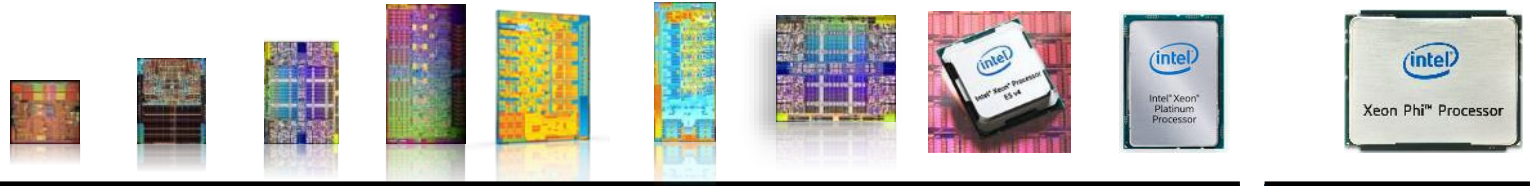
INTEL® ADVISOR

Vectorization Optimization

Klaus-Dieter Oertel
Intel IAGS
HLRN User Workshop, 3-6 Nov 2020

Changing Hardware Impacts Software

More Cores → More Threads → Wider Vectors



	Intel® Xeon® Processor									Intel® Xeon Phi™
	64-bit	5100 series	5500 series	5600 series	E5-2600	E5-2600 V2	E5-2600 V3	E5-2600 V4	Platinum 8180	processor Knights Landing
Core(s)	1	2	4	6	8	12	18	22	28	72
Threads	2	2	8	12	16	24	36	44	56	288
SIMD Width	128	128	128	128	256	256	256	256	512	512

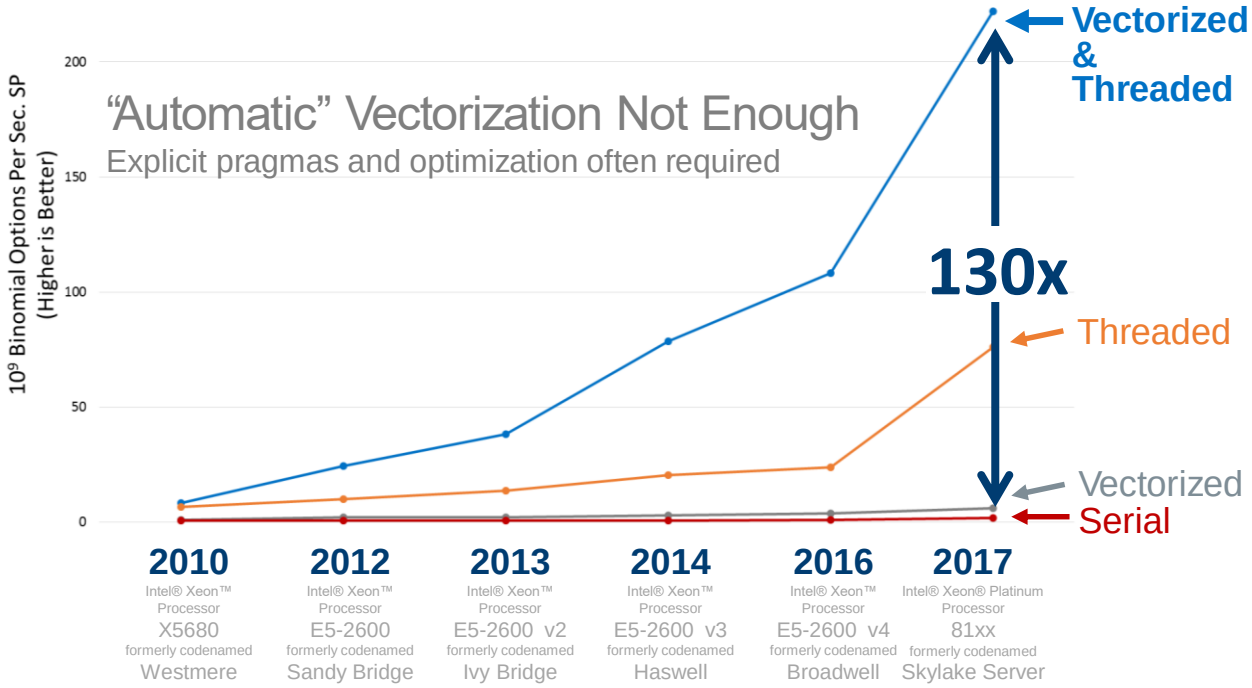
High performance software must be both

- Parallel (multi-thread, multi-process)
- Vectorized

*Product specification for launched and shipped products available on ark.intel.com.

Vectorize & Thread or Performance Dies

Threaded + Vectorized can be much faster than either one alone



The Difference Is Growing With Each New Generation of Hardware

Configurations for 2007-2016 Benchmarks
at the end of this presentation

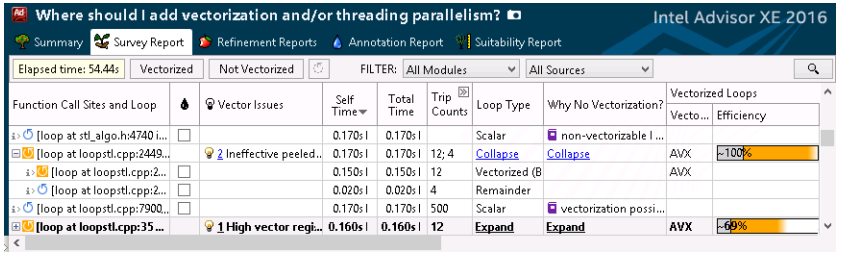
ADVISOR OVERVIEW

Faster Code Faster with Data Driven Design

Intel® Advisor – Vectorization Optimization and Thread Prototyping

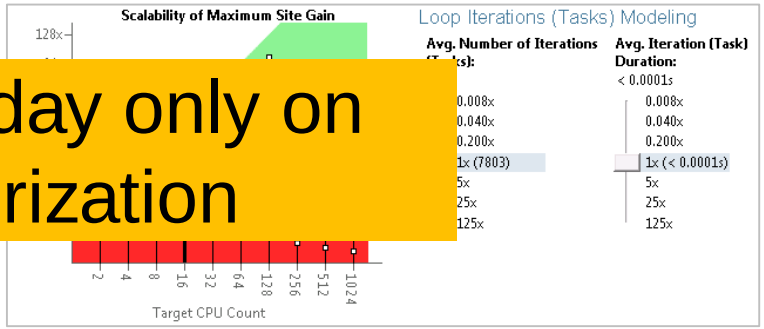
Faster Vectorization Optimization:

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



Breakthrough for Threading Design:

- Quickly prototype multip
- Project scaling on larger
- Find synchronization err
  implementing threading
- Design without disrupting development



Focus today only on vectorization

Less Effort, Less Risk and More Impact

Part of Intel® Parallel Studio for Windows* and Linux* http://intel.ly/advisor-xe

Intel® Advisor – Vectorization Advisor

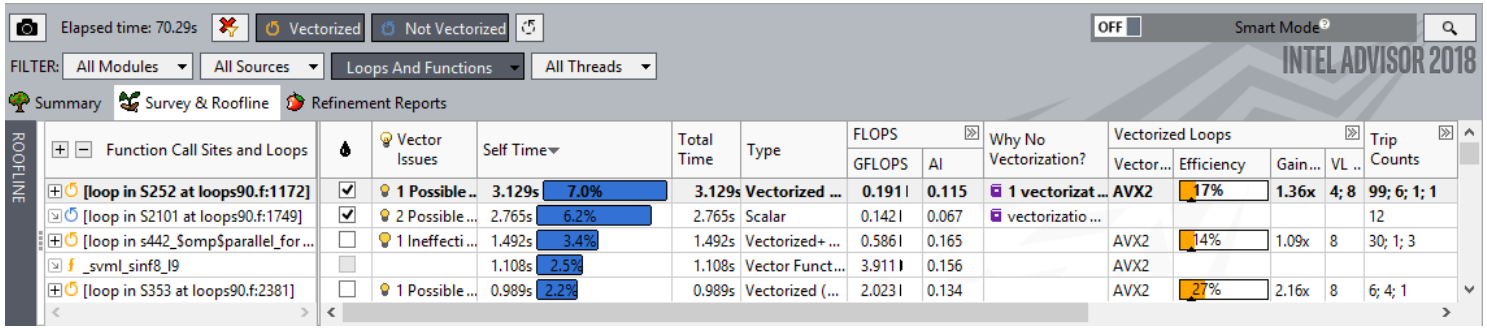
Get breakthrough vectorization performance

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

The data and guidance you need:

- Compiler diagnostics + Performance Data + SIMD efficiency
- Detect problems & recommend fixes
- Loop-Carried Dependency Analysis
- Memory Access Patterns Analysis

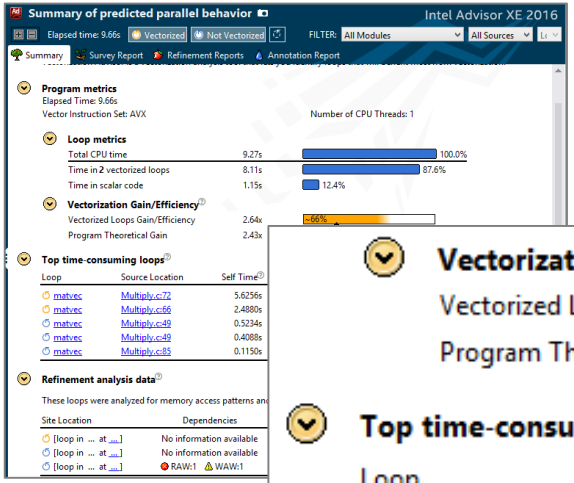


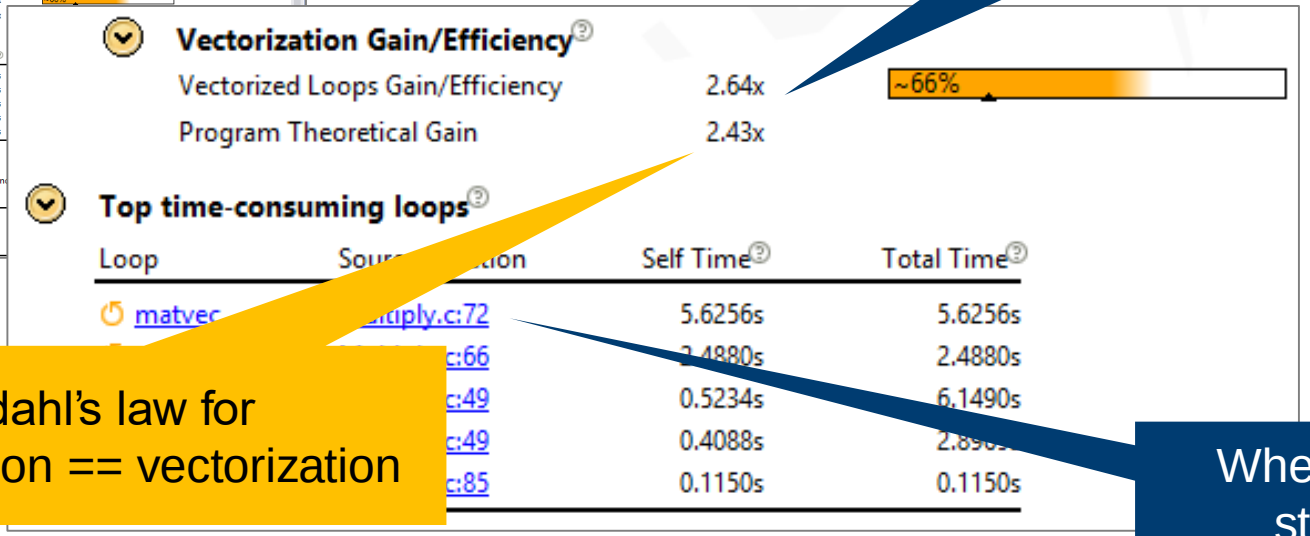
Optimize for AVX-512 with or without access to AVX-512 hardware

Part of Intel® Parallel Studio XE

http://intel.ly/advisor-xe

Summary View: Plan Your Next Steps





What can I expect to gain?

Amdahl's law for parallelization == vectorization

Where do I start?

Amdahl's law

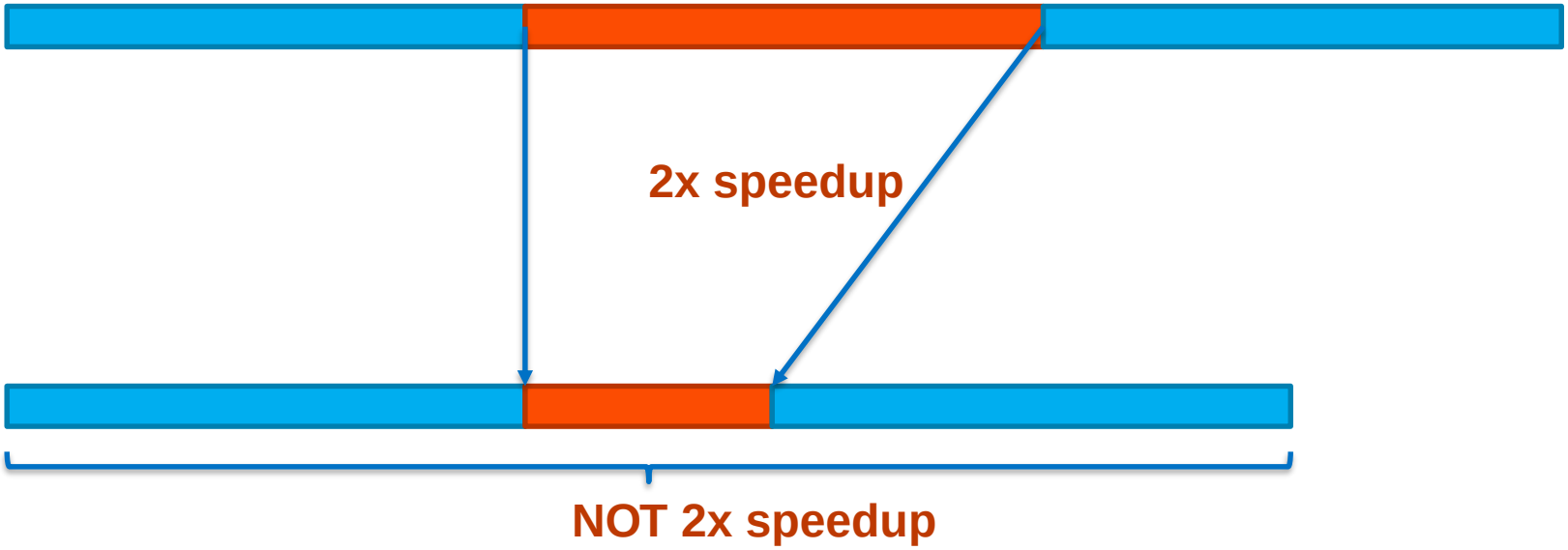
$$S_{total} = \frac{100\%}{(100\% - p) + \frac{p}{s_p}}$$

S = speedup (in parallelized part or total)

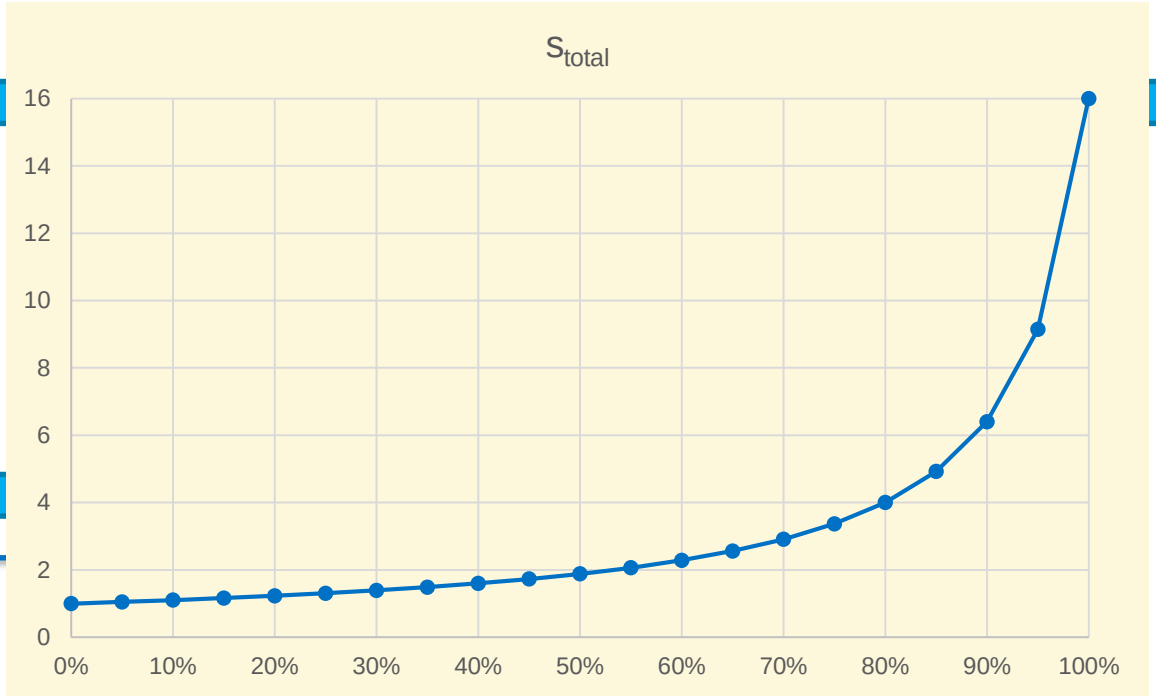
P = proportion of execution time that benefits from parallelization

Example: P=80%, s_p=16 [AVX-512] => S_{total}=4

Amdahl’s law



Amdahl's law



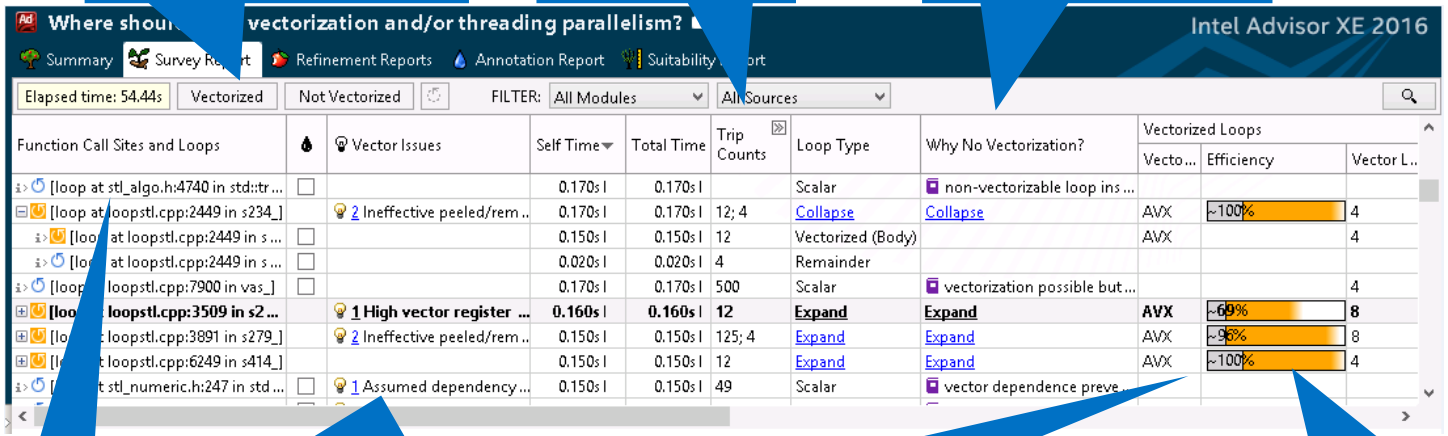
The Right Data At Your Fingertips

Get all the data you need for high impact vectorization

Filter by which loops are vectorized!

