

# Intended Audience: Software Developers

Interested in performance optimizing your application

- Don't need to be a performance expert
- But should be an expert in the application!

Working on a platform with an Intel® Xeon Phi™ code named Knights Landing

Using Intel® VTune™ Amplifier XE performance analyzer

- The performance information here applies to other tools (PTU, etc) but is focused on VTune Amplifier XE
- The last section of this guide also includes information about Intel® Advisor XE

## How to Use this Presentation

Read through the slides once, then again while collecting data

Remember performance analysis is a process that may take several iterations

Software Optimization should begin *after you have*:

- Utilized any compiler optimization options (/O2, /QxAVX2, etc)
- Chosen an appropriate workload
- Measured baseline performance

# Using Intel® VTune™ Amplifier XE to Tune Software on the Intel® Xeon Phi™ code named Knights Landing (KNL)

Software and Services Group

Ver. 1.1

Optimization Notice



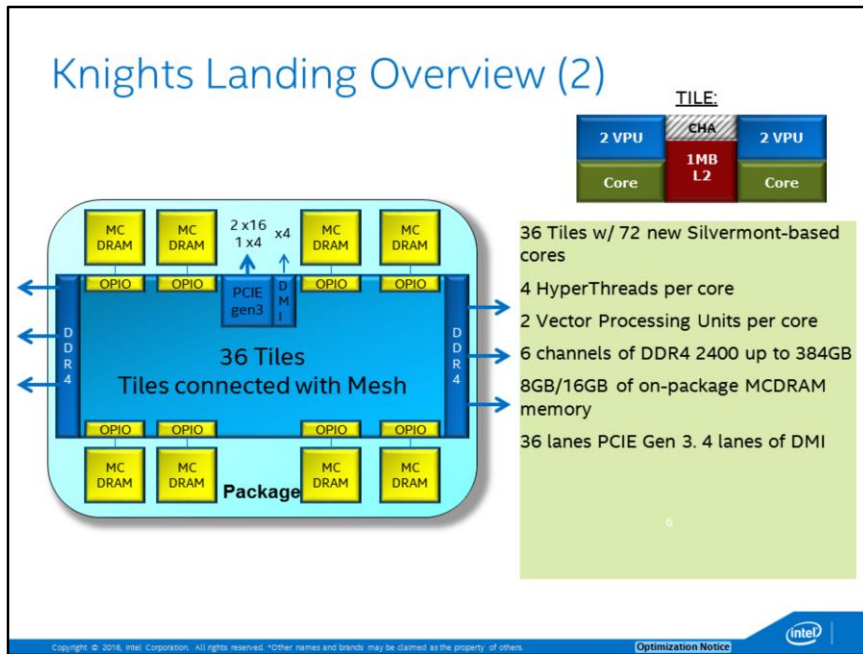
# Agenda

- Intel® Xeon Phi™ Code Named Knights Landing (KNL) Overview
- Intel® VTune™ Amplifier XE
- Software Optimization Steps
  - Profile resource utilization
  - Identify problematic symptoms
  - Locate issues and use recommendations to improve performance
- Additional Tuning Recommendations
- Intel® Advisor

## Knights Landing Overview

- Knights Landing is the next Intel Many Core product after Knights Corner
- First self-boot Intel® Xeon Phi™ that is binary compatible with main line IA
- Significant leap in scalar and vector performance improvement over KNC
- Integration of memory on package: Innovative memory architecture for high bandwidth and high capacity
- Integration of fabric on package

## Knights Landing Overview (2)



KNL is a highly-parallel architecture with large vector units. To get the most performance out of this platform, it is imperative to take advantage of these strengths.

## KNL Tile:



**Core:** Changed from KNC to KNL. Based on Intel microarchitecture code named Silvermont (SLM) core – with *many* changes

### Selected Important features of the Core

- Out of order 2-wide core: 72 inflight uops. 4 threads/core
- Back to back fetch and issue per thread
- 32KB Icache, 32KB Dcache. 2x 64B Loads ports in Dcache. Larger TLBs than in SLM
- L1 Prefetcher (IPP) and L2 Prefetcher.
- Fast unaligned and cache-line split support. Fast Gather/Scatter support
- 2x BW between Dcache and L2 than in SLM: 1 line Rd and ½ line Wr per cycle

**2 VPUs:** 2x 512b Vectors. 32SP and 16DP. X87, SSE and EMU support

# Intel® VTune™ Amplifier XE



## VTune Amplifier XE features:

- Multiple Collection Types
  - Hotspots
  - Bandwidth
  - Event-based Sampling
- Timeline View Integrated into all Analysis Types
- Source/Assembly Viewing
- Compatible with C/C++, Fortran, Java, Assembly, .NET
- Visual Studio Integration, Command-line, or Standalone interface for Windows\* or Linux\*



Copyright © 2016, Intel Corporation. All rights reserved. Other names and brands may be claimed as the property of others.

Optimization Notice



Most screenshots in this presentation were taken from Intel® VTune™ Amplifier XE 2016 Update 4. This is the first public version with KNL support. Screenshots from different versions of the tool may have minor differences.



## Running VTune Amplifier from the command-line

On self-boot KNL machines ensure the `amplxe-cl` command is installed. See the "`amplxe-cl -help`" command for complete details. To collect:

### Hotspots:

```
amplxe-cl -collect advanced-hotspots -- myapp.out
```

### General Exploration:

```
amplxe-cl -collect general-exploration -- myapp.out
```

### Memory Access:

```
amplxe-cl -collect memory-access -- myapp.out
```

Copyright © 2016 Intel Corporation. All rights reserved. Other names and brands may be claimed as the property of others.

Optimization Notice



Results will be created in a directory named `r###ah`, `r###ge`, or `r###macc`

Results can be viewed from the command-line or GUI on the KNL machine, but it is generally more efficient to copy results to another machine with the GUI installed for analysis.

It is also recommended to add the `-no-auto-finalize` flag to collections that will be creating large results. The finalization step is compute intensive and runs serially which may take a long time on the KNL. Finalization can be done on another machine after copying the results off of the KNL.

The data collected may be very large for longer runs with many threads active. If you find that you are reaching the data limit, use the flag `-data-limit=<integer>`. The default limit is 500MB. The integer specifies the size in MB. Use 0 for no limit.

# Advanced Hotspots Analysis

- Supports OpenMP\* analysis
- Stack-sampling is enabled. However, call counts and trip counts are not supported.

Advanced Hotspots Hotspots viewpoint (change) Intel VTune Amplifier XE 2016

Collection Log Analysis Target Analysis Type Summary Bottom-up Caller/Callee Top-down Tree Platform

Elapsed Time: 4.506s

- CPU Time: 566.499s
  - Effective Time: 195.181s
  - Spin Time: 352.865s

A significant portion of CPU time is spent waiting. Use this metric to discover which synchronizations are spinning. Consider adjusting spin wait parameters, changing the lock implementation (for example, by backing off then descheduling), or adjusting the synchronization granularity.
- Overhead Time: 18.453s
- Instructions Retired: 320,824,000,000
- CPI Rate: 2.642

The CPI may be too high. This could be caused by issues such as memory stalls, instruction starvation, branch misprediction or long latency instructions. Explore the other hardware-related metrics to identify what is causing high CPI.

- CPU Frequency Ratio: 1.071
- Total Thread Count: 130
- Paused Time: 0s

Total Thread Count much higher on Intel® Xeon Phi™. Threading is vital for performance.

Copyright © 2016 Intel Corporation. All rights reserved. Other names and brands may be claimed as the property of others. Optimization Notice

# Advanced Hotspots Analysis

Advanced Hotspots Hotspots viewpoint (change) Intel VTune Amplifier XE 2016

Collection Log Analysis Target Analysis Type Summary Bottom-up Collect

Advanced OpenMP performance analysis

OpenMP Analysis. Collection Time: 4.506

Serial Time (outside any parallel region) 1.107s (24.6%)

Serial Time of your application is high. It directly impacts application Elapsed Time and scalability. Explore options for parallelization, algorithm or microarchitecture tuning of the serial part of the application.

Parallel Region Time: 3.399s (75.4%)

Estimated Ideal Time 1.217s (27.0%)

OpenMP Potential Gain 2.182s (48.4%)

The time wasted on load imbalance or parallel work arrangement is significant and negatively impacts the application performance and scalability. Explore OpenMP regions with the highest metric values. Make sure the workload of the regions is enough and the loop schedule is optimal.

