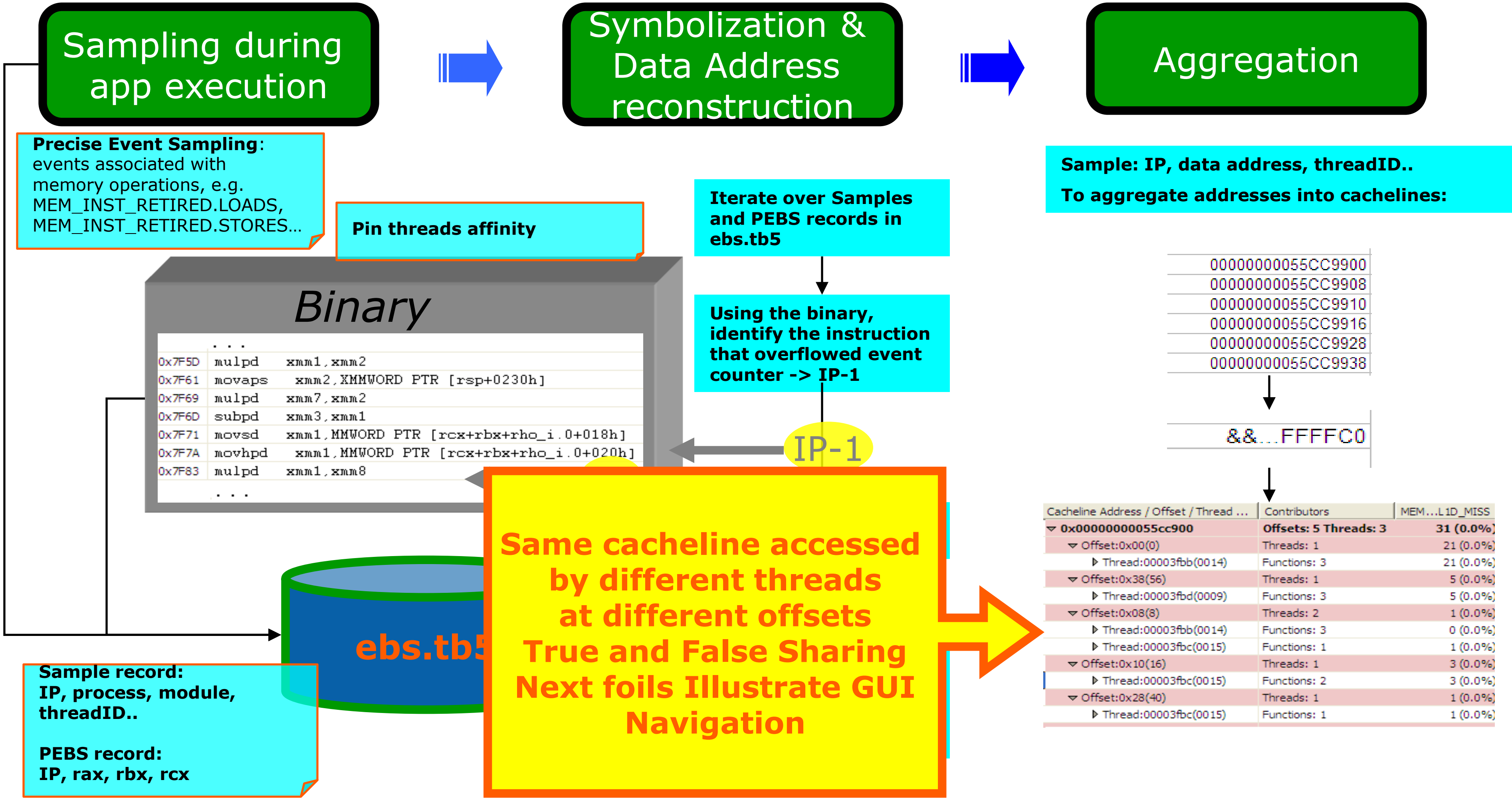
Each **Sample** now described by **IP** and **Data address** (plus other characteristics)

- **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

This foil is best viewed in animation mode

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

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
        . . .
```

Iterate over Samples and PEBS records in ebs.tb5

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

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

PEBS record: IP, rax, rbx, rcx

Sample: IP, data address, threadID..

To aggregate addresses into cachelines:

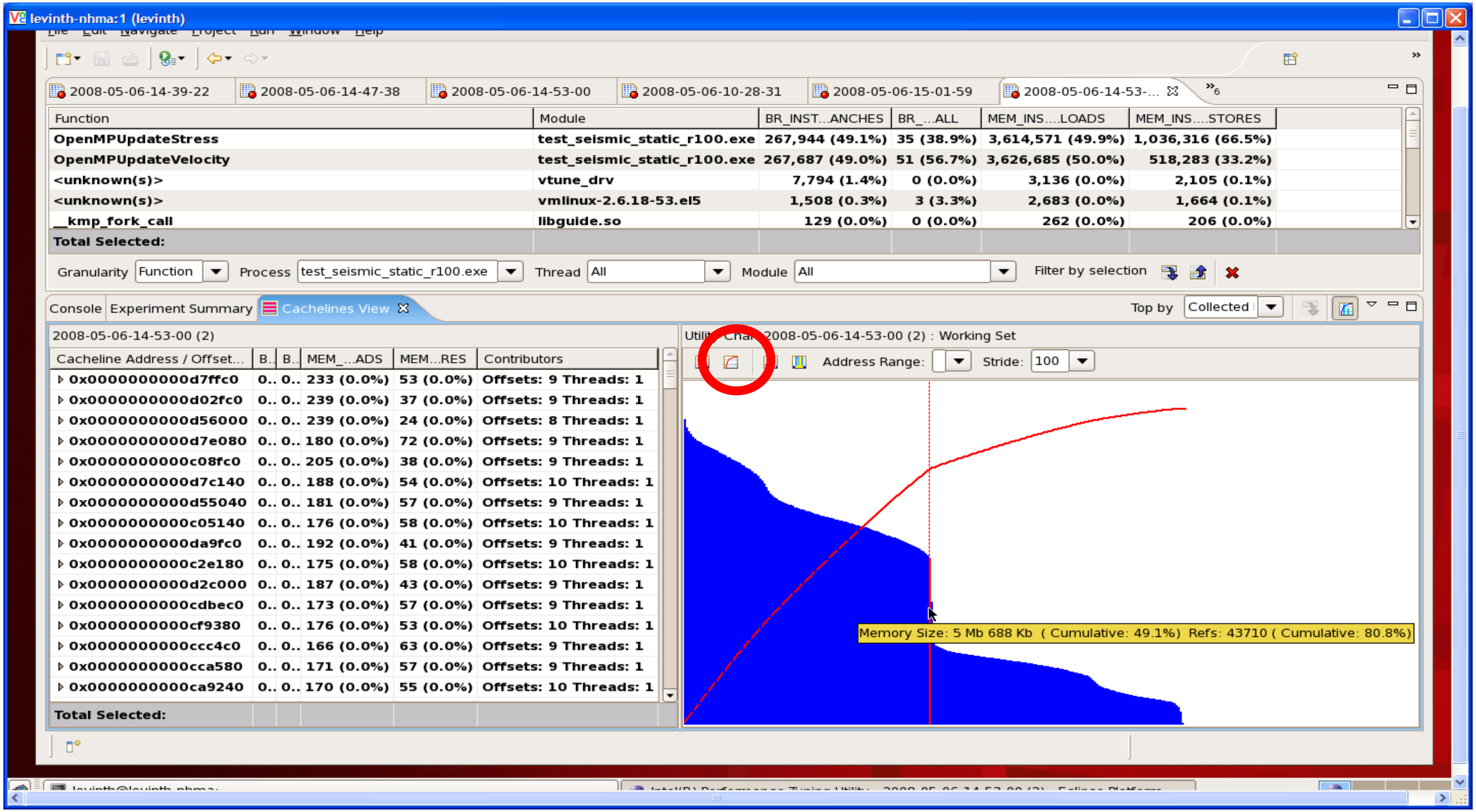
```
00000000055CC9900
00000000055CC9908
00000000055CC9910
00000000055CC9916
00000000055CC9928
00000000055CC9938
```

```
&&...FFFFC0
```

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%)

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

Use Cacheline Access Count to Measure Working Set Size



levinth-nhma:1 (levinth)

File Edit Navigate Project Run Window Help

2008-05-06-14-39-22 | 2008-05-06-14-47-38 | 2008-05-06-14-53-00 | 2008-05-06-10-28-31 | 2008-05-06-15-01-59 | 2008-05-06-14-53-...