Trip Counts

What prevents vectorization?



Focus on hot loops

What vectorization issues do I have?

Which Vector instructions are being used?

How efficient is the code?

Get Faster Code Faster!

THE ROOFLINE MODEL

WHAT IS THE ROOFLINE MODEL ?

Do you know how fast you should run ?

Comes from Berkeley

Performance is limited by equations/implementation & code generation/hardware

2 hardware limitations

- PEAK Flops
- PEAK Bandwidth

The application performance is bounded by hardware specifications

$$\textbf{Gflop/s} = min \begin{cases} Platform\ PEAK \\ Platform\ BW * AI \end{cases}$$

Arithmetic Intensity (Flops/Bytes)

PLATFORM PEAK FLOPS

How many floating point operations per second

$$\text{Gflop/s} = min \begin{cases} Platform\ PEAK \\ Platform\ BW * AI \end{cases}$$

Theoretical value can be computed by specification

Example with 2 sockets Intel® Xeon® Processor E5-2697 v2

PEAK FLOP = 2 x 2.7 x 12 x 8 x 2 = **1036.8 Gflop/s**

- 2: Number of sockets
- 2.7: Core Frequency
- 12: Number of cores
- 8: Number of single precision element in a SIMD register
- 2: 1 port for addition, 1 for multiplication

More realistic value can be obtained by running Linpack

=~ 930 Gflop/s on a 2 sockets Intel® Xeon® Processor E5-2697 v2

PLATFORM PEAK BANDWIDTH

How many bytes can be transferred per second

$$\text{Gflop/s} = min \begin{cases} Platform\ PEAK \\ Platform\ BW * AI \end{cases}$$

Theoretical value can be computed by specification

Example with 2 sockets Intel® Xeon® Processor E5-2697 v2

PEAK BW = 2 x 1.866 x 8 x 4 = 119 GB/s

Number of sockets

Memory Frequency

Byte per channel

Number of mem channels

More realistic value can be obtained by running Stream

=~ 100 GB/s on a 2 sockets Intel® Xeon® Processor E5-2697 v2

DRAWING THE ROOFLINE

Defining the speed of light

$$\text{Gflop/s} = min \begin{cases} Platform\ PEAK \\ Platform\ BW * AI \end{cases}$$

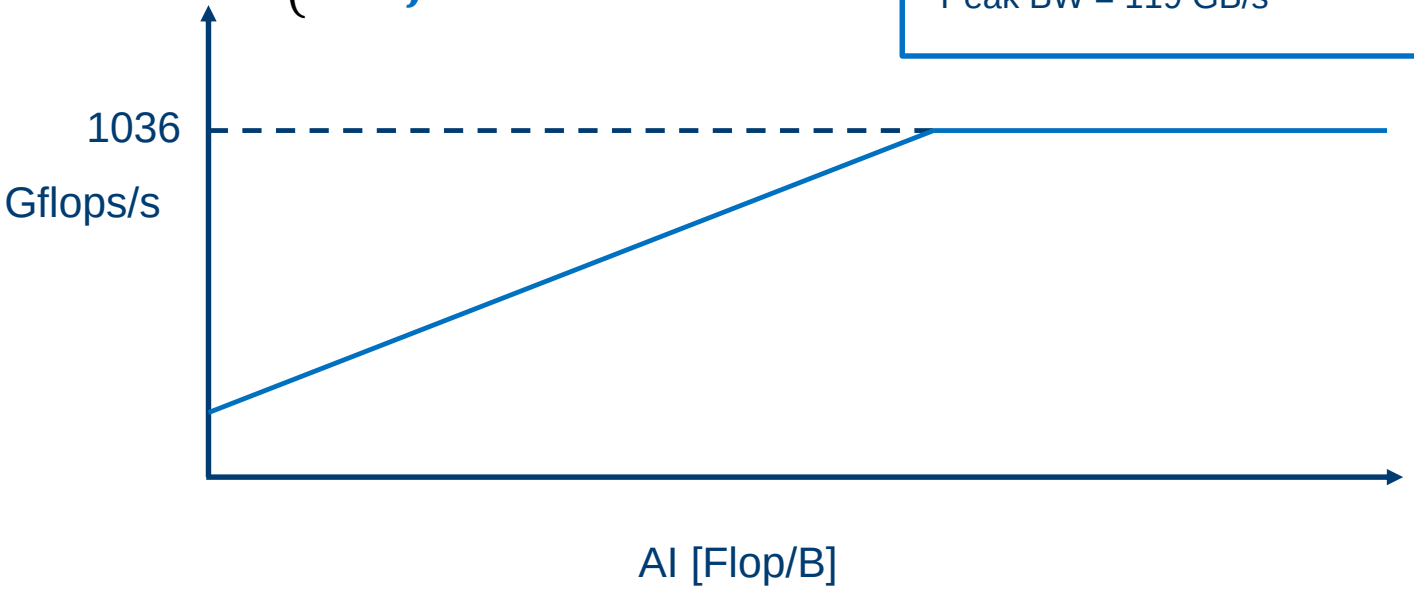
2 sockets Intel® Xeon® Processor E5-2697 v2
Peak Flop = 1036 Gflop/s
Peak BW = 119 GB/s

DRAWING THE ROOFLINE

Defining the speed of light

$$\text{Gflop/s} = min \begin{cases} Platform\ PEAK \\ Platform\ BW * AI \end{cases}$$

2 sockets Intel® Xeon® Processor E5-2697 v2
Peak Flop = 1036 Gflop/s
Peak BW = 119 GB/s

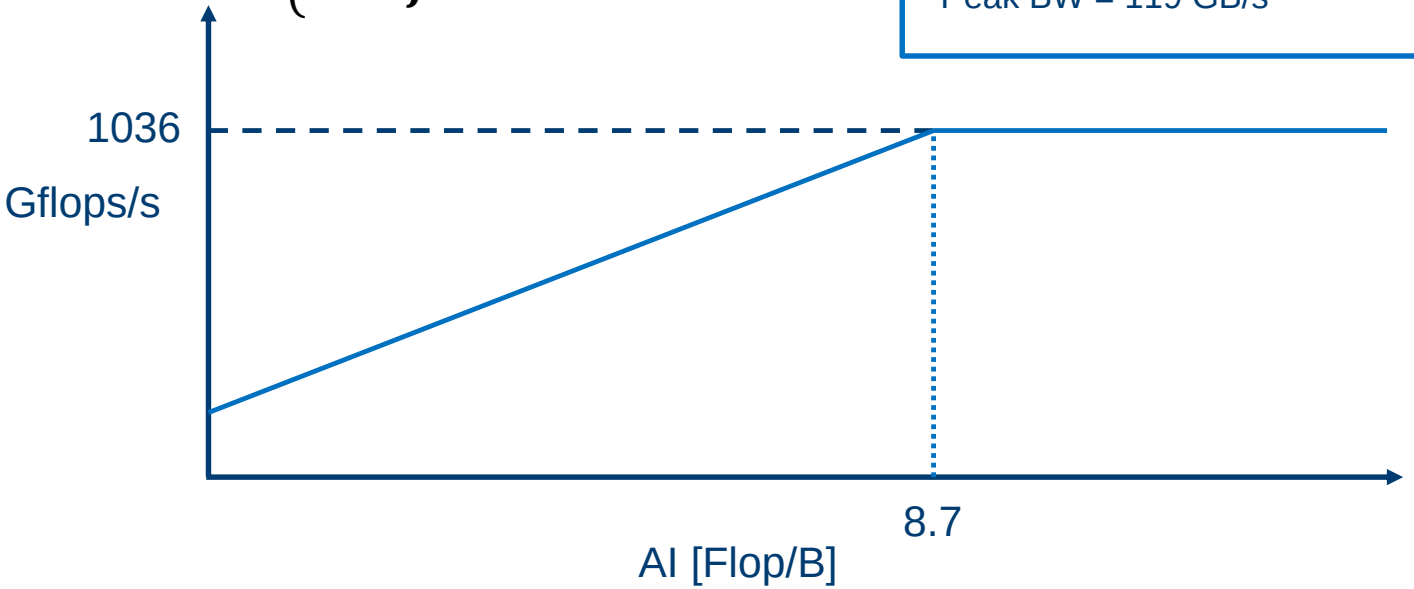


DRAWING THE ROOFLINE

Defining the speed of light

$$\text{Gflop/s} = min \begin{cases} Platform\ PEAK \\ Platform\ BW * AI \end{cases}$$

2 sockets Intel® Xeon® Processor E5-2697 v2
Peak Flop = 1036 Gflop/s
Peak BW = 119 GB/s



WHAT IS THE PERFORMANCE BOUNDARY?

Manual way to do it

Manual counting on matrix/matrix multiplication

```
for(i=0; i<N; i++)
    for(j=0; j<N; j++)
        for(k=0; k<N; k++)
            c[i][j] = c[i][j] + a[i][k] * b[k][j]
```

add = N * N * N

mul = N * N * N

#Read = 3 * N * N * 4 bytes

#Write = N * N * 4 bytes

$$AI = \frac{2N^3}{16N^2} = \frac{1}{8}N$$

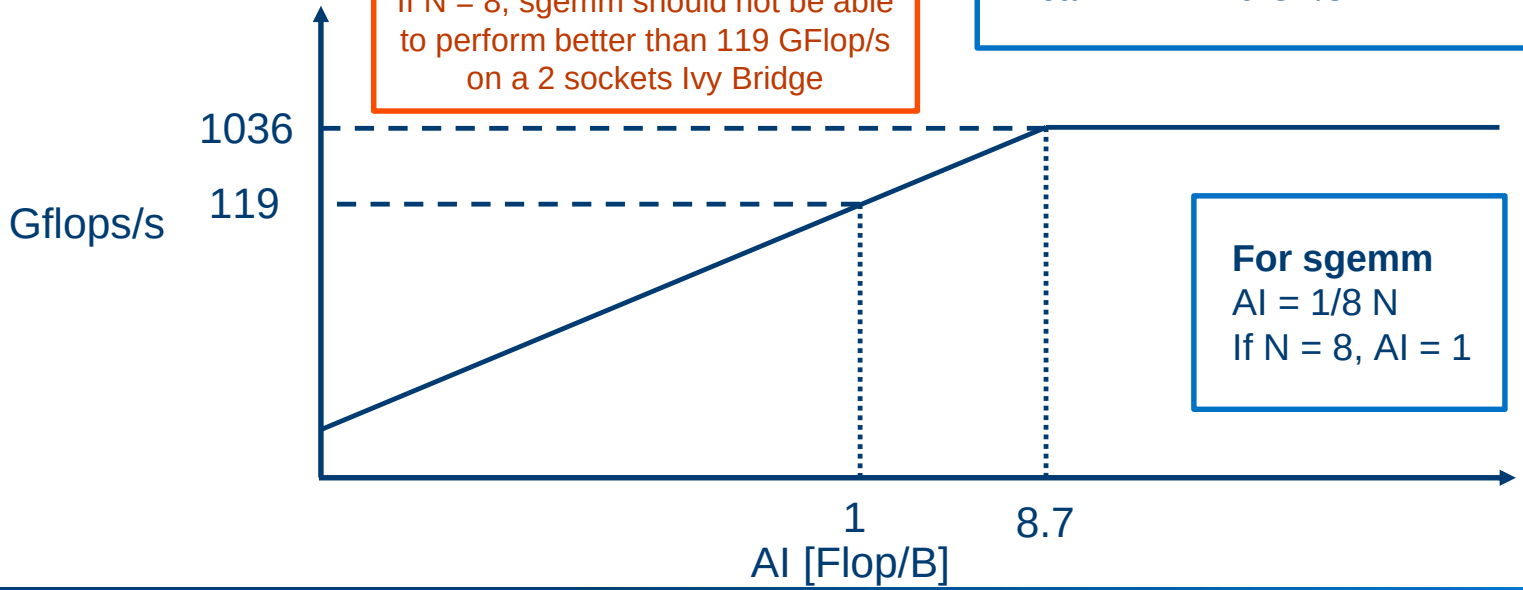
COMPUTE THE MAXIMUM PERFORMANCE

BW * Arithmetic Intensity

$$\text{Gflop/s} = min \begin{cases} Platform\ PEAK \\ Platform\ BW * AI \end{cases}$$

2 sockets Intel® Xeon® Processor E5-2697 v2
Peak Flop = 1036 Gflop/s
Peak BW = 119 GB/s

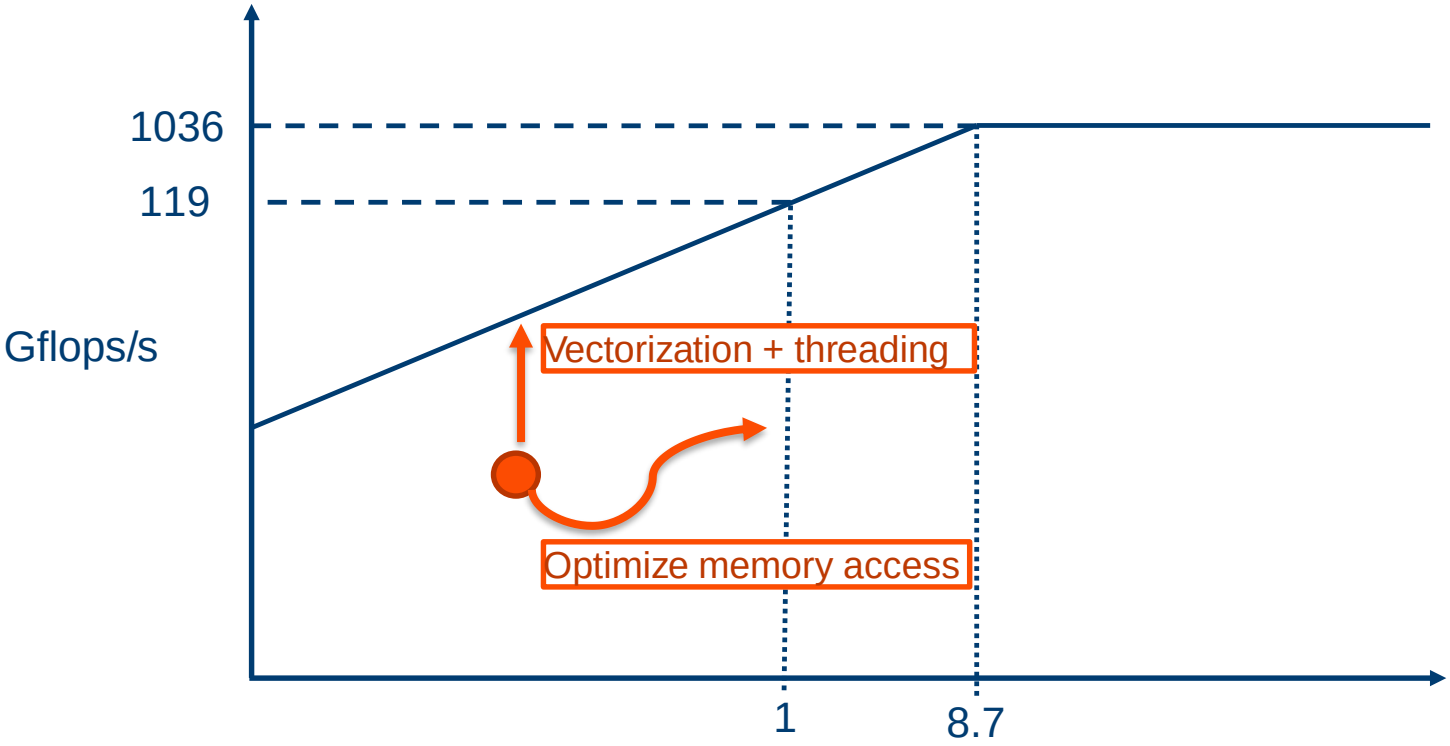
If N = 8, sgemm should not be able to perform better than 119 GFlop/s on a 2 sockets Ivy Bridge



For sgemm
AI = 1/8 N
If N = 8, AI = 1

AND NOW?