Top OpenMP Regions by Potential Gain

This section lists OpenMP regions with the highest potential for performance improvement. The Potential Gain metric shows the elapsed time that could be saved if the region was optimized to have no load imbalance assuming no runtime overhead.

OpenMP Region	OpenMP Potential Gain (%)	OpenMP Region Time
<a href="#">main\$omp\$parallel.130@unknown.19.25</a>	2.182s 48.4%	3.399s

Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

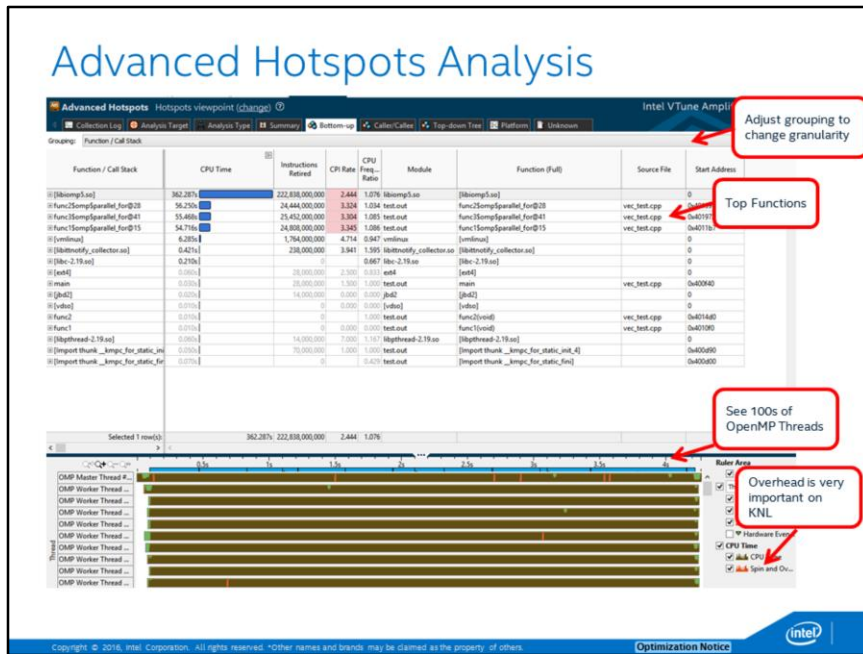
Function	Module	CPU Time
<a href="#">_kmp_wait_template-kmp_flag_64</a>	libiomp5.so	282.763s
<a href="#">main\$omp\$parallel_for@19</a>	test.out	186.721s
<a href="#">kmp_base_flag-unsigned-long-long-notdone_check</a>	libiomp5.so	13.521s
<a href="#">_kmp_hierarchical_barrier_release</a>	libiomp5.so	8.720s
<a href="#">_kmp_yield</a>	libiomp5.so	7.838s
[Others]	N/A	65.935s

\*N/A is applied to non-summable metrics.

Copyright © 2016 Intel Corporation. All rights reserved. Other names and brands may be claimed as the property of others. Optimization Notice intel

The Advanced Hotspots Analysis will show where your application is spending its time, including information related to OpenMP parallelism. Ensure that the OpenMP runtime library used in the application (e.g. libiomp5.so) is available on the system doing the analysis. This is required to accurately analyze OpenMP overhead.

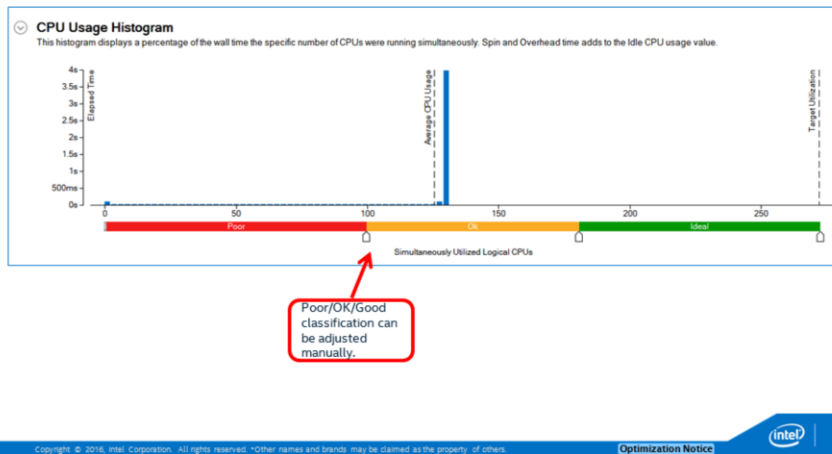
# Advanced Hotspots Analysis



Use the Bottom-up view to see time spent at various granularities; for example Function or Module granularities. This can be changed in the Grouping drop-down menu. Focus tuning efforts on the hot portions of your application.

## Profile Resource Utilization

### Advanced Hotspots > Summary Tab



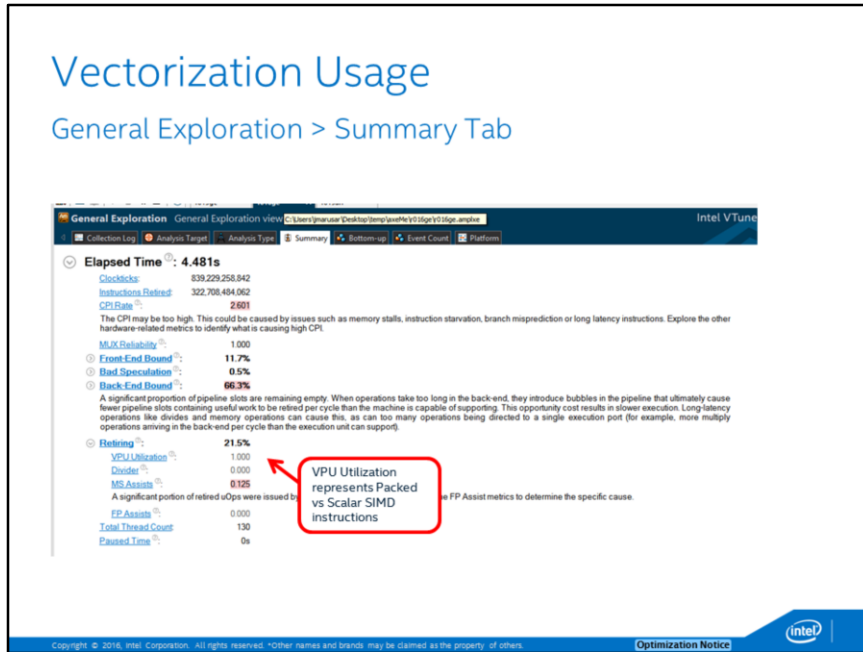
To get the best performance from KNL, it is important to have highly threaded parallel applications. The CPU Usage Histogram in the Summary shows how much time was spent with various numbers of logical cores active. As a general guideline, the vast majority of time should be spent with more than 50% of all available logical cores active. Because each KNL core has 4 HyperThreads, it isn't always beneficial to have all logical cores active if the bottleneck is the execution core, which is shared between HyperThreads. If memory accesses are the bottlenecks, more threads may alleviate the problem.

Memory Bandwidth may not be helped by more threads, but Memory Latency can.

To identify memory latency as the issue look at L2 misses. If L2 misses are high and bandwidth is high, bandwidth may be the bottleneck. If L2 misses are high, but bandwidth is low, latency may be the issue, and more threads may help.

# Vectorization Usage

## General Exploration > Summary Tab



KNL supports 512 bit vector instructions. To optimize for KNL, an application should take advantage of these large vector units with heavily vectorized code. Look at the metric VPU Utilization to determine the areas of high and low vectorization in your application.

