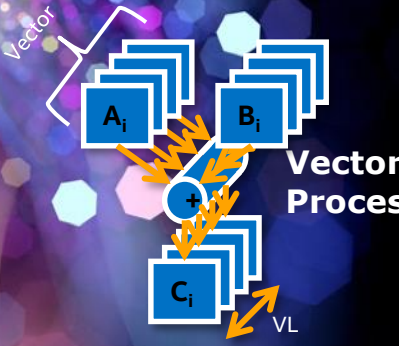

intel
experience
what's inside™
OPENMP MADE EASY
WITH INTEL® ADVISOR
Zakhar Matveev, PhD, Intel CVCG,
November 2018, SC'18 OpenMP booth

intel®
experience
what's inside™

WHY DO WE CARE?



Motivation (instead of Agenda)

- Starting from **4.x**, **OpenMP** introduces support for both levels of parallelism:
  - Multi-Core (think of “pragma/directive omp parallel for”)
  - SIMD (think of “pragma/directive omp simd”)
  - 2 pillars of **OpenMP SIMD** programming model
- **Hardware** with Intel ® AVX-512 support gives you *theoretically* 8x speed-up over SSE baseline (less or even **more** in practice)
- **Intel® Advisor** is here to assist you in:
  - **Enabling SIMD** parallelism with **OpenMP** (if not yet)
  - **Improving** performance of **already** vectorized OpenMP SIMD code
  - And will also help to optimize for **Memory** Sub-system (Advisor **Roofline**)

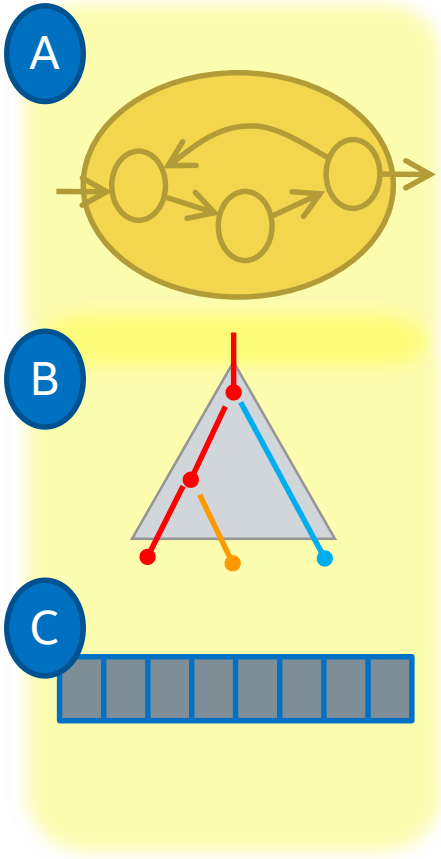
Don’t use a single Vector lane!

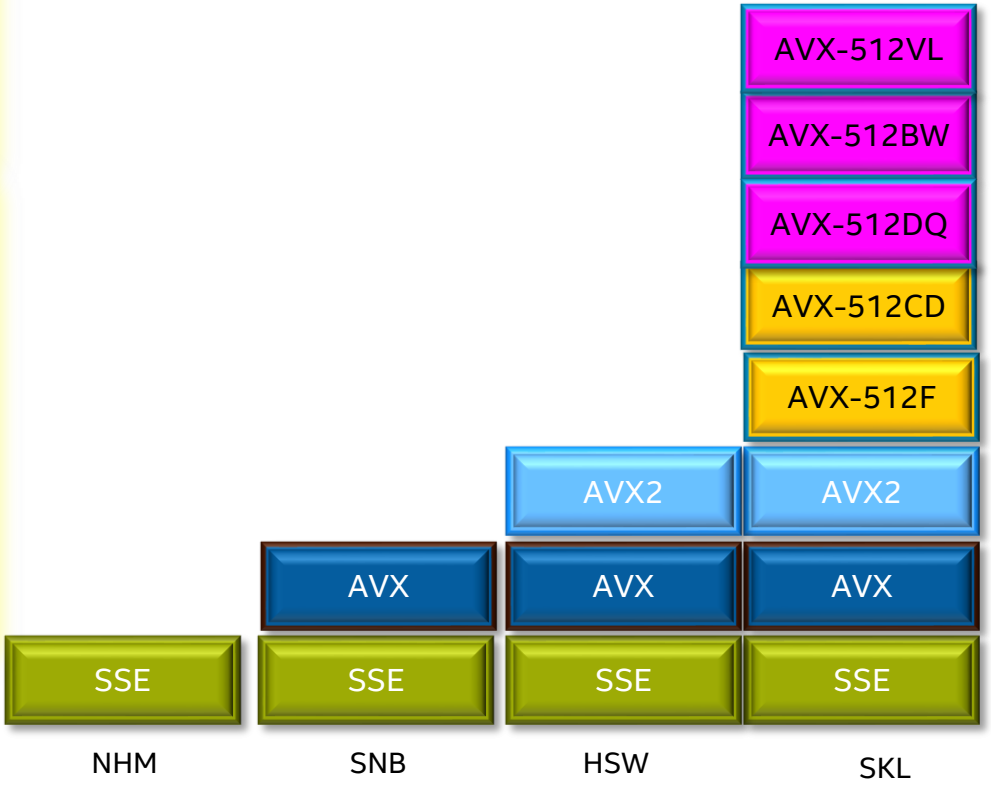
Un-vectorized and un-threaded software will under perform

