

Performance Tuning on KNL with Intel® Parallel Studio XE 2017 Cluster Edition

James Tullos

Technical Consulting Engineer

Intel Corporation

Champaign, IL, USA

Takeaways from Workshop

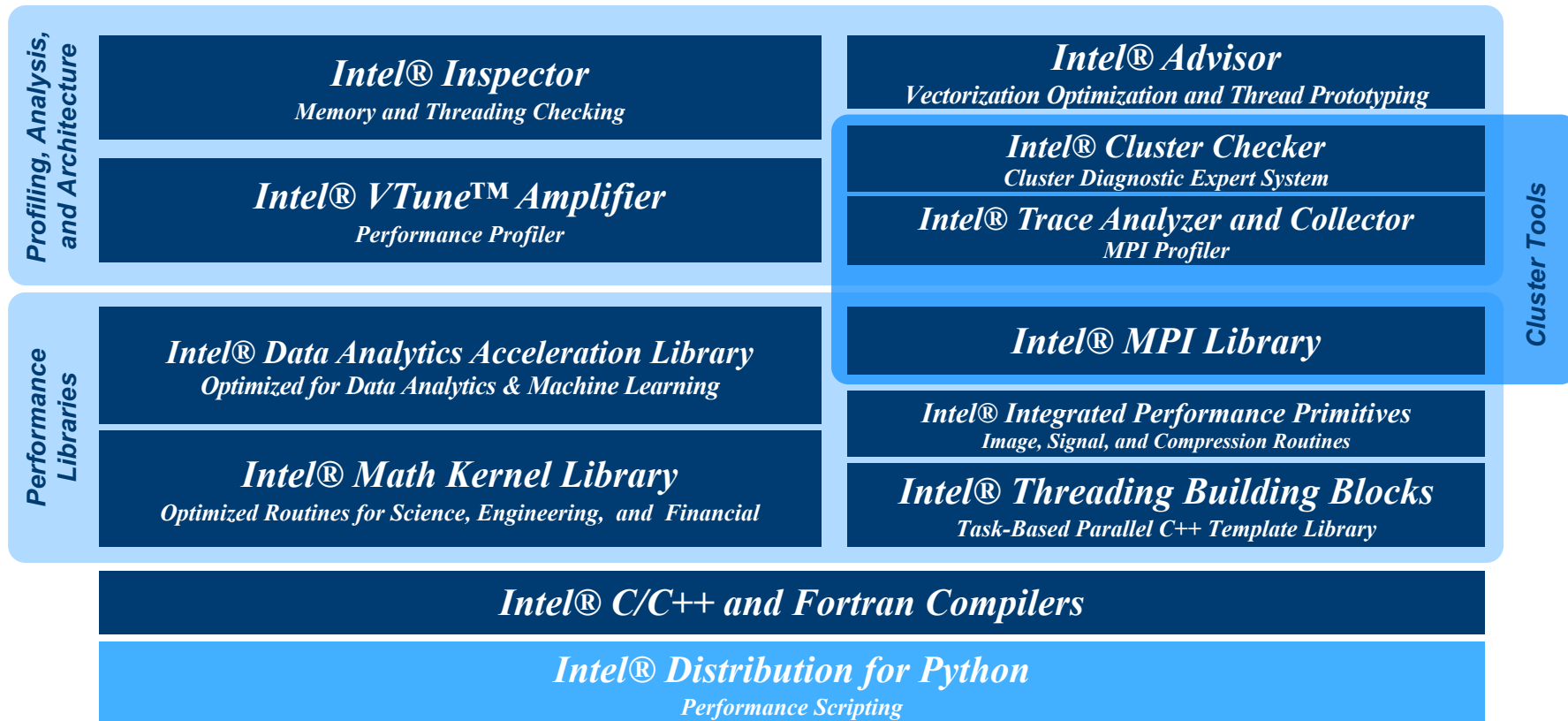
Awesome
Learning
Experience

Performance
is not Simple

Variety of
Techniques

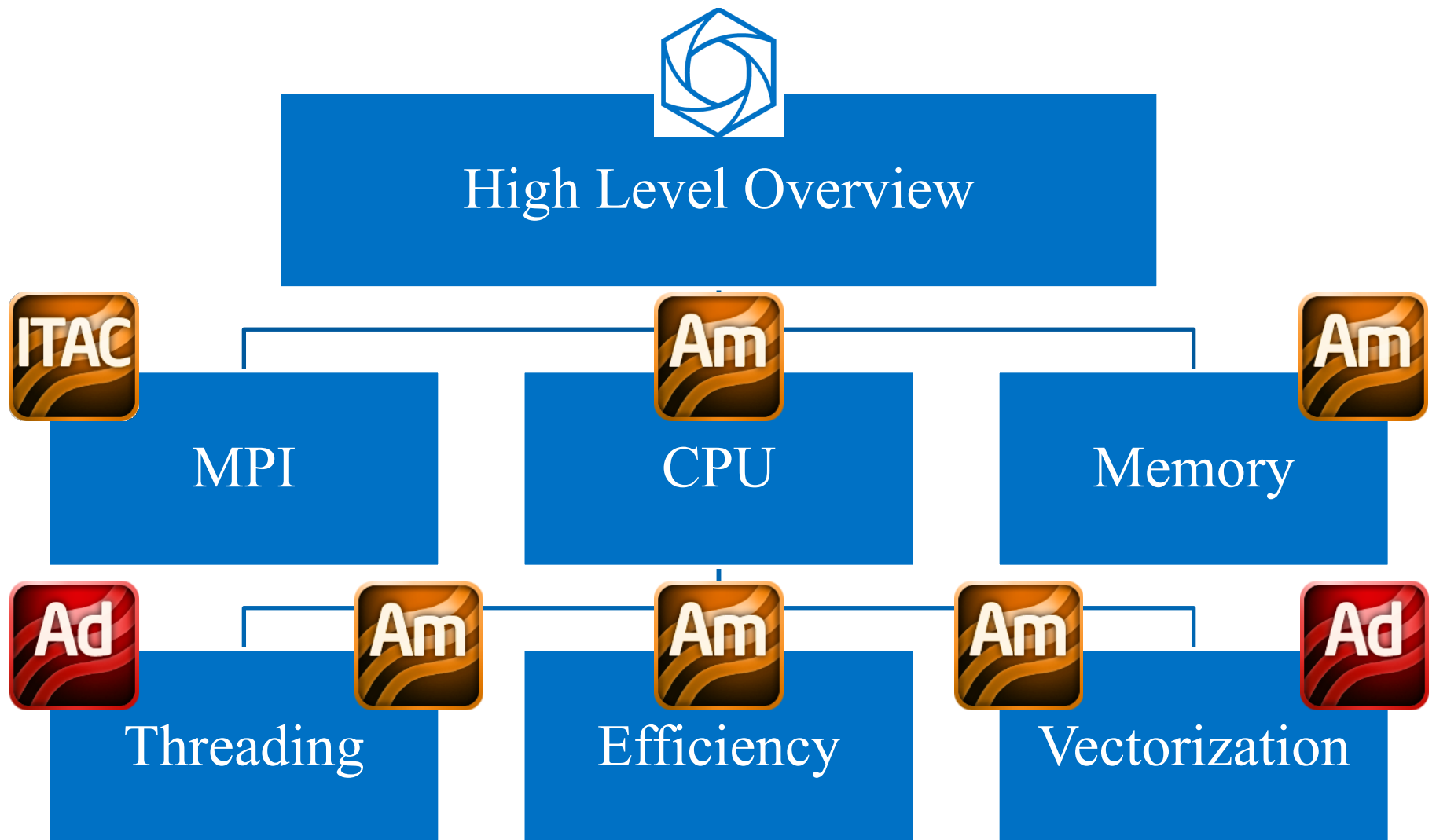
Variety of
Tools

Intel® Parallel Studio XE 2017 Cluster Edition



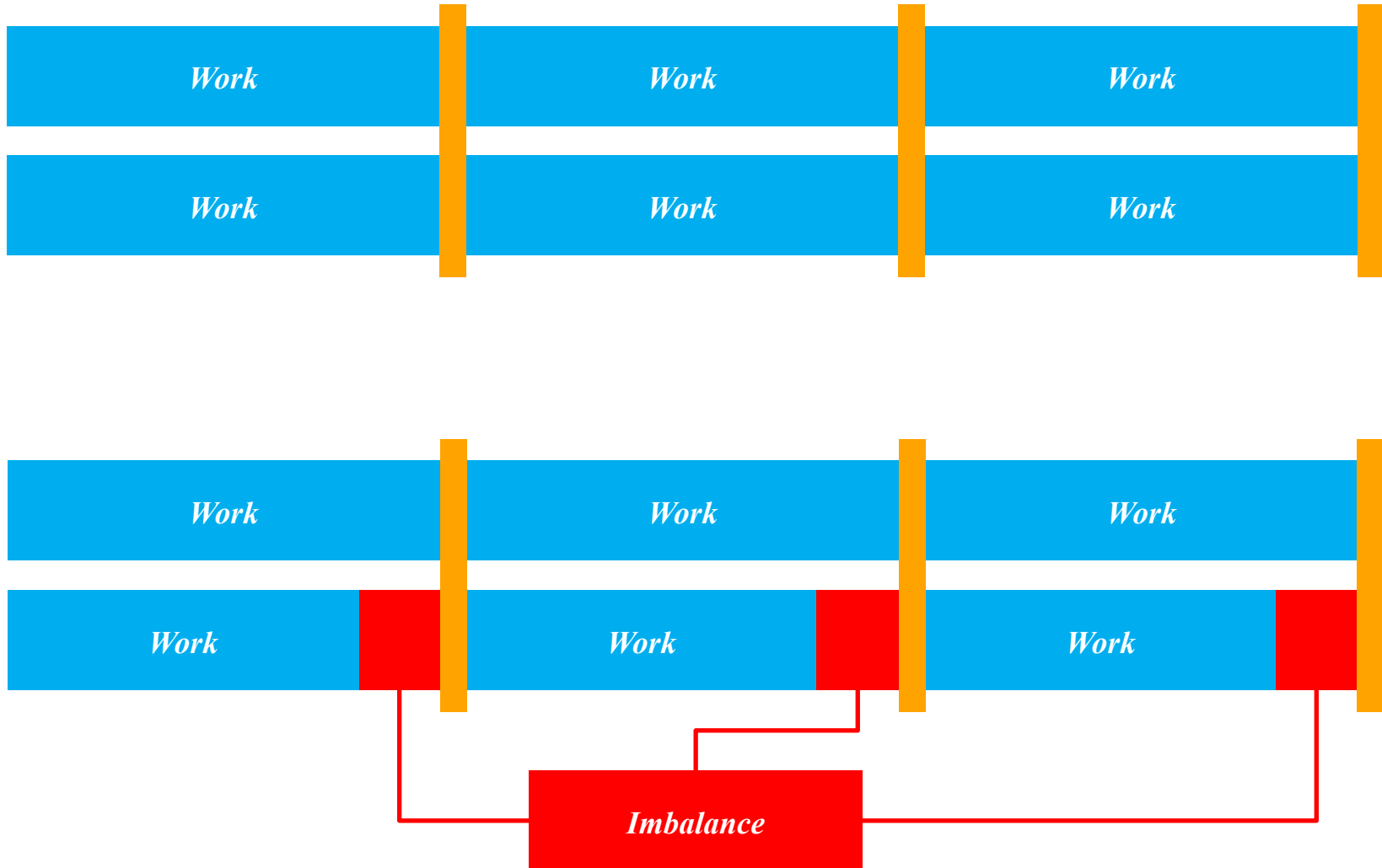
Tools for Performance Tuning on KNL

Tuning Methodology



Intel Performance Snapshots

Defining Imbalance in Parallelism



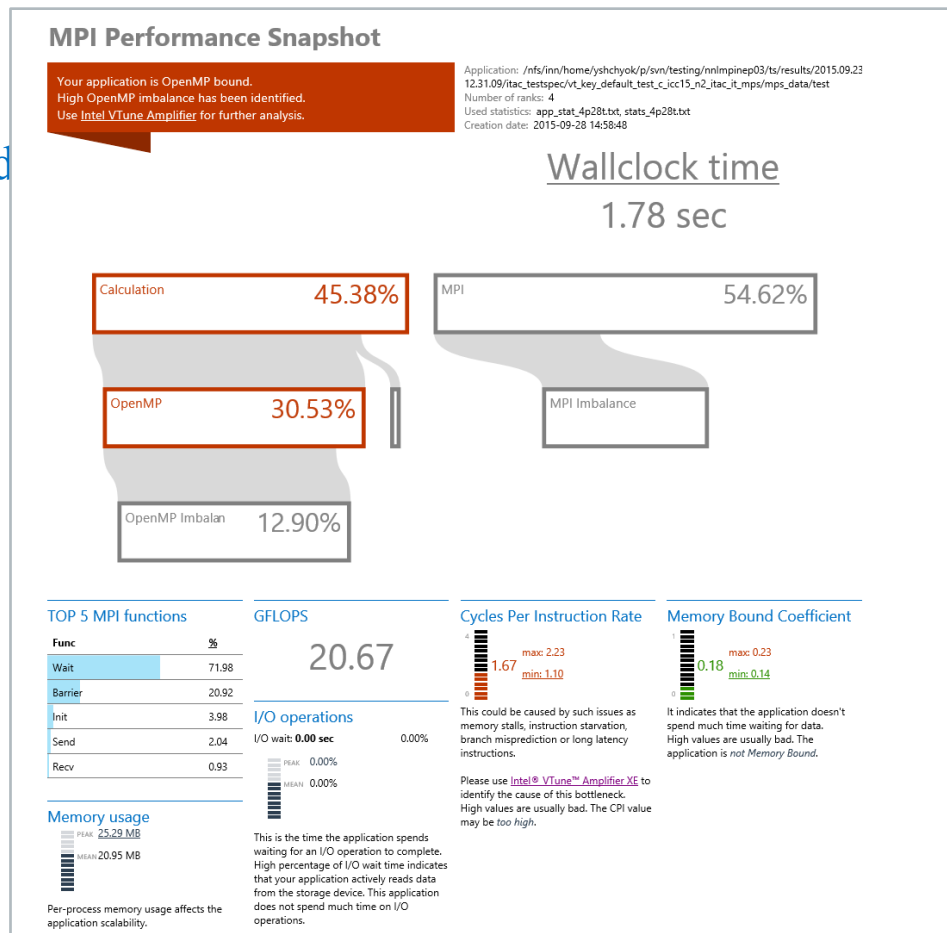
MPI Performance Snapshot

Collector can work with any MPICH-based MPI implementation

New mechanism of MPI imbalance collection based on MPI_T mechanism – Intel® MPI Library only

OpenMP* imbalance from Intel OpenMP* implementation

Simplified HTML report



Application Performance Snapshot

Discover opportunities for better performance with vectorization & threading

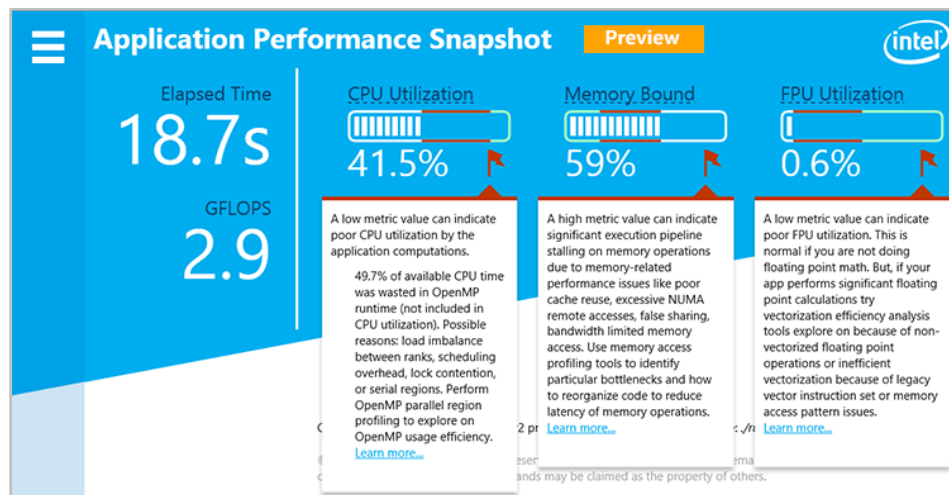
Preview!

Objectives

- Simple enough to run during a coffee break
- Highlight where code modernization can help

Users

- Performance teams – fast prioritization of which apps will benefit most
- All Developers – size the potential performance gain from code modernization



Non-Objectives

- *Actionable tuning data – that is another tool. Snapshot is just a fast “health” check.*