How to get better performance?

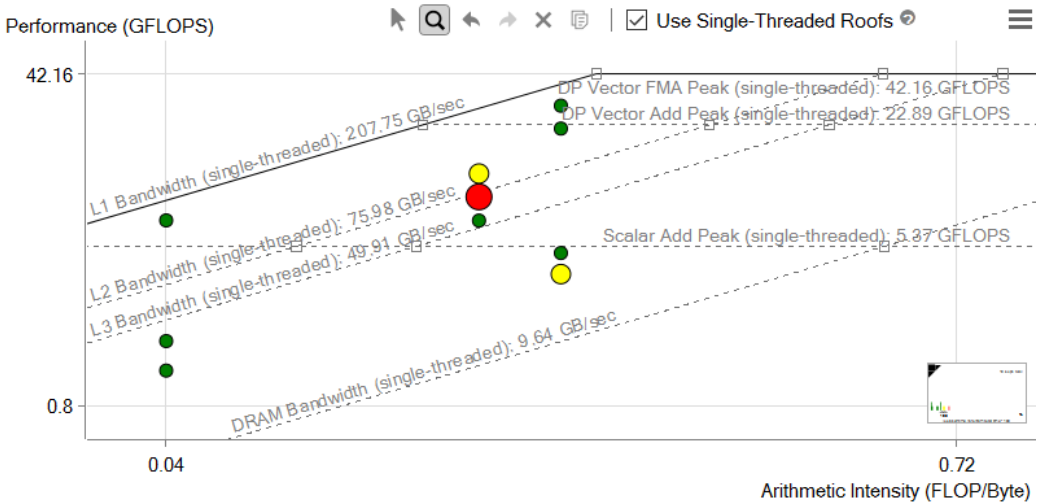


ROOFLINE IN INTEL® ADVISOR

What is a Roofline Chart?

A Roofline Chart plots application performance against hardware limitations.

- Where are the bottlenecks?
- How much performance is being left on the table?
- Which bottlenecks can be addressed, and which *should* be addressed?
- What's the most likely cause?
- What are the next steps?



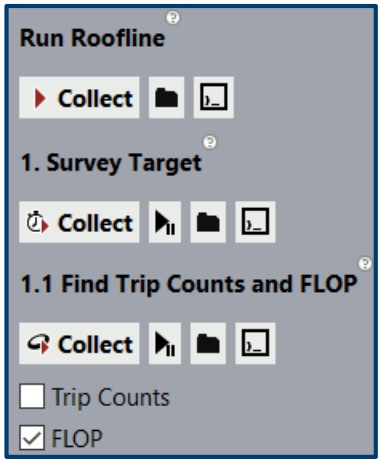
Roofline first proposed by University of California at Berkeley:
Roofline: An Insightful Visual Performance Model for Multicore Architectures, 2009
Cache-aware variant proposed by University of Lisbon:
Cache-Aware Roofline Model: Upgrading the Loft, 2013

Roofline Metrics

Roofline is based on FLOPS and Arithmetic Intensity (AI).

- **FLOPS**: **Fl**oating-Point **Op**erations / **S**econd
- **Arithmetic Intensity**: FLOP / Byte Accessed

Collecting this information in Intel® Advisor requires two analyses.



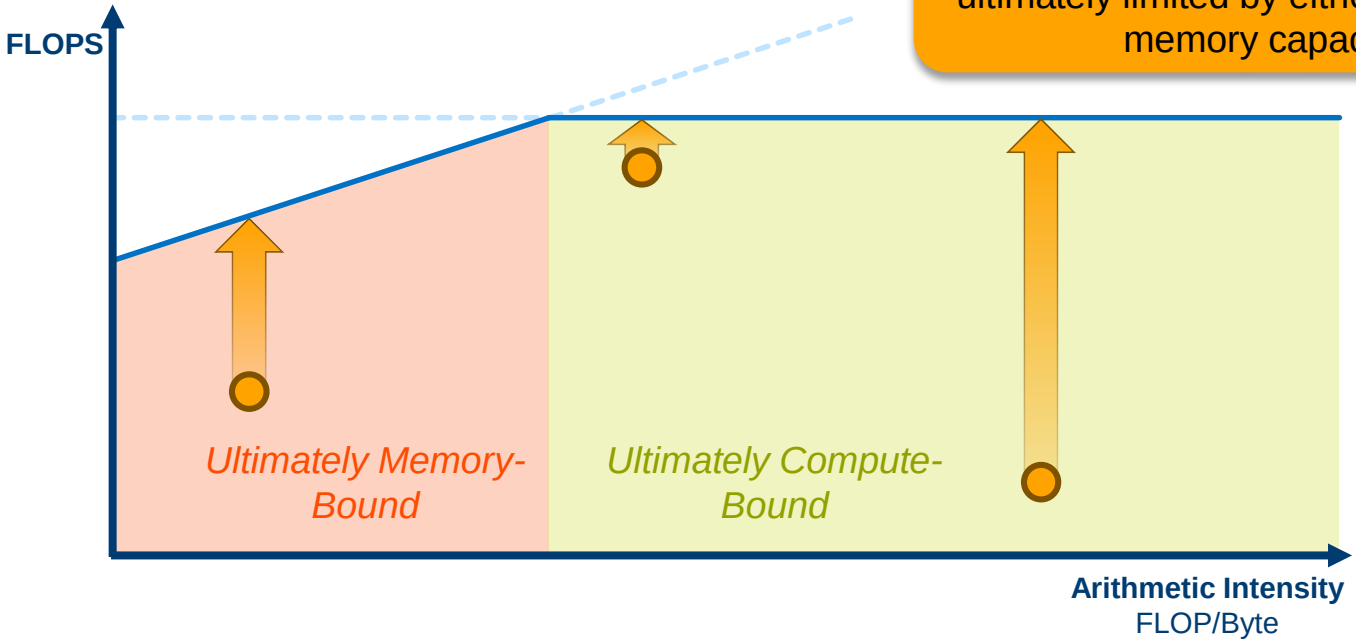
Shortcut to run Survey followed by Trip Counts + FLOPs

Runs system benchmarks and collects timing data.

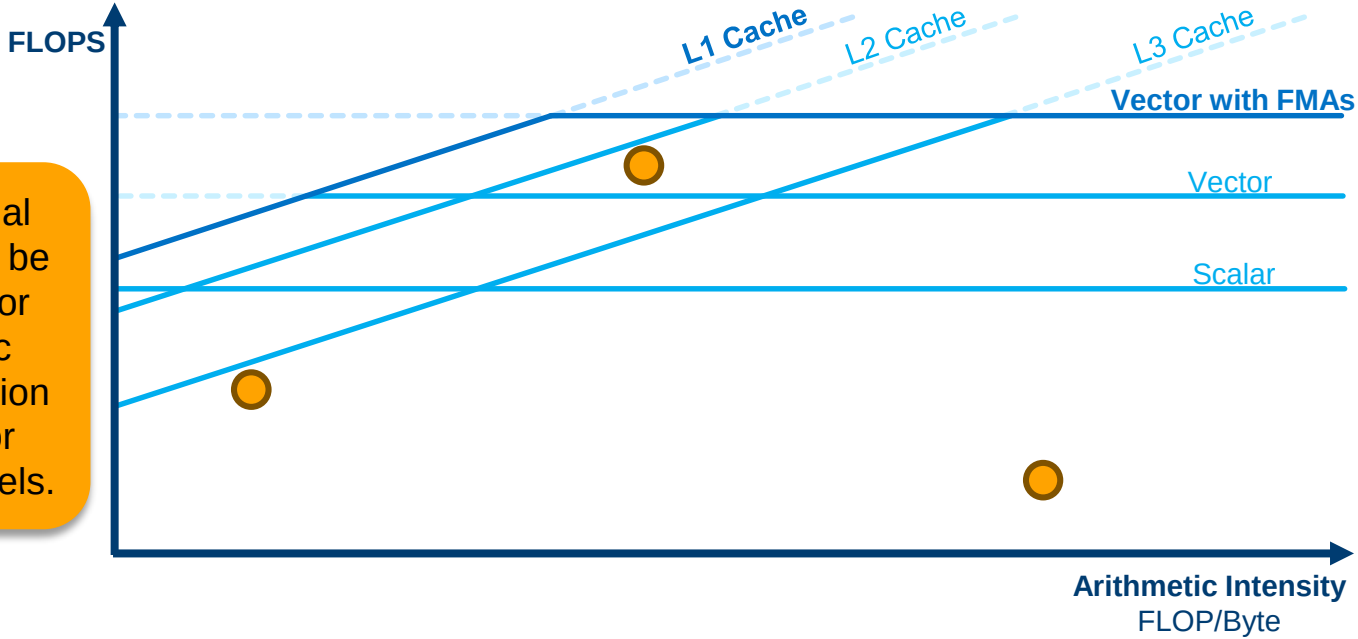
Collects memory traffic and FLOP data.
Must be run separately due to higher overhead that would interfere with timing measurements.

Ultimate Performance Limits

Performance cannot exceed the machine's capabilities, so each loop is ultimately limited by either compute or memory capacity.



Sub-Roofs and Current Limits

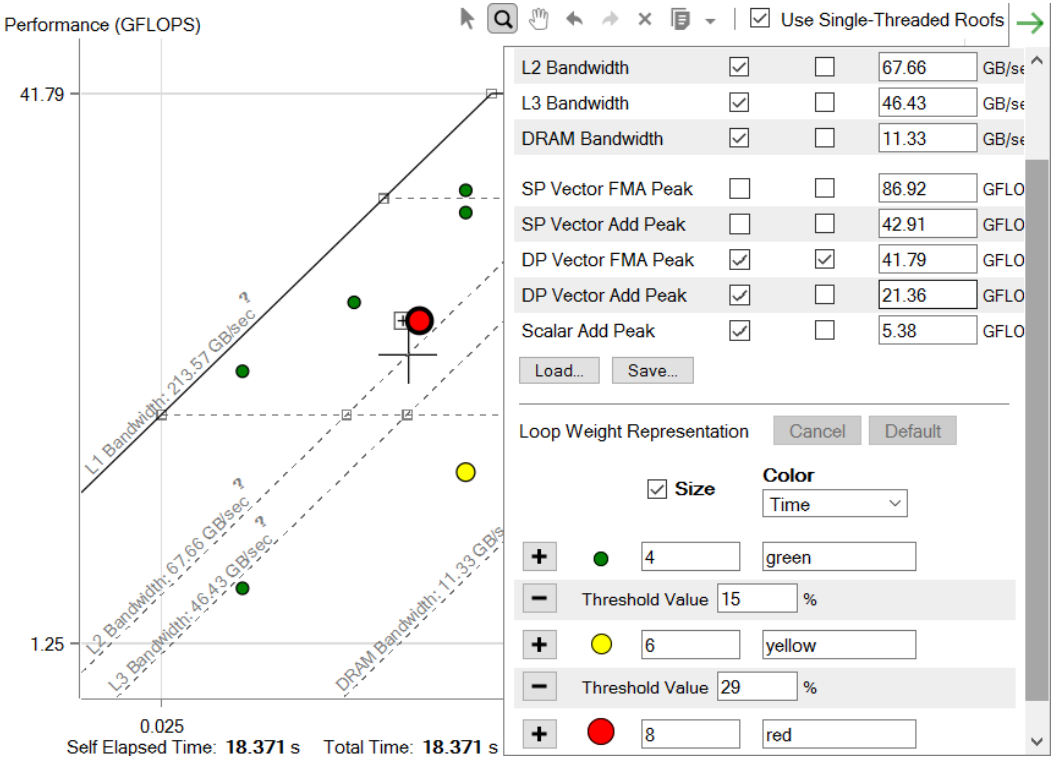


Additional roofs can be plotted for specific computation types or cache levels.

These sub-roofs can be used to help diagnose bottlenecks.

The Intel® Advisor Roofline Interface

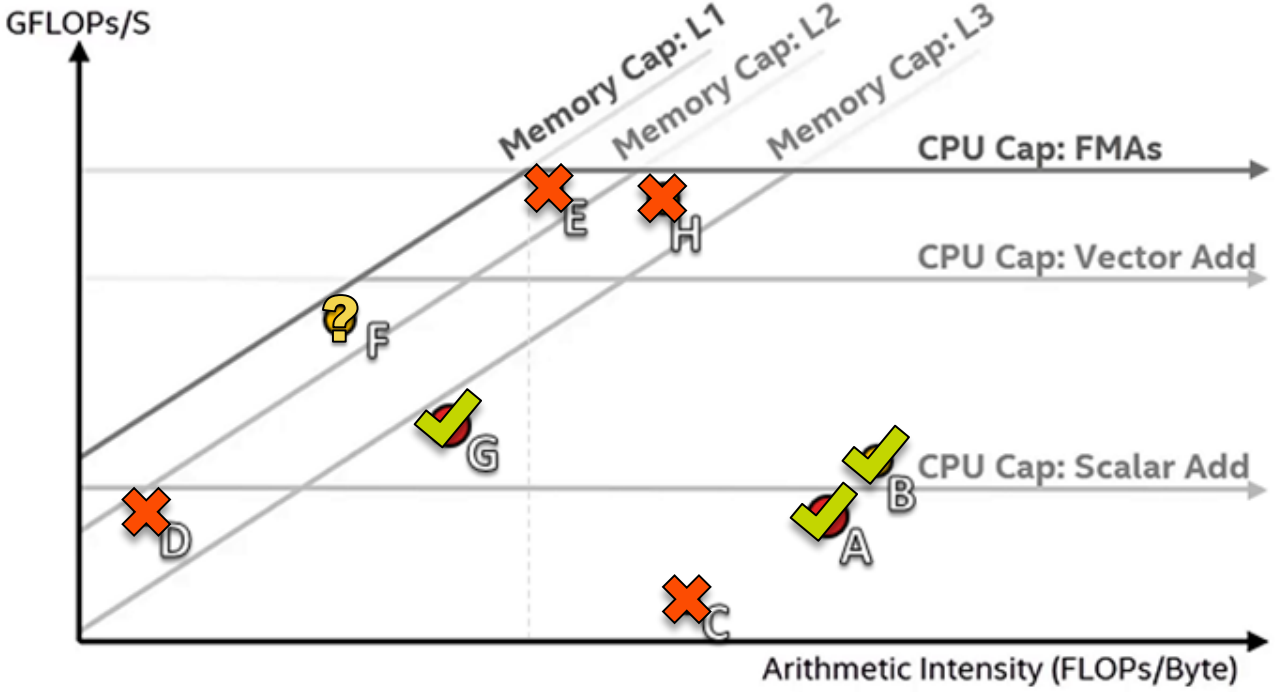
- Roofs are based on benchmarks run before the application.
  - Roofs can be hidden, highlighted, or adjusted.
- Intel® Advisor has size- and color-coding for dots.
  - Color code by duration or vectorization status
  - Categories, cutoffs, and visual style can be modified.



Identifying Good Optimization Candidates

Focus optimization effort where it makes the most difference.

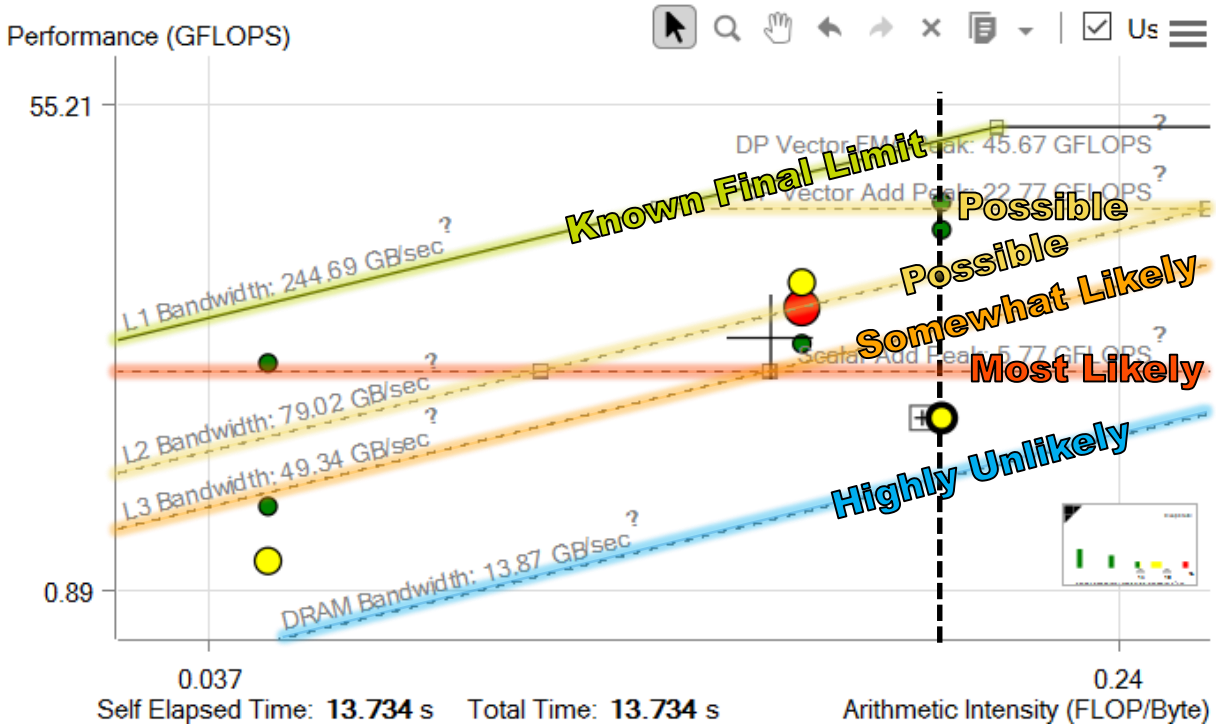
- Large, red loops have the most impact.
- Loops far from the upper roofs have more room to improve.