# Vectorization Usage

General Exploration General Exploration viewpoint (change)

Collection Log Analysis Target Analysis Type Summary Bottom-up Event Count Platform

Grouping: Function / Call Stack

Function / Call Stack	Clockticks	Instructions Retired	CPI Rate	Retiring			
				VPU Utilization	MS Assi...	FP Assists	
@@_kmp_hyper_barrier_release	1,566,588,340,879	763,205,144,806	2.053	0.000	0.000	0.178	0.000
@@compute_rhs_Somp\$parallel@17	1,515,622,273,430	96,228,144,342	15.759	1.000	0.000	0.105	0.000
@@y_solve_Somp\$parallel_for@27	1,396,328,094,489	59,428,089,142	23.496	0.801	0.000	0.223	0.000
@@z_solve_Somp\$parallel_for@31	1,379,010,068,512	64,234,096,351	21.469	0.814	0.000	0.159	0.000
@@x_solve_Somp\$parallel_for@27	1,066,933,600,398	70,884,106,326	15.052	0.831	0.000	0.160	0.000
@@vmlinval	197,668,296,502	36,600,054,900	5.401	1.000	0.000	0.238	0.000
@@tzetar_Somp\$parallel_for@22	165,968,248,952	5,790,008,685	28.665	1.000	0.000	0.200	0.000
@@binvr_Somp\$parallel_for@22	140,052,210,078	6,356,009,534	22.035	1.000	0.000	0.223	0.000
@@add_Somp\$parallel_for@19	125,748,188,622	4,930,007,395	25.507	1.000	0.000	0.014	0.000
@@nimv_Somp\$parallel_for@20	81,542,122,313	3,352,005,028	24.326	1.000	0.000	0.344	0.000
@@nimv_Somp\$parallel_for@20	79,990,119,985	3,260,004,899	24.537	1.000	0.000	0.299	0.000
@@_kmp_hyper_barrier_gather	68,008,102,012	32,918,049,377	2.066	0.000	0.000	0.162	0.000
@@_kmp_wait_yield_4	55,700,082,550	28,304,030,456	2.743	0.000	0.000	0.274	0.000
@@_kmp_yield	40,500,060,750	488,000,732	82.992	0.000	0.000	0.712	0.000
@@fbc-2.19.sc	8,500,012,750	1,692,002,538	5.024	0.000	0.000	0.518	0.000
@@tsinit	7,398,011,097	48,000,072	154.125	1.000	0.000	1.000	0.000
@@tsinitj	6,558,009,837	56,000,084	117.107	1.000	0.000	1.000	0.000
@@exact_rhs_Somp\$parallel@20	3,654,005,481	1,064,001,596	3.434	0.861	0.000	0.158	0.000
@@initialize_Somp\$parallel@22	3,162,004,743	1,282,001,923	2.466	0.951	0.000	0.065	0.000
@@exact_solution	3,084,004,626	2,514,003,771	1.227	0.751	0.000	0.000	0.000
@@_kmp_bakery_check	2,862,004,293	62,000,093	46.161	0.000	0.000	0.745	0.000
@@_kmp_for_static_init_4	1,618,002,427	32,000,048	50.563	1.000	0.000	0.000	0.000
@@_kmp_join_barrier	1,214,001,821	4,000,000	303.500	0.000	0.000	0.000	0.000
@@fblmnotify_collector.sc	1,150,001,725	38,000,054	31.944	0.000	0.000	0.000	0.000
@@_kmpc_for_static_fini	746,001,119			0.000	0.000	0.000	0.000

Copyright © 2018 Intel Corporation. All rights reserved. Other names and brands may be claimed as the property of others. Optimization Notice intel

The VPU Utilization metric is also available in the Bottom-up view of the General Exploration viewpoint. Locate hotspots with low VPU Utilization and try to improve their usage of the AVX512 capabilities.

## Identify the Hotspots

**What:** Hotspots are where your application spends the most time

**Why:** You should aim your optimization efforts there!

- Why improve a function that only takes 2% of your application's runtime?

**How:** VTune Amplifier XE *Advanced Hotspots* analysis type

- Usually hotspots are defined in terms of the CPU\_CLK\_UNHALTED.THREAD event (aka "clockticks")

Copyright © 2016, Intel Corporation. All rights reserved. Other names and brands may be claimed as the property of others.

Optimization Notice



For this processor, the CPU\_CLK\_UNHALTED.THREAD counter measures unhalting clockticks on a per hardware thread basis. The CPU\_CLK\_UNHALTED.THREAD counter allows you to see where cycles are being spent on each individual hardware thread.

There is also a CPU\_CLK\_UNHALTED.REF counter, which counts unhalting clockticks per thread, at the reference frequency for the CPU. In other words, the CPU\_CLK\_UNHALTED.REF counter should not increase or decrease as a result of frequency changes due to throttling. This counter can be useful for removing the variance introduced due to throttling when comparing multiple analyses.



# The “Software on Hardware” Tuning Process

## For each Hotspot

- Determine efficiency
- If inefficient:
  - Determine primary bottleneck
  - Identify architectural reason for inefficiency
  - Optimize the issue

Repeat

## Efficiency Method 1: % Retiring Pipeline Slots

**Why:** Helps you understand how efficiently your app is using the processors

**How:** *General Exploration* profile, Metric: *Retiring*

### What Now:

- For a given hotspot:
- If 10% or more of pipeline slots are retiring (.10 or higher), look at the 3 other top-level metric for tuning options.

Function / Call Stack	Clockticks	Instructions Retired	CPI Rate	Retiring
@@_kmp_hyper_barrier_release	1,566,588,348,879	763,205,144,806	2.03	28.2%
@@compute_rhs_Somp\$parallel@17	1,515,622,273,430	96,228,144,342	15.76	3.3%
@@y_solve_Somp\$parallel_for@17	1,390,328,094,409	59,403,098,142	23.46	2.6%
@@z_solve_Somp\$parallel_for@31	1,379,010,068,512	64,234,096,351	21.49	2.7%
@@x_solve_Somp\$parallel_for@27	1,066,933,600,398	70,884,106,326	15.02	3.7%
@@[vmlina]	197,668,296,502	36,600,054,900	5.41	11.8%
@@testnr_Somp\$parallel_for@32	165,968,248,952	5,790,088,065	28.85	2.2%
@@bmmr_Somp\$parallel_for@22	149,052,210,078	6,356,009,534	22.25	2.5%
@@add_Somp\$parallel_for@19	125,748,188,622	4,930,007,395	25.97	1.7%
@@mmmr_Somp\$parallel_for@20	81,542,122,313	3,352,005,028	24.36	2.4%
@@pmmr_Somp\$parallel_for@20	79,990,119,985	3,260,004,890	24.57	2.8%
@@_kmp_hyper_barrier_gather	68,008,102,012	32,818,049,377	2.64	27.7%
@@_kmp_wait_yield_4	55,700,083,550	20,304,030,456	2.33	24.8%
@@_kmp_yield	40,500,060,750	488,000,732	82.82	2.4%
@@[libc-2.19.so]	8,500,012,750	1,692,002,538	5.04	24.9%
@@Bmmnt	7,398,011,097	48,000,072	154.25	0.1%
@@Bmmg	6,598,008,837	54,000,084	117.97	0.1%
@@exact_rhs_Somp\$parallel@20	3,654,005,481	1,064,001,596	3.44	23.4%
@@initialize_Somp\$parallel@22	3,162,004,743	1,382,001,923	2.46	29.4%
@@exact_solution	3,084,004,626	2,514,003,771	1.27	37.9%
@@_kmp_bakery_check	2,862,004,293	62,000,093	46.11	26.7%

Formula:  
 $(UOPS\_RETIRED.ALL / (2 * CPU\_CLK\_UNHALTED.THREAD))$

Thresholds: Investigate if -  
 % Retiring < .10

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

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

**Why:** Another measure of efficiency that can be useful when comparing 2 sets of data

- Shows average time it takes one of your workload's instructions to execute

**How:** *General Exploration* profile, Metric: *CPI Rate*

**What Now:**

- CPI can vary widely depending on the application and platform!
- If code size stays constant, optimizations should focus on reducing CPI

Function / Call Stack	Clockticks	Instructions Retired	CPI Rate
__kmp_hyper_barrier_release	1,566,588,340,879	763,205,144,806	2.053
compute_rhs_Somp\$parallel@17	1,515,622,273,430	96,228,144,342	15.750
y_solve_Somp\$parallel_for@27	1,396,328,094,489	59,428,089,142	23.496
x_solve_Somp\$parallel_for@31	1,379,010,068,512	64,234,096,351	21.469
z_solve_Somp\$parallel_for@27	1,066,933,600,398	70,884,106,326	15.052
[yminus]	197,668,296,502	36,600,054,900	5.401
testnr_Somp\$parallel_for@22	165,968,248,952	5,790,008,605	28.665
testnr_Somp\$parallel_for@22	140,052,210,078	6,356,009,534	22.035
add4_Somp\$parallel_for@19	125,748,188,622	4,930,007,395	25.507
ninvr_Somp\$parallel_for@20	81,542,122,313	3,352,005,028	24.326
pinvr_Somp\$parallel_for@20	79,990,119,985	3,260,004,890	24.517
__kmp_hyper_barrier_gather	68,008,102,012	32,918,049,377	2.066
__kmp_wait_yield_4	55,700,083,550	20,304,030,456	2.743

Copyright © 2018, Intel Corporation. All rights reserved. Other names and brands may be claimed as the property of others.