Free download: <http://www.intel.com/performance-snapshot>
Also included with Intel® Parallel Studio and Intel® VTune™ Amplifier products.

Tuning MPI Application Performance with Intel® Trace Analyzer and Collector

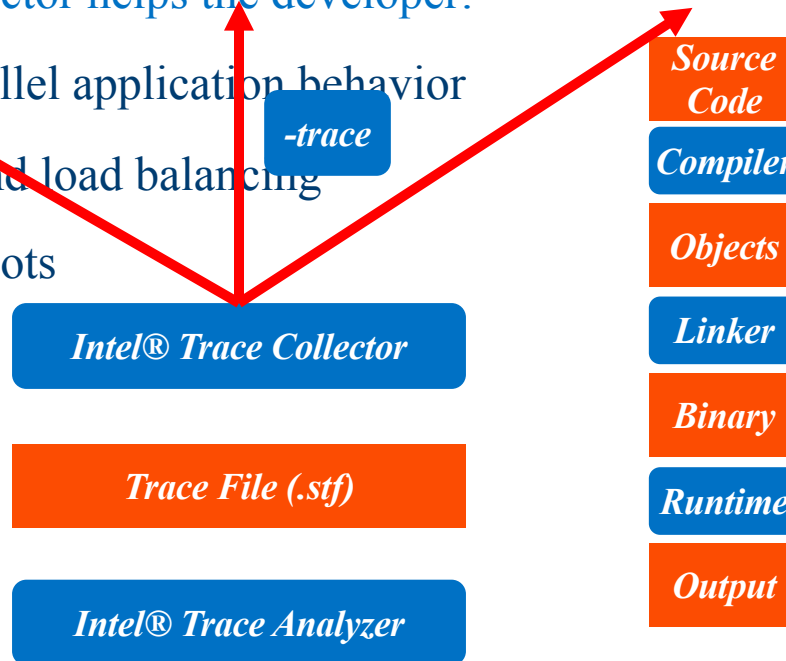
Intel® Trace Analyzer and Collector Overview

Intel® Trace Analyzer and Collector helps the developer:

- Visualize and understand parallel application behavior
- Evaluate profiling statistics and load balancing
- Identify communication spots

Features

- Event-based approach
- Low overhead
- Excellent scalability
- Powerful aggregation and filtering functions
- Performance Assistance and Imbalance Tuning
- NEW in 9.1: MPI Performance Snapshot



Views and Charts

Helps navigating through the trace data and keep orientation

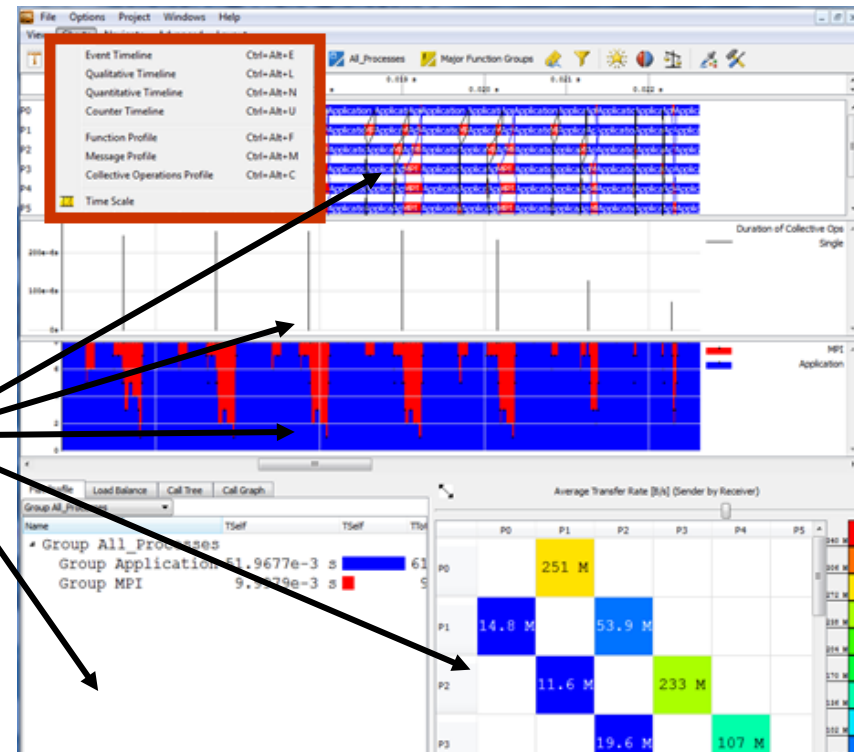
Every View can contain several Charts

All Charts in a View are linked to a single:

- time-span
- set of threads
- set of functions

All Charts follow changes to View (e.g. zooming)

Chart



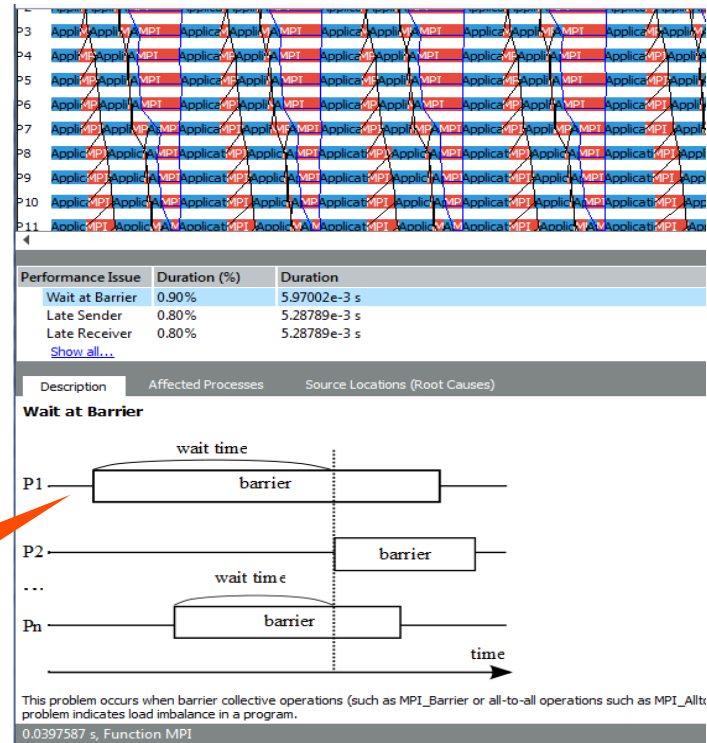
MPI Performance Assistant

Automatic Performance Assistant

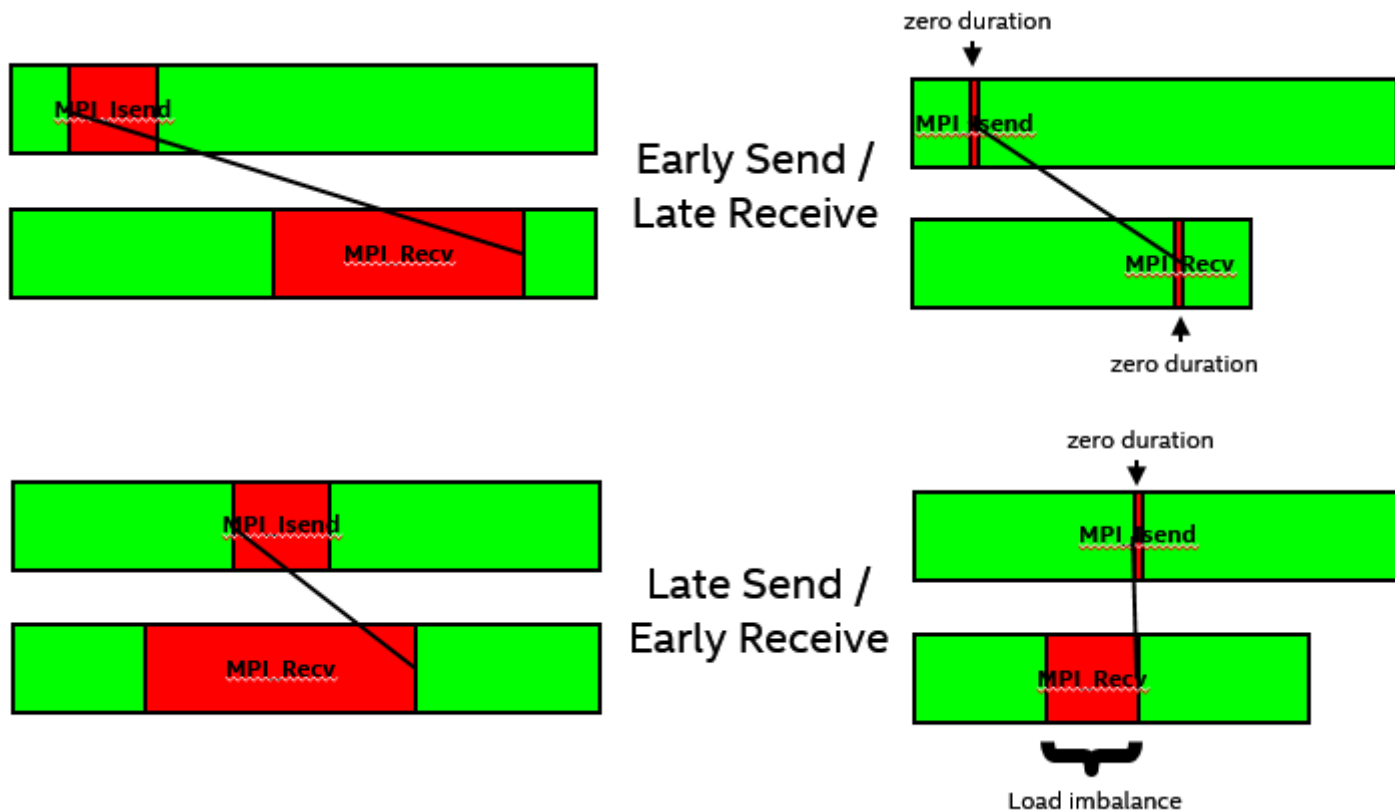
Detect common MPI performance issues

Automated tips on potential solutions

Automatically detect performance issues and their impact on runtime



MPI Idealization/Imbalance



Profiling Performance on Intel® Xeon Phi™ with Intel® VTune™ Amplifier XE

Intel® VTune™ Amplifier Tunes Knights Landing Processors

New!