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%)**
Total Selected:					

Granularity: Function | Process: test_seismic_static_r100.exe | Thread: All | Module: All | Filter by selection

Console | Experiment Summary | Cachelines View | Top by: Collected

2008-05-06-14-53-00 (2)

Cacheline Address / Offset...	B..	B..	MEM_...ADS	MEM...RES	Contributors
0x0000000000d7ffc0	**0..**	**0..**	**233 (0.0%)**	**53 (0.0%)**	**Offsets: 9 Threads: 1**
0x0000000000d02fc0	**0..**	**0..**	**239 (0.0%)**	**37 (0.0%)**	**Offsets: 9 Threads: 1**
0x0000000000d56000	**0..**	**0..**	**239 (0.0%)**	**24 (0.0%)**	**Offsets: 8 Threads: 1**
0x0000000000d7e080	**0..**	**0..**	**180 (0.0%)**	**72 (0.0%)**	**Offsets: 9 Threads: 1**
0x0000000000c08fc0	**0..**	**0..**	**205 (0.0%)**	**38 (0.0%)**	**Offsets: 9 Threads: 1**
0x0000000000d7c140	**0..**	**0..**	**188 (0.0%)**	**54 (0.0%)**	**Offsets: 10 Threads: 1**
0x0000000000d55040	**0..**	**0..**	**181 (0.0%)**	**57 (0.0%)**	**Offsets: 9 Threads: 1**
0x0000000000c05140	**0..**	**0..**	**176 (0.0%)**	**58 (0.0%)**	**Offsets: 10 Threads: 1**
0x0000000000da9fc0	**0..**	**0..**	**192 (0.0%)**	**41 (0.0%)**	**Offsets: 9 Threads: 1**
0x0000000000c2e180	**0..**	**0..**	**175 (0.0%)**	**58 (0.0%)**	**Offsets: 10 Threads: 1**
0x0000000000d2c000	**0..**	**0..**	**187 (0.0%)**	**43 (0.0%)**	**Offsets: 9 Threads: 1**
0x0000000000cdbec0	**0..**	**0..**	**173 (0.0%)**	**57 (0.0%)**	**Offsets: 9 Threads: 1**
0x0000000000cf9380	**0..**	**0..**	**176 (0.0%)**	**53 (0.0%)**	**Offsets: 10 Threads: 1**
0x0000000000ccc4c0	**0..**	**0..**	**166 (0.0%)**	**63 (0.0%)**	**Offsets: 9 Threads: 1**
0x0000000000cca580	**0..**	**0..**	**171 (0.0%)**	**57 (0.0%)**	**Offsets: 9 Threads: 1**
0x0000000000ca9240	**0..**	**0..**	**170 (0.0%)**	**55 (0.0%)**	**Offsets: 10 Threads: 1**
Total Selected:					

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

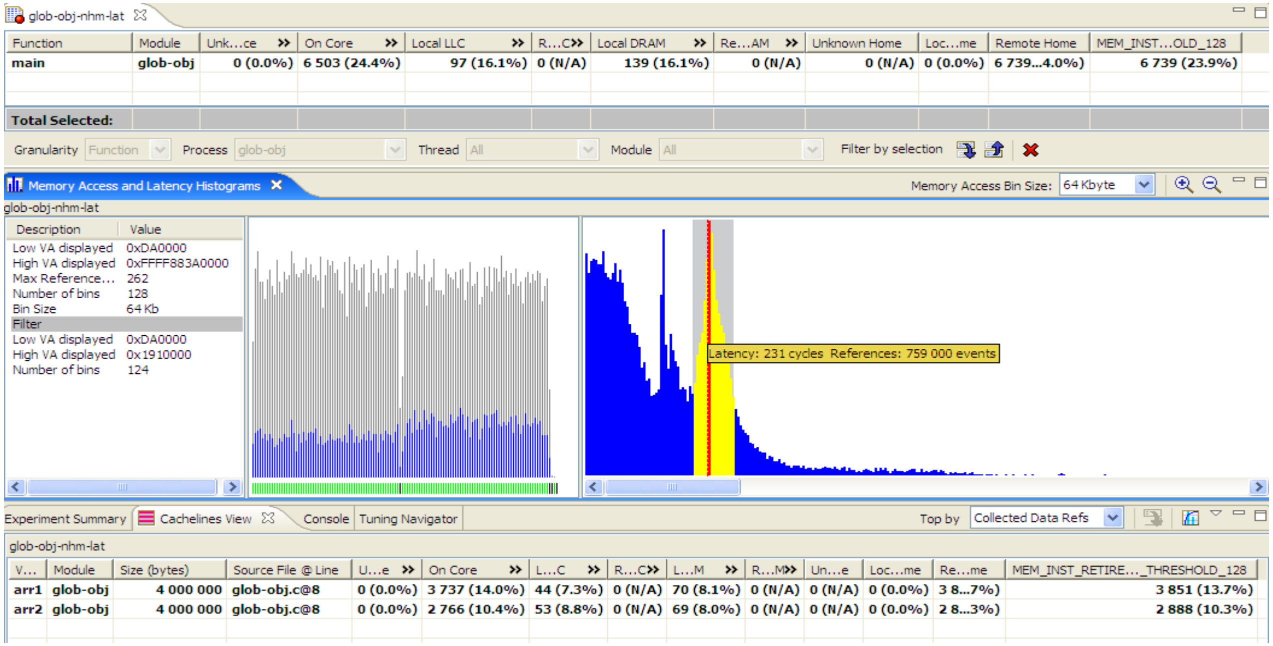
Address Range: | Stride: 100

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

Performance comparison difference may be due to Cache Size

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)



glob-obj-nhm-lat

Function	Module	Unk…ce	On Core	Local LLC	R…C	Local DRAM	Re…AM	Unknown Home	Loc…me	Remote Home	MEM_INST…OLD_128
main	glob-obj	0 (0.0%)	6 503 (24.4%)	97 (16.1%)	0 (N/A)	139 (16.1%)	0 (N/A)	0 (N/A)	0 (0.0%)	6 739…4.0%)	6 739 (23.9%)