Optimization Notice



Formula:

$CPU\_CLK\_UNHALTED.THREAD/INST\_RETIRED.ANY$

Threshold:

In the interface, CPI will be highlighted if it is greater than 6. This is a very general rule based on the fact that many tuned applications should be able to get below this threshold. However, many applications will naturally have a CPI of over 6 – it is very dependent on workload and platform. It is best used as a comparison factor – know your app's CPI and see if over time it is moving upward (that is bad) or reducing (good!).

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

Additionally, CPI can be affected if using Intel® Hyper-threading. In a serial workload, or a workload with Intel® Hyper-threading disabled the theoretical best CPI on a hardware thread is 0.5 because the core can allocate and retire 2 instructions per cycle. In a workload with Intel® Hyper-threading enabled which utilizes all 4 hardware threads effectively, the ideal CPI per-thread would be 2 instead of 0.5. This is because the hardware threads share allocation and

retirement resources on the core.

Note: Optimized code (e.g. with AVX512 instructions) may actually increase the CPI, and increase stall % – but improve the performance. This is because a single vector instruction will generally take more cycles than a single scalar instruction, but it also often performs more work. For example, a vector instruction may take twice as many cycles, but perform the work of four scalar instructions. In that case, the average CPI will increase, but the application will still be running faster.

CPI is just a general efficiency metric – the real measure of efficiency is work taking less time.

# The “Software on Hardware” Tuning Process

## For each Hotspot

- Determine efficiency
  - If inefficient:
    - Determine primary bottleneck
    - Identify architectural reason for inefficiency
    - Optimize the issue

Repeat

## Determine the Primary Bottleneck

If Methods 1 or 2 are used to determine code is inefficient, first determine the primary bottleneck.

The Top-Down hierarchy implemented in General Exploration classifies your application's utilization of the CPU cores into 4 categories:

- Front-End Bound
- Back-End Bound
- Bad Speculation
- Retiring

**The primary bottleneck has the highest fraction of pipeline slots, and should be investigated first!**

Copyright © 2016, Intel Corporation. All rights reserved. Other names and brands may be claimed as the property of others.

Optimization Notice



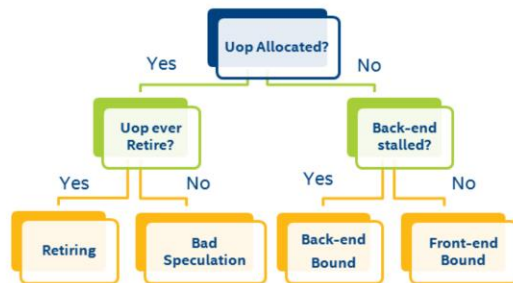
For a hotspot that is inefficient, determining the primary bottleneck is the first step. Optimizing code to fix issues outside the primary bottleneck category may not boost performance – the biggest boost will come from resolving the biggest bottleneck. Generally, if Retiring is the primary bottleneck, that is good. See next slides.

## Issue Classification

A **Pipeline Slot** is an abstract concept – it represents the hardware resources needed to process one micro-operation

On this CPU, there are 2 pipeline slots available on each core, each cycle

Performance is classified according to what happened for each slot available to the application or hotspot:



Copyright © 2016 Intel Corporation. All rights reserved. Other names and brands may be claimed as the property of others.

Optimization Notice



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

# The “Software on Hardware” Tuning Process

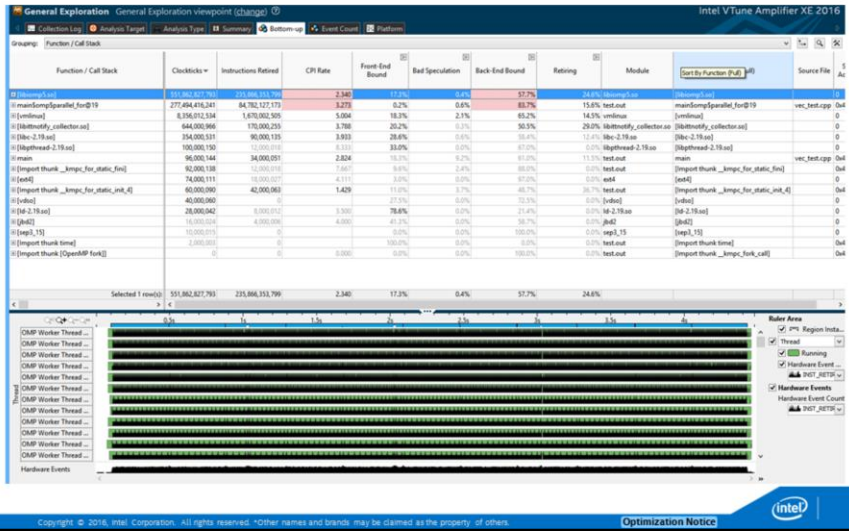
## For each Hotspot

- Determine efficiency
  - If inefficient:
    - Determine primary bottleneck
    - Identify architectural reason for inefficiency
    - Optimize the issue

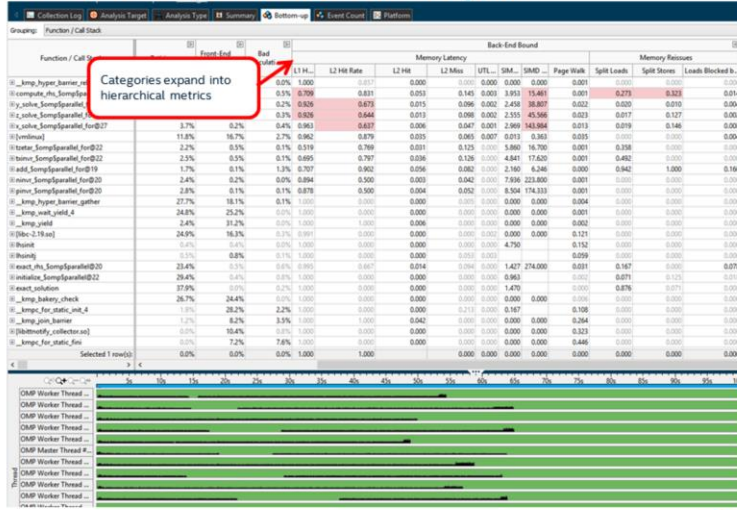
Repeat



# General Exploration Analysis



# General Exploration Analysis



# General Exploration Analysis

## Front-End Bound

### ICache Misses

**Description:**

Missing instruction fetches from the Instruction Cache (ICache) causes stalls in the pipeline. This may be the result of branch-heavy code or poor code layout by the compiler.

**Formula:**
$$\text{ICache Misses} = \frac{\text{FETCH\_STALL\_ICACHE\_FILL\_PENDING\_CYCLES}}{\text{INST\_RETIRED.ANY}}$$
**Threshold:**

Investigate if  $> 0.1$

# General Exploration Analysis

## Bad Speculation

### Branch Mispredict

#### Description:

Mispredicting branch targets causes the processor to execute instructions that will never retire, because they are on the incorrect code path. This represents wasted work and should be minimized.

#### Formula:

Branch Mispredict=  
 $(2 * NO\_ALLOC\_CYCLES.MISPREDICTS)/(2 * CPU\_CLK\_UNHALTED.THREAD)$

#### Threshold:

Investigate if > 0.05

# General Exploration Analysis

## Back-End Bound

### L2 Hit Rate

**Description:**

The L2 is the last, and longest-latency, level in the memory hierarchy before DRAM or MCDRAM. This metrics provides the ratio of demand load requests that hit in L2 to the total number of demand load requests serviced by L2. This metric does not include instruction fetches.

**Formula:**

L2 Hit Rate =  
$$\frac{\text{MEM\_UOPS\_RETIRED.L2\_HIT\_LOADS\_PS}}{\text{MEM\_UOPS\_RETIRED.L2\_HIT\_LOADS\_PS} + \text{MEM\_UOPS\_RETIRED.L2\_MISS\_LOADS\_PS}}$$

**Threshold:**

Investigate if < 0.80

# General Exploration Analysis

## Back-End Bound

### L2 Hit

#### Description:

The L2 is the last, and longest-latency, level in the memory hierarchy before DRAM or MCDRAM. While L2 hits are serviced much more quickly than hits in DRAM, they can still incur a significant performance penalty. This metrics provides the ratio of cycles spent in servicing demand load requests that hit in L2 to the total number of cycles.