Four Critical Optimizations for Intel® Xeon Phi™ Processors

1) High-bandwidth memory

- Decide which data structures to place in MCDRAM.
- See performance problems by memory hierarchy.
- Measure DRAM and MCDRAM bandwidth.



2) Scalability of MPI* and OpenMP*

- Serial versus parallel time
- Imbalance, overhead cost, parallel loop parameters

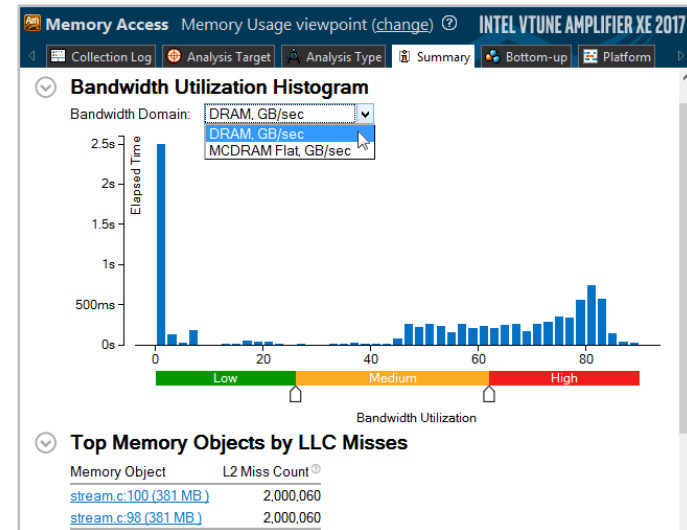
3) Micro architecture efficiency

- See the efficiency of your code in the core pipeline.

- Zero in on details with custom PMU events.

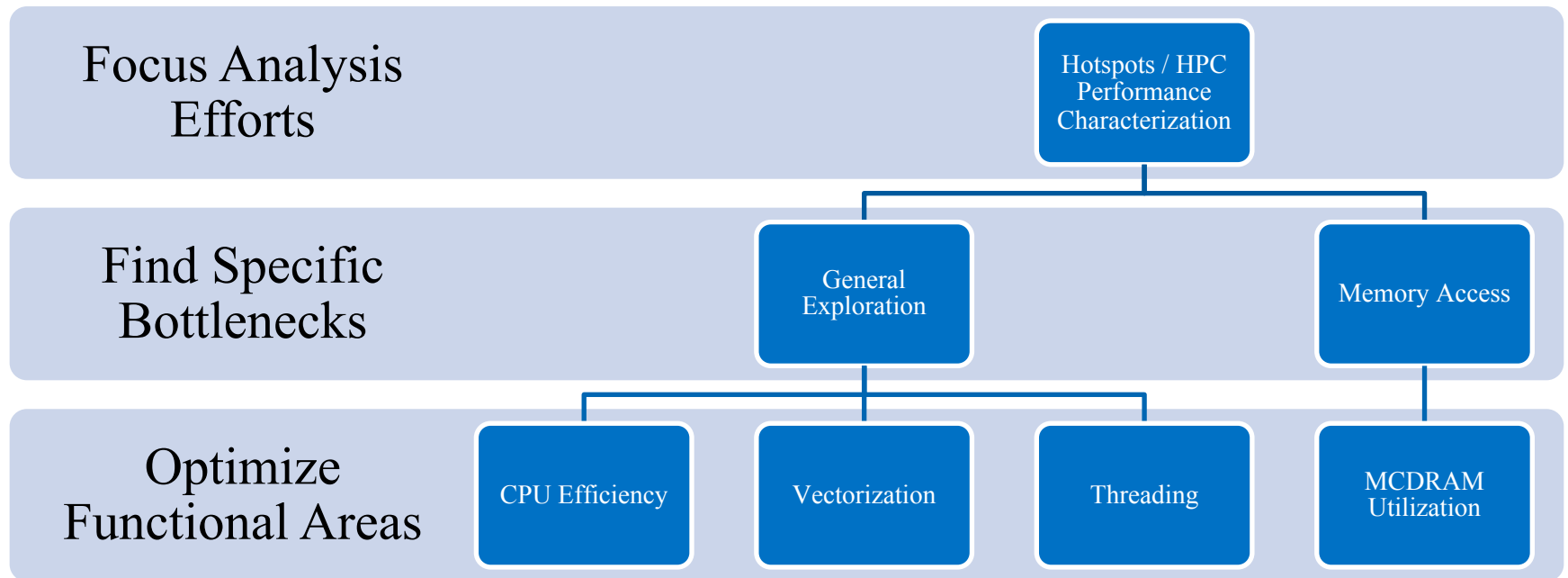
4) Vectorization efficiency: Use [Intel® Advisor](#)

- Optimize for Intel® AVX-512 with or without Intel® AVX-512 hardware.



KNL Tuning Methodology

Using Intel® VTune™ Amplifier XE



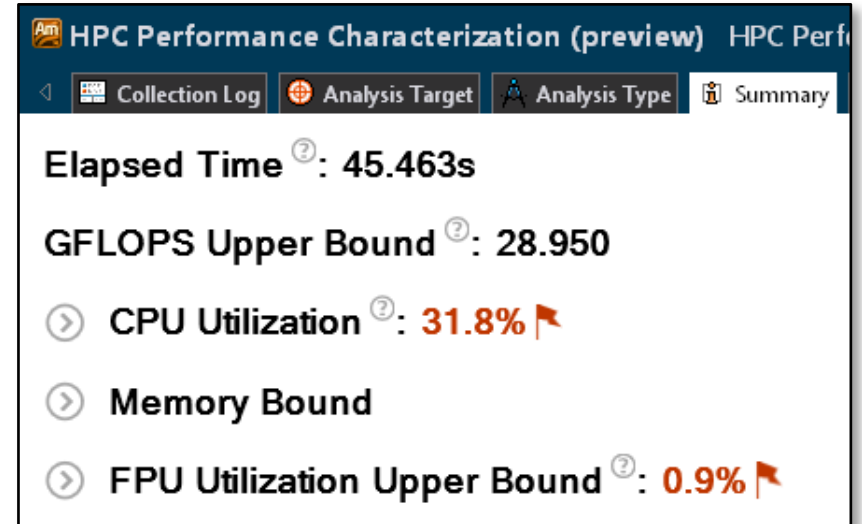
HPC Performance Characterization

Two characterization metrics

- Elapsed Time
- GFLOPs Upper Bound*

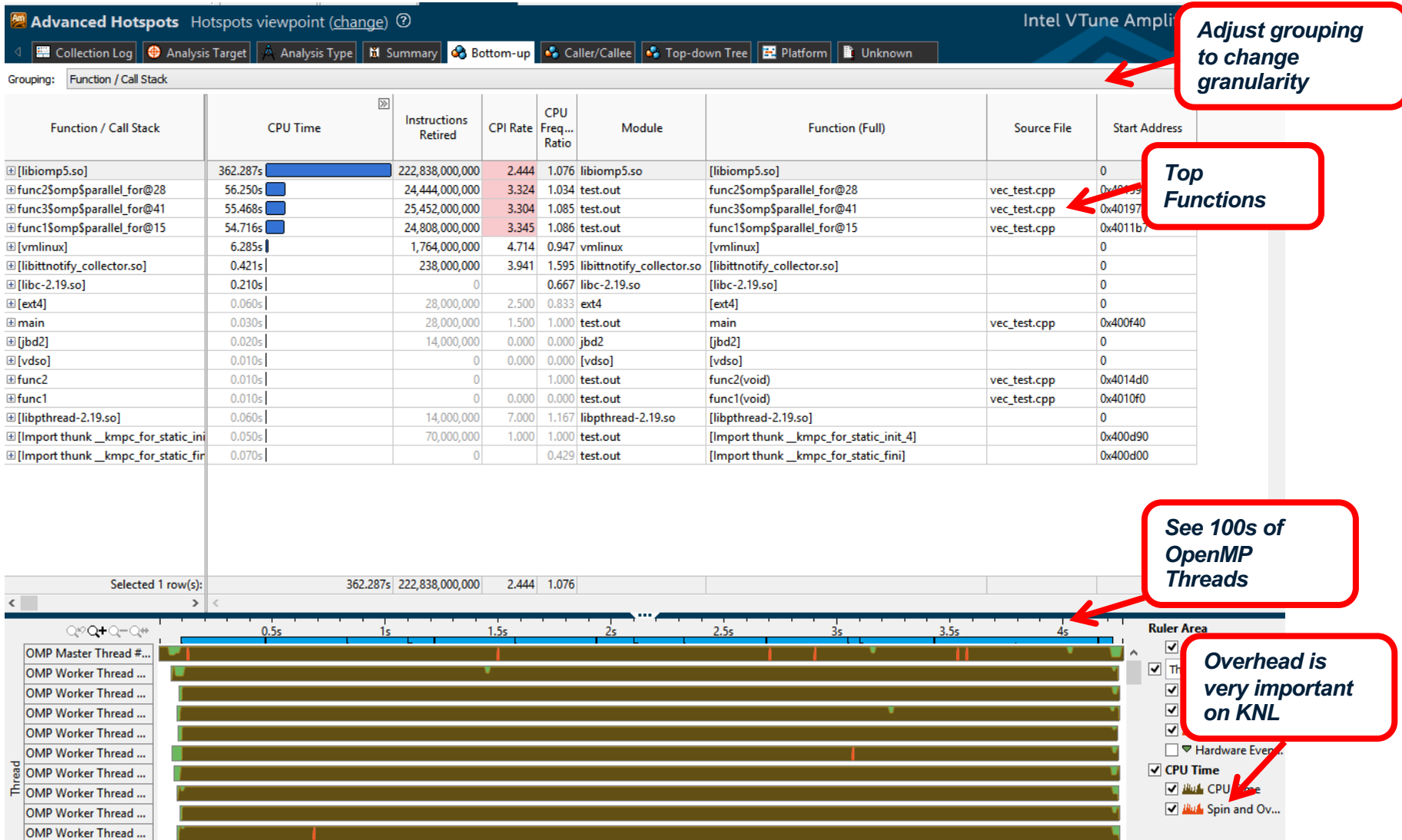
Three performance aspects

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



*Calculated based on FLOP HW counters assuming full vector utilization

Advanced Hotspots Analysis



Instruction 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:

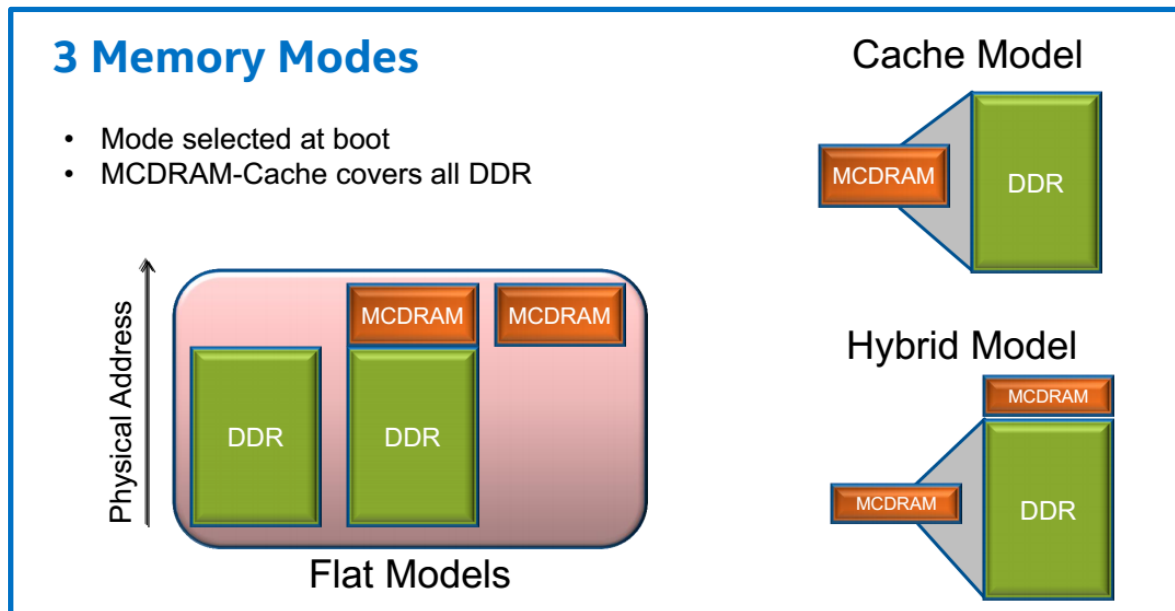


Vectorization Usage

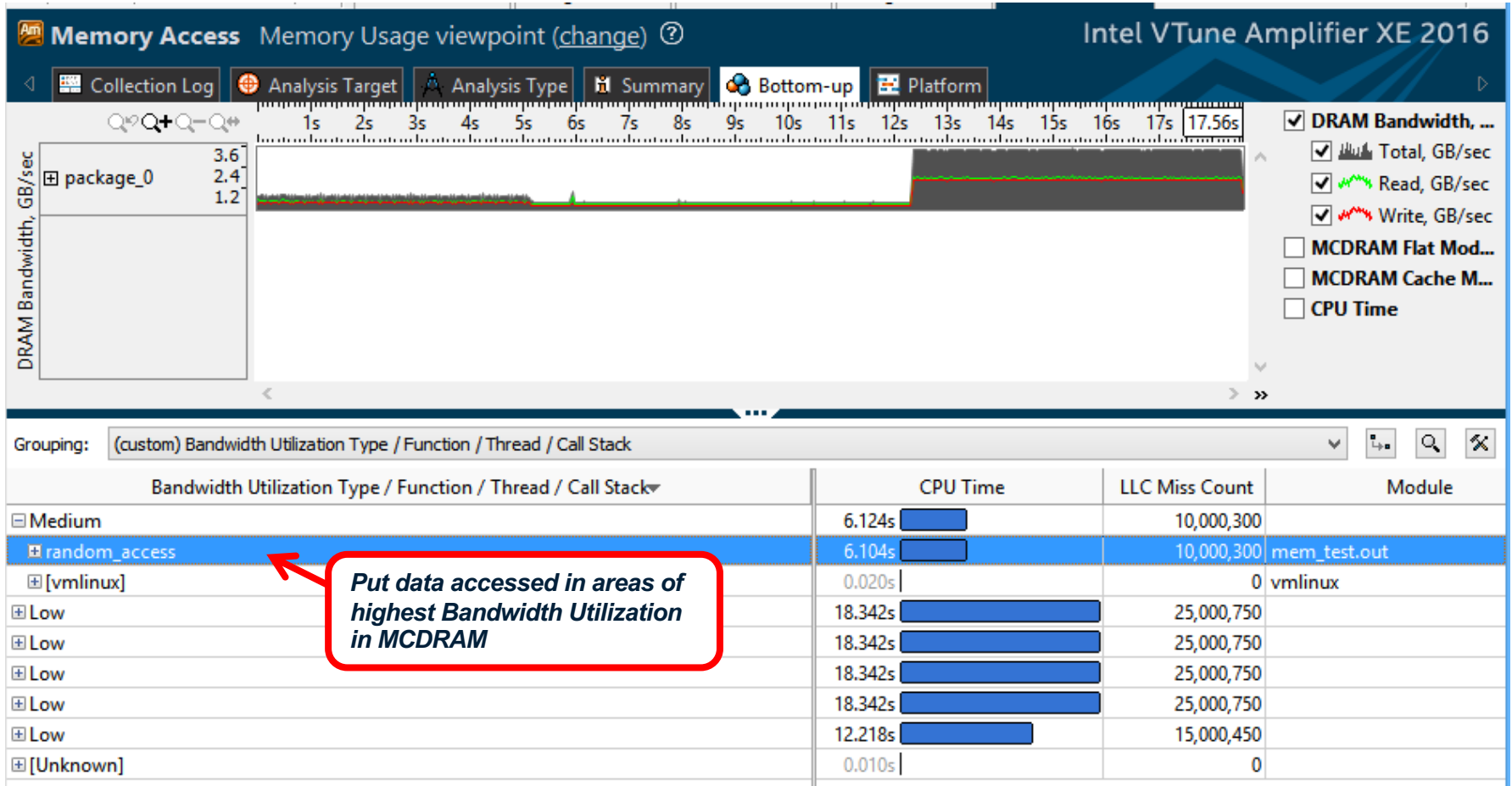
General Exploration General Exploration viewpoint (change) ?							
<a>Collection Log <a>Analysis Target <a>Analysis Type <a>Summary <a>Bottom-up <a>Event Count <a>Platform							
Grouping: Function / Call Stack							
Function / Call Stack	Clockticks ▼	Instructions Retired	CPI Rate	Retiring			
				VPU Utilization	Divi...	MS Assi...	FP Assists
+ __kmp_hyper_barrier_release	1,566,588,349,879	763,205,144,806	2.053	0.000	0.000	0.178	0.000
+ compute_rhs_\$omp\$parallel@17	1,515,622,273,430	96,228,144,342	15.750	1.000	0.000	0.105	0.000
+ y_solve_\$omp\$parallel_for@27	1,396,328,094,489	59,428,089,142	23.496	0.801	0.000	0.223	0.000
+ z_solve_\$omp\$parallel_for@31	1,379,010,068,512	64,234,096,351	21.469	0.814	0.000	0.159	0.000
+ x_solve_\$omp\$parallel_for@27	1,066,933,600,398	70,884,106,326	15.052	0.831	0.000	0.160	0.000
+ [vmlinux]	197,668,296,502	36,600,054,900	5.401	1.000	0.000	0.238	0.000
+ tzetar_\$omp\$parallel_for@22	165,968,248,952	5,790,008,685	28.665	1.000	0.000	0.200	0.000
+ txinvr_\$omp\$parallel_for@22	140,052,210,078	6,356,009,534	22.035	1.000	0.000	0.223	0.000
+ add_\$omp\$parallel_for@19	125,748,188,622	4,930,007,395	25.507	1.000	0.000	0.014	0.000
+ ninvr_\$omp\$parallel_for@20	81,542,122,313	3,352,005,028	24.326	1.000	0.000	0.344	0.000
+ pinvr_\$omp\$parallel_for@20	79,990,119,985	3,260,004,890	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	55,700,083,550	20,304,030,456	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
+ [libc-2.19.so]	8,500,012,750	1,692,002,538	5.024	0.000	0.000	0.518	0.000
+ lhsinit	7,398,011,097	48,000,072	154.125	1.000	0.000	1.000	0.000
+ lhsinitj	6,558,009,837	56,000,084	117.107	1.000	0.000	1.000	0.000
+ exact_rhs_\$omp\$parallel@20	3,654,005,481	1,064,001,596	3.434	0.861	0.000	0.158	0.000
+ initialize_\$omp\$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
+ __kmpc_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	4,000,006	303.500	0.000	0.000	0.000	0.000
+ [libitnotify_collector.so]	1,150,001,725	36,000,054	31.944	0.000	0.000	0.000	0.000
+ __kmpc_for_static_fini	746,001,119	0		0.000	0.000	0.000	0.000

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.*



Identifying Objects to put in MCDRAM



VTune Amplifier Tips

VTune Finalization:

- Finalization is very slow on KNL. Finalize on Xeon.
- Disable auto finalization with: `-finalize-mode=none`

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.

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.

Vectorization Optimization and Thread Prototyping with Intel® Advisor

Faster Code Faster with Data-Driven Design

Intel® Advisor: Vectorization Optimization and Thread Prototyping

Faster vectorization optimization

- Vectorize where it will pay off most.
- Quickly ID what is blocking vectorization.
- Tips for effective vectorization.
- Safely force compiler vectorization.
- Optimize memory stride.

Where should I add vectorization and/or threading parallelism? Intel Advisor XE 2016

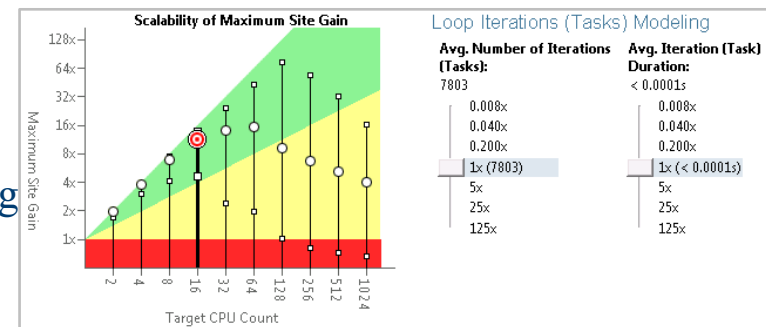
Summary Survey Report Refinement Reports Annotation Report Suitability Report

Elapsed time: 54.44s Vectorized Not Vectorized FILTER: All Modules All Sources

Function Call Sites and Loop	Vector Issues	Self Time	Total Time	Trip Counts	Loop Type	Why No Vectorization?	Vectorized Loops	Efficiency
[loop at stl_algo.h:4740 i...		0.170s	0.170s	1	Scalar	non-vectorizable l...		
[loop at loopstl.cpp:2449...	Ineffective peeled...	0.170s	0.170s	12; 4	Vectorized (B)	Collapse Collapse	AVX	100%
[loop at loopstl.cpp:2...		0.150s	0.150s	12	Vectorized (B)		AVX	
[loop at loopstl.cpp:2...		0.020s	0.020s	4	Remainder			
[loop at loopstl.cpp:7900...		0.170s	0.170s	500	Scalar	vectorization possi...		
[loop at loopstl.cpp:35...	High vector regi...	0.160s	0.160s	12	Expand	Expand	AVX	69%

Breakthrough for threading design

- Quickly prototype multiple options.
- Project scaling on larger systems.
- Find synchronization errors before implementing threading.
- Design without disrupting development.



Less Effort, Less Risk and More Impact

Part of Intel® Parallel Studio XE for Windows* and Linux*

<http://intel.ly/advisor-xe>

New for 2017: AVX-512, FLOPS, & More...

Intel® Advisor: Vectorization Optimization



New!