Total Selected:											

Granularity: Function Process: glob-obj Thread: All Module: All Filter by selection

Memory Access and Latency Histograms Memory Access Bin Size: 64 Kbyte

glob-obj-nhm-lat

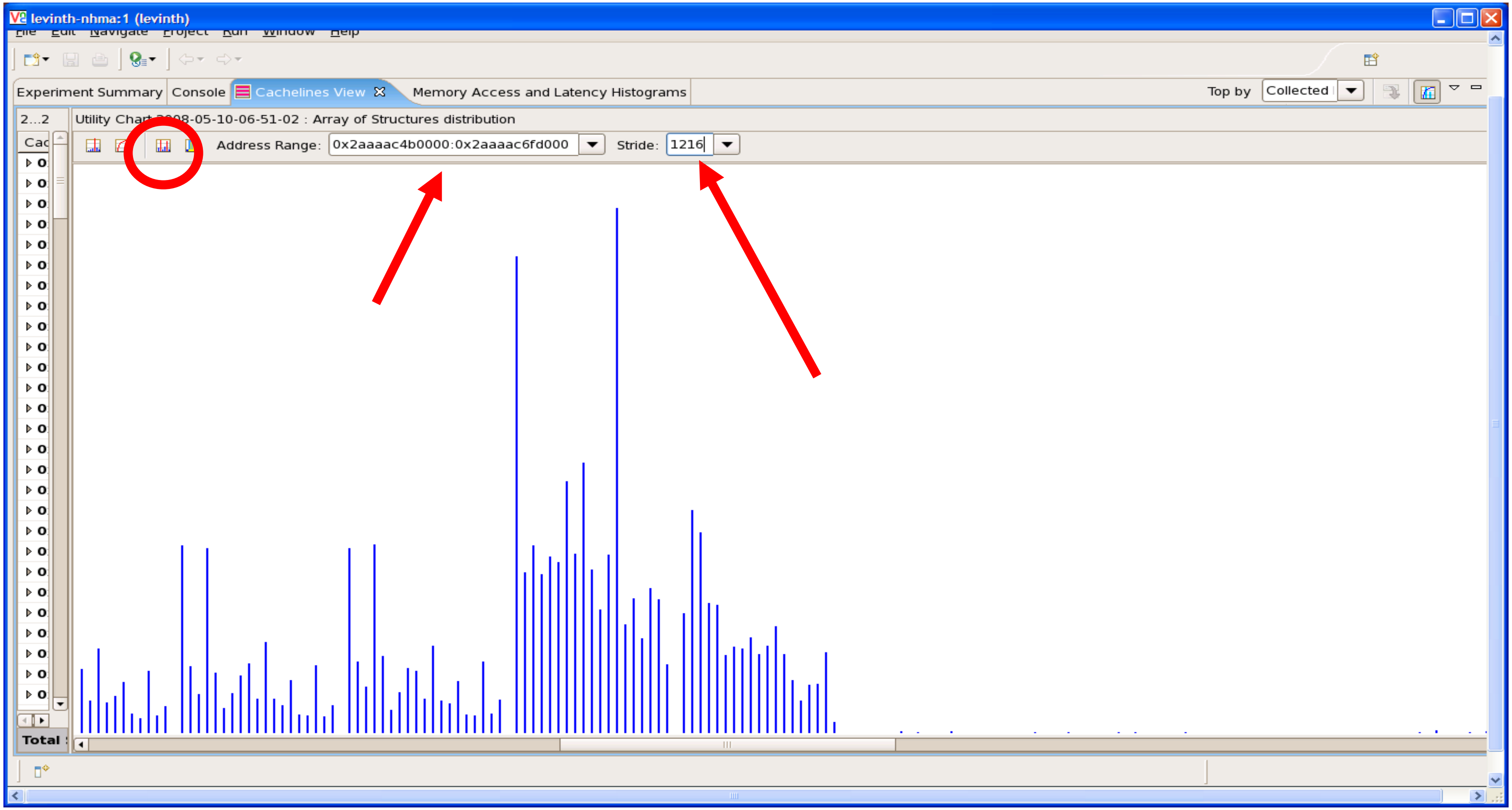
Description	Value
Low VA displayed	0xDA0000
High VA displayed	0xFFFF883A0000
Max Reference…	262
Number of bins	128
Bin Size	64 Kb
Filter	
Low VA displayed	0xDA0000
High VA displayed	0x1910000
Number of bins	124

Experiment Summary | Cachelines View | Console | Tuning Navigator Top by: Collected Data Refs

glob-obj-nhm-lat

V…	Module	Size (bytes)	Source File @ Line	U…e	On Core	L…C	R…C	L…M	R…M	Un…e	Loc…me	Re…me	MEM_INST_RETIRE…_THRESHOLD_128
arr1	glob-obj	4 000 000	glob-obj.c@8	0 (0.0%)	3 737 (14.0%)	44 (7.3%)	0 (N/A)	70 (8.1%)	0 (N/A)	0 (N/A)	0 (0.0%)	3 8…7%)	3 851 (13.7%)
arr2	glob-obj	4 000 000	glob-obj.c@8	0 (0.0%)	2 766 (10.4%)	53 (8.8%)	0 (N/A)	69 (8.0%)	0 (N/A)	0 (N/A)	0 (0.0%)	2 8…3%)	2 888 (10.3%)

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

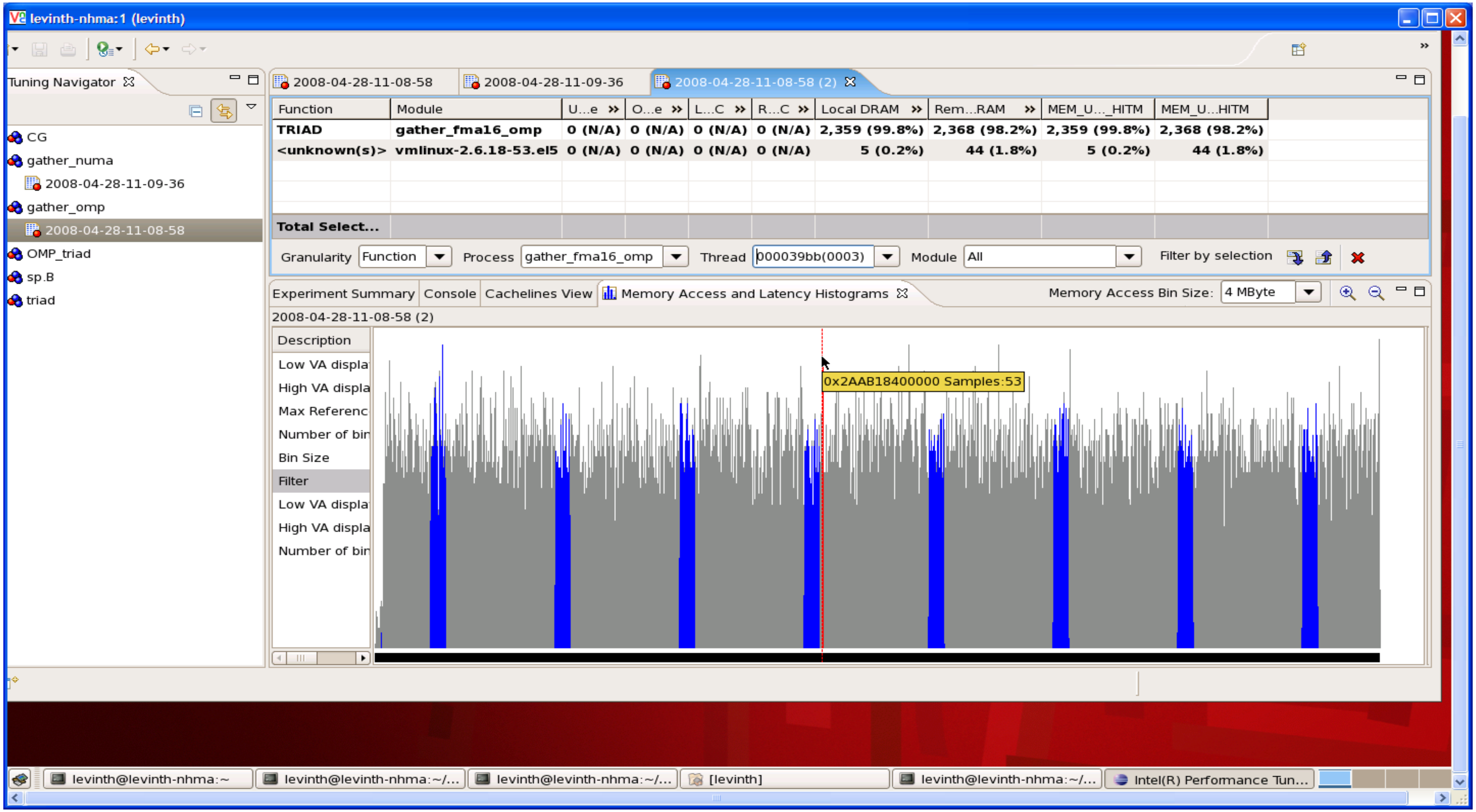


Utility Chart 2008-05-10-06-51-02 : Array of Structures distribution

Address Range: 0x2aaaac4b0000:0x2aaaac6fd000

Stride: 1216

Filtering to a Single Thread Displays the Data Decomposition



levinth-nhma:1 (levinth)

Tuning Navigator

- CG
- gather_numa
  - 2008-04-28-11-09-36
- gather_omp
  - 2008-04-28-11-08-58
- OMP_triad
- sp.B
- triad

2008-04-28-11-08-58 | 2008-04-28-11-09-36 | 2008-04-28-11-08-58 (2)

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%)
Total Select...									

Granularity: Function | Process: gather_fma16_omp | Thread: 000039bb(0003) | 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)

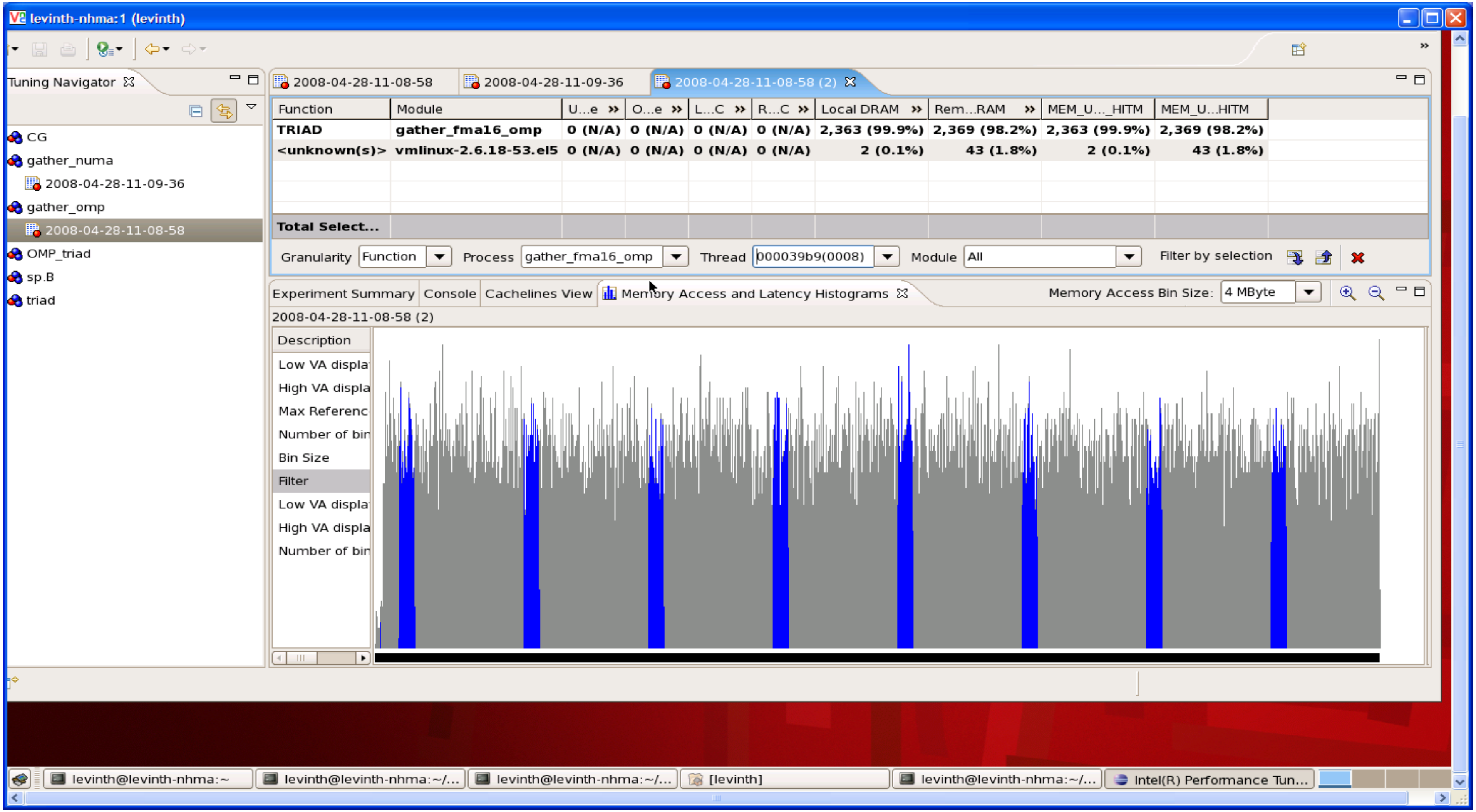
Description

- Low VA displa
- High VA displa
- Max Referenc
- Number of bir
- Bin Size
- Filter
- Low VA displa
- High VA displa
- Number of bir

0x2AAB18400000 Samples:53

levinth@levinth-nhma:~
levinth@levinth-nhma:~/...
levinth@levinth-nhma:~/...
[levinth]
levinth@levinth-nhma:~/...
Intel(R) Performance Tun...

A Different Thread



levinth-nhma:1 (levinth)

Tuning Navigator

- CG
- gather_numa
  - 2008-04-28-11-09-36
- gather_omp
  - 2008-04-28-11-08-58
- OMP_triad
- sp.B
- triad

2008-04-28-11-08-58 | 2008-04-28-11-09-36 | 2008-04-28-11-08-58 (2)

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%)
Total Select...									

