Event Based Analysis

Stephen Blair-Chappell

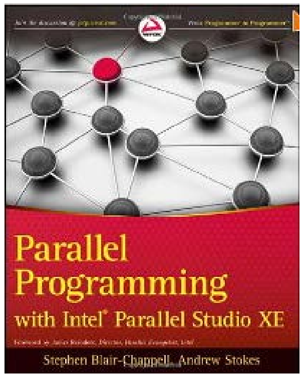
Intel Compiler Labs

This training relies on you owning a copy of the following…

Parallel Programming with Parallel Studio XE

Stephen Blair-Chappell & Andrew Stokes

Wiley ISBN: *9780470891650*


Parallel
Programming
with Intel Parallel Studio XE
Stephen Blair-Chappell, Andrew Stokes

VTune Amplifier is a simple tool

Imagine you have a cool car and you want to drive a little faster or fuel effective

All what you'd need you can find here.

VTune as other simple tools can provide basic information on performance of your engine.

VTune Amplifier is a complex tool

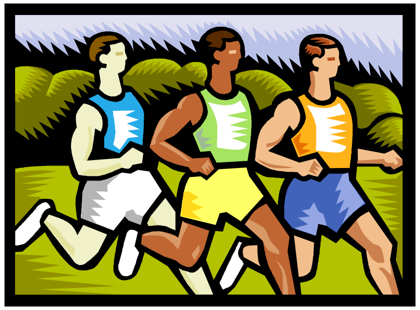
However, if you want your car to win a race…

Your tools set has to be much more complex to analyze all aspects of engine functioning.

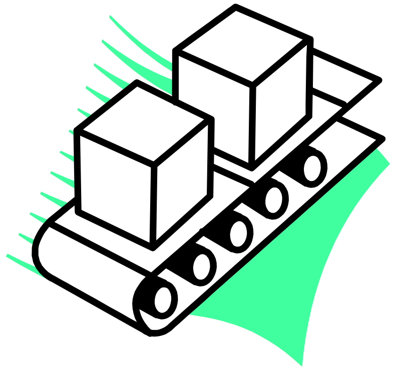
You need to be more proficient in both: the tool's functionality and the engine internals!

Testing the Health of an Application

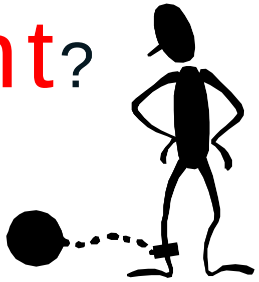
Does it run Fast?



Does it get through lots of work?



Is any part of the code inefficient?



A program’s performance

A program’s performance can be impacted by

- System-wide activity
- Application Heuristics
- CPU architecture

Any analysis
Should be
Done in
This order

Are you Sick?

Fever?

High Pressure?

Aches & Pains

Optimization Notice

‘Is my program unwell?’

- Number of Cycles (clock ticks) a program consumes
- Number of Retired Instructions
- CPI – Cycles Per Instruction
  - Num Cycles / Num retired instr
  - Low good, High bad
  - Theoretical best 0.25***
  - Anything below 1.00 pretty good.

****NOTE: Xeon Phi best CPI is 0.5*

Using CPI can be misleading

- Some optimisation steps can lead to an increase in CPI
- Always keep an eye on the fundamentals!
- How long did my program take to run?

All programs consume cycles

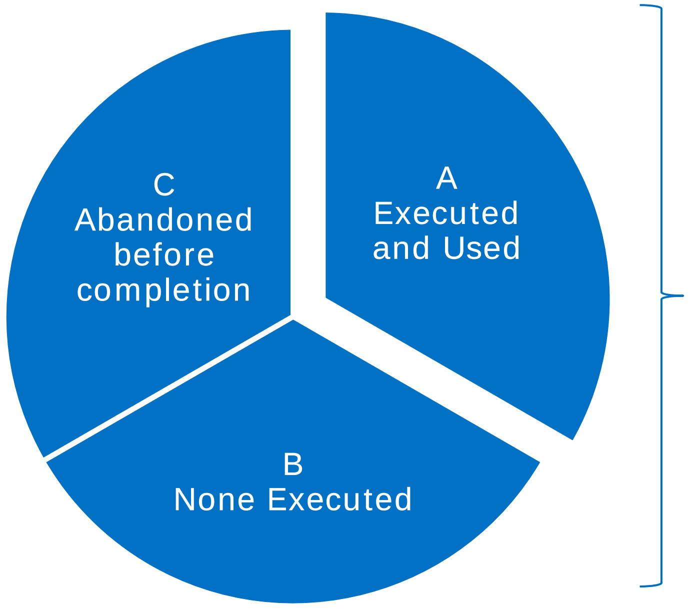
These cycles consist of

Cycles where instructions are usefully executed

Cycles when nothing happens

Cycles where instructions are executed, but the results never used

Goal of performance tuning is to reduce each of these



Total number of CPU cycles

Total Cycles Consumed = A + B + C

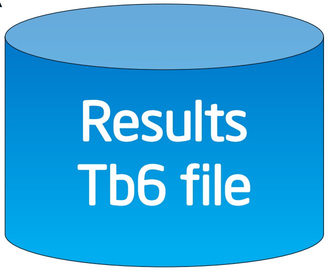
Optimization Notice

Using Amplifier XE

1 Profile

Collector

- Launch from Command line \ GUI *(Does not need GUI installed to run)*



2 View the Results

0 You can add API to your source code *(optional)*

```
#include <ittnotify.h>
int main()
{
  __itt_domain* pD = __itt_domain_create( "Time" );
  pD->flags = 1; // enable domain

  for(int i=0;i< 100000;i++)
  {
    // mark the begining of the frame
    __itt_frame_begin_v3( pD,NULL);

    // simulate frames with different timings
    if(i%3)
      for(int j =0; j < 30000; j++);  // a delay
    else
      for(int j =0; j < 11200; j++);  // another delay

    // mark the end of the frame
    __itt_frame_end_v3( pD,NULL);
  }
  return 0;
} ,
```