- Next-gen Intel® Xeon Phi™ support
- Tune for Intel® AVX-512 with or without Intel AVX-512 hardware
- Precise FLOPS calculation
- Enhanced memory access analysis
- Easier selection of high-impact loops
- Batch mode workflow saves time
- Fast answers with loop analytics

<div> <div>Elapsed time: 39.44s</div> <div>Vectorized</div> <div>Not Vectorized</div> <div>FILTER: All Modules All Sources Loops All Threads</div> <div>OFF Smart Mode</div> </div> <div> <div>Summary</div> <div>Survey Report</div> <div>Refinement Reports</div> </div> <div>INTEL ADVISOR 2017</div>													
Function Call Sites and Loops	Vector Issues	Self Time	Total Time	Type	FLOPS		Why No Vectorization?	Vectorized Loops				Trip Counts	Instr Trait
					GFLOPS	AI		Vect...	Efficiency	Gain...	VL (...)		
[loop in S252 at loops90.f:1172]	1 Possib...	3.080s 8.5%	3.080s	Vectorized Ve...	0.191	0.10...	1 vectori...	AVX2	17%	1.38x	8	999; 62; 1	Divis
[loop in S252 at loops90.f:1172]	1 Possib...	2.970s 8.2%	2.970s	Scalar	0.067	0.0833	vectorizat...					999	Divis
[loop in S252 at loops90.f:1172]	1 Possib...	0.090s	0.090s	Vectorized (Bo...	4.333	0.1250		AVX2			8	1; 62	FMA
[loop in S252 at loops90.f:1172]	1 Possib...	0.020s	0.020s	Scalar			vectorizat...					1	Divis
[loop in S2101 at loops90.f:1749]	2 Possib...	2.580s 7.1%	2.580s	Scalar	0.152	0.0625	vectorizat...						FMA
[loop in S126 at loops90.f:447]	1 Assu...	1.068s	1.068s	Scalar	0.370	0.1667	vector de...						
[loop in S343 at loops90.f:2300]	1 Assu...	1.020s	1.020s	Scalar			vector de...						
[loop in S353 at loops90.f:2381]	1 Possib...	0.880s	0.880s	Vectorized (Bo...	2.274	0.1250		AVX2	35%	2.78x	8	62; 4; 1	FMA

Boost Vectorization with Intel® Advisor

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

1. Compiler diagnostics + Performance Data + SIMD efficiency information

Function Call Sites and Loops	Self Time	Total Time	Compiler Vectorization
[loop in runCForallLambdaLoops]	0.084s	0.084s	Scalar
[loop in runCForallLambdaLoops]			Scalar
[loop in std::Complex_base<double,struct...>]			Scalar
[loop in std::basic_string<char,struct std::char_traits<char>,std::allocator<char>>]			Scalar
[loop in std::basic_string<char,struct std::char_traits<char>,std::allocator<char>>]			Scalar
[loop in std::num_put<char,class std::ostreambuf_iterator<char>>]			Scalar

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

Issue: Peeled/Remainder loop(s) present

All or some source loop iterations are not executing source loop iterations from peeled/remainder loops. Utilizing Full Vectors...

Recommendation: Align memory access

Projected maximum performance gain: High
Projection confidence: Medium

The compiler created a peeled loop because one start at a data boundary. Align the memory access. This example aligns memory using a 32-byte boundary.

```
float *array;  
array = (float *)_mm_malloc(ARRAY_SIZE*  
// Somewhere else  
_assume_aligned(array, 32);  
// Use array in loop
```

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

Total Time	Trip Counts	Median	Min
3,151s	1	1	1
0,440s	1	1	1
0,010s	1	1	1
0,010s	2	1	1
0,010s	3	1	1

4. Loop-Carried Dependency Analysis

ID	Type	Site Name
P1	Parallel site information	site2
P2	Read after write dependency	site2
P3	Read after write dependency	site2
P4	Write after write dependency	site2
P5	Write after write dependency	site2
P6	Write after read dependency	site2
P7	Write after read dependency	site2

5. Memory Access Patterns Analysis

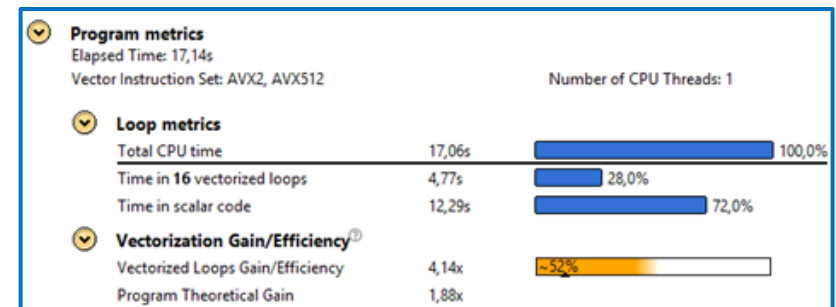
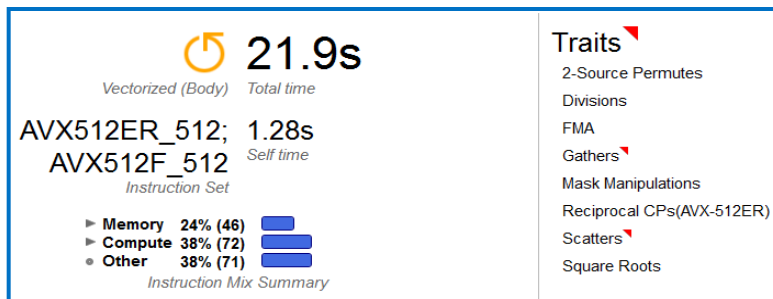
Site Name	Site Function	Site Info	Loop-Carried Dependencies	Strides Distribution	Access Pattern
loop_site_203	runCrawLoops	runCrawLoops.coc1063	RAW:1	No information available	No information available
loop_site_139	runCrawLoops	runCrawLoops.coc622	No information available	39% / 36% / 25%	Mixed strides
loop_site_160	runCrawLoops	runCrawLoops.coc925	No information available	100% / 0% / 0%	All unit strides

ID	Stride	Type	Source	Modules	Alignment
P22	0; 0; 1	Unit stride	runCrawLoops.coc637	lcal.exe	
P23	0; 0	Unit stride	runCrawLoops.coc638	lcal.exe	
P30	-1575; -63; -26; -25; -1; 0; 1; 25; 26; 63; 2164801	Variable stride	runCrawLoops.coc628	lcal.exe	

```
635 j2 = ( j2 & 64-1 ) ;  
636 p[ip][0] += y[12+32];  
637 p[ip][1] += x[12+32];  
638 i2 += e[12+32];  
639 j2 += f[j2+32];  
626 i1 &= 64-1;  
627 j1 &= 64-1;  
628 p[ip][2] += b[j1][11];
```

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

Start Tuning for AVX-512 without AVX-512 hardware

Intel® Advisor - Vectorization Advisor

Use `-axCOMMON-AVX512 -xAVX` compiler flags to generate both code-paths

- AVX(2) code path (executed on Haswell and earlier processors)
- AVX-512 code path for newer hardware

Compare AVX and AVX-512 code with Intel Advisor

Loops		Self Time	Loop Type	Vectorized Loops					Instruction Set Analysis				Advanced
				Vect...	Efficiency	Gain...	VL (...)	Compiler Es...	Traits	Data T...	Vector W...	Instruction Sets	
[-] [loop in s352_at loopstl.cpp:5939]	<input type="checkbox"/>	0,641s	Vectorized (Body)	AVX2	~54%	2,15x	4	2,15x	FMA; Inserts	Float32	128	AVX; FMA	
[-] [loop in s352_at loopstl.cpp:5939]	<input type="checkbox"/>	n/a	Remainder [Not Executed]				4		FMA				
[-] [loop in s352_at loopstl.cpp:5939]	<input type="checkbox"/>	0,641s	Vectorized (Body)	AVX2			4	2,15x	Inserts; FMA				
[-] [loop in s352_at loopstl.cpp:5939]	<input type="checkbox"/>	n/a	Vectorized (Body) [Not Executed]	AVX512			16	3,20x	Gathers; FMA				
[-] [loop in s352_at loopstl.cpp:5939]	<input type="checkbox"/>	n/a	Vectorized (Remainder) [Not Executed]	AVX512			16	2,70x	Gathers; FMA				
[-] [loop in s125_A\$omp\$parallel_for@...]	<input type="checkbox"/>	0,496s	Vectorized Versions	AVX2	~100%	13,54x	8	<13,54x	FMA; NT-stores				
[-] [loop in s125_A\$omp\$parallel_for@...]	<input type="checkbox"/>	n/a	Peeled [Not Executed]				8		FMA				
[-] [loop in s125_A\$omp\$parallel_for@...]	<input type="checkbox"/>	n/a	Remainder [Not Executed]				8		FMA				
[-] [loop in s125_A\$omp\$parallel_for@...]	<input type="checkbox"/>	0,465s	Vectorized (Body)	AVX2			8	13,54x					
[-] [loop in s125_Z\$omp\$parallel_for@...]	<input type="checkbox"/>	n/a	Vectorized (Peeled) [Not Executed]	AVX512			16	6,77x	FMA				
[-] [loop in s125_Z\$omp\$parallel_for@...]	<input type="checkbox"/>	n/a	Vectorized (Body) [Not Executed]	AVX512			32	30,61x	NT-stores				
[-] [loop in s125_Z\$omp\$parallel_for@...]	<input type="checkbox"/>	n/a	Vectorized (Remainder) [Not Executed]	AVX512			16	9,78x	FMA				

*Inserts (AVX2) vs.
Gathers (AVX-512)*

*Speed-up estimate: 13.5x
(AVX2) vs.
30.6x (AVX-512)*

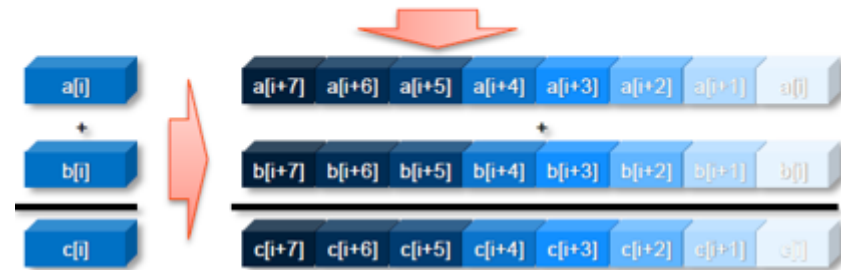
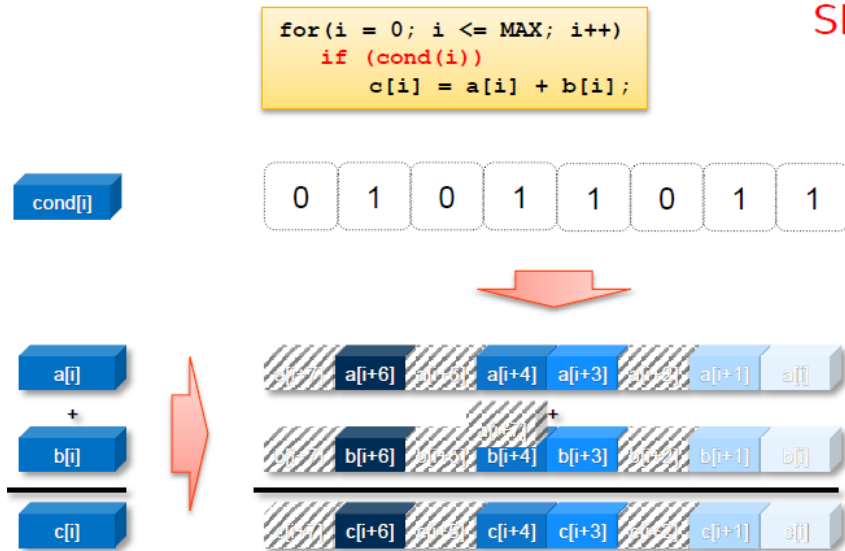
Why is Mask Utilization important?

3 elements suppressed

SIMD Utilization = $5/8$
(62.5%)

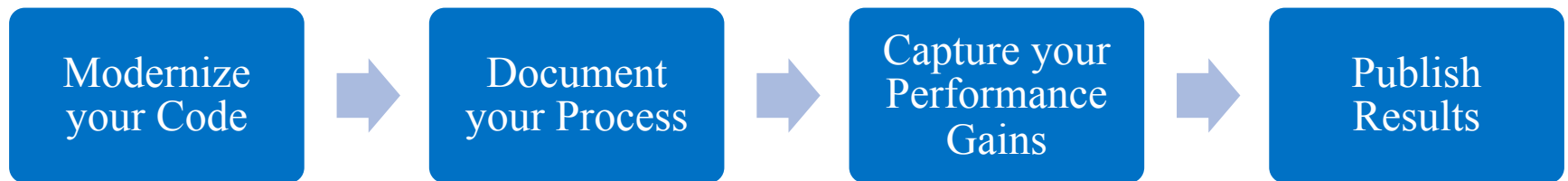
*Not utilizing full
vectors!!*

Fully utilized!



My Challenge to You

Challenge



Additional Resources

Educating with Webinar Series about 2017 Tools

Expert talks about the new features

Series of live webinars, September 13 – November 8, 2016

Attend live or watch after the fact.

<https://software.intel.com/events/hpc-webinars>

What's New In Intel® Parallel Studio XE 2017 (Online)

Sep 13, 2016 (9:00am - 10:00am PST)

This webinar will go over the latest features of the new release of Intel® Parallel Studio XE 2017.



Parallel Programming and Optimization for Intel® Architecture (Workshop)

São Paulo, Brazil

Sep 21, 2016 (11:00am - 5:00pm EBT)

Offered by UNESP in partnership with Intel software Brazil, aims to a practical approach to parallel programming on Intel® Xeon® and Intel® Xeon Phi™ based systems



Code for Speed with High Bandwidth Memory on Intel® Xeon Phi™ Processors (Online)

Oct 11, 2016 (9:00am - 10:00am PST)

Cover methods for users to analyze suitable memory mode and "memkind" library interface, a user-extensible heap manager built on top of jemalloc.