Granularity: Function | Process: gather_fma16_omp | Thread: 000039b9(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

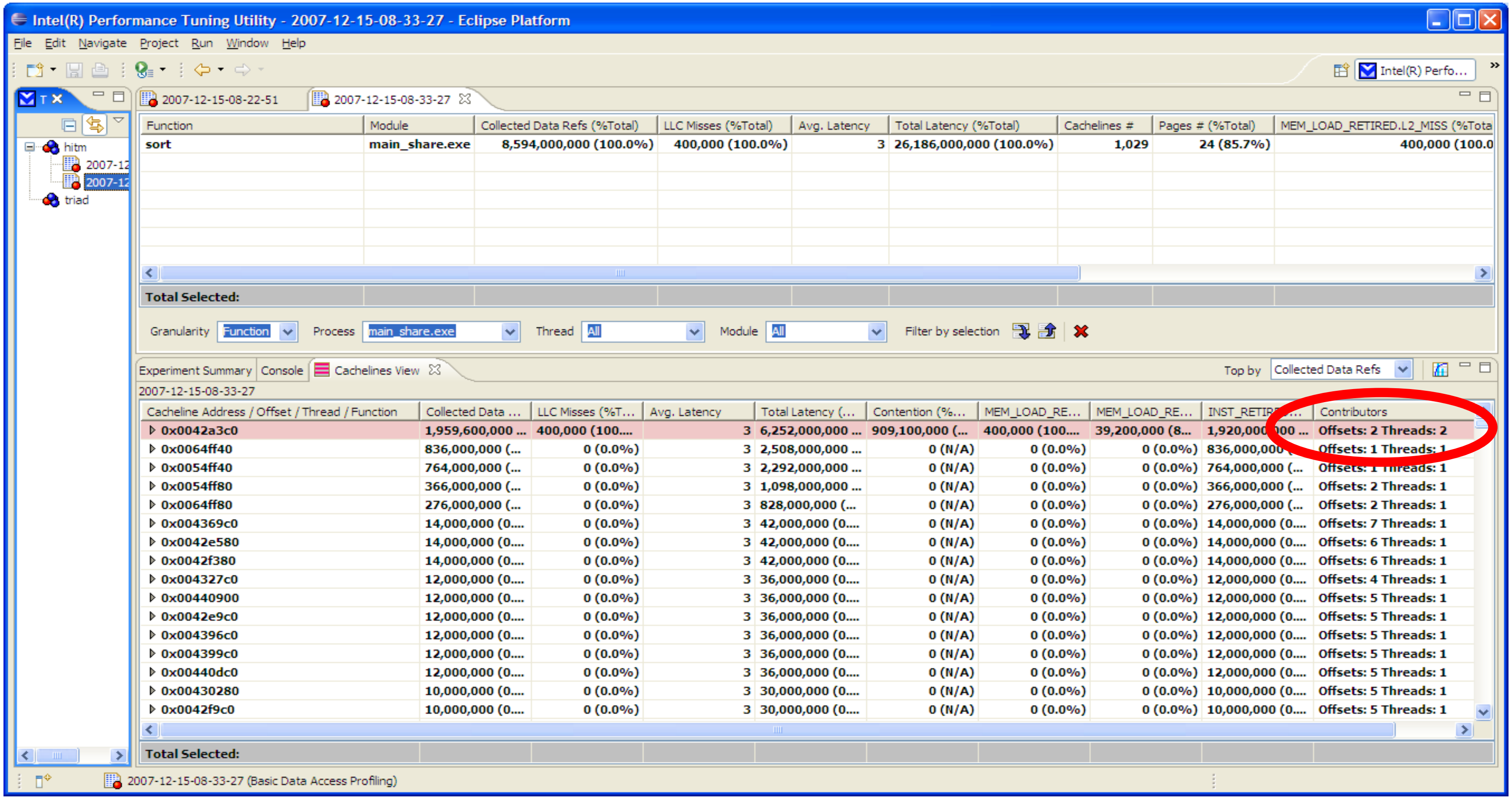
levinth@levinth-nhma:~
levinth@levinth-nhma:~/...
levinth@levinth-nhma:~/...
[levinth]
levinth@levinth-nhma:~/...
Intel(R) Performance Tun...

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
Intel(R) Perfo...
hitm
2007-12
2007-12
triad
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 (%Tota
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

Top by: Collected Data Refs

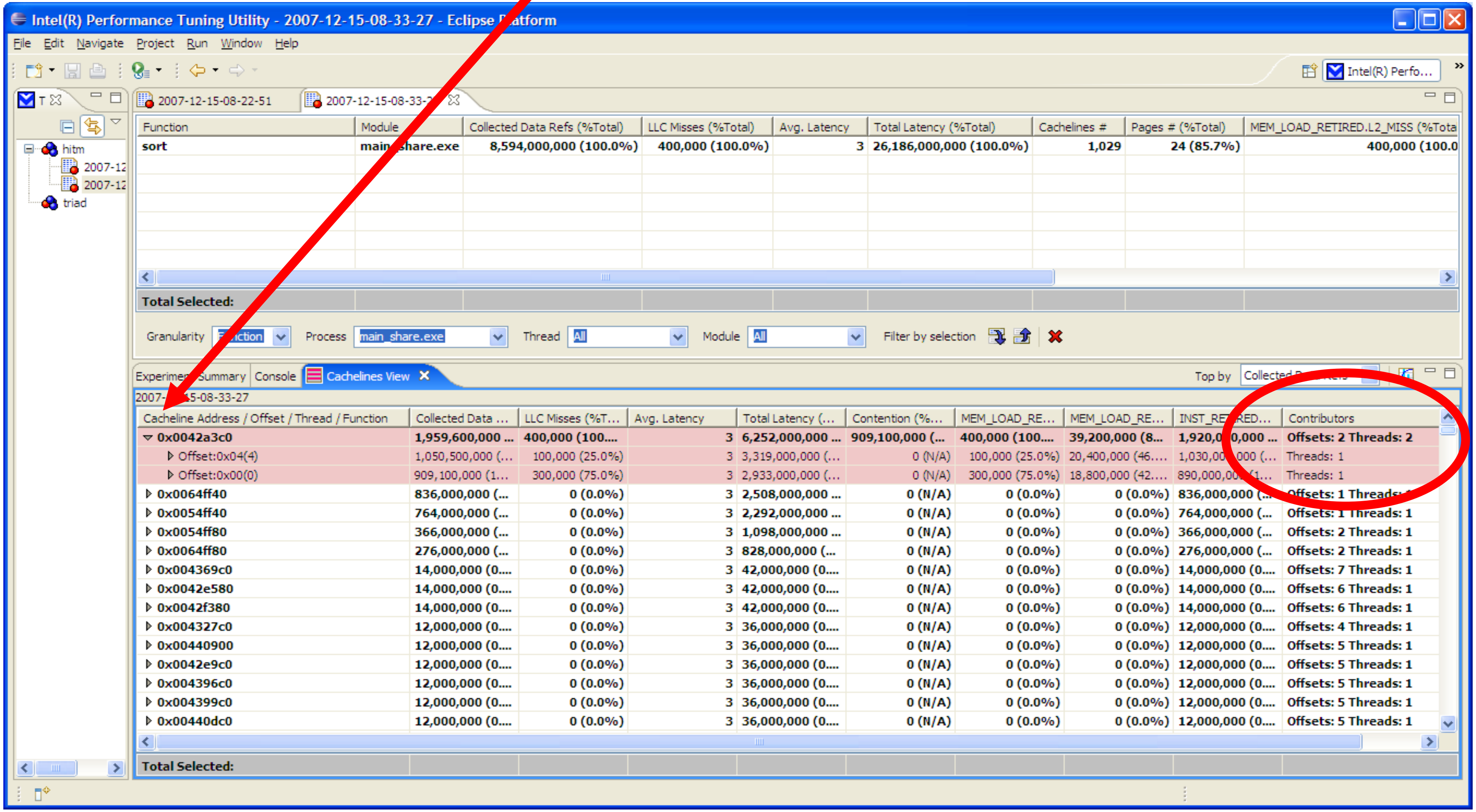
2007-12-15-08-33-27

Cacheline Address / Offset / Thread / Function	Collected Data ...	LLC Misses (%T...	Avg. Latency	Total Latency (...	Contention (%...	MEM_LOAD_RE...	MEM_LOAD_RE...	INST_RETIR...	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



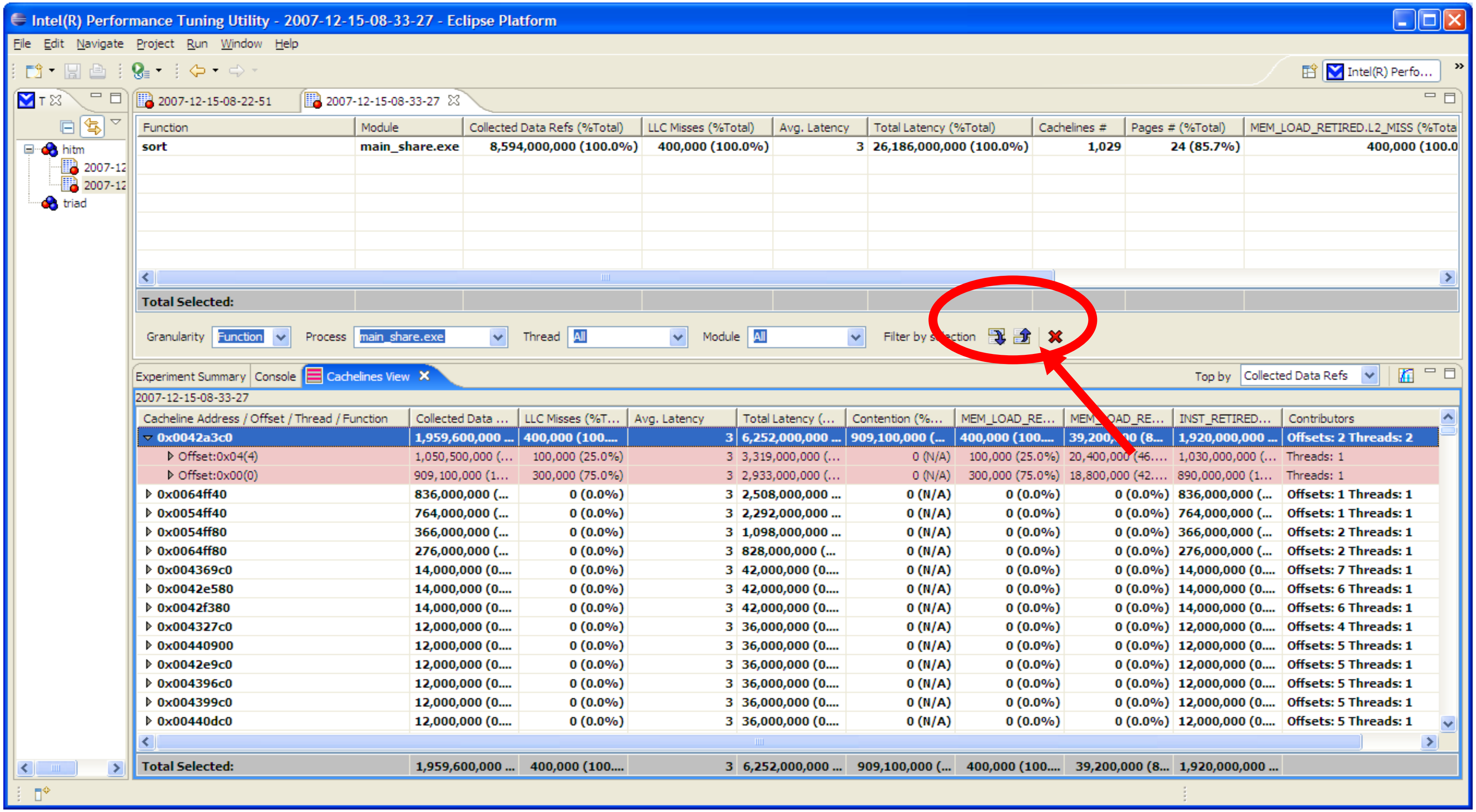
Function	Module	Collected Data Refs (%Total)	LLC Misses (%Total)	Avg. Latency	Total Latency (%Total)	Cachelines #	Pages # (%Total)	MEM_LOAD_RETIRED.L2_MISS (%Tota
sort	main_share.exe	8,594,000,000 (100.0%)	400,000 (100.0%)	3	26,186,000,000 (100.0%)	1,029	24 (85.7%)	400,000 (100.0
Total Selected:								

Granularity: Function; Process: main_share.exe; Thread: All; Module: All

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
▷ 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
Total Selected:									

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



Intel(R) Performance Tuning Utility - 2007-12-15-08-33-27 - Eclipse Platform
File Edit Navigate Project Run Window Help
Intel(R) Perfo...