Some tips to get you up and running on VTune

Linux – Vtune is not recognised

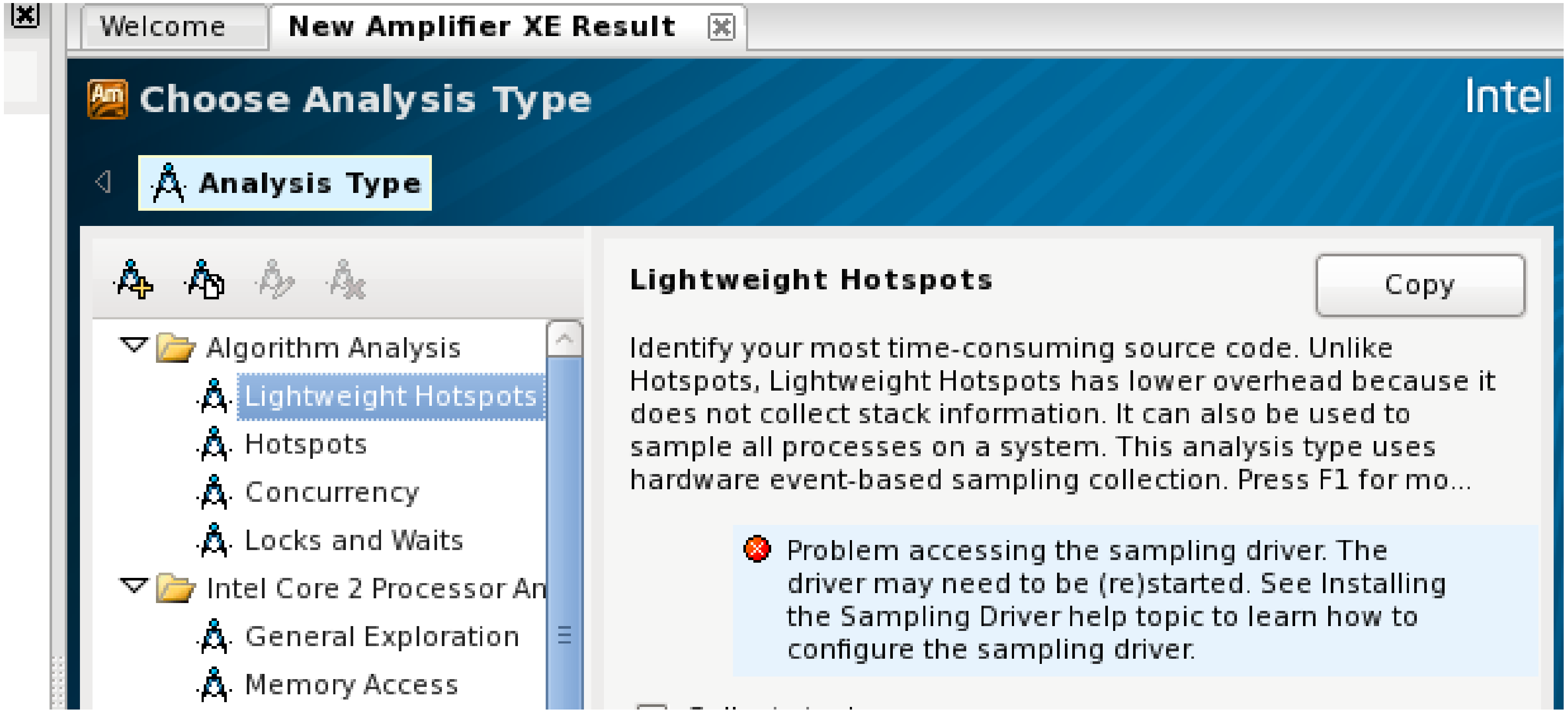
- You must ***source*** the path!

```
source /opt/intel/vtune_amplifier_xe/amplxe-var.sh
```

- To start from prompt.

```
Amplxe-gui &
```

Linux – problem accessing the sample driver!


Welcome | New Amplifier XE Result

Choose Analysis Type

Analysis Type

- Algorithm Analysis
  - Lightweight Hotspots
  - Hotspots
  - Concurrency
  - Locks and Waits
- Intel Core 2 Processor An
  - General Exploration
  - Memory Access

Lightweight Hotspots

Copy

Identify your most time-consuming source code. Unlike Hotspots, Lightweight Hotspots has lower overhead because it does not collect stack information. It can also be used to sample all processes on a system. This analysis type uses hardware event-based sampling collection. Press F1 for mo...

Problem accessing the sampling driver. The driver may need to be (re)started. See Installing the Sampling Driver help topic to learn how to configure the sampling driver.

Optimization Notice

Linux – problem accessing the sample driver!

1. Check if driver is loaded

```
cd /opt/intel/vtune_amplifier_xe/sepdk/prebuilt/
./insmod-sep3 –q
```

NOTE: The default installation expects users to be in the group ‘vtune’

2. If not reload it

<group>

```
./insmod-sep3 -r -g democenter
```

3. if watchdog is causing a problem, disable it

“Warning: NMI watchdog timer is enabled. Turn off the nmi_watchdog tim
before running sampling.”

```
echo 0 >  /proc/sys/kernel/nmi_watchdog
```

4. If not available, rebuild (see next slide)

Linux – problem accessing the sample driver!

4. If not available, rebuild

```
cd /opt/intel/vtune_amplifier_xe/sepdk/src
 ./build-driver -ni --install-dir=../prebuilt
```

5. Load the driver

```
cd ../prebuilt
./insmod-sep3 -r -g democenter
```

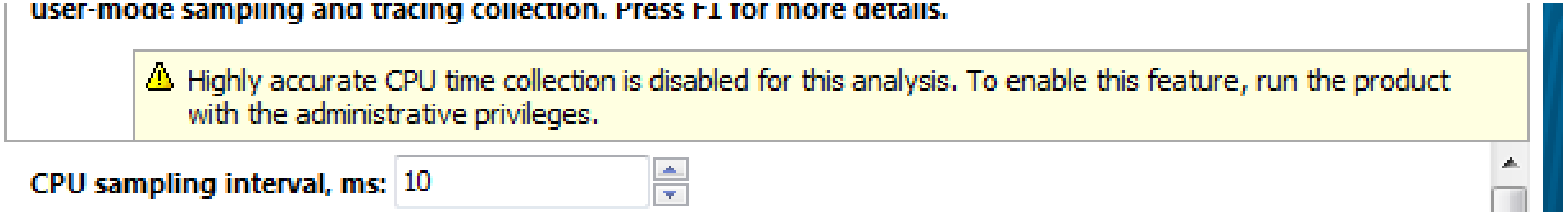
6. To load automatically on reboot

```
./boot-script --install -g democenter
```

<group>

Optimization Notice

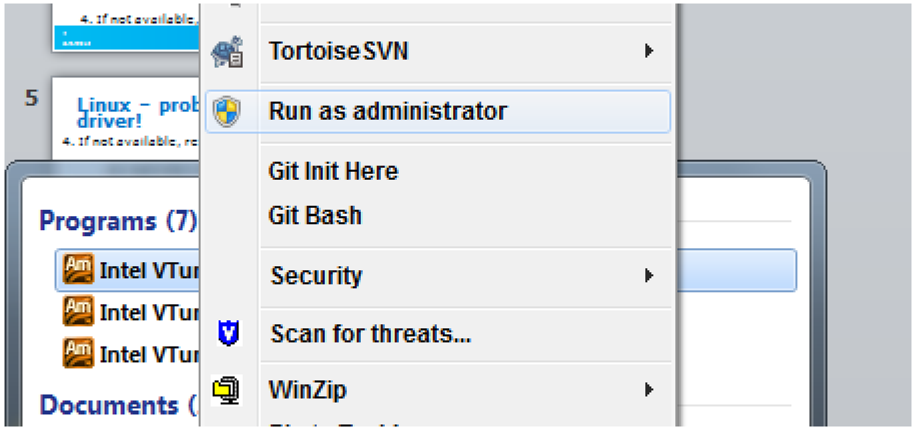
Windows – no accurate CPU time collection.


user-mode sampling and tracing collection. Press F1 for more details.