Vectorization, the "Other" Parallelism You Need (Online)

Oct 18, 2016 (9:00am - 10:00am PST)

This session demonstrates how process of identifying and modifying code to take advantage of the vector hardware will boost application performance.



Roofline analysis: A new way to visualize performance optimization tradeoffs (Online)

Oct 25, 2016 (9:00am - 10:00am PST)

Join us in the webinar to see a demonstration and an introduction to how to use roofline analysis to make your own code more efficient.



Educating with High-Performance Programming Book

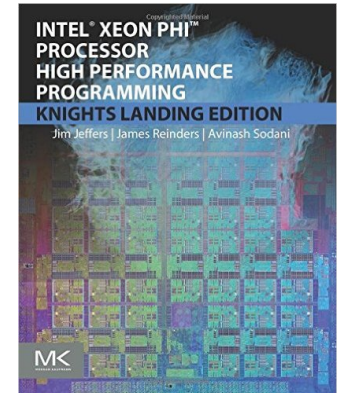
Knights-Landing-specific details, programming advice, and real-world examples.

Intel® Xeon Phi™ Processor High Performance Programming

- Techniques to generally increase program performance on any system and prepare you better for Intel Xeon Phi processors.

Available as of June 2016

<http://lotsofcores.com>



” I believe you will find this book is an invaluable reference to help develop your own Unfair Advantage.”

**James A.
Manager
Sandia National Laboratories**

More Education with software.intel.com/moderncode

- Online community growing collection of tools, trainings, support
 - Features Black Belts in parallelism from Intel and the industry
- Intel® HPC Developer Conferences developers share proven techniques and best practices
 - hpcdevcon.intel.com
- Hands-on training for developers and partners with remote access to Intel® Xeon® processor and Xeon Phi™ coprocessor-based clusters.
 - software.intel.com/icmp
- Developer Access Program provides early access to Intel® Xeon Phi™ processor codenamed Knights Landing plus one-year license for Intel® Parallel Studio XE Cluster Edition.
 - <http://dap.xeonphi.com/>



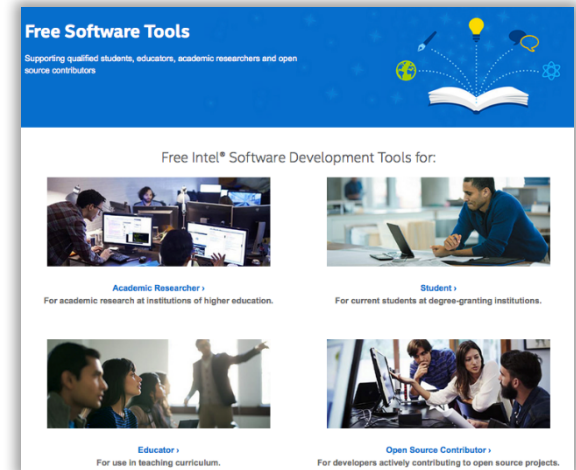
Choices to Fit Needs: Intel® Tools

All Products with support – worldwide, for purchase.

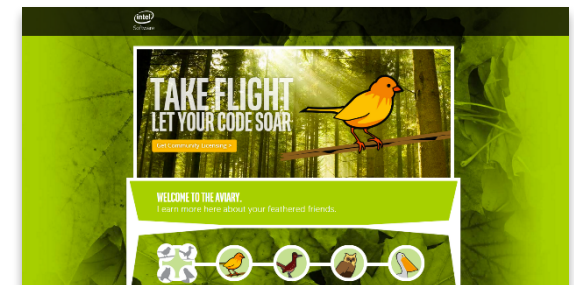
- Intel® Premier Support - private direct support from Intel
- support for past versions
- software.intel.com/products

Most Products without Premier support – via special programs for those who qualify

- students, educators, classroom use, open source developers, and academic researchers
- software.intel.com/qualify-for-free-software
- Intel® Performance Libraries without Premier support -Community licensing for Intel performance libraries
 - no royalties, no restrictions based on company or project size
 - software.intel.com/nest



**Community support only – all tools:
Students, Educators, classroom use,
Open Source Developers,
Academic Researchers (qualification required)**



**Community support only – Intel Performance Libraries:
Community Licensing (no qualification required)**

Tuning Guides and Performance Analysis Papers

Tuning Guides specific to processor families available

- <https://software.intel.com/en-us/articles/processor-specific-performance-analysis-papers>
- Also has general tuning papers and issue-specific tuning information

Intel® VTune™ Amplifier XE Tuning Guides

Our tuning guides explain how to identify common software performance issues using VTune Amplifier XE and give suggestions for optimization.

Tuning Guide for Intel® Microarchitecture	Processors Covered
code name Knights Landing	Intel® Xeon Phi™ Processor
code name Broadwell-E* (server)	Intel® Xeon Processor E5 v4 Family
code name Skylake	6th Generation Intel® Core™ Processor Family
code name Broadwell	5th Generation Intel® Core™ Processor Family
code name Haswell-E* (server)	Intel® Xeon® Processor E5 v3 Family
code name Ivy Bridge-E* (server)	Intel® Xeon® Processor E5/E7 v2 Family
code name Haswell	4th Generation Intel® Core™ Processor Family
Many Integrated Core Architecture	Intel® Xeon Phi™ coprocessor
code name Sandy Bridge-EP/EX/EN (server)	Intel® Xeon® Processor E5 Family
code name Ivy Bridge	3rd Generation Intel® Core™ Processor Families
code name Sandy Bridge	2nd Generation Intel® Core™ Processor Families

Acknowledgements

Argonne National Lab

Intel contributors

IPCCs

Thank you for Attending!



Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Copyright © 2016, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

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

Backup and Reference Information

Intel® Compilers

Intel® Compilers for Intel®Parallel Studio XE 2017

Intel® C++ 17.0 and Intel® Fortran 17.0

Common Updates

- Enhanced support for the latest Intel® AVX2 and AVX512 instruction sets for the latest Intel® processors (including Intel® Xeon Phi™ processor)
- Enhanced optimization/vectorization reports make Intel compilers indispensable towards modernizing your code
- Support for OpenMP* 4.5, offering improved vectorization control, and new SIMD instructions

Intel® C++ Compiler

- *SIMD Data Layout Template to facilitate vectorization for your C++ code*
- *Virtual function vectorization capability*
- *Full support for the latest C11 and C++14 standards; initial C++17 support*

Intel® Fortran Compiler

- *Substantial coarray performance improvement – up to twice as fast as previous versions on non-trivial coarray Fortran programs*
- *Almost complete Fortran 2008 support*
- *Further interoperability with C (part of draft Fortran 2015)*

Compiling for Intel® AVX-512

The Intel® Xeon Phi™ processor codenamed Knights Landing (KNL) supports the Intel® AVX-512 instruction set

- Slightly different from next generation Intel® Xeon® processors; incompatible with Intel® Xeon Phi™ x100 coprocessors

Also supports Intel® SSE, AVX, AVX2 instruction sets

- Apps built for Intel Xeon processors codenamed Broadwell, Haswell and earlier can run on KNL

-xmic-avx512 KNL only

-xcore-avx512 next generation Xeon only

-xcommon-avx512 Intel AVX-512 subset common to KNL and next
generation Xeon for binaries that will run on both

-m, -march, /arch not yet

-ax...avx512 same options supported as for -x

No more -mmic !

Intel® Distribution for Python*

Highlights: Intel® Distribution for Python* 2017

Focus on Advancing Python Performance Closer to Native Speeds

Easy, out-of-the-box access to high-performance Python

- Prebuilt, accelerated Distribution for numerical and scientific computing, data analytics, HPC. Optimized for Intel® architecture.
- Drop-in replacement for your existing Python. No code changes required.

Drive performance with multiple optimization techniques

- Accelerated NumPy/SciPy/scikit-learn with Intel® Math Kernel Library.
- Data analytics with pyDAAL, enhanced thread scheduling with TBB, Jupyter* notebook interface, Numba, Cython.
- Scale easily with optimized mpi4py and Jupyter notebooks.

Faster access to latest optimizations for Intel® architecture

- Distribution and individual optimized packages available through conda and Anaconda Cloud.
- Optimizations upstreamed back to main Python* trunk.

A Two-Pronged Approach for Faster Python* Performance

Accelerated Python Distribution plus Performance Profiling

*Step 1: Use Intel® Distribution for Python**

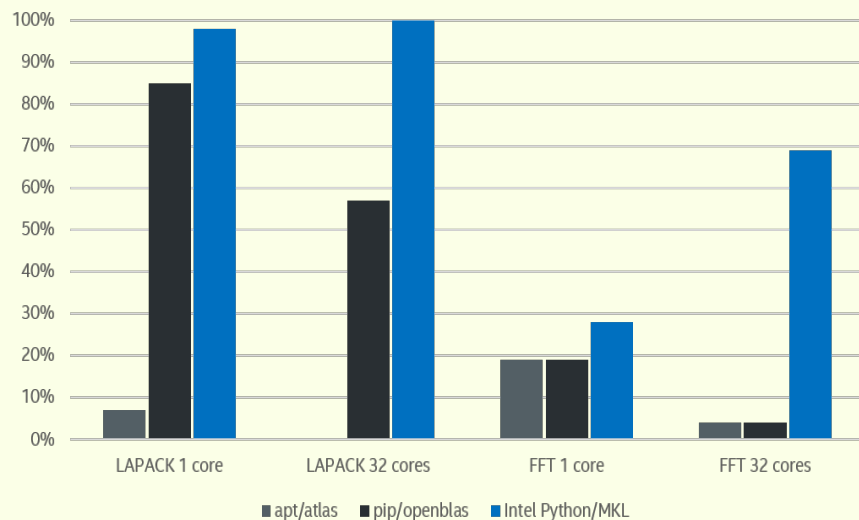
- Leverage optimized native libraries for performance.
- Drop-in replacement for your current Python*.
- Latest optimizations for Intel® processors and compilers.

Step 2: Use Intel® VTune™ Amplifier for Profiling

- Get detailed summary of entire application execution profile.
- Auto-detects and profiles Python/C/C++ mixed code and extensions.
- Accurately detect hotspots. Line-level analysis helps make smart optimization decisions fast.
- Available in Intel® Parallel Studio XE 2017 suite.

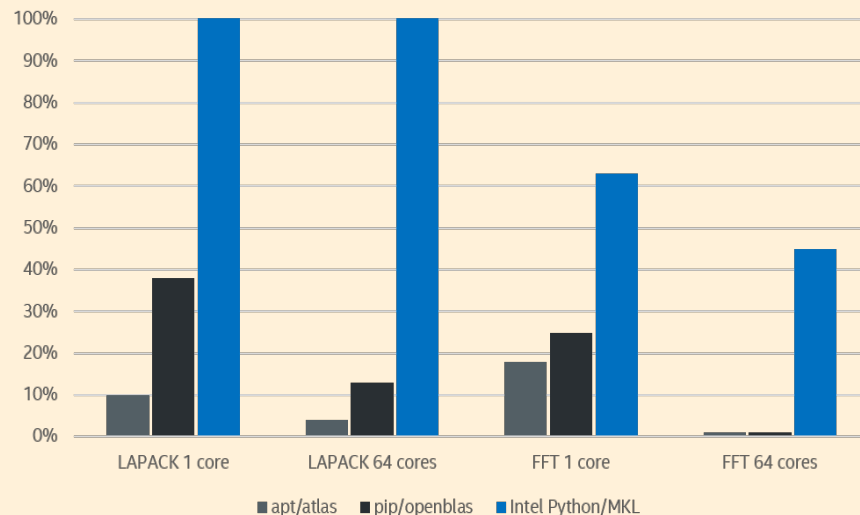
Near-Native Performance Speedups on IA

Python* Performance as a Percentage of C/Intel® MKL for Intel® Xeon® Processors (Higher is Better)



Intel® Xeon® Processor

Python* Performance as a Percentage of C/Intel® MKL for Intel® Xeon Phi™ Product Family (Higher is Better)



Intel® Xeon Phi™ Product Family

Configuration Info: apt/atlas: installed with apt-get, Ubuntu 16.10, python 3.5.2, numpy 1.11.0, scipy 0.17.0; pip/openblas: installed with pip, Ubuntu 16.10, python 3.5.2, numpy 1.11.1, scipy 0.18.0; Intel Python: Intel Distribution for Python 2017;. Hardware: Xeon: Intel Xeon CPU E5-2698 v3 @ 2.30 GHz (2 sockets, 16 cores each, HT=off), 64 GB of RAM, 8 DIMMS of 8GB@2133MHz; Xeon Phi: Intel Intel® Xeon Phi™ CPU 7210 1.30 GHz, 96 GB of RAM, 6 DIMMS of 16GB@1200MHz

Problem Size and Configuration Information

Intel® Distribution for Python* Benchmarks

Hardware/Problem Size	dot	lu	det	inv	cholesky	fft
Intel® Xeon® processor (32 core) and Intel® Xeon Phi™ processor (64 core)	(20k, 10k) and (10k, 20k)	(35k, 35k)	(15k, 15k)	(25k, 25k)	(40k, 40k)	520k
Intel Xeon processor (1 core)	(20k, 5k) and (5,20k)	(20k, 20k)		(10k, 10k)	(10k, 10k)	
Intel Xeon Phi processor (1 core)	(20k, 300) and (300, 20k)	(6k, 6k)	(4k, 4k)	(2k, 2k)		

Configuration Info: apt/atlas: installed with apt-get, Ubuntu* 16.10, Python* 3.5.2, numpy* 1.11.0, scipy* 0.17.0; pip*/openblas*: installed with pip, Ubuntu 16.10, python 3.5.2, numpy 1.11.1, scipy 0.18.0; Intel Python: Intel® Distribution for Python 2017;. Hardware: Intel Xeon processor: Intel Xeon processor E5-2698 v3 @ 2.30 GHz (2 sockets, 16 cores each, HT=off), 64 GB of RAM, 8 DIMMS of 8GB@2133MHz; Intel Xeon Phi processor: Intel Intel® Xeon Phi™ processor 7210 1.30 GHz, 96 GB of RAM, 6 DIMMS of 16GB@1200MHz

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. * Other brands and names are the property of their respective owners. Benchmark source: Intel Corporation.