#### Formula:

$$\text{L2 Hit Penalty} = (17 * \text{MEM\_UOPS\_RETIRED.L2\_HIT\_LOADS\_PS} / \text{CPU\_CLK\_UNHALTED.THREAD})$$

#### Threshold:

Investigate if  $> 0.10$

# General Exploration Analysis

## Back-End Bound

### L2 Miss

#### Description:

The L2 is the last and longest-latency level in the memory hierarchy before the main memory (DRAM) and MCDRAM. Any memory requests missing here must be serviced by either DRAM or MCDRAM, with significant latency. The L2 Miss metric shows ratio of cycles spent in servicing demand load requests that miss in L2 to the total number of cycles.

#### Formula:

L2 Miss Penalty =  
 $(230 * \text{MEM\_UOPS\_RETIRED.L2\_MISS\_LOADS\_PS} / \text{CPU\_CLK\_UNHALTED.THREAD})$

#### Threshold:

Investigate if  $> 0.15$

# General Exploration Analysis

## Retiring

### VPU Utilization

#### Description:

This metric measures the fraction of micro-ops (uops) that performed packed vector operations of any vector length and any mask. VPU utilization metric can be in conjunction with the compiler's vectorization report to assess VPU utilization and to understand the compiler's judgement about the code. Note that this metric includes integer packed simd uops but does not account for loads and stores. Also, this metric does not take into consideration the uop masking behavior or vector length of the uops.

#### Formula:

$$\text{Vector VPU Compute Percentage} = \frac{\text{UOPS\_RETIRED.PACKED\_SIMD}}{\text{UOPS\_RETIRED.PACKED\_SIMD} + \text{UOPS\_RETIRED.SCALAR\_SIMD}}$$

#### Threshold:

Investigate if < 0.5



# General Exploration Analysis

## Retiring

### Divider

#### Description:

Not all arithmetic operations take the same amount of time. Divides and square roots, both performed by the DIV unit, take considerably longer than integer or floating point addition, subtraction, or multiplication. This metric measures the fraction of total cycles when DIV unit was active. Note that this metric accounts only for the following division operations: integer div, x87 div, divss, divsd, sqrtss, sqrtsd.

#### Formula:

Divider = (CYCLES\_DIV\_BUSY.ALL) / (CPU\_CLK\_UNHALTED.THREAD)

#### Threshold:

Investigate if > 0.05

# General Exploration Analysis

## Retiring

### FP Assists

#### Description:

Certain floating point operations cannot be handled natively by the execution pipeline and must be performed by microcode (small programs injected into the execution stream). For example, when working with very small floating point values (so-called denormals), the floating-point units are not set up to perform these operations natively. Instead, a sequence of instructions to perform the computation on the denormal is injected into the pipeline. Since these microcode sequences might be hundreds of instructions long, these microcode assists are extremely detrimental to performance. This metric also accounts for other FP assists such as Flush-To-Zero (FTZ).

#### Formula:

$$\text{FP Assists} = (\text{MACHINE\_CLEARS.FP\_ASSIST}) / (\text{INST\_RETIRED.ANY})$$

#### Threshold:

Investigate if  $> 0.05$

## Additional Topic: Metric Reliability

General Exploration General Exploration viewpoint (change) ⓘ

Collection Log Analysis Target Analysis Type Summary Bottom-up PMU Events Platform

Grouping: Function / Call Stack

Function / Call Stack	Clocktic...	Instructions Retired	CPI Rate	Filled Pipeline Slots		Unfilled Pipeline Slots (Stalls)	
				Retiring	Bad Speculation	Back-End Bound	Front-End Bound
grid_intersect	14,076,021,114	12,468,018,702	1.129	0.210	0.076	0.650	0.063
sphere_intersect	9,306,013,959	9,206,013,809	1.011	0.282	0.038	0.615	0.065
grid_bounds_intersect	1,098,001,647	690,001,035	1.591	0.123	0.020	0.781	0.075
func@0x1002e3d5	922,001,383	700,001,050	1.317	0.000	0.000	1.000	0.000
_kmp_x86_pause	354,000,531	212,000,318	1.670	0.000	0.000	1.000	0.000
tri_intersect	222,000,333	152,000,228	1.461	0.405	0.000	0.561	0.101
pos2grid	212,000,318	186,000,279	1.140	0.248	0.000	0.717	0.035
pos2grid	14,076,021,114	12,468,018,702	1.129	0.210	0.076	0.650	0.063

Selected 1 row(s):

Grayed out metric values represent low reliability of the metrics for each value in the grid.

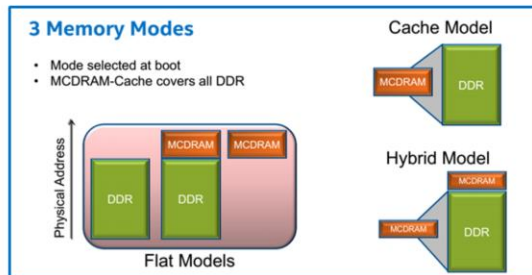
Copyright © 2018, Intel Corporation. All rights reserved. Other names and brands may be claimed as the property of others. Optimization Notice intel

The General Exploration analysis type multiplexes hardware events during collection, which can result in imprecise results if too few samples are collected. The GUI will gray out metrics if the reliability is low based on the number of samples collected. If a metric is grayed out for your area of interest, consider increasing the runtime of the analysis or allowing multiple runs via the project properties.

Previous versions of the tool used a MUX Reliability metric for each row, however this was unable to distinguish between different metrics on the same row.

## Memory Access Analysis

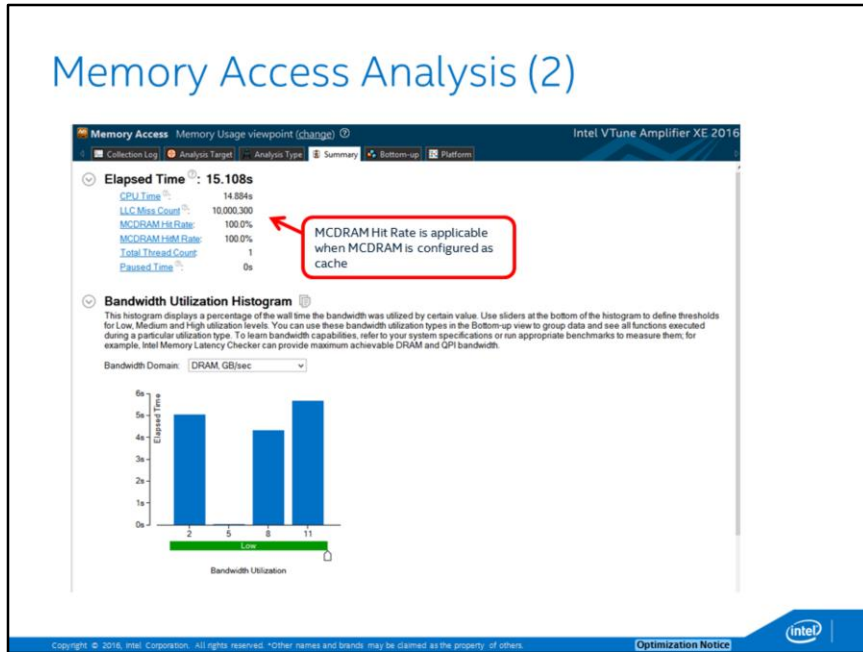
- Provides individual bandwidth information for both, MCDRAM and DDR.
- VTune cannot yet identify the system configuration: cluster mode and memory modes. Hence, shows the bandwidth information for both cache and flat mode. Users need to choose the correct data based on system configuration.