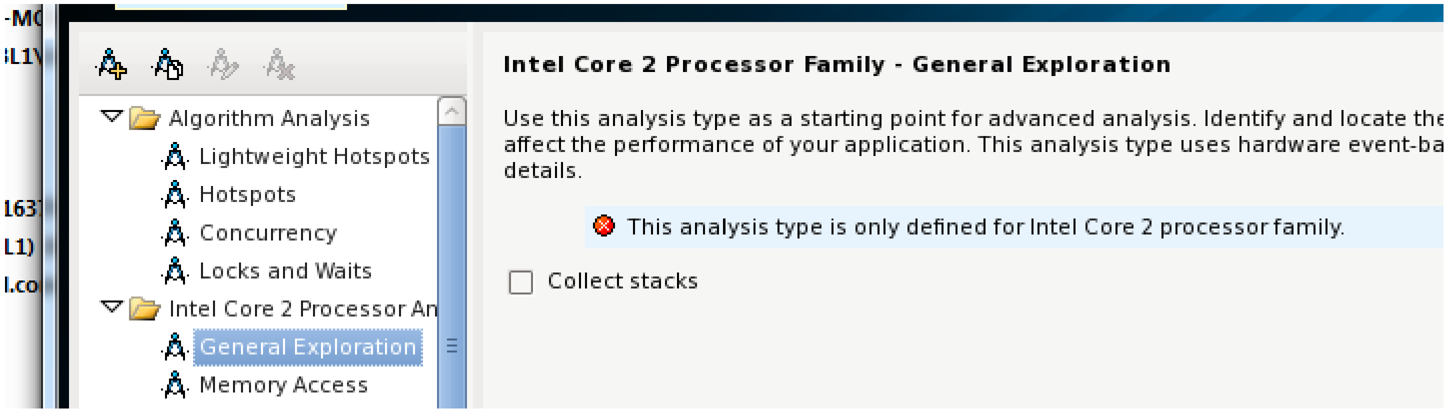
⚠ Highly accurate CPU time collection is disabled for this analysis. To enable this feature, run the product with the administrative privileges.

CPU sampling interval, ms: 10

Start the program in Administrator mode:



Architectural Analysis not available



-MC
L1
163
L1)
l.co

Algorithm Analysis
Lightweight Hotspots
Hotspots
Concurrency
Locks and Waits
Intel Core 2 Processor An
General Exploration
Memory Access

Intel Core 2 Processor Family - General Exploration

Use this analysis type as a starting point for advanced analysis. Identify and locate the
affect the performance of your application. This analysis type uses hardware event-ba
details.

This analysis type is only defined for Intel Core 2 processor family.

☐ Collect stacks

You are highlighting the wrong analysis type!