Identifying Potential Bottlenecks

Final roofs *do* apply;
sub-roofs *may* apply.

- Roofs above indicate *potential* bottlenecks
- Closer roofs are the most likely suspects
- Roofs below may contribute but are generally not primary bottlenecks

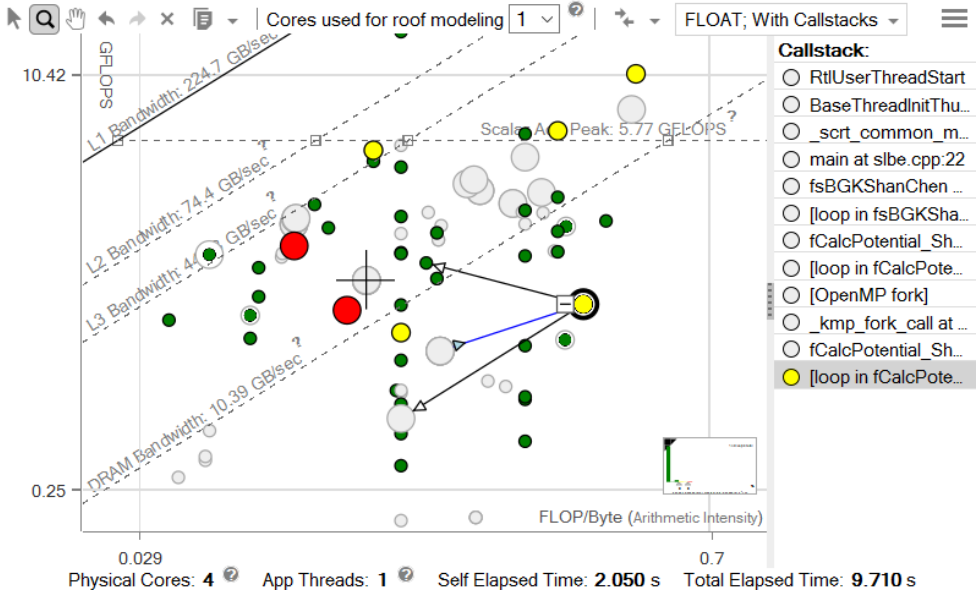


ROOFLINE WITH CALL STACKS

Roofline with Call Stacks

Advisor 2018 Update 1 added call stacks to Roofline.

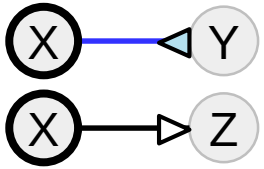
- Granularity of data can be adjusted.
- Reveals inefficiencies that originate higher in the call chain.
- More accurate view of functions or loops that behave differently under different circumstances
  - Differentiates between instances with different call chains.



Reading the Roofline with Call Stacks

Visualizing the Call Chain

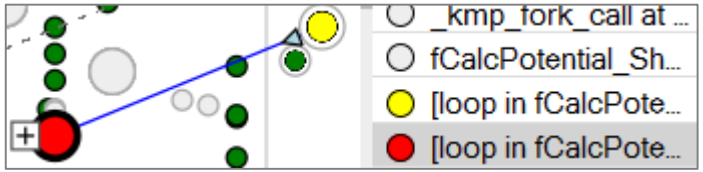
Arrows indicate relationships between dots.

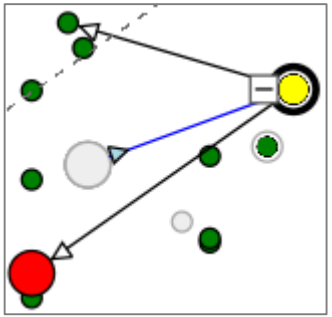


X is called directly by Y.

X directly calls Z

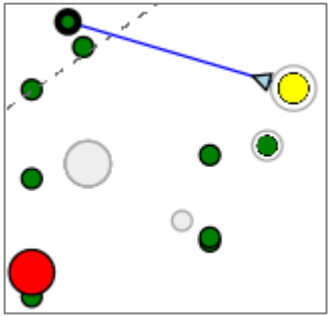
The call stack displays the call chain for the selected loop. Clicking an entry causes it to flash on the Roofline for easy identification.





The selected yellow dot was called by the gray dot, and it calls the red and green dots.

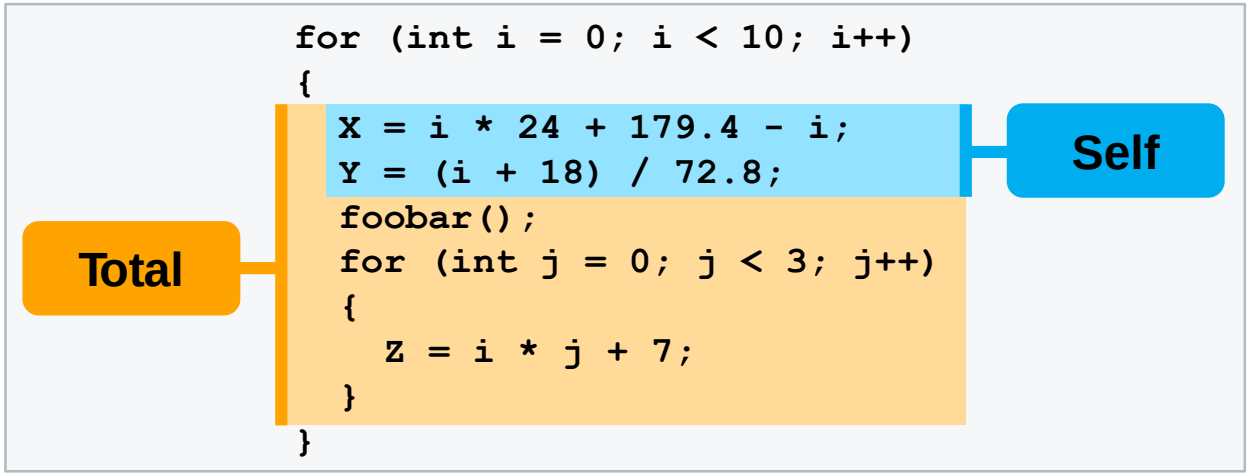
Selecting the green dot shows that it is called by the yellow dot, and doesn't call anything itself.



Self Data vs Total Data

The original Roofline used only **self data**: only work done directly is recorded.

The Roofline with call stacks uses both **self data and total data**, which includes work done in functions or loops called as well as work done directly.



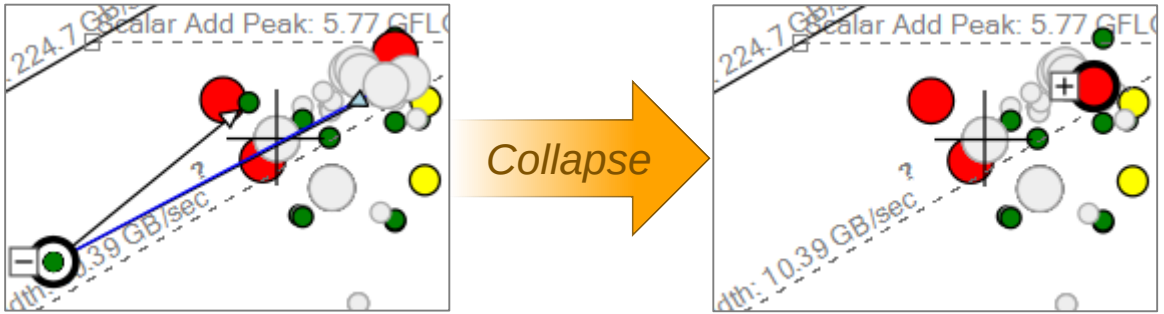
Reading the Roofline with Call Stacks

Expanding and Collapsing Outer Loops

Collapsing and expanding dots switches between self- and total-data mode.

Dots with no self data are grayed out when expanded and in color when collapsed.

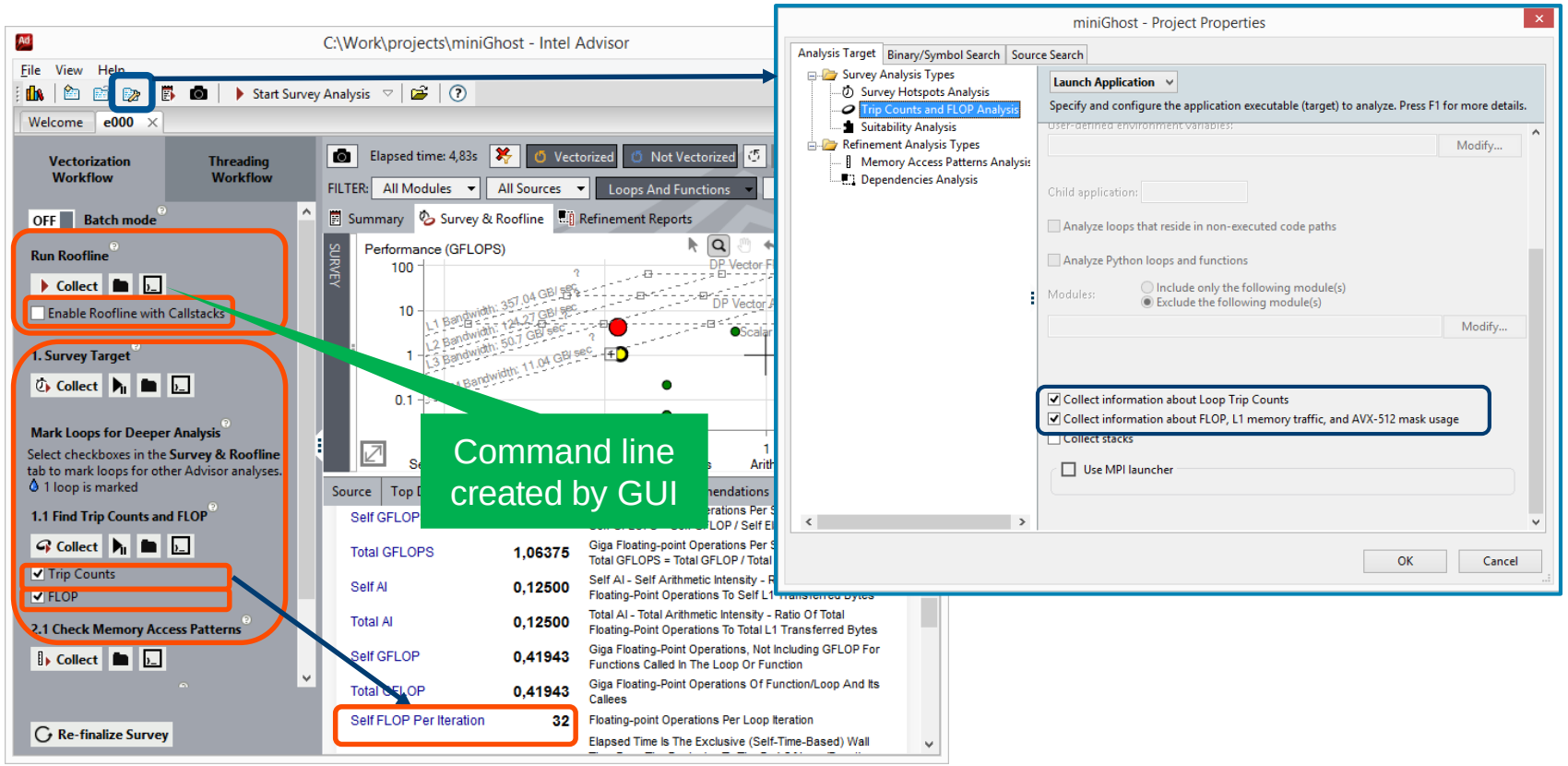
Dots that have self data have the appearance and location based on it when expanded, with a halo of the size related to their total data.



When collapsed, their appearance and location changes to reflect the total data.

GUI AND COMMAND LINE

Get Roofline data using **GUI**



Get roofline data using **command line**. Example:

```
> source advixe-vars.sh
```

1st method:

```
> advixe-cl -collect roofline -project-dir ./your_project -- <your-executable-with-parameters>
```

2nd method (more flexible):

```
> advixe-cl -collect survey -project-dir ./your_project -- <your-executable-with-parameters>
```

```
> advixe-cl -collect tripcounts -flop -project-dir ./your_project -- <your-executable-with-parameters>
```

```
> advixe-gui ./your_project
```

Running Intel Advisor XE on a cluster

Example: Collect from middle rank of 3x3x3 cube of processes:

```
mpirun -n 27 advixe-cl -collect survey-project-dir ./my_proj ./your_app

mpirun -n 13 ./your_app \
          : -n 1 advixe-cl -collect survey -project-dir ./my_proj ./your_app \
          : -n 13 ./your_app
```

Intel MPI-specific (adding corner rank and middle surface rank):

```
mpirun –n 27 \
 –gtool “advixe-cl –collect survey --project-dir ./my_proj :1,5,14” ./your_app
```

or: I_MPI_GTOOL=“advixe-cl –collect survey --project-dir ./my_proj :1,5,14”

FLOPS AND MASK UTILIZATION PROFILER

Precise Repeatable FLOPS Metrics

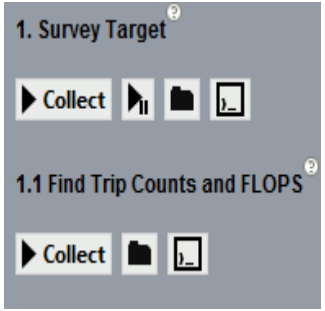
Intel® Advisor – Vectorization Optimization

- FLOPS by loop and function
- All recent Intel processors (not co-processors)
- Instrumentation (count FLOPs) plus sampling (time with low overhead)
- Adjusted for masking with AVX-512 processors

INTEL ADVISOR 2017

Function Call Sites and Loops	FLOPS						
	GFLOPS	AI	L1 GB/s	GFLOP	FLOP Per Iteration	L1 GB	L1 Bytes Per Iteration
[loop in matvec at Multiply.c:69]	0.826	0.1633	5.0586	3.0720	32	18.8160	196
[loop in matvec at Multiply.c:60]	0.912	0.1633	5.5853	3.0720	32	18.8160	196
[loop in matvec at Multiply.c:69]	1.248	0.2500	4.9920	1.3440	4	5.3760	16
[loop in matvec at Multiply.c:60]	1.592	0.2500	6.3699	1.3440	4	5.3760	16
[loop in matvec at Multiply.c:69]	3.055	0.2500	12.2205	0.0960	16	0.3840	64
[loop in matvec at Multiply.c:60]	6.282	0.2500	25.1279	0.0960	16	0.3840	64

Getting FLOP/S in Advisor



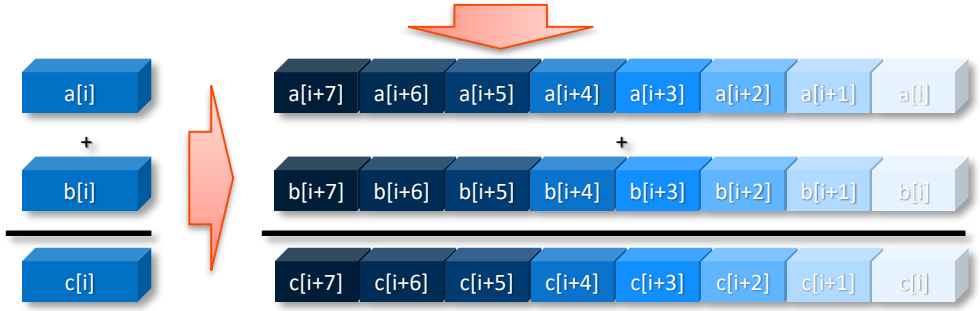
FLOP/S **= #FLOP/Seconds**	**Seconds**	**#FLOP** **- and Mask Utilization** **- and #Bytes**
Step 1: Survey - Not intrusive, sampling based, ~5-10% overhead - Time/Performance-representative	✓	
Step 2: Trip counts+FLOPS - Precise, instrumentation based - Physically count Num-Instructions - Possible to do all types of dynamic analysis including mask register tracking etc		✓

Why Mask Utilization is Important?

100%

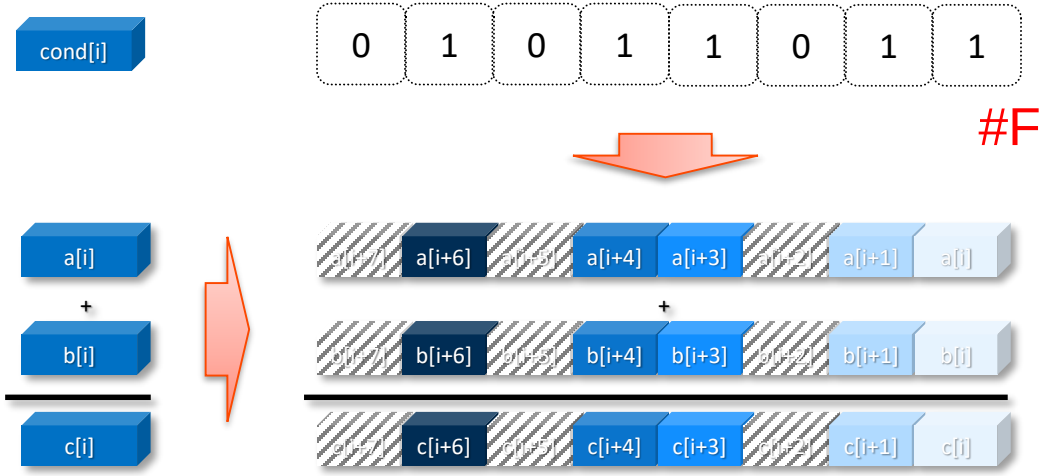
```
for(i = 0; i <= MAX; i++)
    c[i] = a[i] + b[i];
```

#FLOP (MAX = 8): **8**



Why Mask Utilization Important?

```
for(i = 0; i <= MAX; i++)
   if (cond(i))
      c[i] = a[i] + b[i];
```



3 elements suppressed

Mask Utilization = 5/8

62.5%

#FLOP (MAX = 8): **5**

VECTORIZATION EFFICIENCY

Spend your time in the most efficient place! A typical vectorized loop consists of…

Main vector body

- Fastest among the three!