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804

Intel® MPI Library

Intel® MPI Library Overview

Optimized MPI application performance

- Application-specific tuning
- Automatic tuning
- New! - Support for Intel® Xeon Phi™ processor (code-named Knights Landing)
- New: Support for Intel® Omni-Path Architecture Fabric

Lower-latency and multi-vendor interoperability

- Industry leading latency
- Performance optimized support for the fabric capabilities through OpenFabrics*(OFI)

Faster MPI communication

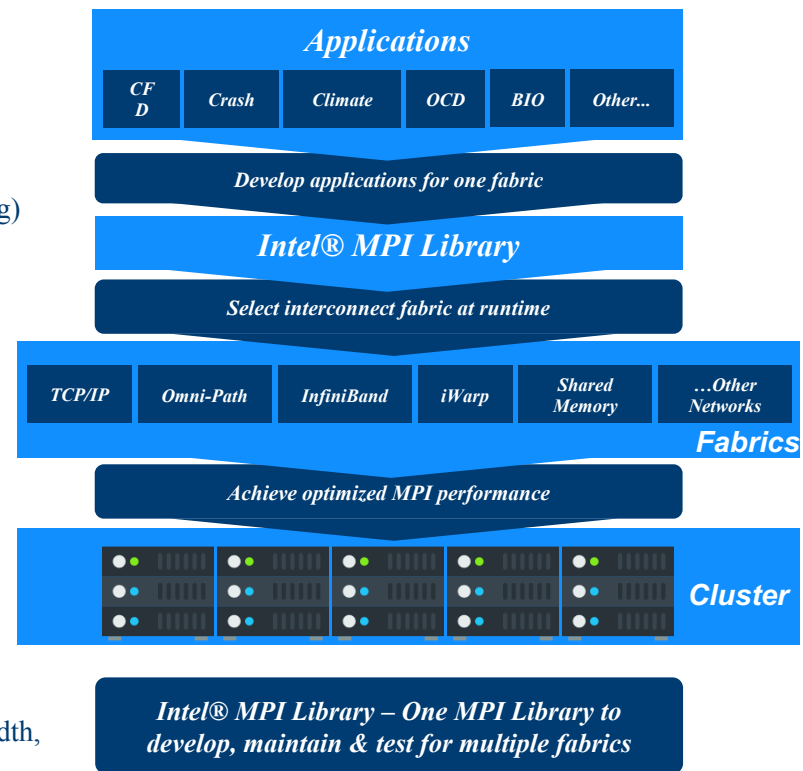
- Optimized collectives

Sustainable scalability up to 340K cores

- Native InfiniBand* interface support allows for lower latencies, higher bandwidth, and reduced memory requirements\

More robust MPI applications

- Seamless interoperability with Intel® Trace Analyzer and Collector



What's New: Intel® MPI Library 2017

- Support for Intel® Xeon Phi™ processors (code-named Knights Landing (KNL))
- Support for Intel® Omni-Path Architecture fabric
- Usage of specially optimized memcpy for KNL
- Tuning of shared memory collectives on single KNL nodes
- General optimization of RMA
- General optimization and speed up startup time and MPI tune utility

HBM awareness

MPI_Win_allocate_shared()/MPI_Win_allocate() MPI_Info key numa_bind_policy

I_MPI_HBW_POLICY environment variable

- Can control user and MPI buffers as well
- Only hbw_* options

I_MPI_BIND_WIN_ALLOCATE environment variable

Value	Effect
localalloc	Allocate memory on local node
hbw_preferred	Allocate in HBW if available, fallback to DRAM
hbw_bind	Allocate only in HBW
hbw_interleave	Allocate on HBW and DRAM in round-robin
<NUMA node id>	Allocate on specified NUMA node

Intel® VTune™ Amplifier

Faster, Scalable Code—Faster

Get the data you need

- Hotspot (Statistical call tree), call counts (statistical)
- Thread profiling – concurrency and locks and waits analysis
- Cache miss, bandwidth analysis...¹
- GPU offload and OpenCL* kernel tracing

Find answers fast

- View results on the source/assembly
- OpenMP* scalability analysis, graphical frame analysis
- Filter out extraneous data – organize data with viewpoints
- Visualize thread and task activity on the timeline

Easy to use

- No special compiles – C, C++, C#, Fortran*, Java*, ASM*
- Visual Studio* integration or stand-alone
- Graphical interface and command line
- Local and remote data collection
- Analyze Windows* and Linux* data on macOS*²

¹ Events vary by processor. ² No data collection on macOS*

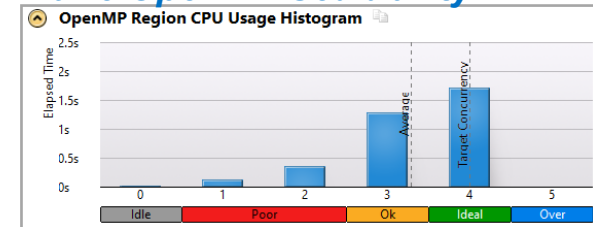
Quickly Find Tuning Opportunities

Function / Call Stack	CPU Time					Spin Time	Overhead Time
	Effective Time by Utilization						
	Idle	Poor	Ok	Ideal	Over		
FireObject:checkCollision	4.507s					0s	0s
FireObject:ProcessFireCollisionsRange	3.444s					0s	0s
NtWaitForSingleObject	0s					3.406s	0s
std::basic_ifstream<char,struct std::char_traits	3.359s					0s	0s
Ogre:FileSystemArchive::open	3.359s					0s	0s
CBaseDevice::Present	2.335s					0.671s	0s
Selected 1 row(s):						1.151s	0.728s

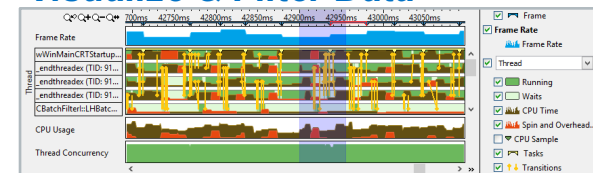
See Results On The Source Code

Source		Assembly		Assembly grouping: Address		
Source Line	Source	CPU Time: Total by Utilization				
		Idle	Poor	Ok	Ideal	Over
81	for (int i = 0; i < mem_array_i_max; i++)	0.300s				
82	{					
83	for (int j = 0; j < mem_array_j_max; j++)	4.936s				
84	{					
85	mem_array[j*mem_array_j_max+i] = *fill_val	7.207s				

Tune OpenMP Scalability



Visualize & Filter Data



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”)

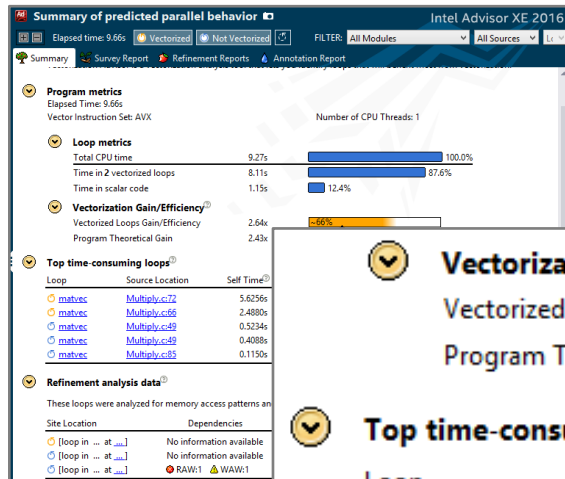
The “Software on Hardware” Tuning Process

For each Hotspot

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

Repeat

Summary View: Plan Your Next Steps



What can I expect to gain?

Vectorization Gain/Efficiency^②

Vectorized Loops Gain/Efficiency 2.64x

Program Theoretical Gain 2.43x

~66%

Top time-consuming loops^②

Loop	Source Location	Self Time ^②	Total Time ^②
matvec	Multiply.c:72	5.6256s	5.6256s
matvec	Multiply.c:66	2.4880s	2.4880s
matvec	Multiply.c:49	0.5234s	6.1490s
matvec	Multiply.c:49	0.4088s	2.8968s
matvec	Multiply.c:85	0.1150s	0.1150s

Where do I start?

Vector Efficiency: All The Data In One Place

My “performance thermometer”

Vector Issues	Self Time▼	Total Time	Type	Why No Vectorization?	Vectorized Loops				Trip Counts	Instruction Set Analysis		
					Vect...	Efficiency	Gain...	VL (...)		Traits	Data T...	Num.
1 Inefficient ...	5.281s	5.281s	Vectorized (Bo...		AVX	~55%	2.19x	4	25; 2	Inserts	Float64	3; 7
1 Ineffective ...	2.828s	2.828s	Vectorized (Bo...		AVX	~91%	3.65x	4	25; 2		Float64	3

~91%

~91%: Achieved Vectorization Efficiency

Achieved Vectorization Efficiency = (Estimated Gain/Vector Length) * 100%

Estimated Gain = 3.65x

Vector Length = 4

Orange color = Achieved vectorization efficiency is higher than reference efficiency for original scalar loop

⚠ Efficiency is approximately 91%, which means actual efficiency may be lower

▲ (25%): Reference Efficiency for original scalar loop

Reference Efficiency = (1x/Vector Length) * 100%

□ (100%): Theoretical Maximum Vectorization Efficiency

Maximum Vectorization Efficiency = (Theoretical Maximum Gain/Vector Length) * 100%

Theoretical Maximum Gain = Currently selected Vector Length = 4

Look at Vector Issues and Traits to find out why

- All kinds of “memory manipulations”
- Usually an indication of “bad” access pattern

Check if It Is It Safe to Vectorize

Loop-Carried Dependencies Analysis Verifies Correctness

Intel Advisor XE 2016

Where should I add vectorization and/or threading parallelism?

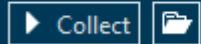
Summary Survey Report Refinement Reports Annotation Report Suitability Report

Program time: 12.82s Vectorized Not Vectorized FILTER: All Modules All Sources

Function Call Sites and Loops	Self Time	Total Time			Trip Counts	Compiler Vectorization	
						Loop Type	Why No Vectorization?
z> [loop at Multiply.c:53 in matvec]	0.047s	0.047s			3	Vectorized (Body)	
z> [loop at Multiply.c:53 in matvec]	0.413s	0.413s			101	Scalar	
[loop at Multiply.c:45 in matvec]	0.109s	12.373s				Collapse	Collapse
z> [loop at Multiply.c:45 in matvec]	0.078s	11.930s			12	Vectorized (Body)	
z> [loop at Multiply.c:45 in matvec]	0.031s	0.444s			2	Remainder	
[loop at Driver.c:146 in main]	0.016s	12.483s			1000000	Scalar	vector dependence prevents vectoriza ...

2.1 Check Dependencies

Identify and explore loop-carried dependencies for marked loops. Fix the reported problems.