T
hitm
2007-12
2007-12
triad
2007-12-15-08-22-51
2007-12-15-08-33-27

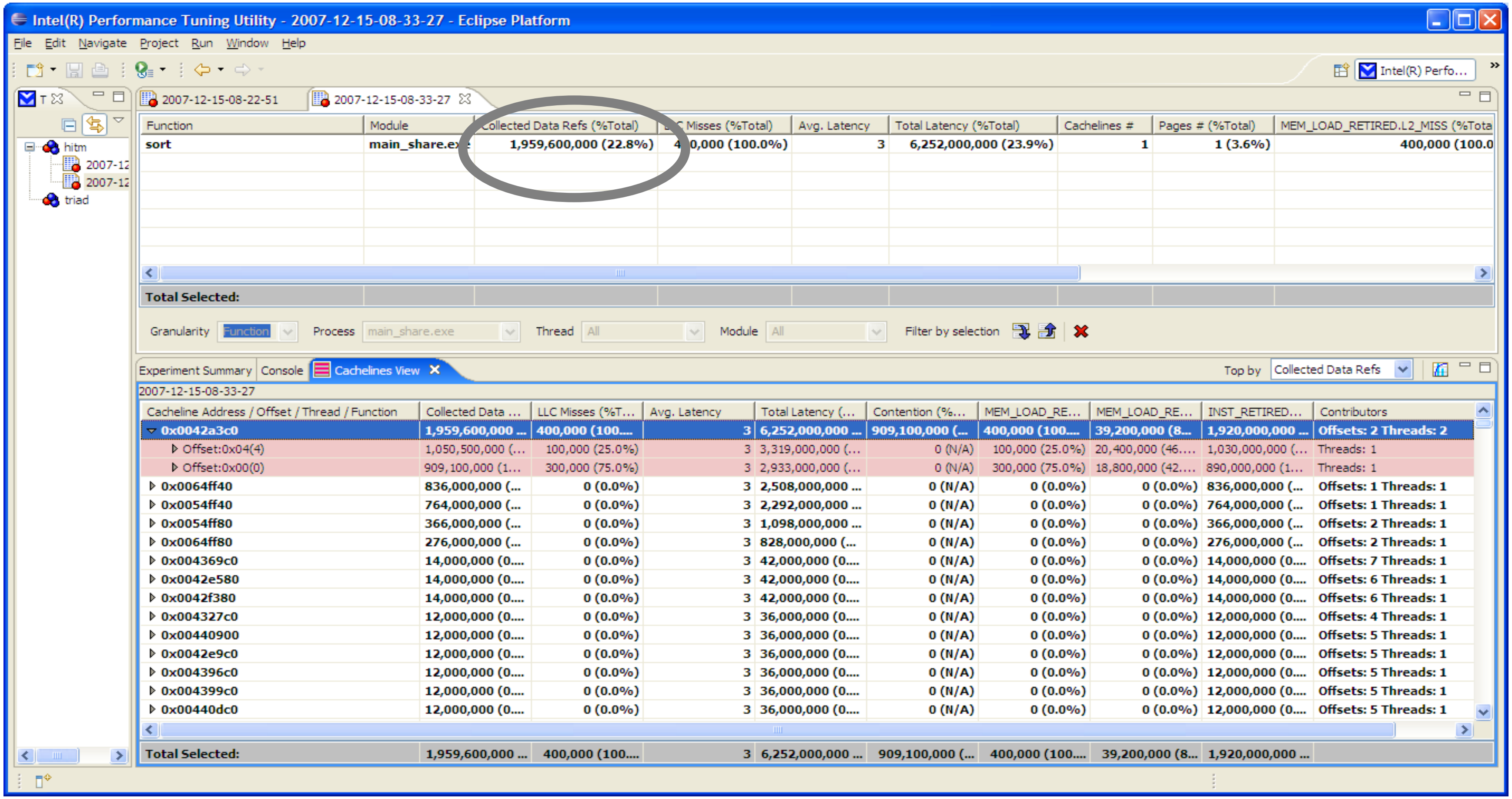
Function	Module	Collected Data Refs (%Total)	LLC Misses (%Total)	Avg. Latency	Total Latency (%Total)	Cachelines #	Pages # (%Total)	MEM_LOAD_RETIRED.L2_MISS (%Tota
sort	main_share.exe	8,594,000,000 (100.0%)	400,000 (100.0%)	3	26,186,000,000 (100.0%)	1,029	24 (85.7%)	400,000 (100.0
Total Selected:								

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

2007-12-15-08-33-27

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
Total Selected:	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 ...	

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



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

Function	Module	Collected Data Refs (%Total)	LLC Misses (%Total)	Avg. Latency	Total Latency (%Total)	Cachelines #	Pages # (%Total)	MEM_LOAD_RETIRED.L2_MISS (%Tota
sort	**main_share.ex**	**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**
Total Selected:								

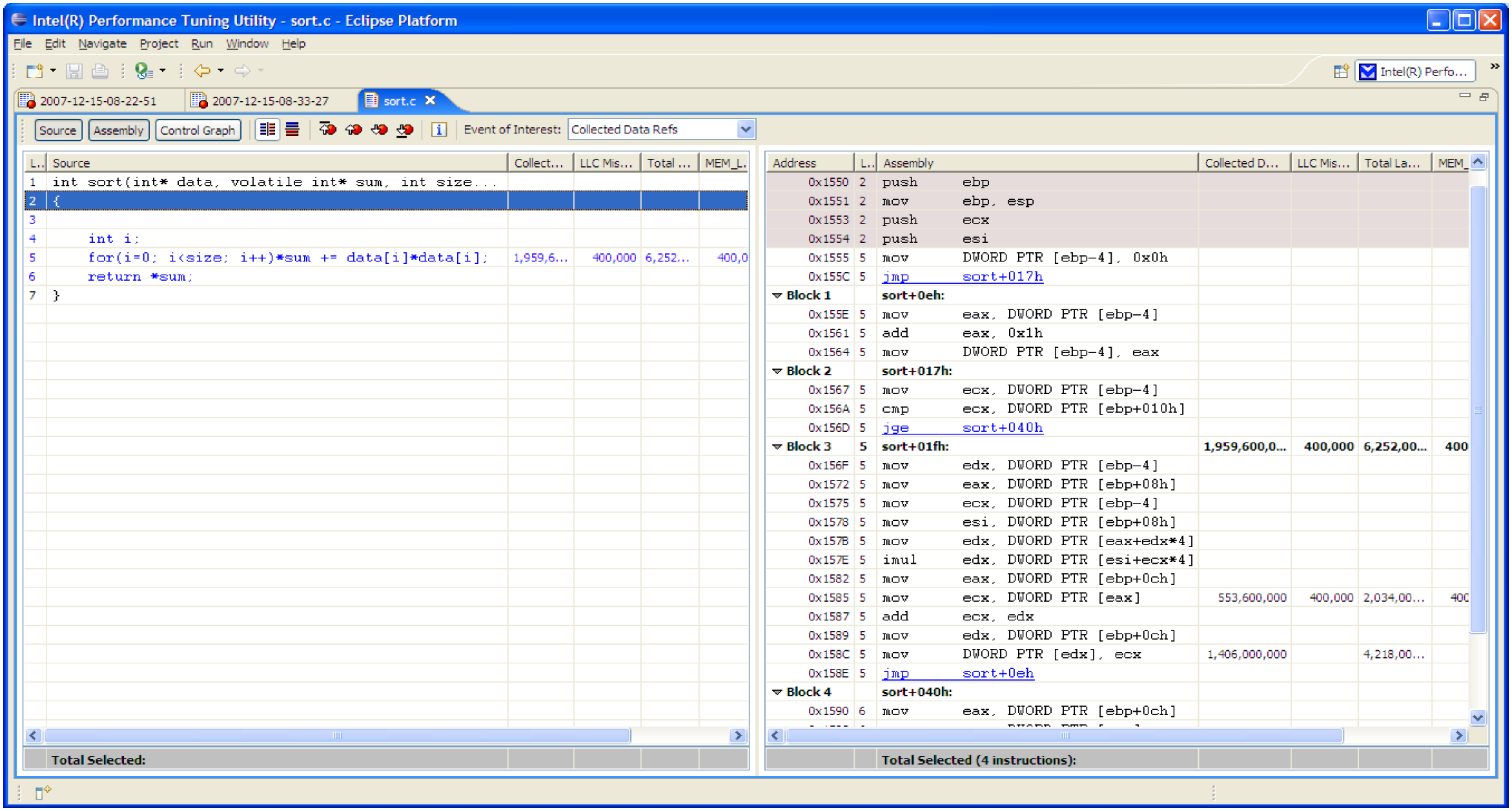
Granularity: Function | Process: main_share.exe | Thread: All | Module: All | Filter by selection

Top by: Collected Data Refs

2007-12-15-08-33-27

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**
Total Selected:	**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 ...**	

The Pointer “sum” is Causing the False Sharing


Intel(R) Performance Tuning Utility - sort.c - Eclipse Platform
File Edit Navigate Project Run Window Help
Intel(R) Perfo...