Optimization Notice

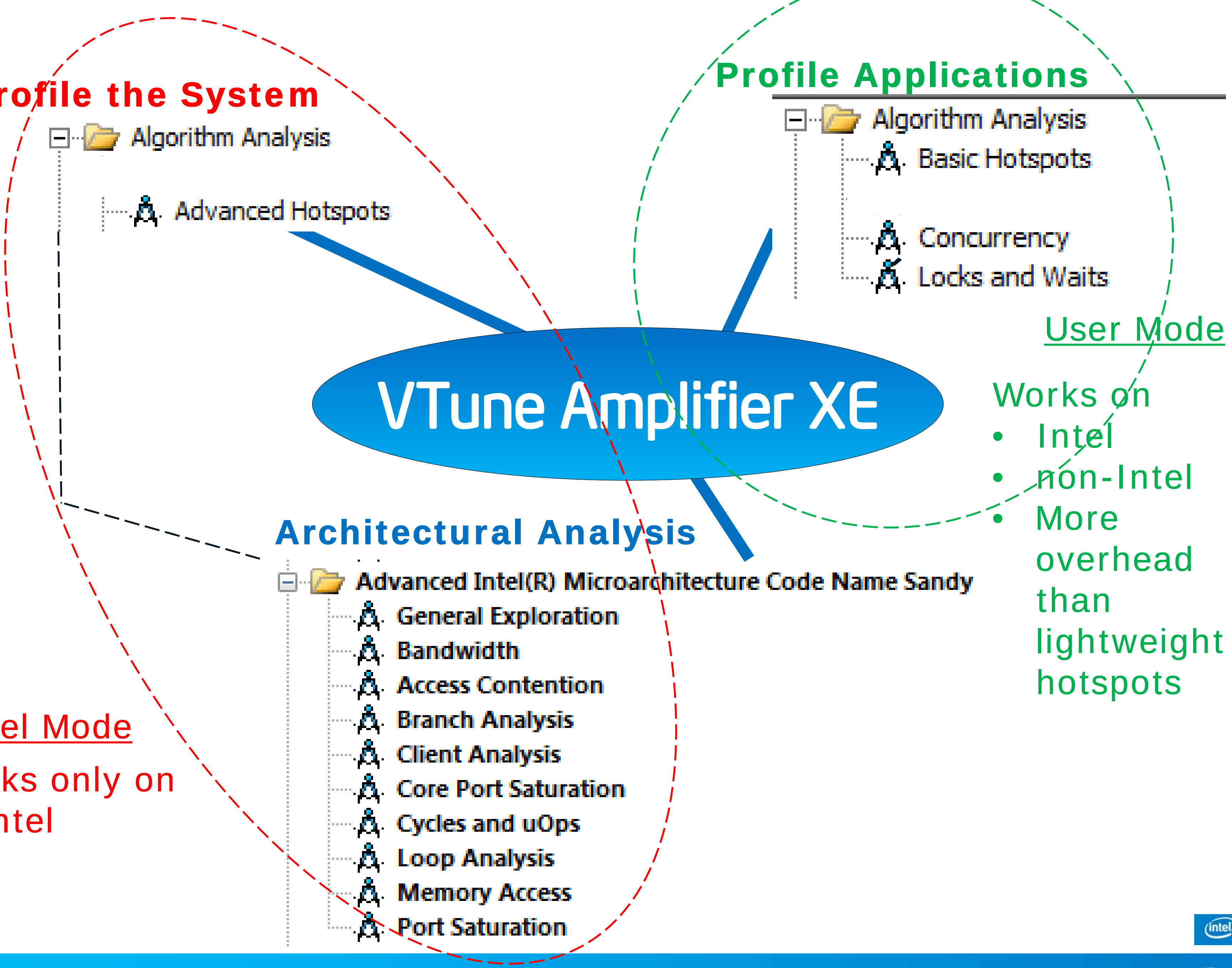
Linux – The source editor won’t open

Set the EDITOR or VISUAL environment variable!

```
export EDITOR=gedit
```

or

```
export EDITOR=vi
```



Profile the System

- Algorithm Analysis
  - Advanced Hotspots

Profile Applications

- Algorithm Analysis
  - Basic Hotspots
  - Concurrency
  - Locks and Waits

Architectural Analysis

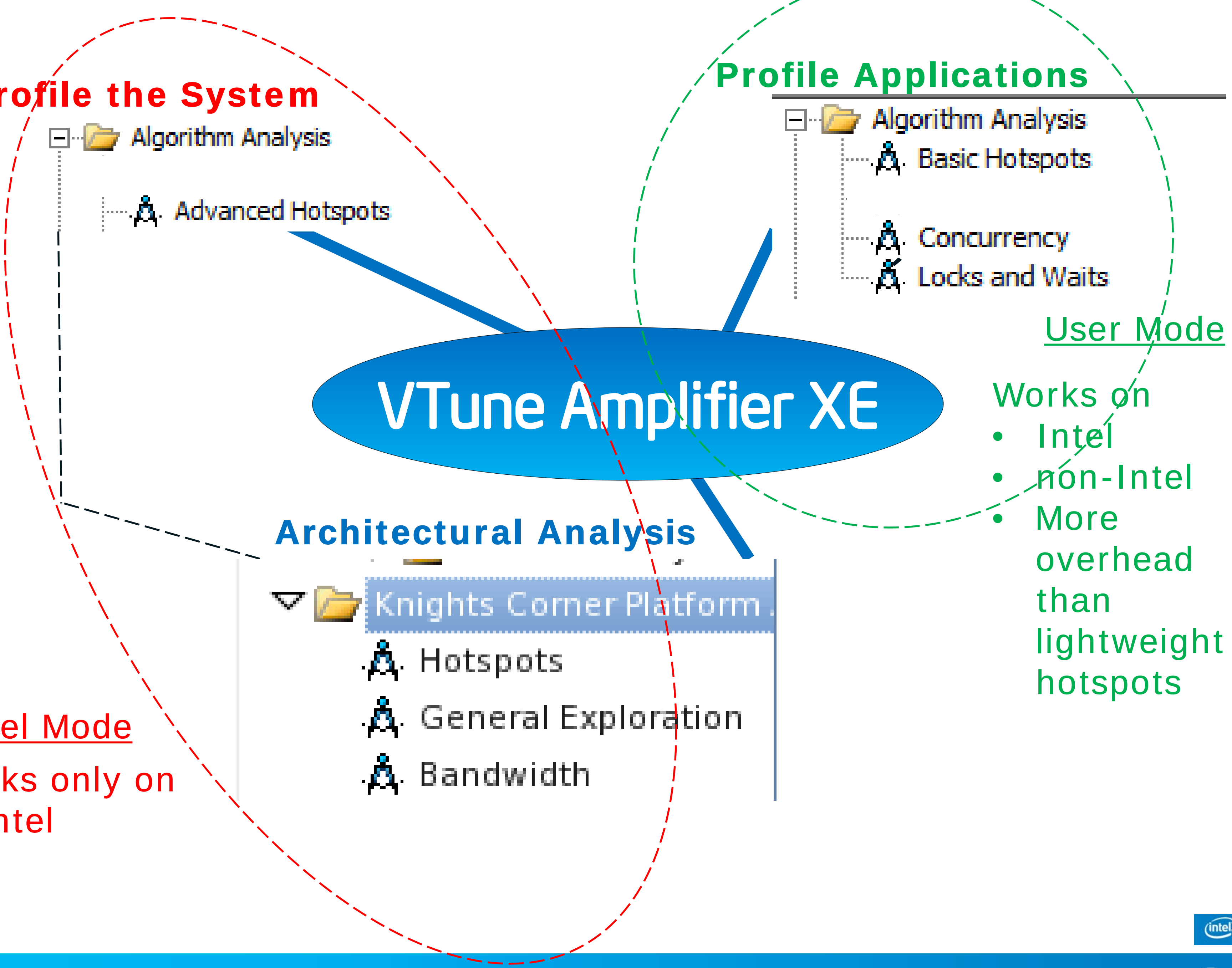
- Advanced Intel(R) Microarchitecture Code Name Sandy
  - General Exploration
  - Bandwidth
  - Access Contention
  - Branch Analysis
  - Client Analysis
  - Core Port Saturation
  - Cycles and uOps
  - Loop Analysis
  - Memory Access
  - Port Saturation

User Mode

Works on
- Intel
- non-Intel
- More overhead than lightweight hotspots

Kernel Mode

Works only on
- Intel



User Mode

Works on
- Intel
- non-Intel
- More overhead than lightweight hotspots

Kernel Mode

Works only on
- Intel

Intel® Xeon Phi™ Coprocessor Performance Monitoring Unit

Visualizing Performance Opportunities using Intel® VTune™ Amplifier

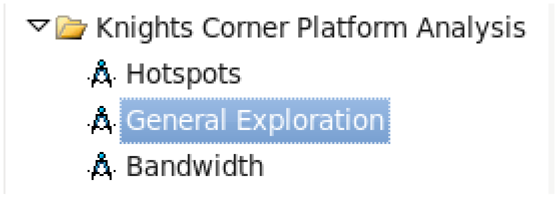
Once hot spots have been identified, performance events can be examined to identify poor resource use

Two performance counters per thread currently on coprocessor

- VTune™ Amplifier event multiplexing enables sampling more than two events at a time

Focus optimization on hot spots displaying problematic events

- Indicators of resource use can be derived from perf. Events
- Many useful indicators pulled together in General Exploration analysis



Cycles Per Instruction (CPI), a standard measure, has some special kinks

- Threads on each Intel® Xeon™ Phi core share a clock
  - If all 4 HW threads are active, each gets ¼ total cycles
- Multi-stage instruction decode requires two threads to utilize the whole core – one thread only gets half
- With two ops/per cycle (U-V-pipe dual issue):

Threads per Core	Best CPI per Core	Best CPI per Thread
1 x	1.0	= 1.0
2 x	0.5	= 1.0
3 x	0.5	= 1.5
4 x	0.5	= 2.0

- **To get thread CPI, multiply by the active threads**

As an efficiency metric, CPI must be considered carefully: it IS a ratio

- Changes in CPI absent major code changes can indicate general latency gains/losses

Metric	Formula	Investigate if
CPI per Thread	CPU_CLK_UNHALTED/ INSTRUCTIONS_EXECUTED	> 4.0, or increasing
CPI per Core	(CPI per Thread) / Number of hardware threads used	> 1.0, or increasing