Command Line

Select loop for
Dependency
Analysis and
press play!

Vector Dependence
prevents
Vectorization!

Memory Analysis Is Critical

Determine Possible Bandwidth or Latency Issues

Footprint	Small enough	Big enough					
Access Pattern			Source	Stride	Operand Type	Operand Size ...	Aggregated footprint
Unit Stride	Effective SIMD No Latency and BW bottlenecks	Effective SIMD Bandwidth bottleneck	m=1; m<=half; m++) { = fCpMod(i + lbv[3*m], Xmax); = fCpMod(j + lbv[3*m+1], Ymax); = fCpMod(k + lbv[3*m+2], Zmax); = (nextx * Ymax + nexty) * Zmax + nextz;	[0] [0] [3] [0] [3] [0] [3]	int int int int	32 32;64 32 32	48 888 888 888
			lbeitlength + 1*lbay.ng + m + half], lbf[i1next*lbaiteleng	[0] [1] [-4]	float64:int	32;64	9MB
Const stride	Medium SIMD Latency bottleneck possible	Medium SIMD Latency and Bandwidth bottleneck possible	lbeitlength + 1*lbay.ng + m + 1], lbf[i1*lbaitelength + 1				
Irregular Access, Gather/Scatter	Bad SIMD Latency bottleneck possible	Bad SIMD Latency bottleneck					
			Assembly	Physical Stride	Operand Info	Address range	Memory access
			r3				
			f298 1254 add r14d, r12d				
			f2cb 1256 mov r12, qword ptr [r9+rsi*8]	-43775, 118377...	int*1, int*1, i...	0x27561058 - 0x27e5cf20	9MB
			f2cf 1256 vmovd xmm0, qword ptr [r8+rbx*8]	1	float64*1	0x27561098 - 0x275610d0	64B
			f2d5 1256 mov qword ptr [r8+rbx*8], r12	1	int*1	0x27561098 - 0x275610d0	64B
			f2d9 1256 mov r13d, dword ptr [rip+0x1565bc]	0	int*1	0x18589c - 0x18589c	4B

Detect possible latency or bandwidth issues

Enhanced Memory Analysis

Address	Line	Physical Stride	Operand Info	Vector Length	Operand Size (bits)	Address range	Memory access fo...
0x14002eabd							
0x14002eac1							
0x14002eac7							
0x14002eacb							
0x14002ead0							
0x14002ead0	2		float32*8	8	256	0x23a2d960 - 0x23a ...	288B
0x14002ead6	2		float32*8	8	256	0x23a2d980 - 0x23a ...	288B
0x14002eadd							
0x14002eae1							
0x14002eae5			float32*8;int ...	8	32	0x87af10 - 0x87b038	300B
0x14002eaec							
0x14002eaf0							
0x14002eaf4			float32*8;int ...	8	32	0x87af28 - 0x87b038	
0x14002eafb	2		float32*8	8	256	0x879f70 - 0x87b038	
0x14002eb01	2		float32*8	8	256	0x879f90 - 0x87b038	

Is your memory range small enough to fit in cache?

 Gather (irregular) access

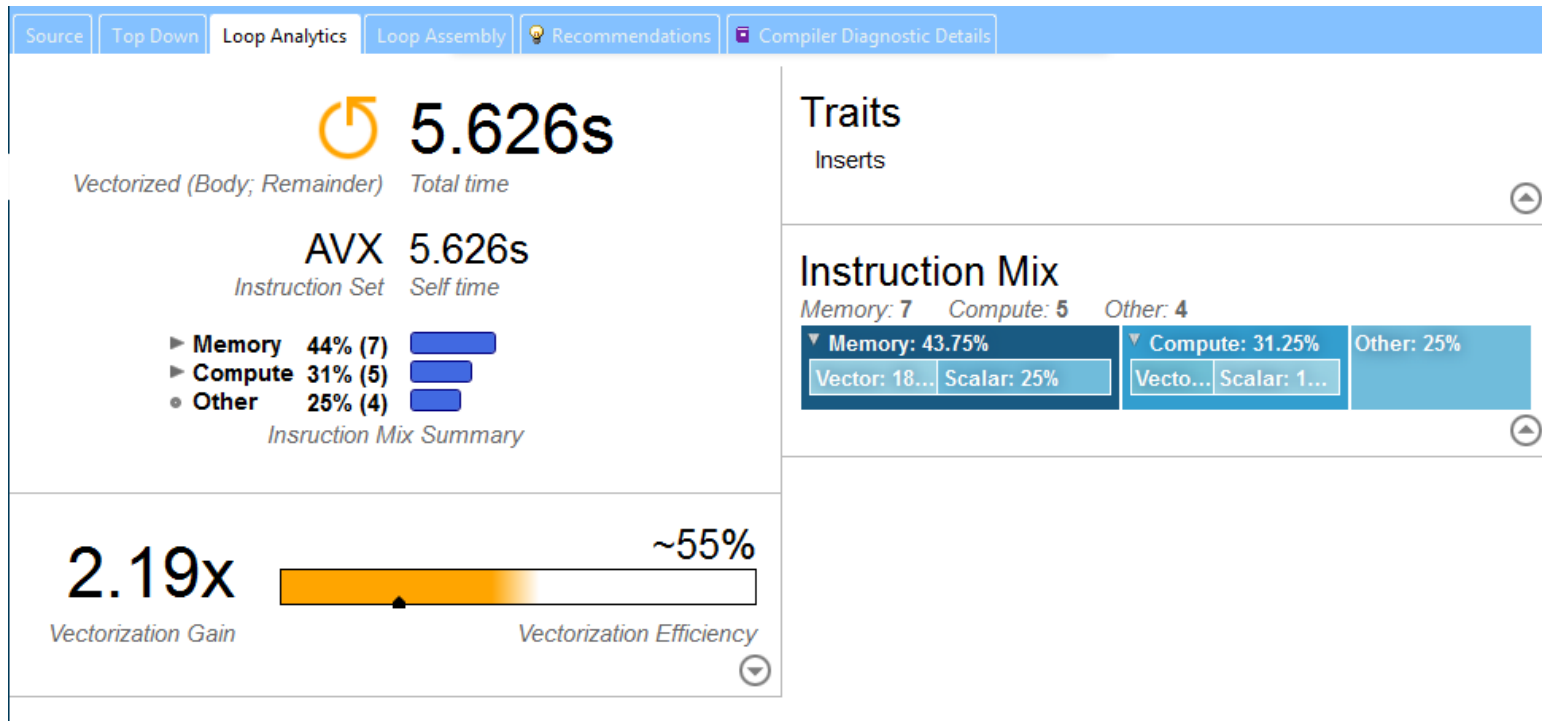
Operand Size (bits): 32
 Operand Type: float32,int32
 Vector Length: 8
 Memory access footprint: 300B

▼ Gather details

Mask is constant
 Mask: [11111111]
 Active elements in the mask: 100.0%

Loop Analytics

Get detailed information about your loops



Gather/Scatter analysis is very important for AVX512

AVX512 Gather/Scatter in wider use than on previous instruction sets

- Many more applications can now be vectorized
- Gives good average performance but far from optimal
- Much greater need for Gather/Scatter profiling
- With Intel® Advisor you get both dynamic and static gather/scatter information

Irregular access patterns decreases performance!

Gather profiling

New type of indicator for stride

Run Memory Access Pattern Analysis

2.2 Check Memory Access Patterns

▶ Collect

🔍 -- Nothing to analyze --

0%: percentage of memory instructions with unit stride or stride 0 accesses

Unit stride (stride 1) = Instruction accesses memory that consistently changes by one element from iteration to iteration

Uniform stride (stride 0) = Instruction accesses the same memory from iteration to iteration

50%: percentage of memory instructions with fixed or constant non-unit stride accesses

Constant stride (stride N) = Instruction accesses memory that consistently changes by N elements from iteration to iteration

Example: for the double floating point type, stride 4 means the memory address accessed by this instruction increased by 32 bytes, (4*sizeof(double)) with each iteration

50%: percentage of memory instructions with irregular (variable or random) stride accesses

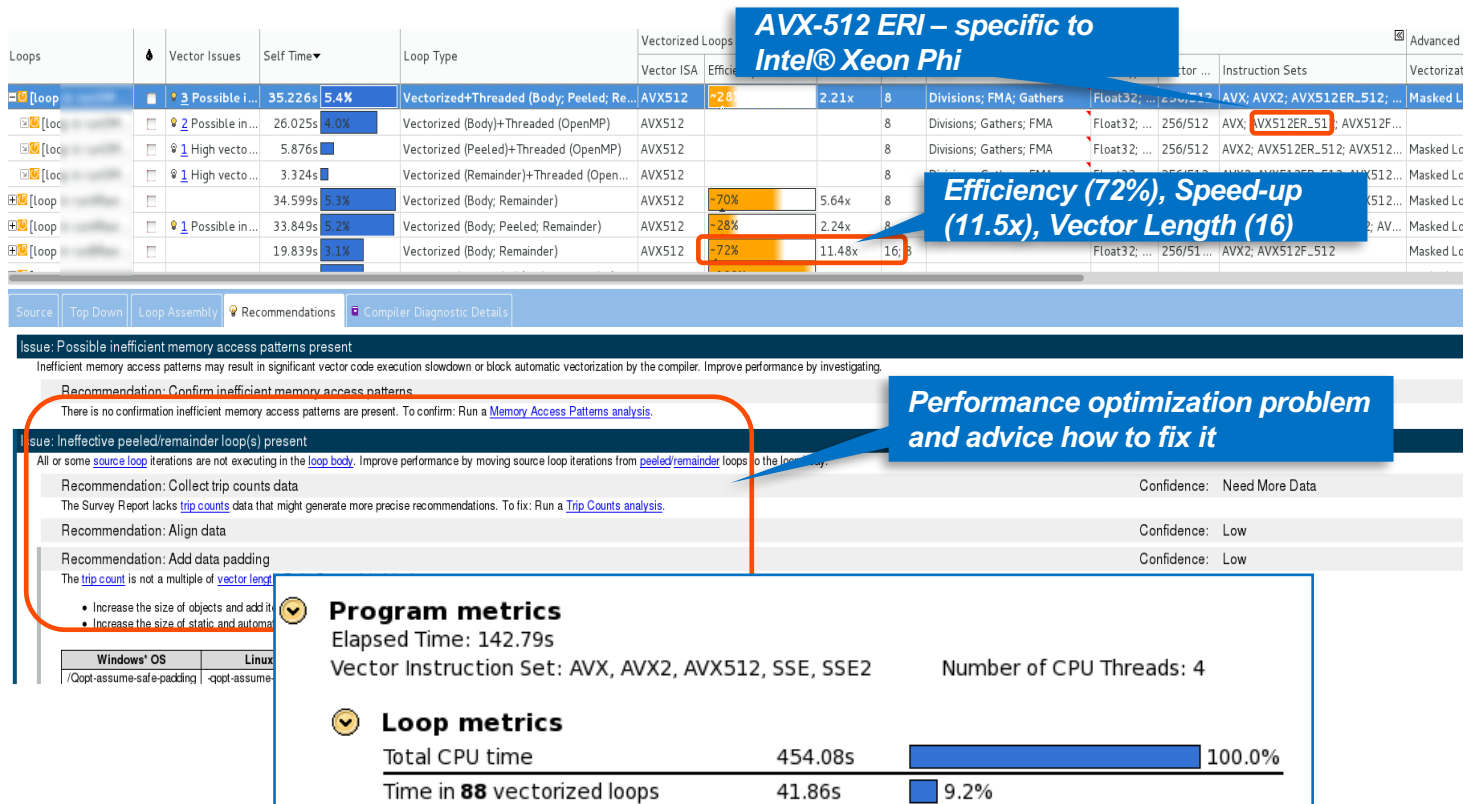
Irregular stride = Instruction accesses memory addresses that change by an unpredictable number of elements from iteration to iteration

Typically observed for indirect indexed array accesses, for example, `a[index[i]]`

- gather (irregular) accesses, detected for `v(p)gather*` instructions on AVX2 Instruction Set Architecture

Vectorization Advisor on KNL AVX-512

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



Mask Utilization and FLOPS profiler

- Long-waiting in HPC: accurate HW independent FLOPs measurement tool
- Not just count FLOPs. Has following additions:
 - (AVX-512 only) Mask-aware. Masked-Memory/Unmasked-Compute pattern aware
 - Unique capability to correlate FLOPs with performance data (obtained without instrumentation). Gives FLOPs/s.
- Lightweight instrumentation, PIN-based, benefits from “threadchecker tools” and more generally Advisor framework integration.