Optional peel part

- Used for the unaligned references in your loop. Uses Scalar or slower vector

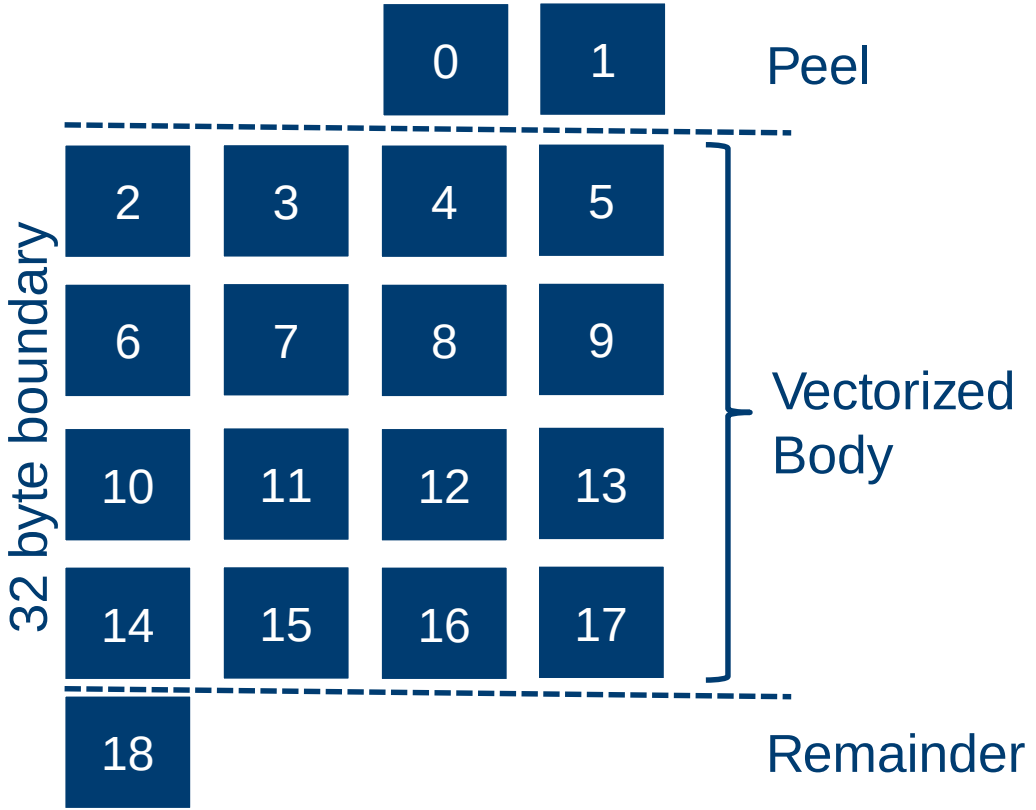
Remainder part

- Due to the number of iterations (trip count) not being divisible by vector length. Uses Scalar or slower vector.

Larger vector register means more iterations in peel/remainder

- Make sure you align your data! (and you tell the compiler it is aligned!)
- Make the number of iterations divisible by the vector length!

What are peels and remainders?

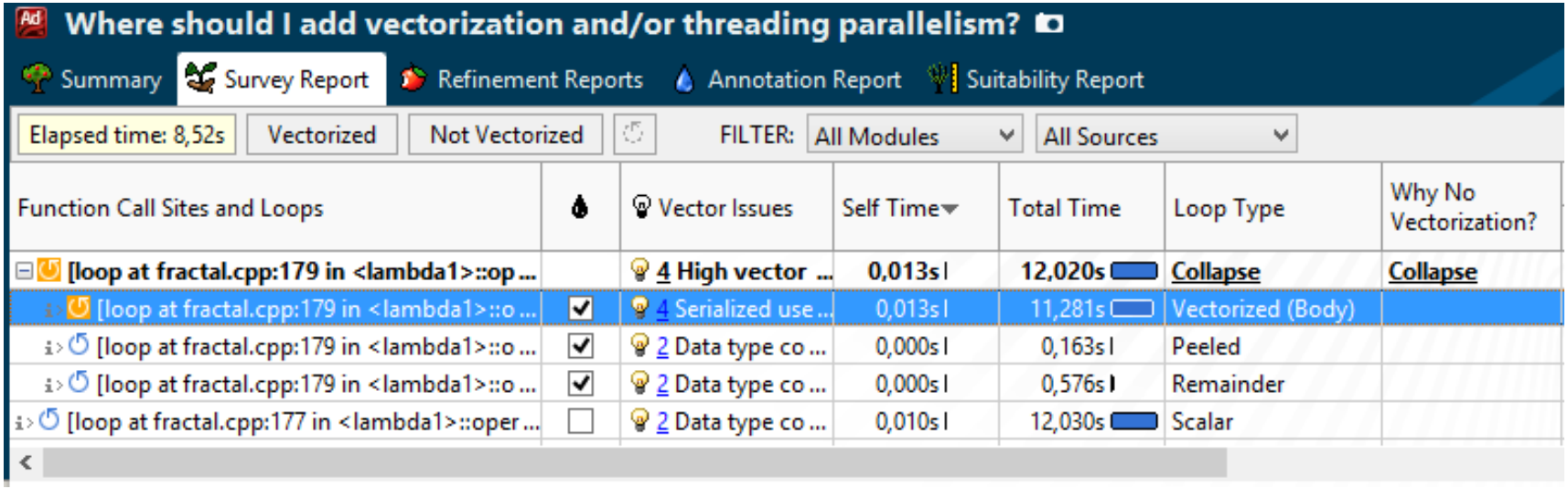



```
// xAVX
// 256 bits wide regs
// holds 4 x 64bit vals

void Func(double *pA)
{
  for (int i=0; i<19;i++)
              pA[i] = …;
}
```

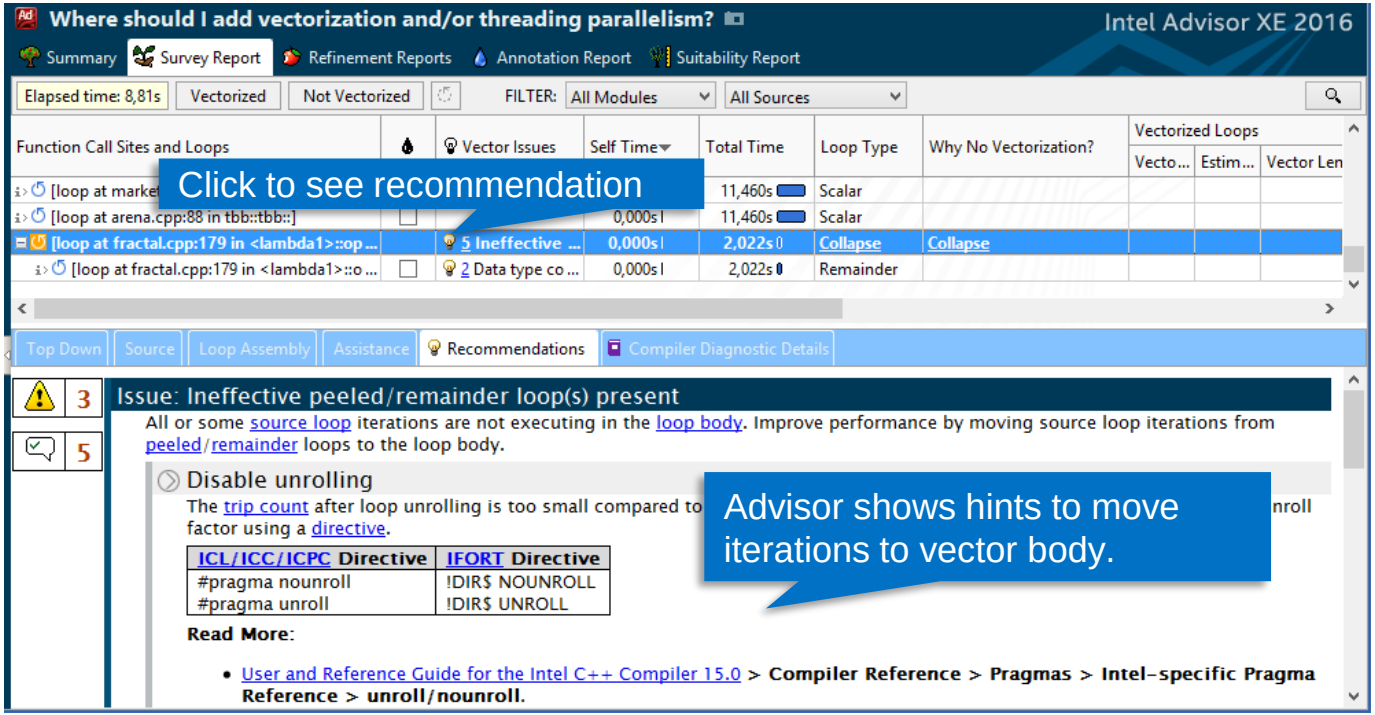
Don’t Just Vectorize, Vectorize Efficiently

See detailed times for each part of your loops. Is it worth more effort?



Get Specific Advice For Improving Vectorization

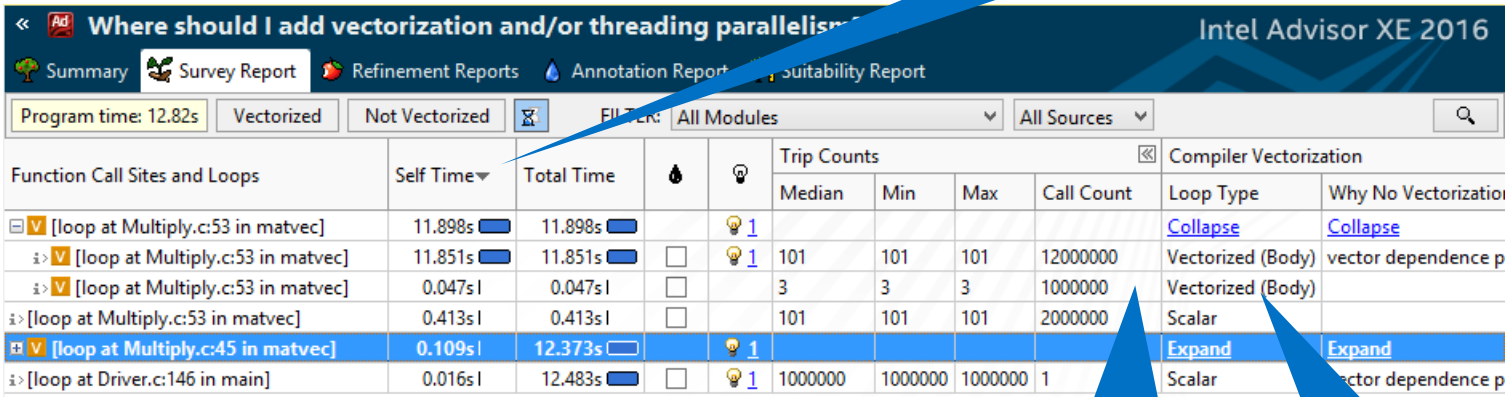
Intel® Advisor – Vectorization Advisor



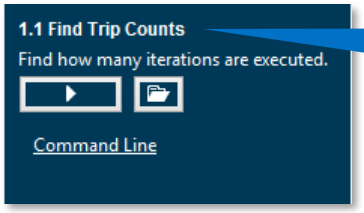
Critical Data Made Easy

Loop Trip Counts

Knowing the time spent in a loop is not enough!



Function Call Sites and Loops	Self Time	Total Time			Trip Counts				Compiler Vectorization	
					Median	Min	Max	Call Count	Loop Type	Why No Vectorizatio
[loop at Multiply.c:53 in matvec]	11.898s	11.898s		1					Collapse	Collapse
[loop at Multiply.c:53 in matvec]	11.851s	11.851s		1	101	101	101	12000000	Vectorized (Body)	vector dependence p
[loop at Multiply.c:53 in matvec]	0.047s	0.047s			3	3	3	1000000	Vectorized (Body)	
[loop at Multiply.c:53 in matvec]	0.413s	0.413s			101	101	101	2000000	Scalar	
[loop at Multiply.c:45 in matvec]	0.109s	12.373s		1					Expand	Expand
[loop at Driver.c:146 in main]	0.016s	12.483s		1	1000000	1000000	1000000	1	Scalar	ector dependence p



Check actual trip counts

Loop is iterating 101 times but called > million times

Since the loop is called so many times it would be a big win if we can get it to vectorize.

Factors that **slow-down** your **Vectorized** code

1.A. Indirect memory access

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

i ≠ j ➔ B[i] ≠ B[j]

1.B Memory sub-system Latency / Throughput

```
void scale(int *a, int *b)
{
   for (int i = 0; i < VERY_BIG; i++){
         c[i] = z * a[i][j];
         b[i] = z * a[i];
   }
}
```

2. Serialized or “sub-optimal” function calls

```
for (i = 1; i < nx; i++) {
     sumx = sumx +
          serialized_func_call(x, y, xp);
}
```

3. Small trip counts not multiple of VL

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

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

```
for(i = 0; i <= MAX; i++) {
          if ( D[i] < N)
                    do_this(D);
          else if (D[i] > M)
                    do_that();
          //…
 }
```

5. MANY others: spill/fill, fp accuracy trade-offs, FMA, DIV/SQRT, Unrolling, even AVX throttling..

Factors that **slow-down** your **Vectorized** code

1.A. Indirect memory access

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

1.B Memory sub-system Latency / Throughput

```
void scale(int *a, int *b)
{
   for (int i = 0; i < VERY_BIG; i++)
                c[i] = z * a[i][j];
                b[i] = z * a[i];
}
```

2. Serialized or “sub-optimal” function calls

```
for (i = 1; i < nx; i++) {
                sumx = sumx +
                serialized_func_call(x, y,
xp);
}
```

3. Small trip counts not multiple of VL

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

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

```
for(i = 0; i <= MAX; i++) {
                if ( D[i] < N)
                                do_this(D);
                else if (D[i] > M)
                                do_that();
                //…
 }
```

5. MANY others: spill/fill, fp accuracy trade-offs, FMA, DIV/SQRT, Unrolling

Vector Efficiency: All The Data In One Place

My “performance thermometer”

Elapsed time: 8,01s

Loops	Vecto...	Efficiency	Estimated Gain	Vect...	Co	Traits	Vector Widths	Self Time
[loop at lbpSUB.cpp:1280 in fPropagationS ...	AVX	13%	0,53	4	0,53	Blends; Extracts; Inserts; Shuffles	128/256	2,312s
[loop at lbpGET.cpp:152 in fGetFracSite]	AVX	30%	2,38	8	2,34	Blends; Inserts; Masked Stores	128/256	0,030s
[loop at lbpGET.cpp:42 in fGetOneMassSite]	AVX	36%	2,86	8	2,79		256	0,100s
[loop at lbpGET.cpp:78 in fGetTotMassSite]	AVX	36%	2,86	8	2,79		256	0,010s
[loop at lbpGET.cpp:334 in fGetOneDirecSp ...	AVX	38%	3,05	8	2,97	Type Conversions	128/256	0,011s
[loop at lbpBGK.cpp:840 in fCollisionBGK]	AVX	100%	2,05	2	2,05		128	0,080s

Achieved Efficiency

Original (scalar) code efficiency. Corresponds to 1x speed-up.