2007-12-15-08-22-51
2007-12-15-08-33-27
sort.c
Source
Assembly
Control Graph

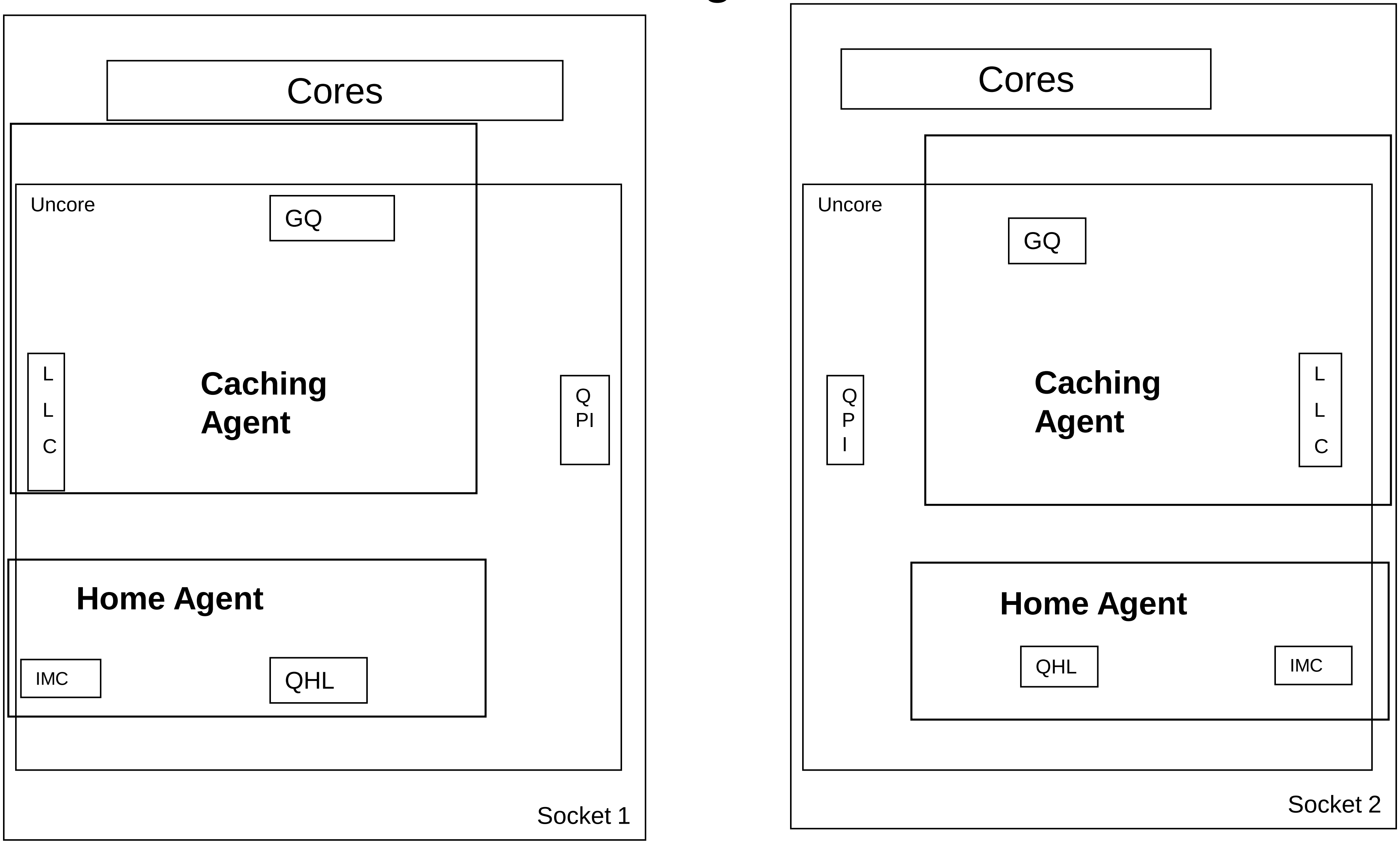
Event of Interest: Collected Data Refs

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	`}`				
	Total Selected:				

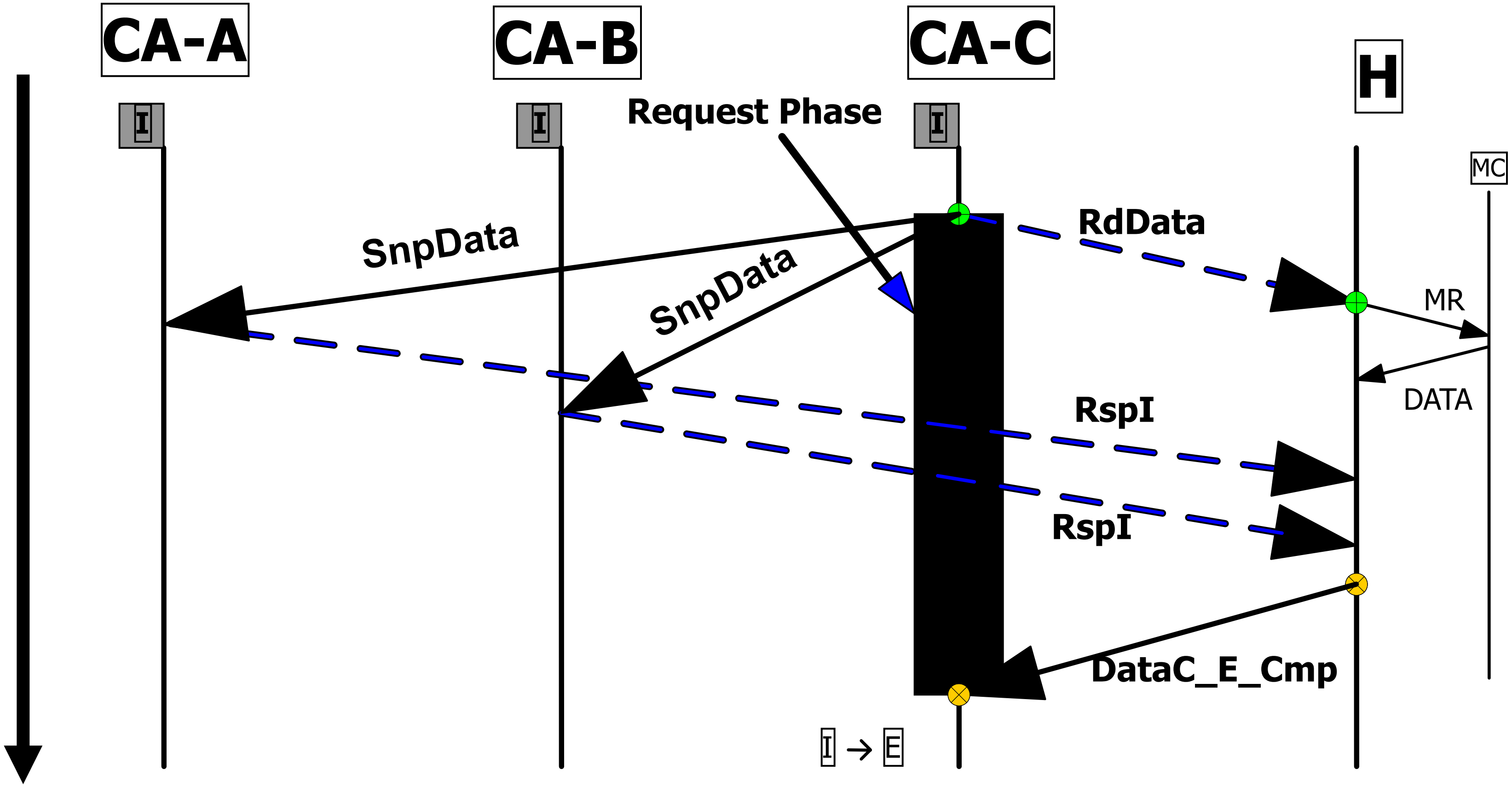
Address	L..	Assembly	Collected D...	LLC Mis...	Total La...	MEM_
0x1550	2	push ebp				
0x1551	2	mov ebp, esp				
0x1553	2	push ecx				
0x1554	2	push esi				
0x1555	5	mov DWORD PTR [ebp-4], 0x0h				
0x155C	5	jmp sort+017h				
▽ **Block 1**		**sort+0eh:**				
0x155E	5	mov eax, DWORD PTR [ebp-4]				
0x1561	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**	**5**	**sort+01fh:**	**1,959,600,0...**	**400,000**	**6,252,00...**	**400**
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+0eh				
▽ **Block 4**		**sort+040h:**				
0x1590	6	mov eax, DWORD PTR [ebp+0ch]				
		Total Selected (4 instructions):				

NUMA cacheline access

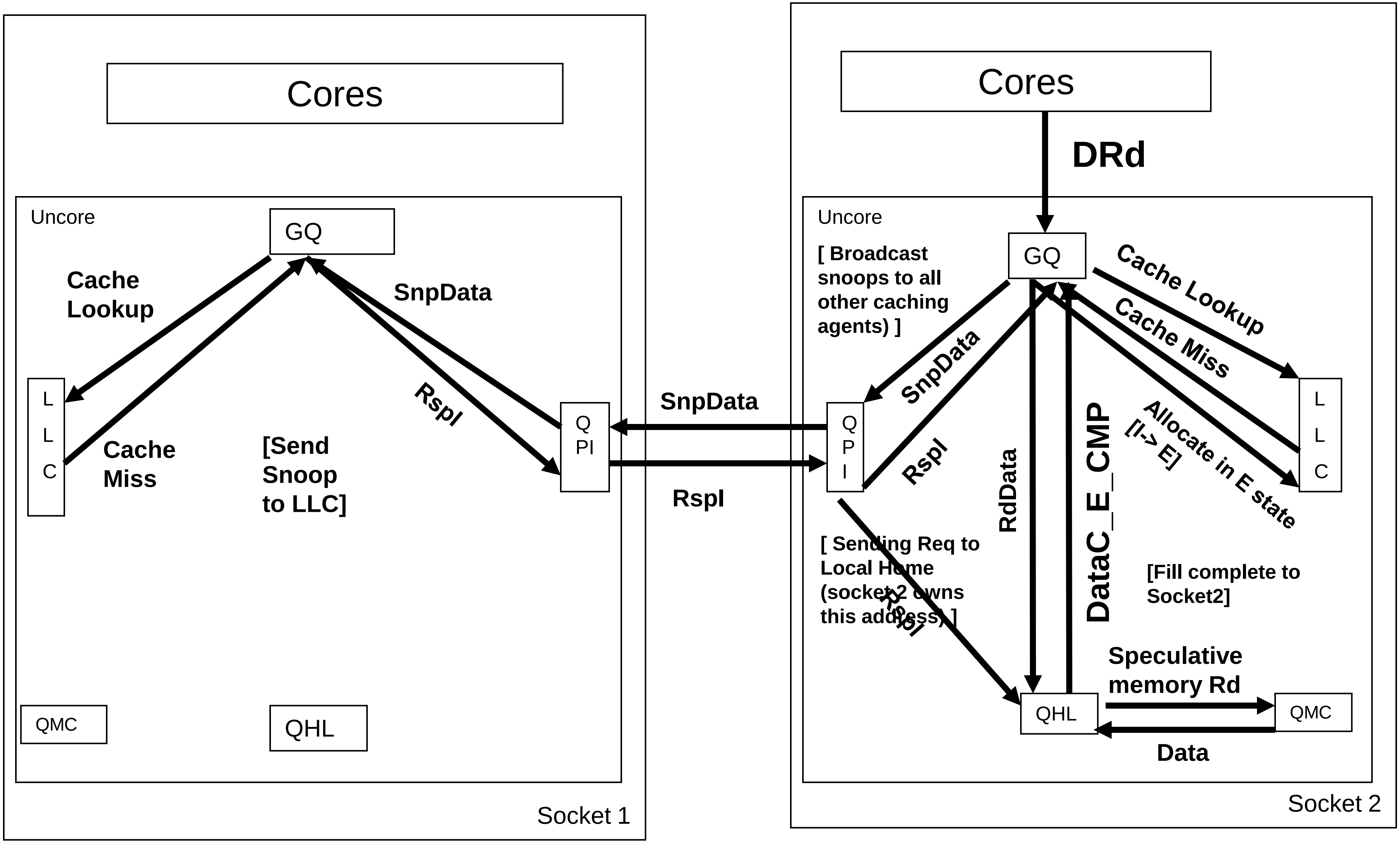
A NHM Socket is a Caching Agent and a Home Agent



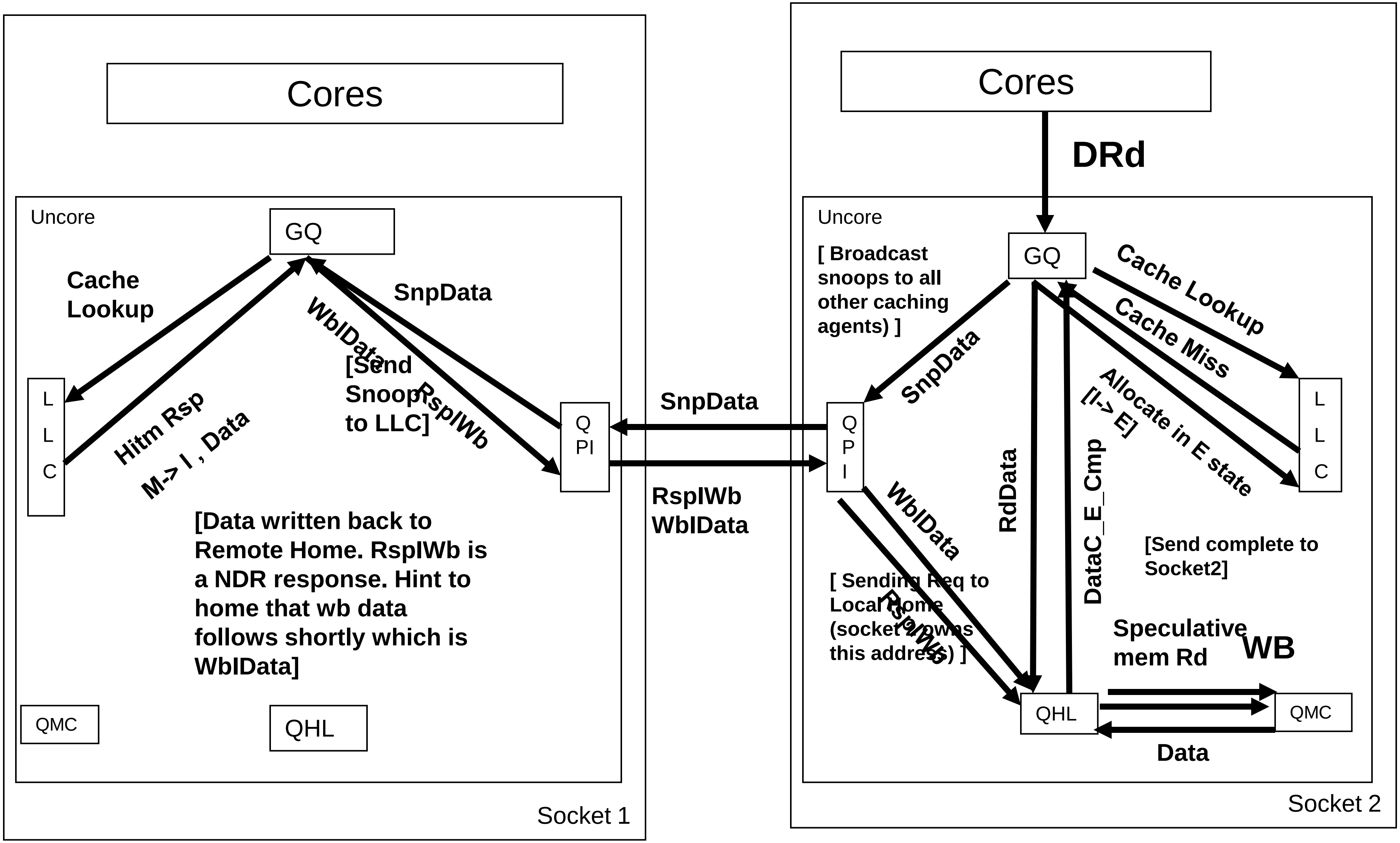
Simple Data Read



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)

Summary

- Event based sampling performance analysis is extremely powerful on Intel® Core™ i7, XEON™ 5500 and 5600 Processor Families
- Correct methodology is essential
- Correct usage of events is essential
- Intel® PTU simplifies task

backup

Low level utilities

- PTU low level utilities can be invoked from the command line by adding the PTU bin directory to the path
- Low level PMU collector is SEP
  - Invoked by vtsarun
  - Data is stored in file called tbsXXXYYYY.tb5
  - sep –start –ex 16 –ec "CPU_CLK_UNHALTED.THREAD:sa=2000000,UOPS_RETIRED.ANY,UOPS_RETIRED.STALL_CYCLES" –app ./myapp –args " arg1 arg2"
    - :sa=VAL explicitly sets SAV value for the event preceding it
    - -ex 16 causes sep to add PEBS buffer to event record
      - Selecting data profile does the same thing

Low level utilities

- sep –start –ex 16 –ec “CPU_CLK_UNHALTED.THREAD:sa=2000000,UOPS_RETIRED.ANY,UOPS_RETIRED.STALL_CYCLES,BR_INST_RETIRED.NEAR_CALL:lbr=2” –app ./myapp –args “ arg1 arg2”
  - Event names must be upper case
  - :lbr=VAL turns on LBR capture with filter value determined by VAL
    - Filter values can be determined with profile editor and show command button

LBR Value	Filter Result
1	All Branches
2	All Calls
3	User Calls
4	All Calls & Ret
5	User Calls & Ret

Low level utilities

- sfdump5 creates test output based on data in tb5 file
- sfdump5 tbsXXXZZZ.tb5 –modules > modules.txt
  - Summary of data
    - Total number of samples and events=samples*SAV
      - Events ordered by “event number”
    - Total number of samples/module/event_type

Example sfdump5 output

```
Event Summary
CPU_CLK_UNHALTED.THREAD
   2396              = Samples collected due to this event
   2000000           = Sample after value used during collection
   4792000000        = Total events (samples*SAV)
INST_RETIRED.ANY
   1327              = Samples collected due to this event
   2000000           = Sample after value used during collection
   2654000000        = Total events (samples*SAV)

Module View (all values in decimal)

Module                              Process
  Event                                               Events%   Samples   Events          Module Path
------------------------------------------------------------------------------------------------------------------------------------
triad                               triad
  CPU_CLK_UNHALTED.THREAD                                       90.40%    2166    4332000000      /home/vtune/snb3/triad_src/triad
  INST_RETIRED.ANY                                              89.98%    1194    2388000000
vmlinux                             triad
  CPU_CLK_UNHALTED.THREAD                                       4.47%     107     214000000       vmlinux
  INST_RETIRED.ANY                                              4.97%     66      132000000