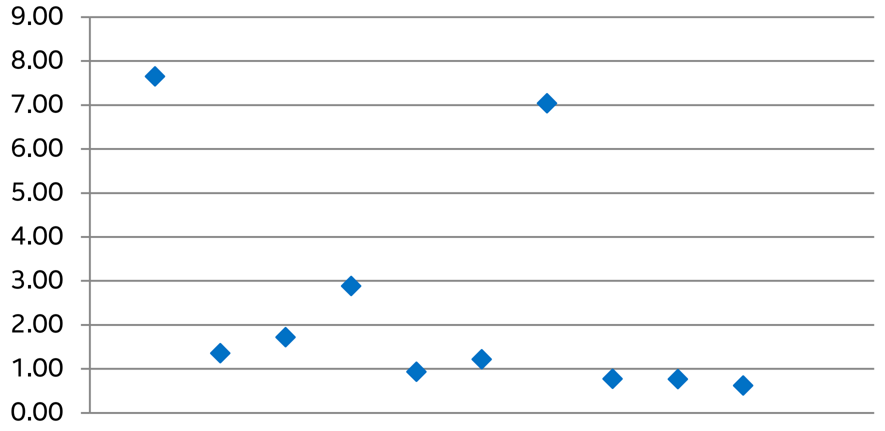
- Note the effect on CPI from applied optimizations
- Reduce high CPI through optimizations that target latency
  - Better prefetch
  - Increase data reuse through better blocking

Two more examples why absolute CPI value is less important than changes

- Scaling data from a typical lab workload:

Metric	1 hardware thread / core	2 hardware threads / core	3 hardware threads / core	4 hardware threads / core
CPI per Thread	5.24	8.80	11.18	13.74
CPI per Core	5.24	4.40	3.73	3.43

- Observed CPIs from several tuned workloads:



Efficiency Metric: Compute to Data Access Ratio

- Measures an application's computational density, and suitability for Intel® Xeon Phi™ coprocessors

Metric	Formula	Investigate if
Vectorization Intensity	VPU_ELEMENTS_ACTIVE / VPU_INSTRUCTIONS_EXECUTED	
L1 Compute to Data Access Ratio	VPU_ELEMENTS_ACTIVE / DATA_READ_OR_WRITE	< Vectorization Intensity
L2 Compute to Data Access Ratio	VPU_ELEMENTS_ACTIVE / DATA_READ_MISS_OR_ WRITE_MISS	< 100x L1 Compute to Data Access Ratio

- Increase computational density through vectorization and reducing data access (see cache issues, also, DATA ALIGNMENT!)

Problem Area: L1 Cache Usage

- Significantly affects data access latency and therefore application performance

Metric	Formula	Investigate if
L1 Misses	DATA_READ_MISS_OR_WRITE_MISS + L1_DATA_HIT_INFLIGHT_PF1	
L1 Hit Rate	(DATA_READ_OR_WRITE – L1 Misses) / DATA_READ_OR_WRITE	< 95%

- Tuning Suggestions:
  - Software prefetching
  - Tile/block data access for cache size
  - Use streaming stores
  - If using 4K access stride, may be experiencing conflict misses
  - Examine Compiler prefetching (Compiler-generated L1 prefetches should not miss)

*tuning suggestions requiring deeper understanding of architectural tradeoffs and application data handling details are highlighted with this “ninja” notation

Problem Area: Data Access Latency

Metric	Formula	Investigate if
Estimated Latency Impact	(CPU_CLK_UNHALTED – EXEC_STAGE_CYCLES – DATA_READ_OR_WRITE) / DATA_READ_OR_WRITE_MISS	>145

- Tuning Suggestions:
  - Software prefetching
  - Tile/block data access for cache size
  - Use streaming stores
  - Check cache locality – turn off prefetching and use CACHE_FILL events - reduce sharing if needed/possible
  - If using 64K access stride, may be experiencing conflict misses

Problem Area: TLB Usage

- Also affects data access latency and therefore application performance

Metric	Formula	Investigate if:
L1 TLB miss ratio	DATA_PAGE_WALK/DATA_READ_OR_WRITE	> 1%
L2 TLB miss ratio	LONG_DATA_PAGE_WALK / DATA_READ_OR_WRITE	> .1%
L1 TLB misses per L2 TLB miss	DATA_PAGE_WALK / LONG_DATA_PAGE_WALK	> 100x

- Tuning Suggestions:
  - Improve cache usage & data access latency
  - If L1 TLB miss/L2 TLB miss is high, try using large pages
  - For loops with multiple streams, try splitting into multiple loops
  - If data access stride is a large power of 2, consider padding between arrays by one 4 KB page

Problem Area: VPU Usage

- Indicates whether an application is vectorized successfully and efficiently

Metric	Formula	Investigate if
Vectorization Intensity	VPU_ELEMENTS_ACTIVE / VPU_INSTRUCTIONS_EXECUTED	<8 (DP), <16(SP)

- Tuning Suggestions:
  - Use the Compiler vectorization report!
  - For data dependencies preventing vectorization, try using Intel® Cilk™ Plus #pragma SIMD (if safe!)
  - Align data and tell the Compiler!
  - Restructure code if possible: Array notations, AOS->SOA