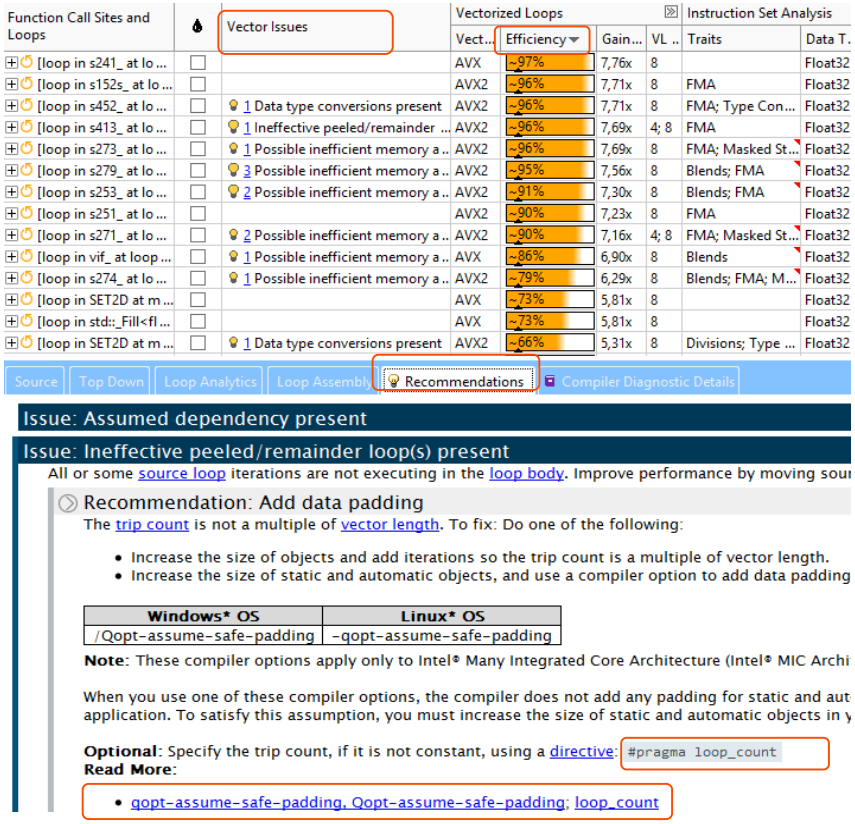
Upper bound: 100% efficiency 4x gain (VL=4)

- **Auto-vectorization**: affected <3% of code
  - With moderate speed-ups
- First attempt to **simply put #pragma simd**:
  - Introduced slow-down
- Look at Vector Issues and Traits to find out why
  - All kinds of “memory manipulations”
  - Usually an indication of “bad” access pattern

Survey: Find out if your code is “under vectorized” and why

Advisor Survey: **Focus** + **Characterize.**

Focus and order **vectorized loops**



Function Call Sites and Loops	Vector Issues	Vectorized Loops: Vect...	Efficiency	Gain...	VL ..	Instruction Set Analysis: Traits	Data T.
[loop in s241_ at lo ...		AVX	~97%	7,76x	8		Float32
[loop in s152s_ at lo ...		AVX2	~96%	7,71x	8	FMA	Float32
[loop in s452_ at lo ...	1 Data type conversions present	AVX2	~96%	7,71x	8	FMA; Type Con...	Float32
[loop in s413_ at lo ...	1 Ineffective peeled/remainder ...	AVX2	~96%	7,69x	4; 8	FMA	Float32
[loop in s273_ at lo ...	1 Possible inefficient memory a...	AVX2	~96%	7,69x	8	FMA; Masked St...	Float32
[loop in s279_ at lo ...	3 Possible inefficient memory a...	AVX2	~95%	7,56x	8	Blends; FMA	Float32
[loop in s253_ at lo ...	2 Possible inefficient memory a...	AVX2	~91%	7,30x	8	Blends; FMA	Float32
[loop in s251_ at lo ...		AVX2	~90%	7,23x	8	FMA	Float32
[loop in s271_ at lo ...	2 Possible inefficient memory a...	AVX2	~90%	7,16x	4; 8	FMA; Masked St...	Float32
[loop in vif_ at loop ...	1 Possible inefficient memory a...	AVX	~86%	6,90x	8	Blends	Float32
[loop in s274_ at lo ...	1 Possible inefficient memory a...	AVX2	~79%	6,29x	8	Blends; FMA; M...	Float32
[loop in SET2D at m ...		AVX	~73%	5,81x	8		Float32
[loop in std::_Fill<fl ...		AVX	~73%	5,81x	8		Float32
[loop in SET2D at m ...	1 Data type conversions present	AVX2	~66%	5,31x	8	Divisions; Type ...	Float32

Source | Top Down | Loop Analytics | Loop Assembly | Recommendations | Compiler Diagnostic Details

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 sour

Recommendation: Add data padding

The trip 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

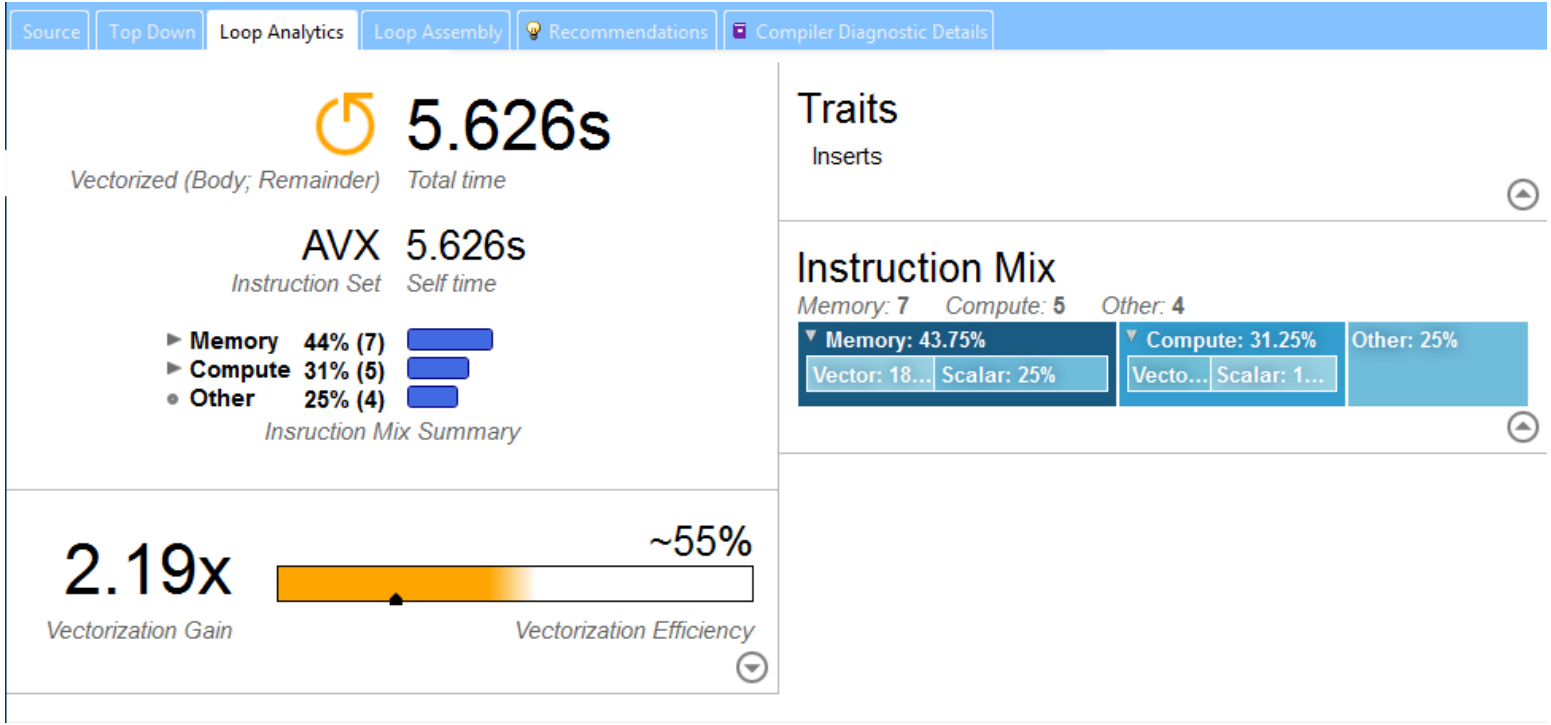
Optional: Specify the trip count, if it is not constant, using a directive: `#pragma loop_count`
Read More:

- qopt-assume-safe-padding, Qopt-assume-safe-padding; loop_count

- **Efficiency** – my performance thermometer
- **Recommendations** – get tip on how to improve performance
  - (also apply to scalar loops)

Loop Analytics

Get detailed information about your loops



DEPENDENCY ANALYSIS

Factors that **prevent Vectorizing** your code

1. Loop-carried dependencies

```
DO I = 1, N
   A(I + M) = A(I) + B(I)
ENDDO
```

M >= SIMDlength?

1.A Pointer aliasing (compiler-specific)

```
void scale(int *a, int *b)
{
   for (int i = 0; i < 1000; i++)
          b[i] = z * a[i];
}
```

2. Function calls (incl. indirect)

```
for (i = 1; i < nx; i++) {
  x = x0 + i * h;
  sumx = sumx + func(x, y, xp);
}
```

3. Loop structure, boundary condition

```
struct _x { int d; int bound; };

void doit(int *a, struct _x *x)
{
  for(int i = 0; i < x->bound; i++)
    a[i] = 0;
}
```

4 Outer vs. inner loops

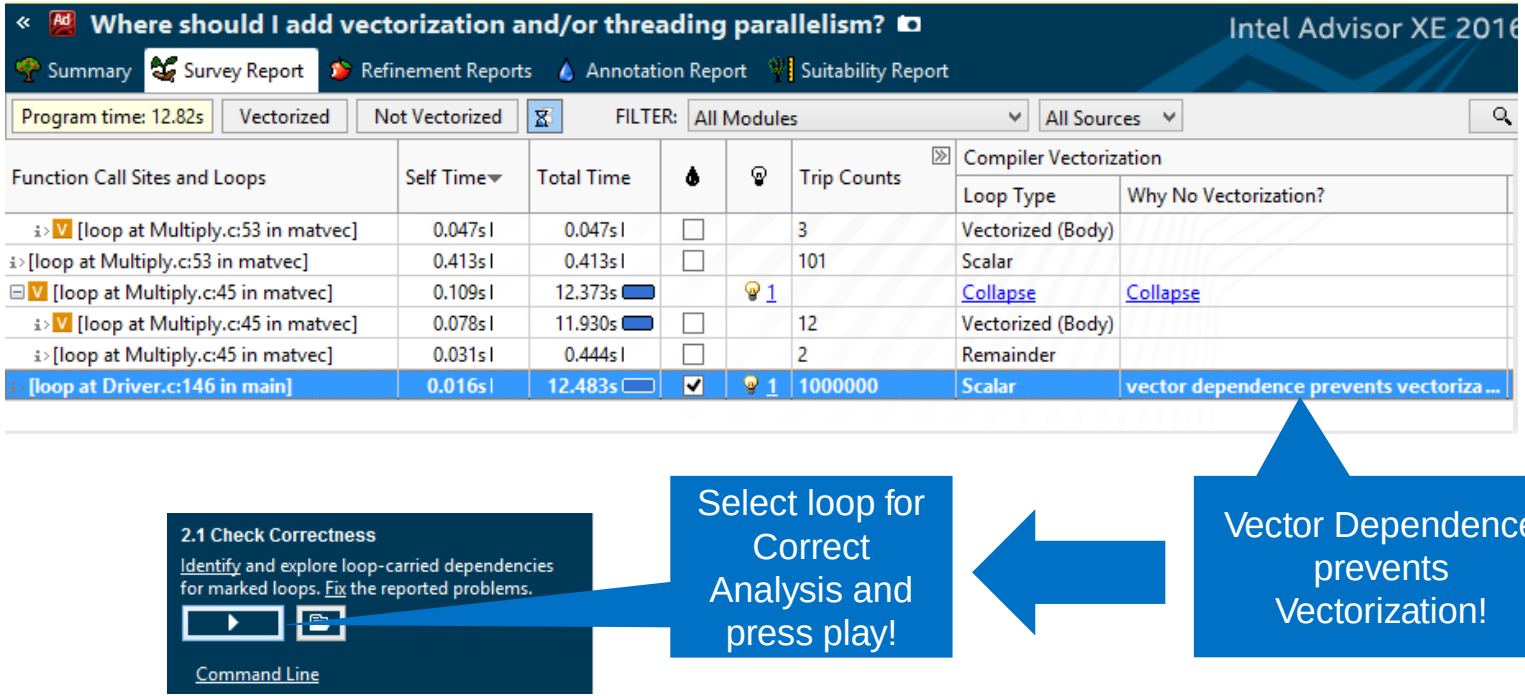
```
for(i = 0; i <= MAX; i++) {
  for(j = 0; j <= MAX; j++) {
    D[j][i] += 1;
  }
}
```

5. Cost-benefit (compiler specific..)

And others……

Is It Safe to Vectorize?

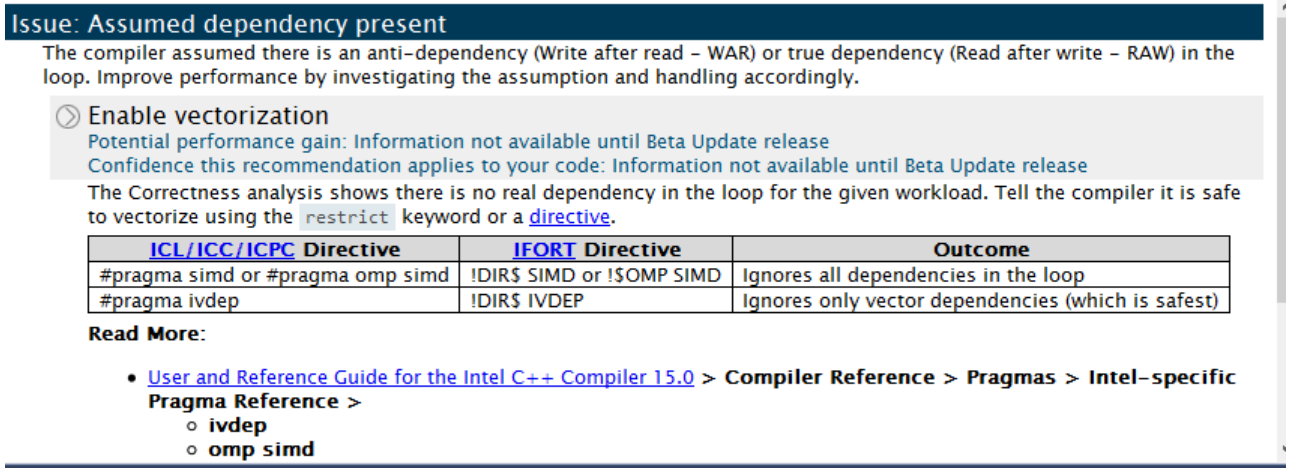
Loop-carried dependencies analysis verifies correctness



Data Dependencies – Tough Problem #1

Is it safe to force the compiler to vectorize?

```
for (i=0;i<N;i++)          // Loop carried dependencies!
    A[i] = A[i-M]*C[i]; // Need to check if it is safe to force
                           // the compiler to vectorize!
```



Issue: Assumed dependency present

The compiler assumed there is an anti-dependency (Write after read – WAR) or true dependency (Read after write – RAW) in the loop. Improve performance by investigating the assumption and handling accordingly.

Enable vectorization

Potential performance gain: Information not available until Beta Update release

Confidence this recommendation applies to your code: Information not available until Beta Update release

The Correctness analysis shows there is no real dependency in the loop for the given workload. Tell the compiler it is safe to vectorize using the `restrict` keyword or a directive.

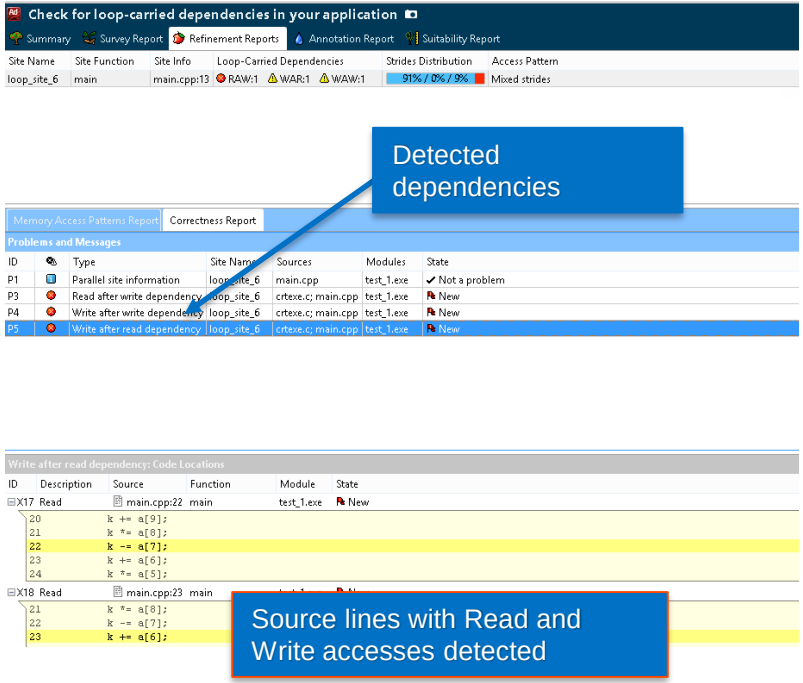
ICL/ICC/ICPC Directive	IFORT Directive	Outcome
#pragma simd or #pragma omp simd	!DIR$ SIMD or !$OMP SIMD	Ignores all dependencies in the loop
#pragma ivdep	!DIR$ IVDEP	Ignores only vector dependencies (which is safest)

Read More:

- User and Reference Guide for the Intel C++ Compiler 15.0 > **Compiler Reference > Pragmas > Intel-specific Pragma Reference >**
  - **ivdep**
  - **omp simd**

Correctness – Is It Safe to Vectorize?

Loop-carried dependencies analysis



Received recommendations to force vectorization of a loop:

1. Mark-up loop and check for REAL dependencies
2. Explore dependencies with code snippets

In this example 3 dependencies were detected:

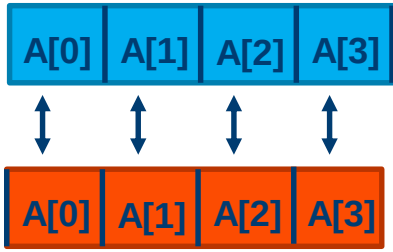
- RAW – Read After Write
- WAR – Write After Read
- WAW – Write After Write

This is NOT a good candidate to force vectorization!