```

- **Thus CPU_CLK_UNHALTED.THREAD is event 0   "ei-00"**
- **Thus Inst_RETIRED.ANY is event 1   "ei-01"**

Low level utilities

- Sfdump5 tbsXXXZZZ.tb5 /dumpsamples > samples.txt
  - Text dump of all samples
  - All sample records in a given file are same length
  - Length = SUM of all required fields for all events
    - If PEBS record is collected for PEBS events, the corresponding fields exist for non PEBS event but are zero filled
    - Events with LBR collection are only collected with other events that have SAME LBR filter value
      - 33 X 64 bits are added

/dumpsamples example output

```
00000208   64--0033:0x0000000000400DF9-0  p-0x0000231C     c-00 t-0x0000231C     sgno-
   0x00000001  ei-00  tsc-0x0003C06F0CF15DD4   triad
```

- 00000208 is the record number
- 64--0033:0x0000000000400DF9-0 tells you this is a 64 bit binary and the IP of the interupt was 0x0000000000400DF9
- p-0x0000231C   gives the process ID
- c-00   the core number of the interupt   in this case 0
- t-0x0000231C the thread ID
- ei-00   the event number
    - thus this is an record triggered by CPU_CLK_UNHALTED.THREAD
    - See –modules output to determine event numbers for a particular collection
- tsc-0x0003C06F0CF15DD4   the Time Stamp Counter
- Triad the load module name

/dumpsamples example output LBRs

```
00000091   64--0033:0x0000000000400694-0  p-0x00000A0A      c-00 t-0x00000A0A      sgno-
    0x00000001  ei-00  tsc-0x000000C43DECAFB1   extra_00-0x0000000000000006 extra_01-
    0x0000000000400A2C extra_02-0x00000000004009C4 extra_03-0x000000000040095C extra_04-
    0x00000000004008E6 extra_05-0x000000000040086E extra_06-0x0000000000400806 extra_07-
    0x000000000040074A extra_08-0x00000000004006E2 extra_09-0x0000000000401061 extra_10-
    0x0000000000400D7F extra_11-0x0000000000400D97 extra_12-0x0000000000400C52 extra_13-
    0x0000000000400BEC extra_14-0x0000000000400B84 extra_15-0x0000000000400AFC extra_16-
    0x0000000000400A94 extra_17-0x0000000000400976 extra_18-0x000000000040090E extra_19-
    0x0000000000400888 extra_20-0x0000000000400820 extra_21-0x00000000004007B8 extra_22-
    0x00000000004006FC extra_23-0x0000000000400694 extra_24-0x0000000000400648 extra_25-
    0x0000000000400D38 extra_26-0x0000000000400CC2 extra_27-0x0000000000400C06 extra_28-
    0x0000000000400B9E extra_29-0x0000000000400B36 extra_30-0x0000000000400AAE extra_31-
    0x0000000000400A46 extra_32-0x00000000004009DE call_chain
```

- record number is 91
- Event number (ei) is 0
- Extra_01 -> extra_16 are the branch source addresses
- Extra_17 -> extra_32 are the branch target addresses
- extra_00 points to the most recent LBR source entry
    - In this case extra_06
- Most recent target is extra_(extra_00+17)
    - Thus last target is extra_23 = extra_23-0x0000000000400694
    - And PEBS IP field is = 64--0033:0x0000000000400694-0

/dumpsamples example output PEBS

```
00000445   64--0033:0x0000000000401665-0  p-0x00000978      c-00 t-0x00000978      sgno-
    0x00000001  ei-00  tsc-0x0000011CF7198F6F   extra_00-0x0000000000000202 extra_01-
    0x0000000000401665 extra_02-0x00000123F1DE149A extra_03-0x0000000000000001 extra_04-
    0x0000000000000000 extra_05-0x00000123F1DE149A extra_06-0x000000001B4E4355 extra_07-
    0x000000004ABCE4E1 extra_08-0x00007FFFA989B710 extra_09-0x00007FFFA989B6A0 extra_10-
    0x0000000000000000 extra_11-0x0000000000000001 extra_12-0x00007FFFA989B400 extra_13-
    0x0000003731E97DD0 extra_14-0x0000000000400720 extra_15-0x00007FFFA989B860 extra_16-
    0x0000000000000000 extra_17-0x0000000000000000 extra_18-0x00007FFFA989B6F8 extra_19-
    0x0000000000000041 extra_20-0x0000000000000038 extra_21-0x000000000000FFFF extra_22-
    0x0000000000000000 store_fwd_lnx2
```

•Event number (ei) is 0 (in this case the latency event)

•Extra_01 is Event IP

  •IP of instruction after the instruction that caused the interupt (“IP+1”)