## Memory Access Analysis

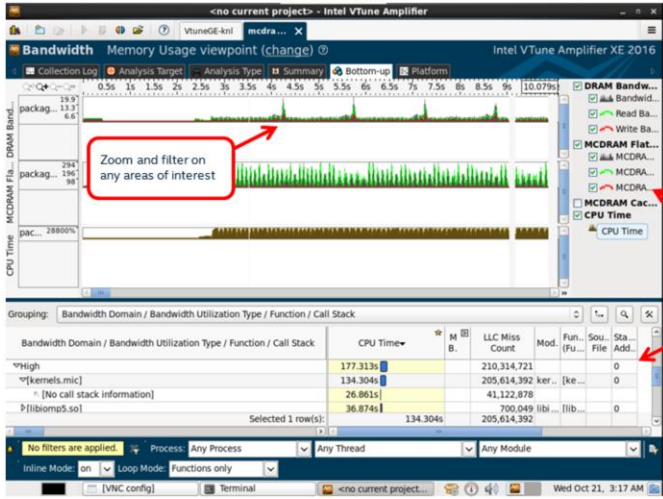
- **Flat Mode**
  - Allows the developers to explicitly control which data structures are in MCDRAM vs. DDR.
  - Requires code modification, otherwise DDR will be used by default.
- **Cache Mode**
  - No code modification. Hardware will use L1, then L2, then MCDRAM cache.
  - Data allocated into MCDRAM cache needs to be highly reusable to see performance benefits.
  - Average L2 miss latency is higher because misses in MCDRAM cache then go to DDR
  - May have aliasing issues if multiple pages are mapped into same cache lines in MCDRAM Cache. Non deterministic (not repeatable)
  - Streaming stores are negatively impacted. Expect lower bandwidth

## Memory Access Analysis (2)



- Notes about MCDRAM Hit Rate
  - This rate counts loads and streaming stores, e.g. vmovnt (non-temporal), but not stores/writebacks
  - If you have streaming stores, your MCDRAM Hit Rate may be lower than expected because streaming stores are expected to miss MCDRAM Cache

# Memory Access Analysis (3)

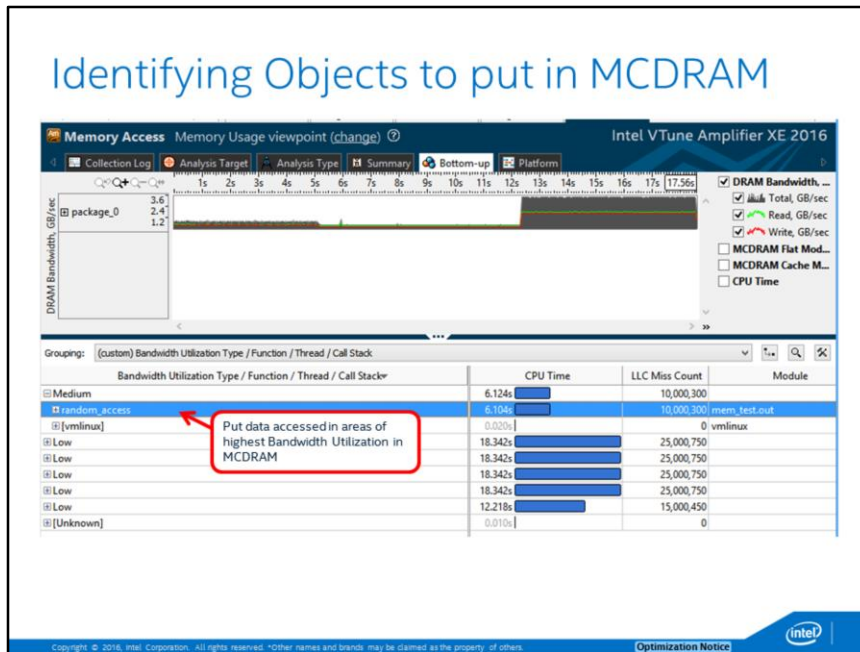


Select Flat or Cache mode for MCDRAM

Memory information at source code level.



## Identifying Objects to put in MCDRAM



Run a Memory Access analysis with MCDRAM configured in flat mode and all allocations occurring in DDR (not using MCDRAM). Create a custom grouping in the Memory Access analysis to see functions causing Medium or High bandwidth utilization. Objects accessed within these functions may be candidates to move into MCDRAM.



## KNL Cluster Mode Performance Tuning

- **Quadrant Cluster Mode**
  - This configuration allows for an increase in usable bandwidth on the mesh because there is less traffic crossing quadrant boundaries. It is generally expected to offer better performance than all-to-all mode but does require that all DDR channels be populated identically.
- **Sub-NUMA Cluster Mode (SNC4)**
  - This mode is expected to be preferable when threads running on the chip can be grouped and affinity-tuned to specific quadrants of tiles and they mostly access their own data. A single data structure or array will normally be mapped to only a pair of MCDRAM channels, or half of the DDR channels. Therefore, the accessible bandwidth to that structure will be less than what it would be in quadrant mode because it is not spread evenly across all channels. While this may seem undesirable, it is important to remember that if the chip is being used to run multiple MPI ranks, or multiple processes, then the total available bandwidth of the system is likely to be highest in this mode. Note that if you try and allocate more memory than is available in your local cluster, the additional memory will be allocated on another cluster. This is expected and does not cause an exception or some other error to occur.
- **All-to-All Cluster Mode**
  - This mode is rarely used for performance. This is the fallback mode in the event of system asymmetries or irregularities.

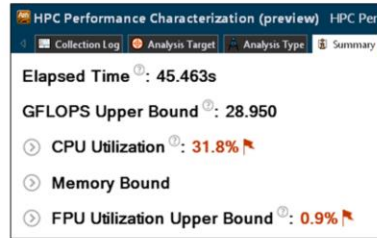
## HPC Performance Characterization

### Two characterization metrics

- Elapsed Time
- GFLOPs Upper Bound\*

### Three performance aspects

- CPU Utilization
- Memory Bound
- FPU Utilization Upper Bound\*



\*Calculated based on FLOP HW counters assuming full vector utilization

Copyright © 2016 Intel Corporation. All rights reserved. Other names and brands may be claimed as the property of others.

Optimization Notice



HPC Performance Characterization provides performance information that is especially important for High Performance Computing (HPC) applications. This analysis can be run from the GUI or using the command line flag “-collect hpc-performance”

# HPC Performance Characterization

## CPU Utilization

- % of "Effective" CPU usage by the application under profiling (threshold 90%)
  - Under assumption that the app should use all available logical cores on a node
  - Subtracting spin/overhead time spent in MPI and threading runtimes

### Metrics in CPU utilization section

- Average CPU usage
- Additional MPI and OpenMP scalability metrics impacting effective CPU utilization
- CPU usage histogram

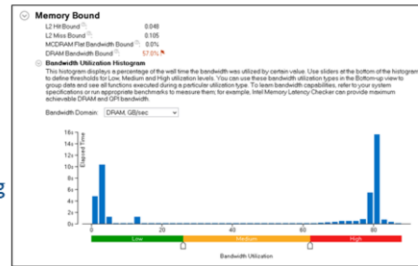


The CPU Utilization metrics provide another way to determine how busy all of the cores are during the performance analysis.

# HPC Performance Characterization

## Metrics in Memory Bound section

- L2 Hit Bound
  - Cost of L1 misses served in L2
- L2 Miss Bound
  - Cost of L2 misses
- MCDRAM Bandwidth Bound
  - % of app elapsed time consuming high MCDRAM Bandwidth
- MCDRAM Bandwidth Bound
  - % of app elapsed time consuming high MCDRAM Bandwidth
- Bandwidth utilization histogram



Copyright © 2016, Intel Corporation. All rights reserved. Other names and brands may be claimed as the property of others.

Optimization Notice



The Memory Bound metrics provide information about how the application is utilizing, and possibly bottlenecked by, the memory subsystem. If issues are exposed here, the Memory Access analysis may provide even more detailed information.

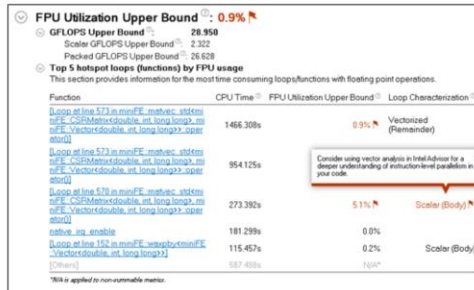
# HPC Performance Characterization

## FPU Utilization Upper Bound

- % of FPU load (100% when FPU is fully loaded, threshold 50%)

## Metrics in FPU utilization section

- GFLOPs broken down by scalar and packed
- Top 5 loops/functions by FPU usage
  - Dynamically generated issue descriptions on low FPU usage help to define the cause and next steps



Copyright © 2018, Intel Corporation. All rights reserved. Other names and brands may be claimed as the property of others.

Optimization Notice



The Floating Point Units (FPU) on KNL are important for getting the best performance. The HPC Performance Characterization provides an upper bound estimate of the utilization. This is an upper bound because the events used are not able to account for masking, and the metric assumes all vector lanes are used in each instruction.

## VTune Amplifier Tips

- **VTune Finalization:**
  - Finalization is very slow on KNL. Finalize on Xeon.
  - Disable auto finalization with: `-no-auto-finalize`
- **Large amount of raw data collected:**
  - Appropriately select the app run duration using: `-target-duration-type=<veryshort/short/medium/long>`
  - Change the default data limit as required.
- **Power throttling:**
  - Keep an eye on the CPU frequency ratio. If this ratio changes significantly during the run then you might be seeing throttling or turbo effects.