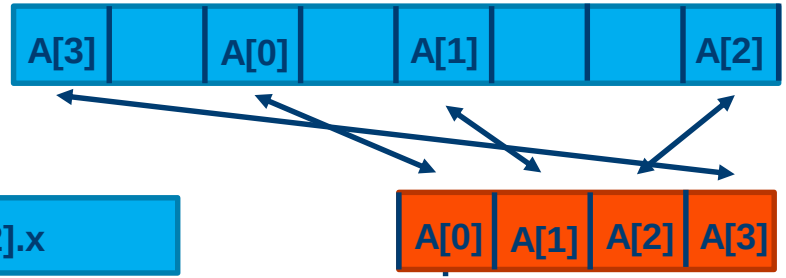
MEMORY ACCESS PATTERN ANALYSIS

Memory access patterns

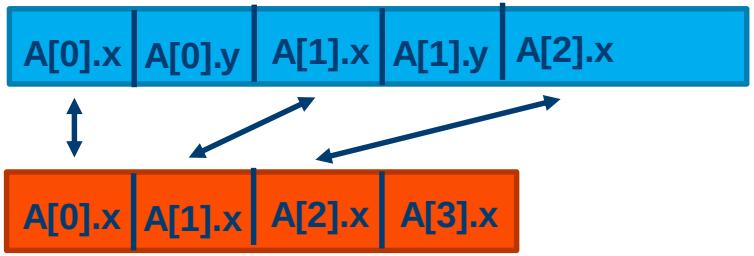
Unit strided (contiguous):



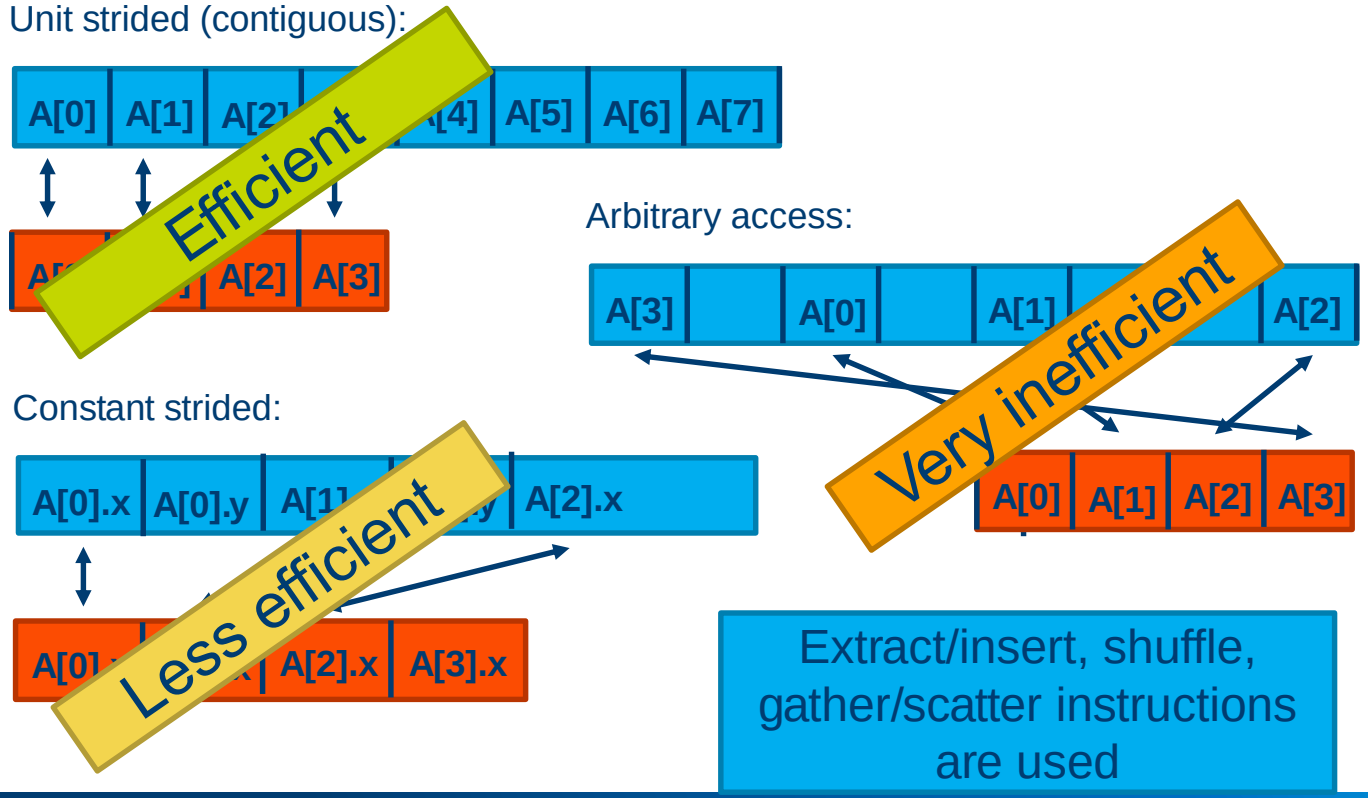
Arbitrary access:



Constant strided:

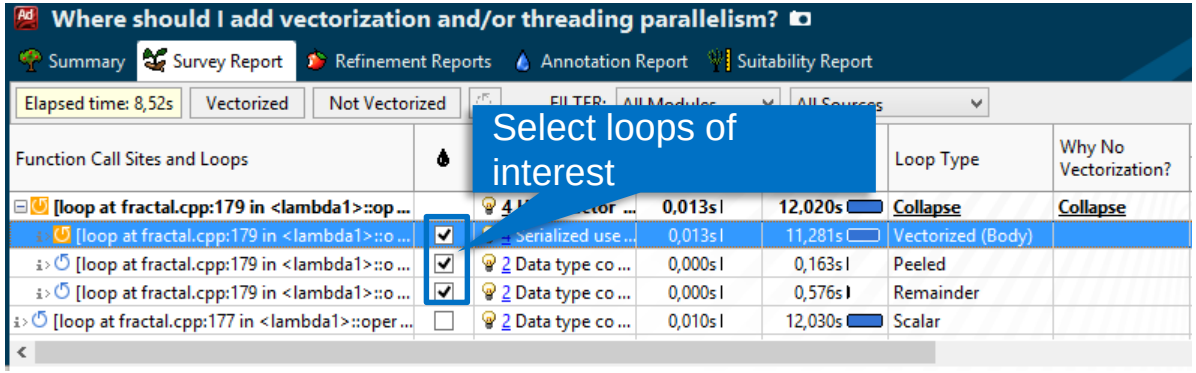


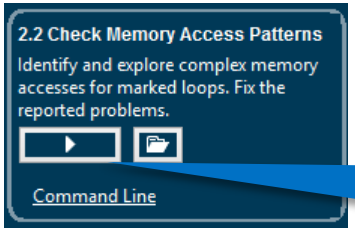
Memory access patterns



Improve Vectorization

Memory Access pattern analysis



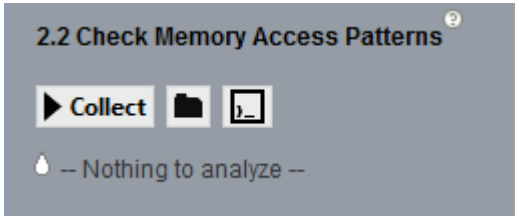


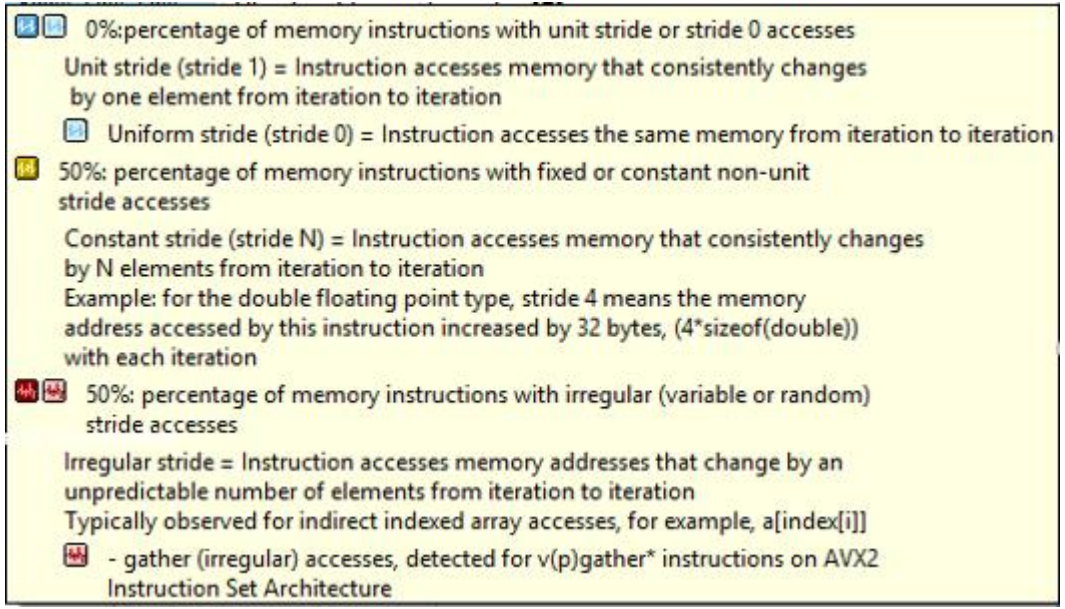
Run Memory Access Patterns analysis, just to check how memory is used in the loop and the called function

Irregular access patterns decreases performance!

Gather profiling

Run Memory Access Pattern Analysis





Enhanced Memory Access Analysis

Are you bandwidth or compute limited?

Measure Footprint

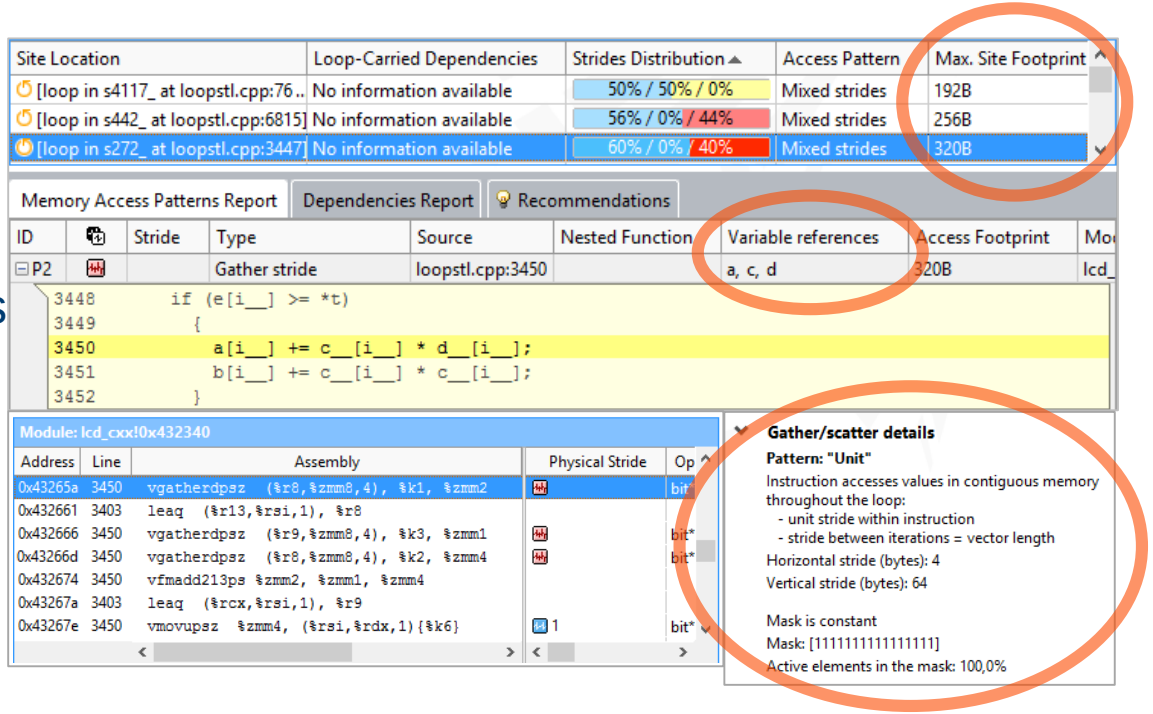
- Compare to cache size
  Does it fit in L2 cache?

Variable References

- Map data to variable names for easier analysis

Gather/Scatter

- Detect unneeded gather/scatters that reduce performance



References

Roofline model proposed by Williams, Waterman, Patterson:
http://www.eecs.berkeley.edu/~waterman/papers/roofline.pdf

"Cache-aware Roofline model: Upgrading the loft" (Ilic, Pratas, Sousa, INESC-ID/IST, Thec Uni of Lisbon)
http://www.inesc-id.pt/ficheiros/publicacoes/9068.pdf

Additional Material

Intel® Advisor – Threading Design & Prototyping:

- Product page – overview, features, FAQs, support…
- Training materials – movies, tech briefs, documentation…
- Evaluation guides – step by step walk through
- Reviews

Additional Analysis Tools:

- Intel® VTune Amplifier – performance profiler
- Intel® Inspector - memory and thread checker / debugger

Additional Development Products:

- Intel® Software Development Products

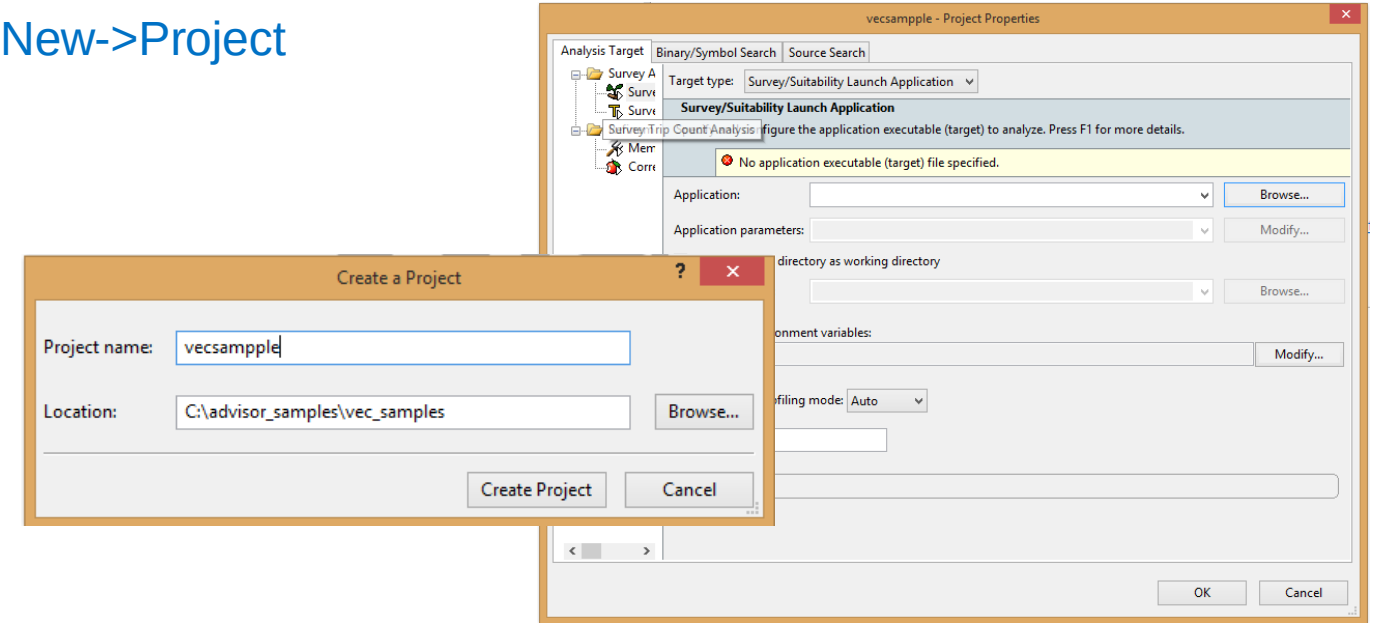
BACKUP

GETTING STARTED

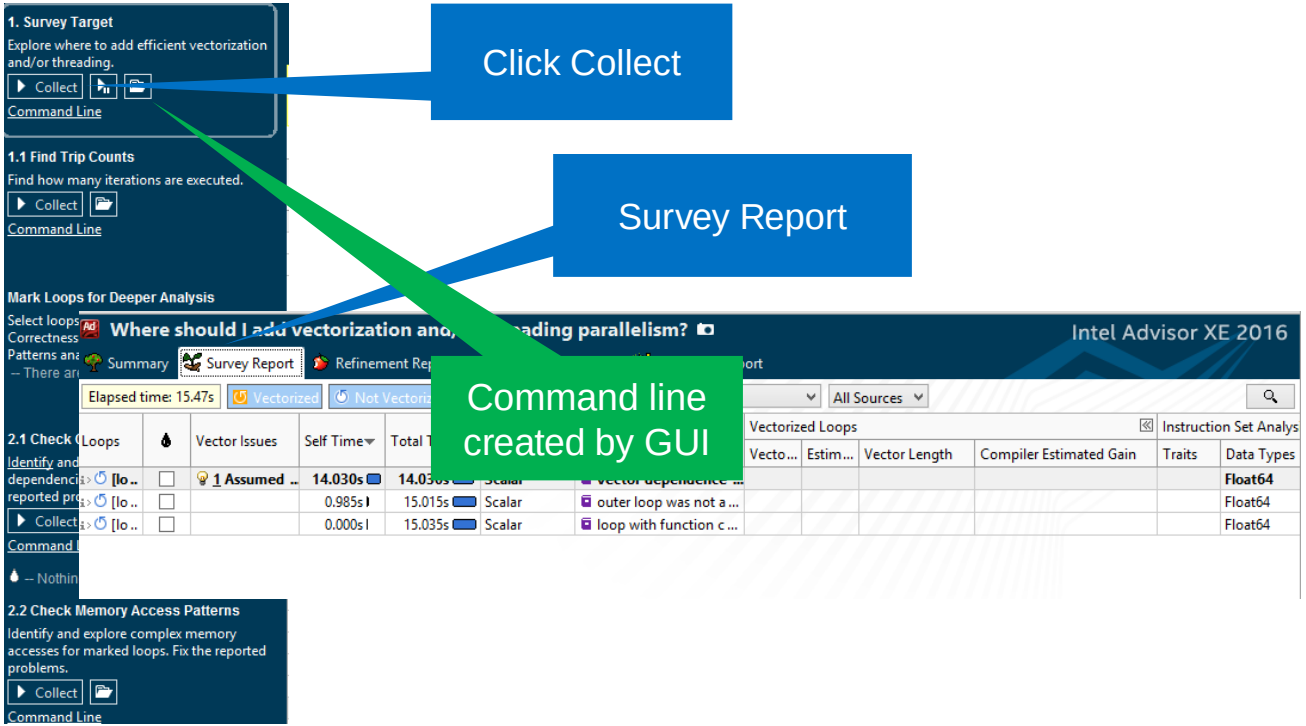
Before you analyze

Create Project

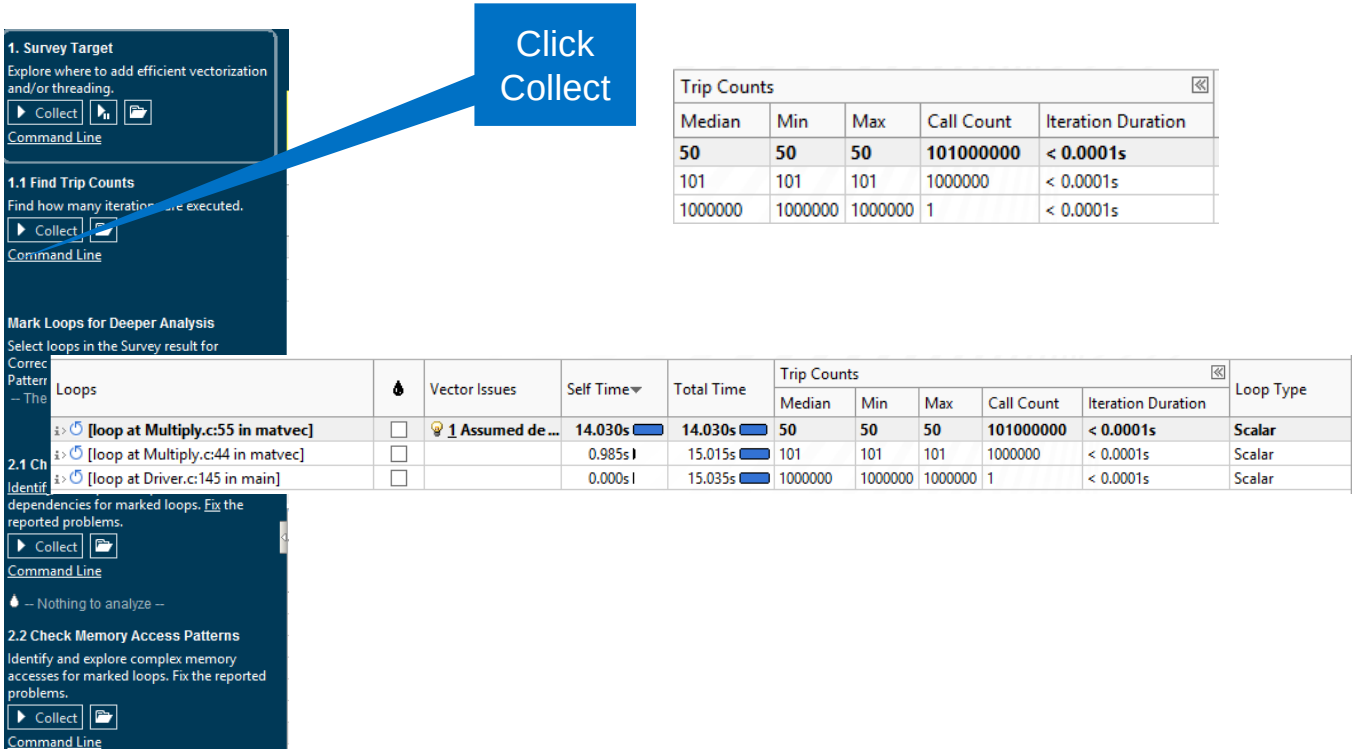
- File->New->Project



Analyze what loops you are spending your time in and how they have been vectorized!



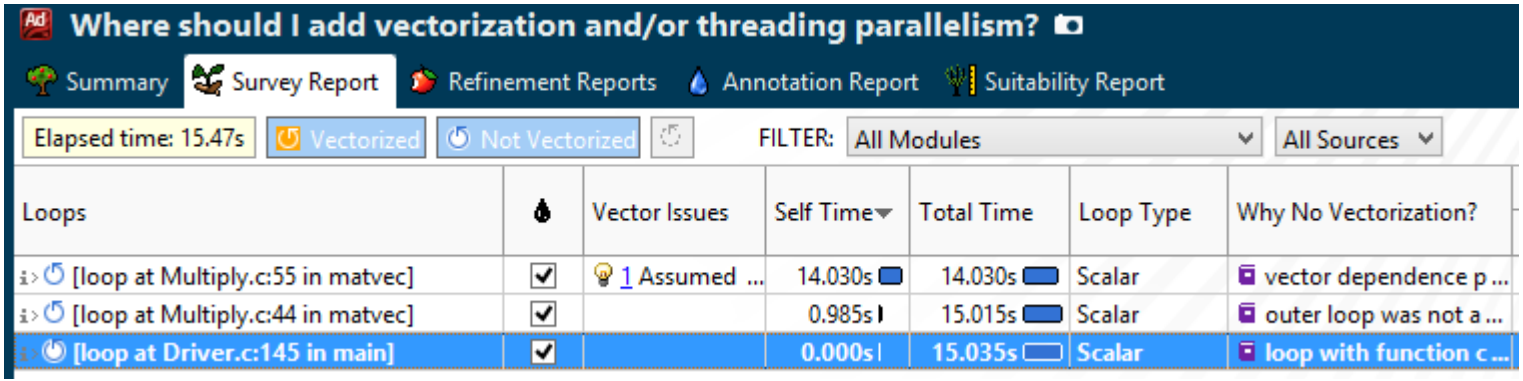
Next analyze how many times your loops are iterating and how many times they are called.



Trip Counts				
Median	Min	Max	Call Count	Iteration Duration
50	**50**	**50**	**101000000**	**< 0.0001s**
101	101	101	1000000	< 0.0001s
1000000	1000000	1000000	1	< 0.0001s

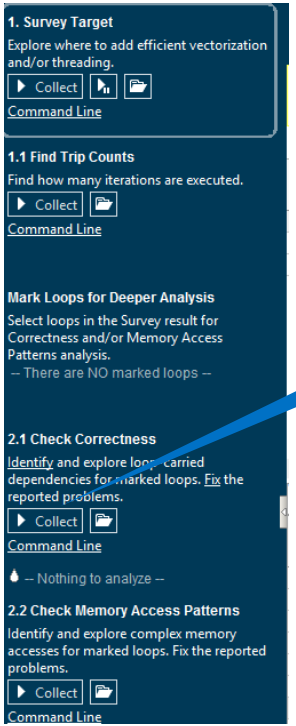
Loops		Vector Issues	Self Time	Total Time	Trip Counts					Loop Type
					Median	Min	Max	Call Count	Iteration Duration	
[loop at Multiply.c:55 in matvec]		**1 Assumed de ...**	**14.030s**	**14.030s**	**50**	**50**	**50**	**101000000**	**< 0.0001s**	**Scalar**
[loop at Multiply.c:44 in matvec]			0.985s	15.015s	101	101	101	1000000	< 0.0001s	Scalar
[loop at Driver.c:145 in main]			0.000s	15.035s	1000000	1000000	1000000	1	< 0.0001s	Scalar

Specify loops for deeper analysis



Deeper analysis

Check dependencies



Click Collect

We marked 3 loops for a dependency analysis. Two of the loops had no dependencies. One of the loops has Read-After-Write dependency and can't be vectorized.

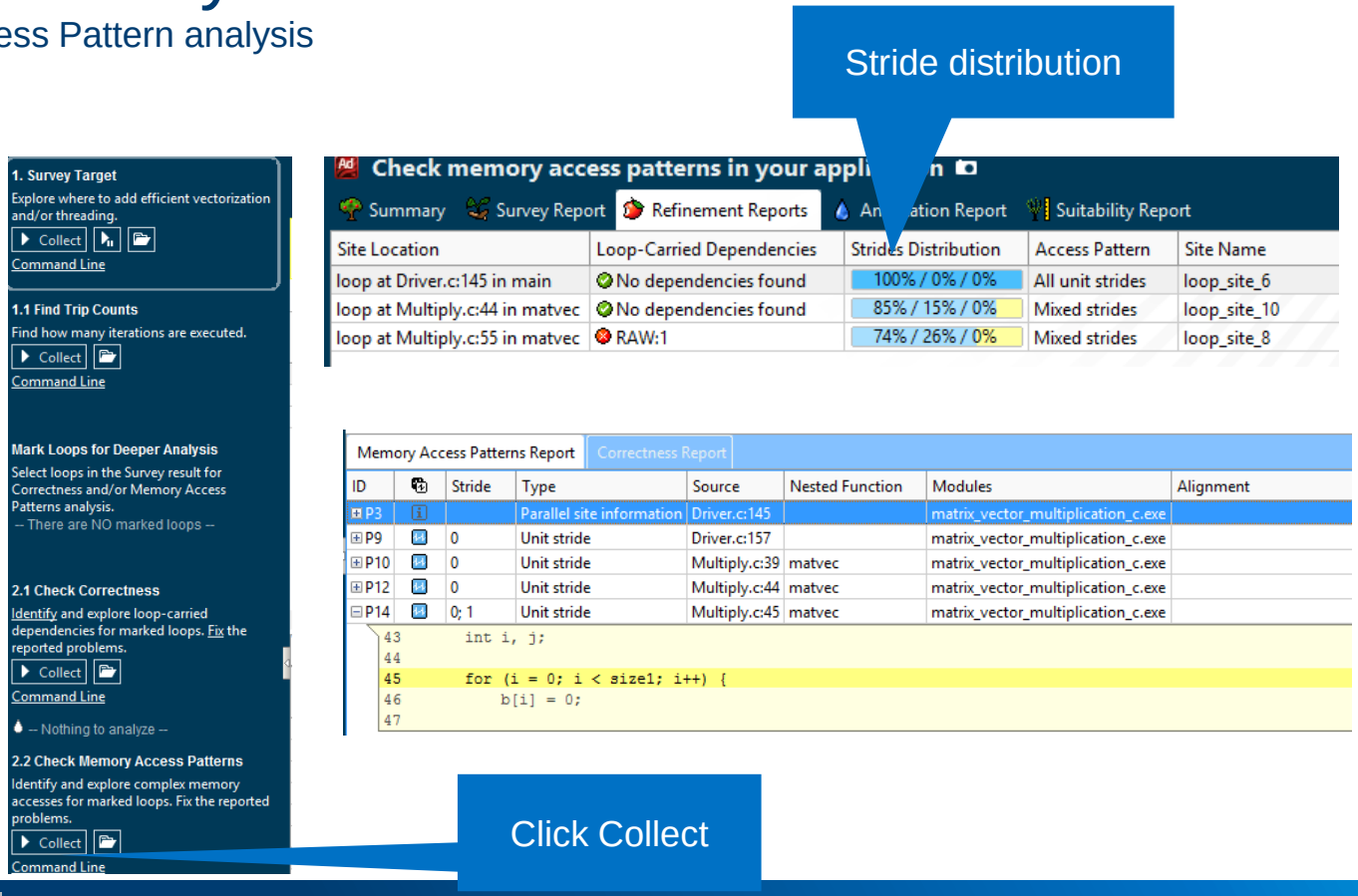
Check memory access patte… in your application

Summary | Survey Report | Refinement Reports | Annotation Report | Suitability Report

Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern	Site Name
loop at Driver.c:145 in main	No dependencies found	No information available	No information available	loop_site_6
loop at Multiply.c:44 in matvec	No dependencies found	No information available	No information available	loop_site_10
loop at Multiply.c:55 in matvec	RAW:1	No information available	No information available	loop_site_8

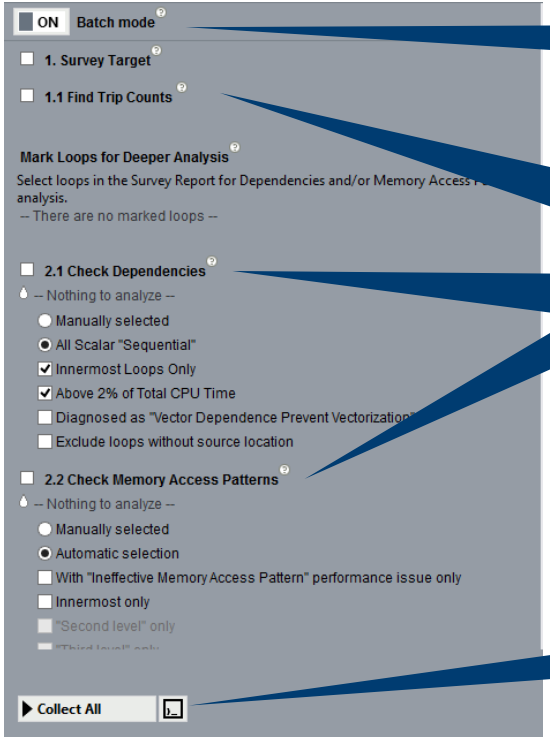
Deeper analysis

Memory Access Pattern analysis



Batch Mode Workflow Saves Time

Intel® Advisor - Vectorization Advisor



Turn On Batch Mode

Select analyses to run

Click Collect all

Run several analyses in batch as a single run

Contains pre-selected criteria for advanced analyses

Command Line: Intel® Advisor XE

Collecting survey and tripcounts

advixe-cl –collect survey –project-dir ./advi -- mult.exe

advixe-cl –collect tripcounts –project-dir ./advi -- mult.exe

Creating snapshot in command line, e.g:

```
advixe-cl --snapshot --project-dir ./advi \
          --pack --cache-sources --cache-binaries -- /tmp/new_snapshot
```

Viewing the results

advixe-gui ./advi

advixe-cl –report survey –project-dir ./advi

Advisor works with GCC and Microsoft Compilers

Adds bonus capabilities with the Intel Compiler

Advisor using GCC, Microsoft or Intel Compiler:

- Finds un-vectorized loops
- Analyze SIMD, AVX, AVX2, AVX-512
- Dependency Analysis – safely force vectorization with a pragma
- Memory Access Pattern Analysis - optimize stride and caching
- Trip Counts
- FLOPS metrics with masking
- Roofline Analysis – balance memory vs. compute optimization

Intel Compiler Adds:

- Usually better optimized vectorization
- Better compiler optimization messages

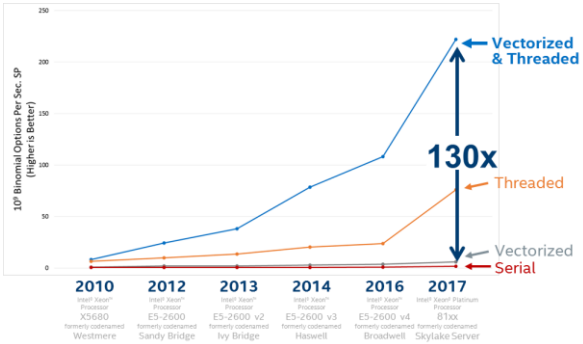
Intel Advisor with Intel Compiler Adds:

- Finds inefficiently vectorized loops and estimates performance gain
- Compiler optimization report messages displayed on the source
- More tips for improving vectorization
- Optimize for AVX-512 even without AVX-512 hardware

Configurations for 2010-2017 Benchmarks

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

Performance measured in Intel Labs by Intel employees



Platform Hardware and Software Configuration

	Platform	Unscaled Core Frequency	Cores/ Socket	Num Sockets	L1 Data Cache	L2 Cache	L3 Cache	Memory	Memory Frequency	Memory Access	H/W Prefetchers Enabled	HT Enabled	Turbo Enabled	C States	O/S Name	Operating System	Compiler Version
WSM	Intel® Xeon™ X5680 Processor	3.33 GHZ	6	2	32K	256K	12 MB	48 MB	1333 MHz	NUMA	Y	Y	Y	Disabled	Fedora 20	3.11.10-301.fc20	icc version 17.0.2
SNB	Intel® Xeon™ E5 2690 Processor	2.9 GHZ	8	2	32K	256K	20 MB	64 GB	1600 MHz	NUMA	Y	Y	Y	Disabled	Fedora 20	3.11.10-301.fc20	icc version 17.0.2
IVB	Intel® Xeon™ E5 2697v2 Processor	2.7 GHZ	12	2	32K	256K	30 MB	64 GB	1867 MHz	NUMA	Y	Y	Y	Disabled	RHEL 7.1	3.10.0-229.el7.x86_64	icc version 17.0.2
HSW	Intel® Xeon™ E5 2600v3 Processor	2.2 GHz	18	2	32K	256K	46 MB	128 GB	2133 MHz	NUMA	Y	Y	Y	Disabled	Fedora 20	3.15.10-200.fc20.x86_64	icc version 17.0.2
BDW	Intel® Xeon™ E5 2600v4 Processor	2.3 GHz	18	2	32K	256K	46 MB	256 GB	2400 MHz	NUMA	Y	Y	Y	Disabled	RHEL 7.0	3.10.0-123. el7.x86_64	icc version 17.0.2
BDW	Intel® Xeon™ E5 2600v4 Processor	2.2 GHz	22	2	32K	256K	56 MB	128 GB	2133 MHz	NUMA	Y	Y	Y	Disabled	CentOS 7.2	3.10.0-327. el7.x86_64	icc version 17.0.2
SKX	Intel® Xeon® Platinum 81xx Processor	2.5 GHz	28	2	32K	1024K	40 MB	192 GB	2666 MHz	NUMA	Y	Y	Y	Disabled	CentOS 7.3	3.10.0-514.10.2.el7.x86_64	icc version 17.0.2

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. For more information go to http://www.intel.com/performance

Legal Disclaimer & Optimization Notice

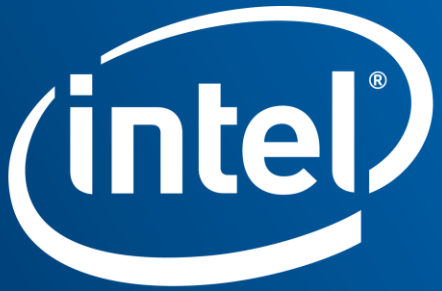
Benchmark results were obtained prior to implementation of recent software patches and firmware updates intended to address exploits referred to as "Spectre" and "Meltdown". Implementation of these updates may make these results inapplicable to your device or system.

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. For more complete information visit www.intel.com/benchmarks.

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.

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



Software