Problem Area: Memory Bandwidth

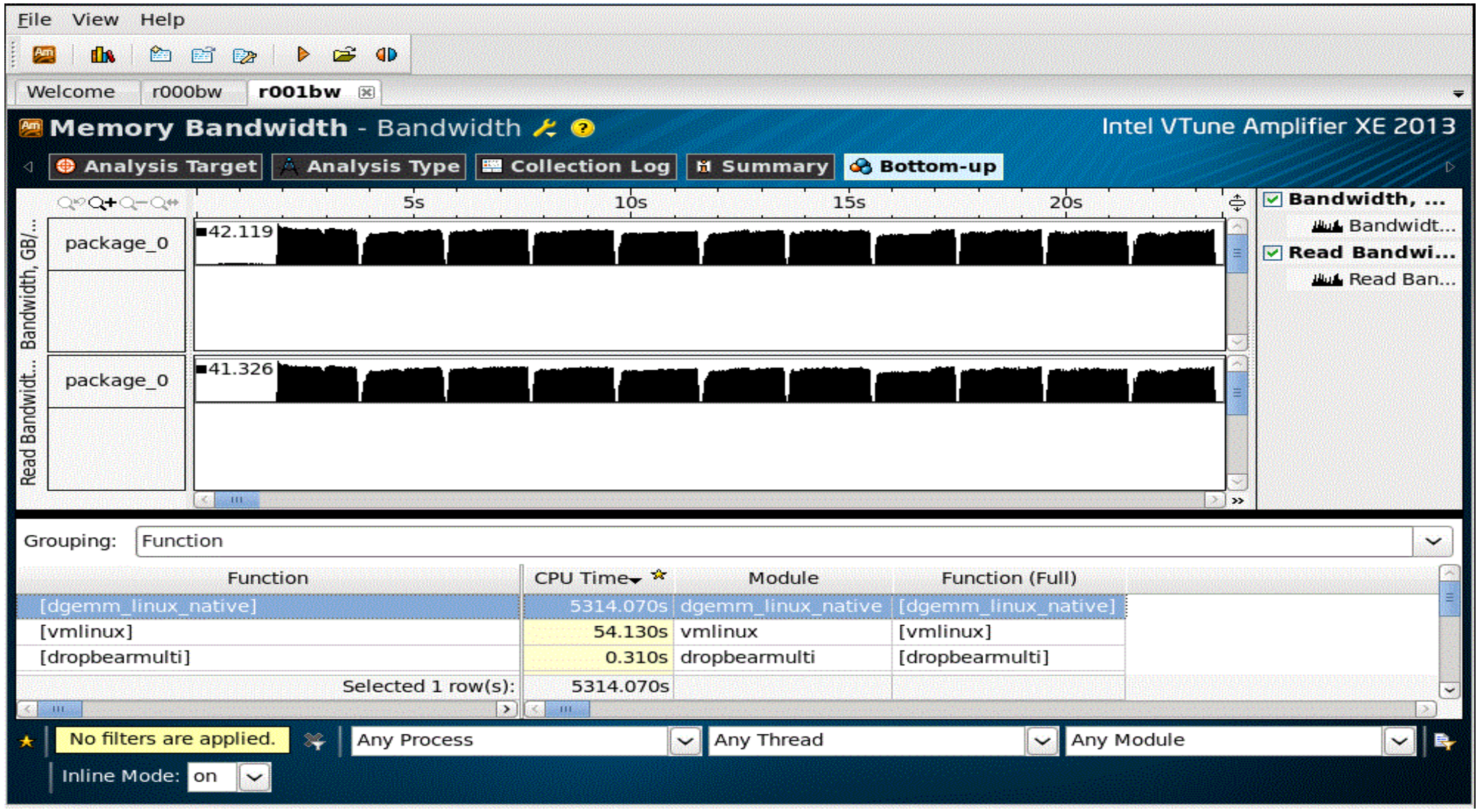
- Can increase data latency in the system or become a performance bottleneck

Metric	Formula	Investigate if
Memory Bandwidth	(UNC_F_CH0_NORMAL_READ + UNC_F_CH0_NORMAL_WRITE+ UNC_F_CH1_NORMAL_READ + UNC_F_CH1_NORMAL_WRITE) X 64/time	< 80GB/sec (practical peak 140GB/sec) (with 8 memory controllers)

- Tuning Suggestions:
  - Improve locality in caches
  - Use streaming stores
  - Improve software prefetching

Final caution: coprocessor collections can generate dense volumes of data

Example: DGEMM on 60+ cores



Grouping: Function

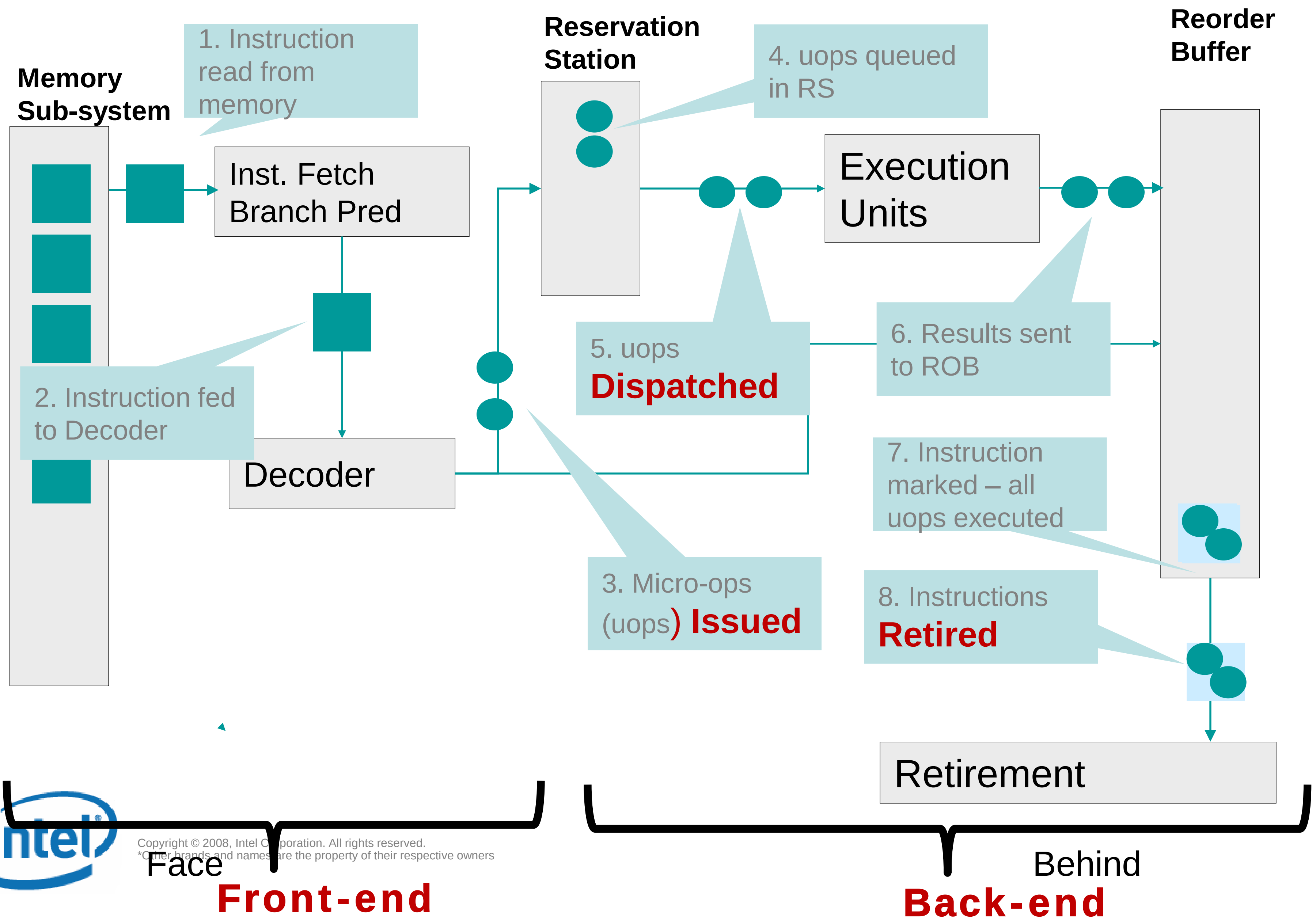
Function	CPU Time	Module	Function (Full)
[dgemm_linux_native]	5314.070s	dgemm_linux_native	[dgemm_linux_native]
[vmlinux]	54.130s	vmlinux	[vmlinux]
[dropbearmulti]	0.310s	dropbearmulti	[dropbearmulti]
Selected 1 row(s):	5314.070s		

No filters are applied. Any Process Any Thread Any Module

Inline Mode: on

Tip: Use a CPU Mask to reduce data volume while maintaining equivalent accuracy.