## VTune Amplifier Tips (cont.)

- Event multiplexing:
  - Similar to KNC, KNL has only 2 general purpose counters. Hence, when collecting a large number of events the data might be statistically invalid.
  - Try changing the target duration type or allow multiple runs.

# Boost Vectorization with Intel® Advisor

Intel Advisor XE has a new feature to help analyze existing vectorization and guide you through improving vectorization use.

**1. Compiler diagnostics + Performance Data + SIMD efficiency information**

**2. Guidance: detect problem and recommend how to fix it**

**3. "Accurate" Trip Counts + FLOPs: understand utilization, parallelism granularity & overheads**

Loop	Trips	FLOPs
loop1	1000000	1000000000
loop2	1000000	1000000000
loop3	1000000	1000000000
loop4	1000000	1000000000
loop5	1000000	1000000000

**4. Loop-Carried Dependency Analysis**

ID	Type	Site Name
P1	Prohibit site information	s1n2
P2	Read after write dependency	s1n2
P3	Read after write dependency	s1n2
P4	Write after write dependency	s1n2
P5	Write after write dependency	s1n2
P6	Write after read dependency	s1n2
P7	Write after read dependency	s1n2

**5. Memory Access Patterns Analysis**

ID	Site	Type	Source	Includes	Alignment
000	000	loop stride	loop/looploop000	looploop	
001	001	loop stride	loop/looploop001	looploop	
002	002	loop stride	loop/looploop002	looploop	
003	003	loop stride	loop/looploop003	looploop	
004	004	loop stride	loop/looploop004	looploop	
005	005	loop stride	loop/looploop005	looploop	
006	006	loop stride	loop/looploop006	looploop	
007	007	loop stride	loop/looploop007	looploop	
008	008	loop stride	loop/looploop008	looploop	
009	009	loop stride	loop/looploop009	looploop	

Copyright © 2016, Intel Corporation. All rights reserved. Other names and brands may be claimed as the property of others. Optimization Notice

Use this 5 step process to determine how well you are vectorizing and where you can improve. Intel® Advisor is available at: <https://software.intel.com/en-us/intel-advisor-xe>



# Survey Analysis

Function Call Sites and Loops	Vector Issues	Vectorized Loops	Instruction Set Analysis
		Vect.   Efficiency %   Gen...   VL...   Traits   Data T.	
loop in s241, at lo...		AVX   95%   7.71x   8	Float32
loop in s152, at lo...		AVX2   95%   7.71x   8   FMA	Float32
loop in s452, at lo...	1 Data type conversions present	AVX2   95%   7.71x   8   FMA, Type Con...	Float32
loop in s413, at lo...	1 Ineffective peeled/remainder ...	AVX2   95%   7.66x   8   FMA	Float32
loop in s273, at lo...	1 Possible insufficient memory a...	AVX2   95%   7.66x   8   FMA, Masked St...	Float32
loop in s279, at lo...	1 Possible insufficient memory a...	AVX2   95%   7.56x   8   Blend: FMA	Float32
loop in s253, at lo...	1 Possible insufficient memory a...	AVX2   95%   7.30x   8   Blend: FMA	Float32
loop in s231, at lo...		AVX2   95%   7.23x   8   FMA	Float32
loop in s271, at lo...	1 Possible insufficient memory a...	AVX2   95%   7.15x   8   FMA, Masked St...	Float32
loop in vlf, at loop...	1 Possible insufficient memory a...	AVX   95%   6.90x   8   Blend:	Float32
loop in s274, at lo...	1 Possible insufficient memory a...	AVX2   95%   6.29x   8   Blend: FMA; M...	Float32
loop in SETD2, at m...		AVX   95%   5.81x   8	Float32
loop in st4_Fill4...		AVX   95%   5.81x   8	Float32
loop in SETD2, at m...	1 Data type conversions present	AVX2   95%   5.31x   8   Divisions; Type ...	Float32

Vectorized
  Not Vectorized

- **Sort** – Look at your hottest vectorized loops
- **Efficiency** – use as a performance thermometer
- **Recommendations** – get tips on how to improve performance

**Issue: Assumed dependency present**

**Issue: Ineffective peeled/remainder loop(s) present**

All or some `source` loop iterations are not executing in the `loop` body. Improve performance by moving sou...

**Recommendation: Add data padding**

The `loop` count is not a multiple of `vector` length. To fix: Do one of the following:

- Increase the size of objects and add iterations so the trip count is a multiple of vector length.
- Increase the size of static and automatic objects, and use a compiler option to add data padding

Windows® OS	Linux® OS
/Qopt-assume-safe-padding	-qopt-assume-safe-padding

**Note:** These compiler options apply only to Intel® Many Integrated Core Architecture (Intel® MIC Archi...

When you use one of these compiler options, the compiler does not add any padding for static and aut application. To satisfy this assumption, you must increase the size of static and automatic objects in y

**Optional:** Specify the trip count, if it is not constant, using a `directive` `#pragma loop_count`

**Read More:**

`-qopt-assume-safe-padding, /Qopt-assume-safe-padding, loop_count`

# Summary View: Plan Your Next Steps

The screenshot displays the Intel VTune Summary View. On the left, a sidebar shows a tree view of the application's execution. The main area is divided into several sections:

- Vectorization Gain/Efficiency:** Shows a progress bar for "Vectorized Loops Gain/Efficiency" at 66% and "Program Theoretical Gain" at 2.64x.
- Top time-consuming loops:** A table listing loops with their source locations, self times, and total times.

Loop	Source Location	Self Time <sup>2)</sup>	Total Time <sup>2)</sup>
matvec	<a href="#">Multiply.c:72</a>	5.6256s	5.6256s
matvec	<a href="#">Multiply.c:66</a>	2.4880s	2.4880s
matvec	<a href="#">Multiply.c:49</a>	0.5234s	6.1490s
matvec	<a href="#">Multiply.c:49</a>	0.4088s	2.4396s
matvec	<a href="#">Multiply.c:85</a>	0.1150s	0.1150s

What can I expect to gain?

Where do I start?

# Factors That Can Affect Efficiency

**2.19x** Vectorization Gain

**~55%** Vectorization Efficiency

**1.A. Indirect memory access**

```
for (i=0; i<N; i++)
    A[i] = C[i]*D[i]
```

**1.B. Memory sub-system Latency / Throughput**

```
void main(int *a, int *b)
{
    for (int i = 0; i < VSIZE; i++)
        a[i] = b * a[i] * i;
        b[i] = b * a[i];
}
```

**2. Serialized or "sub-optimal" function calls**

```
for (i = 1; i < N; i++) {
    a[i] = a[i] +
        serial_func_call(x, y);
}
```

**3. Small trip counts that are not a multiple of the Vector Length**

```
void main(int *a, int *b, int
unknown_small_val)
{
    for (int i = 0; i <
unknown_small_val;
i++)
        a[i] = b*b[i];
}
```

**4. Branchy codes, outer vs. inner loops**

```
for (i = 0; i <= MAX; i++) {
    if ( D[i] < 0)
        do_this_D();
    else if (D[i] > 0)
        do_that_D();
    //...
}
```

**2.2 Check Memory Access Patterns**

Command Line

Input: `loop in a\bin\cap\lib\intel\intel64\parallel\loop1_48K`

Issue: `Intel(R) Parallel Studio: compiler support present.`

All of some `Intel(R) Loop` features are not working in the `loop1_48K`. Improve performance

Recommendation: Add data padding.

The `loop1_48K` is not a multiple of `vector_length`. To fix, do one of the following:

- increase the size of objects and add elements to the trip count in a multiple of `vector_length`.
- increase the size of static and automatic objects, and set a compiler option to a multiple of `vector_length`.

Optimization: Identify the trip count, if it is not constant, using a `break` statement.

Code: `main_array_safety_padding, Data: main_array_safety_padding, loop_count`

- main\_array\_safety\_padding, Data: `main_array_safety_padding, loop_count`
- main\_array\_safety\_padding, Data: `main_array_safety_padding, loop_count`

Recommendation: Collect trip counts data

Recommendation: Disable unrolling

Recommendation: Specify the expected loop trip count

Recommendation: Use a smaller vector length

Copyright © 2016, Intel Corporation. All rights reserved. Other names and brands may be claimed as the property of others. Optimization Notice

Analyze the hot loops for the common issues that can impact vectorization. Use the Memory Access Patterns Analysis and Recommendations to identify problematic behaviors and ways to correct them.