•Extra_02-> extra_17 are the register values at the completion of the offending instruction

PEBS Buffer field definitions

(x)->r_flags	//extra_00
(x)->linear_ip	//extra_01
(x)->rax	//extra_02
(x)->rbx	//extra_03
(x)->rcx	//extra_04
(x)->rdx	//extea_05
(x)->rsi	//extra_06
(x)->rdi	//extra_07
(x)->rbp	//extra_08
(x)->rsp	//extra_09
(x)->r8	//extra_10
(x)->r9	//extra_11
(x)->r10	//extra_12
(x)->r11	//extra_13
(x)->r12	//extra_14
(x)->r13	//extra_15
(x)->r14	//extra_16
(x)->r15	//extra_17
(x)->data_linear_address	//extra_18
(x)->data_source	//extra_19
(x)->latency	//extra_20

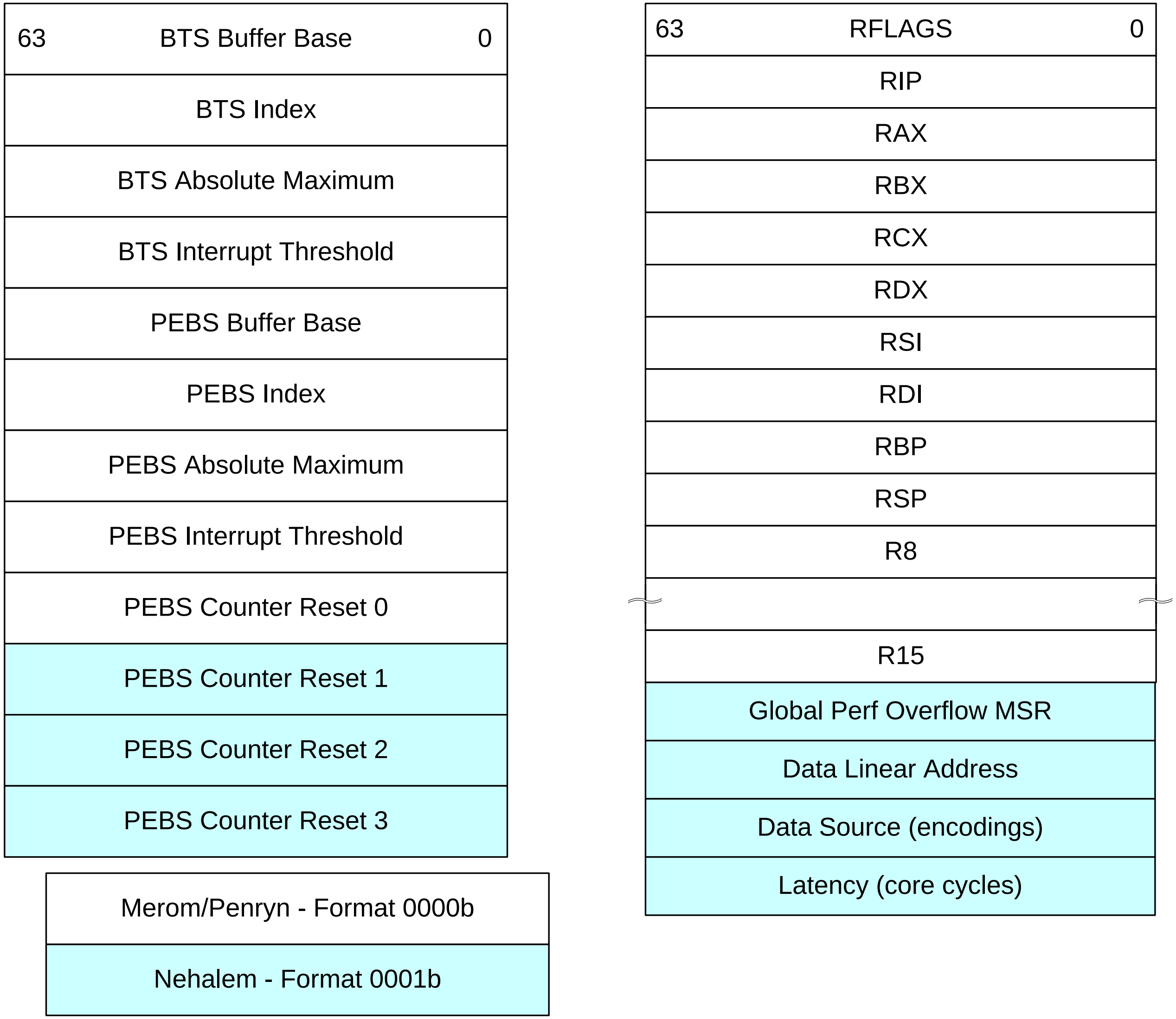
Precise Events

- 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

R15

Global Perf Overflow MSR

Data Linear Address

Data Source (encodings)

Latency (core cycles)

Load Latency Threshold Event:

- **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**

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**

NUMA, Intel® QuickPath Interconnect, and Intel ® Xeon 5500/5600 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 ~2X larger

Pealing 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

Gather and OOO execution

	no prefetch	pref = 8	pref = 16	pref = 32	pref = 64	pref = 96
2 fp ops	34.5	34.9	34.2	37.2	38.7	38.9
4 fp ops	44.5	34.5	33.6	38	42.2	41.4
8 fp ops	74.8	34.8	34.1	38.7	42.7	41.7
16 fp ops	108.9	34.6	34	42.2	50.9	45.6

Data collected on Core™ 2 processor, prefetchers on

Glossary

- PMU: Performance Monitoring Unit
  - Assembly of counters and programmable crossbars that allow counting and profiling using user selectable events
- FE: core pipeline Front End
  - Responsible for branch prediction, instruction fetch, decode to uops, allocation of OOO backend resources
- BE: core pipeline Backend
  - Stage uops waiting for inputs, execute upon availability, retire in order

Glossary

- RS: reservation station
  - Where uops are staged for execution waiting for availability of their inputs
- ROB: Reorder Buffer
  - Where uops wait prior to retirement until all older uops have retired and execution path is confirmed. Second point corrects when uops are executed on a mispredicted path.
- RAT: Resource Allocation Table
  - Allocates BE resources for uops prior to issuing them from front end of pipeline to the backend

Glossary

- Cachelines are 64 bytes
- LLC: Last level Cache
  - L3 on these processors
- LFB: line fill buffer
  - Buffers used for transfering cachelines into and out of L1D
- WB: writeback
  - Modified data is written back to higher level in memory subsystem on eviction
- RFO: Read for Ownership
  - Stores require cachelines are in exclusive ownership state so they can be modified

Glossary

- Prefetch, by hardware (HW) or by explicit instruction (SW)
  - Request cacheline prior to execution of consuming instruction (load/store) with intention of hiding latency
- BW: bandwidth
  - Data moved/unit time. I prefer cachelines/cycle as that is what is measured
- Latency: time required to transfer a single line from source to usage.

Glossary

- SIMD: Single instruction multiple data
  - SSE parallel execution mode
  - AKA vectorization
- X87: legacy floating point computation mode. In contrast to SSE FP instructions
- NT: Non Temporal
  - Data store mode that writebacks data in 64 byte aligned contiguous 64 byte chunks directly to dram without RFO
- HITM: Hit Modified
  - Snoop response when line is found in modified state in another cache

Glossary

- HT: Intel® Hyper-threading Technology
  - Execution mode allowing uops from two threads to be executed in an intermingled flow, without an OS context switch, through a single core pipeline.
- Turbo: Intel® Turbo Boost Technology
  - Adjusting core frequency upwards on active core when other cores are under utilized, while staying within required power envelope. Enhances performance of single threaded execution