The life of a program instruction



For more information on Intel® Xeon Phi™ and VTune™ Amplifier XE

Optimization on the coprocessor: http://software.intel.com/en-us/articles/optimization-and-performance-tuning-for-intel-xeon-phi-coprocessors-part-1-optimization

http://software.intel.com/en-us/articles/optimization-and-performance-tuning-for-intel-xeon-phi-coprocessors-part-2-understanding

Coprocessor Performance Monitoring Unit: http://software.intel.com/sites/default/files/forum/278102/intelr-xeon-phitm-pmu-rev1.01.pdf

For general information: http://software.intel.com/mic-developer

Thank You

Backup

Suggested Order of Fixing Problems

Priority	Problem
1	Cache misses
2	Contested access
3	Other data access issues
4	Allocation Stalls
5	Micro Assists
6	Branch Mispredictions and machine clears
7	Other Front-end stalls

See slides in backup section for a more detailed description

Optimization Notice

Cache Misses

- **Why:** Cache misses raise the CPI of an application
  - Focus on long-latency data accesses coming from 2nd and 3rd level misses
- **How:** General Exploration profile, Metrics: *LLC Hit*, *LLC Miss*
- **What Now:**
  - If either metric is highlighted for your hotspot, consider reducing misses:
    - Use the cacheline replacement analysis outlined in the Intel® 64 and IA-32 Architectures Optimization Reference Manual, section **B.3.4.2**
    - Use software prefetch instructions
    - Block data accesses to fit into cache
    - Use local variables for threads
    - Pad data structures to cacheline boundaries
    - Change your algorithm to reduce data storage

B.3.4.2 Cache-line Replacement Analysis

When an application has many cache misses, it is a good idea to determine where cache lines are being replaced at the highest frequency. The instructions responsible for high amount of cache replacements are not always where the application is spending the majority of its time, since replacements can be driven by the hardware prefetchers and store operations which in the common case do not hold up the pipeline. Typically traversing large arrays or data structures can cause heavy cache line replacements.

Required events

L1D.REPLACEMENT - Replacements in the 1st level data cache.

L2_LINES_IN.ALL - Cache lines being brought into the L2 cache.

OFFCORE_RESPONSE.DATA_IN_SOCKET.LLC_MISS_LOCAL.DRAM_0 - Cache lines being brought into the LLC.

Usages of events

Identifying the replacements that potentially cause performance loss can be done at process, module, and function level. Do it in two steps:

- Use the precise load breakdown to identify the memory hierarchy level at which loads are satisfied and cause the highest penalty.
- Identify, using the formulas below, which portion of code causes the majority of the replacements in the level below the one that satisfies these high penalty loads.

For example, if there is high penalty due to loads hitting the LLC, check the code which is causing replacements in the L2 and the L1. In the formulas below, the nominators are the replacements accounted for a module or function. The sum of the replacements in the denominators is the sum of all replacements in a cache level for all processes. This enables you to identify the portion of code that causes the majority of the replacements.

L1D Cache Replacements

```
%L1D.REPLACEMENT =
        L1D.REPLACEMENT / SumOverAllProcesses(L1D.REPLACEMENT );
```

L2 Cache Replacements

```
%L2.REPLACEMENT =
        L2_LINES_IN.ALL / SumOverAllProcesses(L2_LINES_IN.ALL );
```

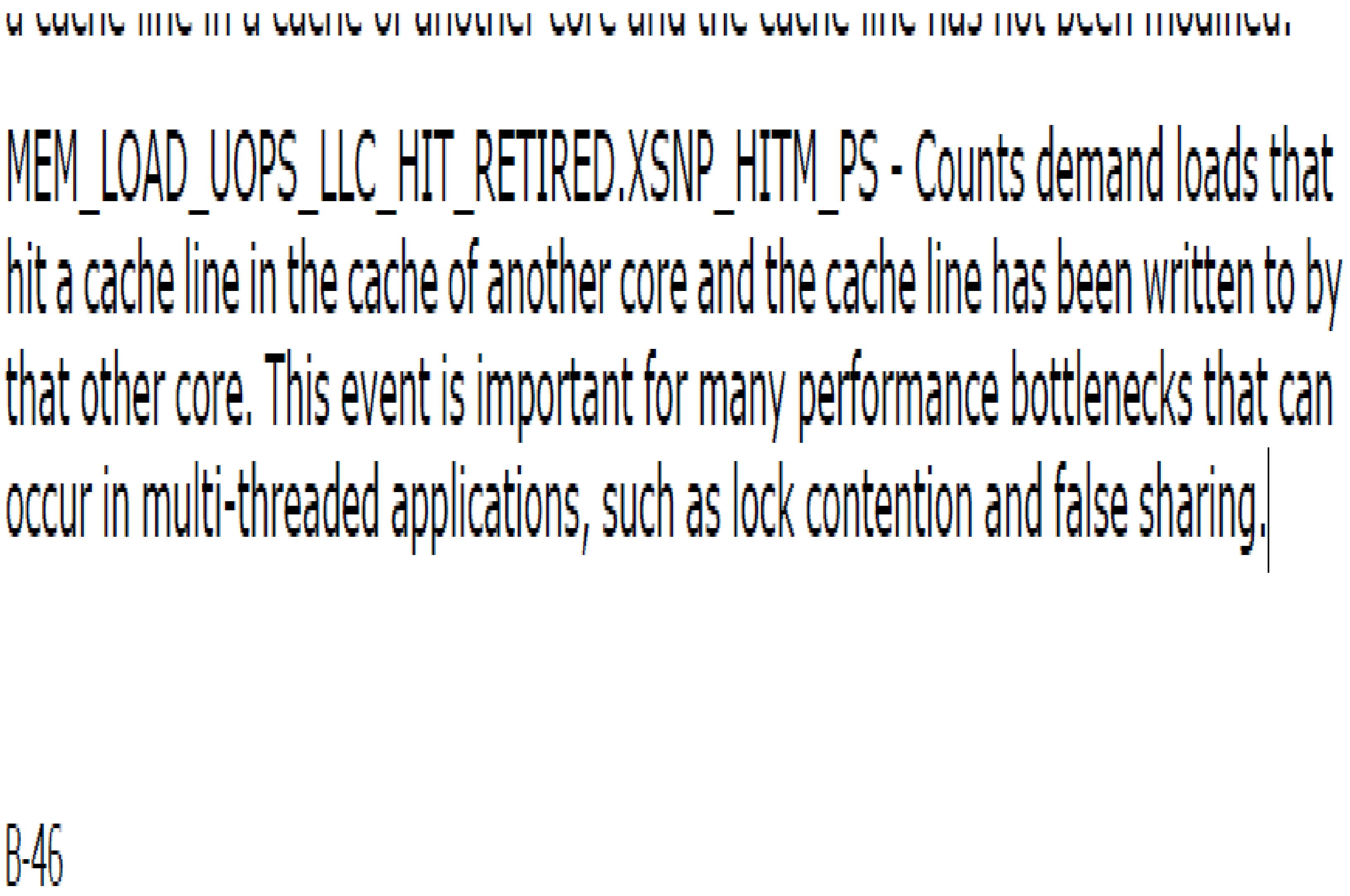
L3 Cache Replacements

```
%L3.REPLACEMENT =
        OFFCORE_RESPONSE.DATA_IN_SOCKET.LLC_MISS_LOCAL.DRAM_0/ SumOv-
erAllProcesses(OFFCORE_RESPONSE.DATA_IN_SOCKET.LLC_MISS_LOCAL.DRAM_0 );
```

Contested Accesses

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

Hit Modified Data



a cache line in a cache of another core and the cache line has not been modified.

MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HITM_PS - Counts demand loads that hit a cache line in the cache of another core and the cache line has been written to by that other core. This event is important for many performance bottlenecks that can occur in multi-threaded applications, such as lock contention and false sharing.

B-46

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

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

2.2.4.4 Store Forwarding

If a load follows a store and reloads the data that the store writes to memory, the Intel Core microarchitecture can forward the data directly from the store to the load. This process, called store to load forwarding, saves cycles by enabling the load to obtain the data directly from the store operation instead of through memory.

The following rules must be met for store to load forwarding to occur:

- The store must be the last store to that address prior to the load.
- The store must be equal or greater in size than the size of data being loaded.
- The load cannot cross a cache line boundary.
- The load cannot cross an 8-Byte boundary. 16-Byte loads are an exception to this rule.
- The load must be aligned to the start of the store address, except for the following exceptions:
  - An aligned 64-bit store may forward either of its 32-bit halves
  - An aligned 128-bit store may forward any of its 32-bit quarters
  - An aligned 128-bit store may forward either of its 64-bit halves

Software can use the exceptions to the last rule to move complex structures without losing the ability to forward the subfields.

Other Data Access Issues: Cache Line Splits

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

Other Data Access Issues: 4K Aliasing

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

Optimization Notice

11.8 4K ALIASING

4-KByte memory aliasing occurs when the code stores to one memory location and shortly after that it loads from a different memory location with a 4-KByte offset between them. For example, a load to linear address 0x400020 follows a store to linear address 0x401020.

The load and store have the same value for bits 5 - 11 of their addresses and the accessed byte offsets should have partial or complete overlap.

4K aliasing may have a five-cycle penalty on the load latency. This penalty may be significant when 4K aliasing happens repeatedly and the loads are on the critical path. If the load spans two cache lines it might be delayed until the conflicting store is committed to the cache. Therefore 4K aliasing that happens on repeated unaligned Intel AVX loads incurs a higher performance penalty.

To detect 4K aliasing, use the LD_BLOCKS_PARTIAL.ADDRESS_ALIAS event that counts the number of times Intel AVX loads were blocked due to 4K aliasing.

To resolve 4K aliasing, try the following methods in the following order:

- Align data to 32 Bytes.
- Change offsets between input and output buffers if possible.
- Use 16-Byte memory accesses on memory which is not 32-Byte aligned.

Optimization Notice

Other Data Access Issues: DTLB Misses

Why: First-level DTLB Load misses (Hits in the STLB) incur a latency penalty. Second-level misses require a page walk that can affect your application's performance.

How: General Exploration profile, Metric: *DTLB Overhead*

What Now:

- If this metric is highlighted for your hotspot, investigate at the sourcecode level.
- To fix these issues, target data locality to TLB size, use the Extended Page Tables (EPT) on virtualized systems, try large pages (database/server apps only), increase data locality by using better memory allocation or Profile-Guided Optimization

Allocation Stalls

Why: Certain types of instructions can cause allocation stalls because they take longer to retire. These increase latencies overall.

How: General Exploration Profile, Metric: *LEA Stalls, Flags Merge Stalls*

What Now:

- If this metric is highlighted for your hotspot, investigate at the sourcecode level.
- Try to eliminate uses of 3-operand LEA instructions, Look for certain uses of an LEA instruction (see section 3.5.1.3 of Intel® 64 and IA-32 Architectures Optimization Reference Manual) or partial register use (see section 3.5.2.4 of Intel® 64 and IA-32 Architectures Optimization Reference Manual) and fix.

Microcode Assists

Why: Assists from the microcode sequencer can have long latency penalties.

How: General Exploration Profile, Metric: *Assists*

What Now:

- If this metric is highlighted for your hotspot, re-sample using the additional assist events to determine the cause.
- If FP_ASSISTS.ANY / INST_RETIRED.ANY is significant, check for denormals. To fix enable FTZ and/or DAZ if using SSE/AVX instructions or scale your results up or down depending on the problem
- If ((OTHER_ASSISTS.AVX_TO_SSE_NP*75) / CPU_CLK_UNHALTED.THREAD) or ((OTHER_ASSISTS.SSE_TO_AVX_NP*75) / CPU_CLK_UNHALTED.THREAD) is greater than .1, reduce transitions between SSE and AVX code

Cancelled

Branch Mispredicts

Why: Mispredicted branches cause pipeline inefficiencies due to wasted work or instruction starvation (while waiting for new instructions to be fetched)

How: General Exploration Profile, Metric: *Branch Mispredict*

What Now:

- If this metric is highlighted for your hotspot try to reduce misprediction impact:
- Use compiler options or profile-guided optimization (PGO) to improve code generation
- Apply hand-tuning by doing things like hoisting the most popular targets in branch statements.

Machine Clears

Cancelled

Why: Machine clears cause the pipeline to be flushed and the store buffers emptied, resulting in a significant latency penalty.

How: General Exploration Profile, Metric: *Machine Clears*

Now What:

- If this metric is highlighted for your hotspot try to determine the cause using the specific events:
- If MACHINE_CLEARS.MEMORY_ORDERING is significant, investigate at the sourcecode level. This could be caused by 4K aliasing conflicts or contention on a lock (both previous issues).
- If MACHINE_CLEARS.SMC is significant, the clears are being caused by self-modifying code, which should be avoided.

Front-end Stalls

Why: Front-end stalls (at the Issue stage of the pipeline) can cause instruction starvation, which may lead to stalls at the execute stage in the pipeline.

How: General Exploration profile, Metric: *Front-end Bound Pipeline Slots*

What Now:

- If this metric is highlighted for your hotspot, try using better code layout and generation techniques:
  - Try using profile-guided optimizations (PGO) with your compiler
  - Use linker ordering techniques (/ORDER on Microsoft's linker or a linker script on gcc)
  - Use switches that reduce code size, such as /O1 or /Os

Legal Disclaimer & Optimization Notice