## Check if it is Safe to Vectorize

Loop-Carried Dependencies Analysis Verifies Correctness

Function Call Sites and Loops	Self Time	Total Time	Trip Counts	Compiler Vectorization
[loop at Multiply.c:53 in matvec]	0.047s	0.047s	3	Vectorized (Body)
[loop at Multiply.c:53 in matvec]	0.413s	0.413s	101	Scalar
[loop at Multiply.c:45 in matvec]	0.109s	12.379s	1	Collapsed
[loop at Multiply.c:45 in matvec]	0.079s	11.830s	12	Vectorized (Body)
[loop at Multiply.c:45 in matvec]	0.031s	0.646s	2	Remainder
loop at Driver.c:148 in main	0.071s	12.483s	1,000,000	Scalar vector dependence prevents vectorization



Data dependencies between loop iterations make it difficult for the compiler to vectorize a loop. For example:

```
for (i = 1; i < N; i++) {
    A[i] = A[i-1] + C[i];
}
```

Each iteration is dependent on the value calculated in the previous iteration. Use Advisor to detect these dependencies.

# Improve Vectorization

## Memory Access Pattern Analysis

Function Call Sites and Loops	Vectorized	Not Vectorized	Loop Type	Why No Vectorization?
Loop at fractal.cpp:179 in <lambda1>:cop...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Collapsed	Collapsed
Loop at fractal.cpp:179 in <lambda1>:cop...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Vectorized (Block)	
Loop at fractal.cpp:179 in <lambda1>:co...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Peel	Peel
Loop at fractal.cpp:179 in <lambda1>:co...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Remainder	Remainder
Loop at fractal.cpp:177 in <lambda1>:oper...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Scalar	Scalar

2.2 Check Memory Access Patterns  
Identify and optimize complex memory access for marked loops. Fix the reported problems.

Command Line

Run Memory Access Patterns analysis to check how memory is used in the loop and the called function

The strides of memory accesses can affect vectorization. Determine the patterns to learn which loops may be difficult to vectorize.

# Memory Analysis Is Critical

## Determine Possible Bandwidth or Latency Issues

Footprint	Small enough	Big enough	Source	Stride	Operand Type	Operand Size	Aggregated footprint
Access Pattern			<pre> int i; for (i = 0; i &lt; N; i++) {     *p1 = *p2 + i;     *p2 = *p1 + i;     *p3 = *p2 + i;     *p4 = *p3 + i; } </pre>		int	32	48
Unit Stride	Effective SIMD No Latency and BW bottlenecks	Effective SIMD Bandwidth bottleneck	<pre> int i; for (i = 0; i &lt; N; i++) {     *p1 = *p2 + i;     *p2 = *p1 + i;     *p3 = *p2 + i;     *p4 = *p3 + i; } </pre>		int	32	96
Const stride	Medium SIMD Latency bottleneck possible	Medium SIMD Latency and Bandwidth bottleneck possible	<pre> int i; for (i = 0; i &lt; N; i++) {     *p1 = *p2 + i;     *p2 = *p1 + i;     *p3 = *p2 + i;     *p4 = *p3 + i; } </pre>				
Irregular Access, Gather/Scatter	Bad SIMD Latency bottleneck possible	Bad SIMD Latency bottleneck	<pre> int i; for (i = 0; i &lt; N; i++) {     *p1 = *p2 + i;     *p2 = *p1 + i;     *p3 = *p2 + i;     *p4 = *p3 + i; } </pre>				

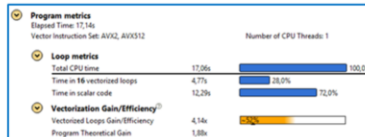
  

Assembly	Physical/Stride	Operand Info	Address range	Memory access
<pre> mov rax, rdi mov rsi, rdi mov rdi, rdi mov rdi, rdi mov rdi, rdi </pre>				
<pre> mov rax, rdi mov rsi, rdi mov rdi, rdi mov rdi, rdi mov rdi, rdi </pre>				
<pre> mov rax, rdi mov rsi, rdi mov rdi, rdi mov rdi, rdi mov rdi, rdi </pre>				
<pre> mov rax, rdi mov rsi, rdi mov rdi, rdi mov rdi, rdi mov rdi, rdi </pre>				

Copyright © 2016, Intel Corporation. All rights reserved. Other names and brands may be claimed as the property of others. Optimization Notice 53

# AVX-512 Specifics for KNL

1. Native AVX-512 profiling on KNL
2. Precise FLOPs and Mask Utilization profiler
3. AVX-512 Advice and Traits
4. AVX-512 Gather/Scatter Profiler



FLOPS And AVX-512 Mask Usage		Vectorized Loops				Instruction Set Analysis	
Gflops	AI	Mask Utilization	Vector...	Efficiency	Gain Estim...	VL (...)	Traits
2.080	0.1243	100.0%	AVX512	100%	17.50x	16; 8	FMA; Mask Manipulations
0.856	0.0809	91.7%	AVX512	100%	17.69x	16; 8	FMA; Mask Manipulations
0.455	0.1398	89.6%	AVX512	100%	14.41x	16; 8	FMA; Mask Manipulations
0.234	0.1472	100.0%					Appr. Reciprocals(AVX-512ER); Expon...
0.148	0.1429						FMA
0.095	0.0722	40.1%					FMA; Square Roots; Type Conversions
0.091	0.0208						FMA
0.074	0.1429						FMA

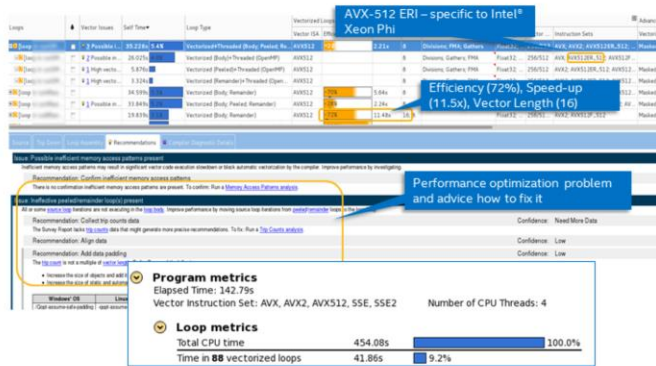
Copyright © 2016 Intel Corporation. All rights reserved. Other names and brands may be claimed as the property of others.

Optimization Notice



# Vectorization Advisor on KNL AVX-512

See the Intel Advisor tutorials and documentation to learn how to analyze your KNL application.





# Good Luck!

## For more information:

VTune Amplifier XE Videos, Forums, and Resources:  
<http://software.intel.com/en-us/intel-vtune-amplifier-xe/#pid-3659-760/>

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

VTune Amplifier XE Tuning Guides for Other microarchitectures:  
<http://software.intel.com/en-us/articles/processor-specific-performance-analysis-papers>

# Legal Disclaimer

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document or other Intel literature may be obtained by calling 1-800-548-4725 or by visiting Intel's website.

Intel® Hyper-Threading Technology requires a computer system with a processor supporting HT Technology and an HT Technology-enabled chipset, BIOS and operating system. Performance will vary depending on the specific hardware and software you use. For more information including details on which processors support HT Technology, see [here](#).

Intel® Virtualization Technology requires a computer system with an enabled Intel® processor, BIOS, virtual machine monitor (VMM) and, for some uses, certain computer system software enabled for it. Functionality, performance or other benefits will vary depending on hardware and software configurations and may require a BIOS update. Software applications may not be compatible with all operating systems. Please check with your application vendor.

64-bit computing on Intel architecture requires a computer system with a processor, chipset, BIOS, operating system, device drivers and applications enabled for Intel® 64 architecture. Performance will vary depending on your hardware and software configurations. Consult with your system vendor for more information.

\*Intel® Turbo Boost Technology requires a PC with a processor with Intel Turbo Boost Technology capability. Intel Turbo Boost Technology performance varies depending on hardware, software and overall system configuration. Check with your PC manufacturer on whether your system delivers Intel Turbo Boost Technology. For more information, see <http://www.intel.com/technology/turboboost>.

Intel, the Intel logo, Xeon, Xeon Inside, VTune, inTru, and Core are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

\*Other names and brands are the property of their respective owners.

Copyright © 2014, Intel Corporation

Copyright © 2014, Intel Corporation. All rights reserved. \*Other names and brands may be claimed as the property of others.

Optimization Notice



## Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