Permission to Design for All Lanes

Threading and Vectorization needed to fully utilize modern hardware

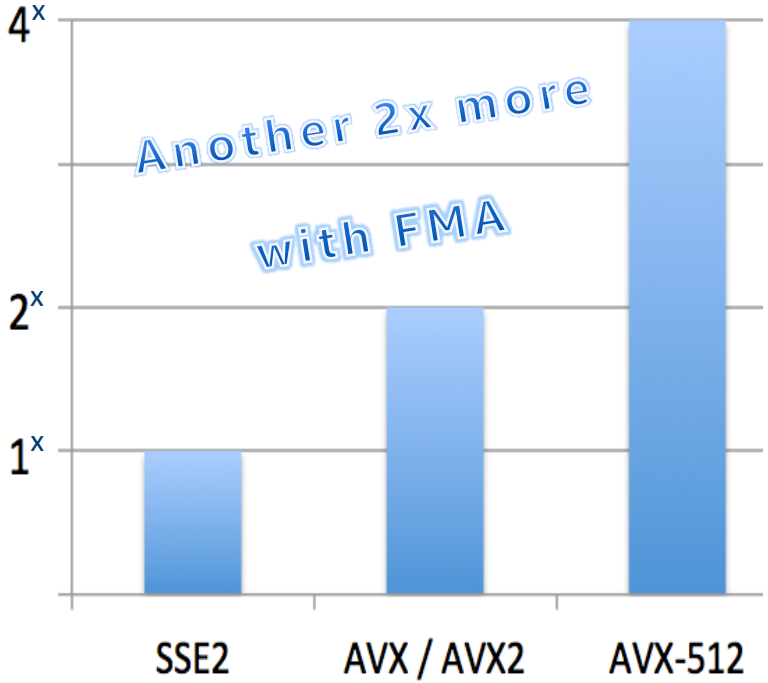
Vector parallelism in x86





Intel® microarchitecture code name …

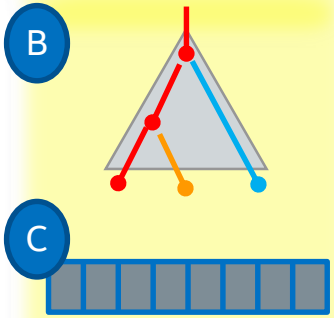
(theoretically) 8x more SIMD FLOP/S compared to your (–O2) optimized baseline

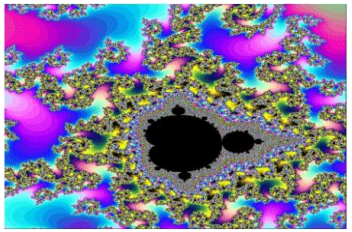


- Significant leap to 512-bit SIMD support for processors
- Intel® Compilers and Intel® Math Kernel Library include AVX-512 support
- Strong compatibility with AVX
- Added EVEX prefix enables additional functionality

Don’t leave it on the table!

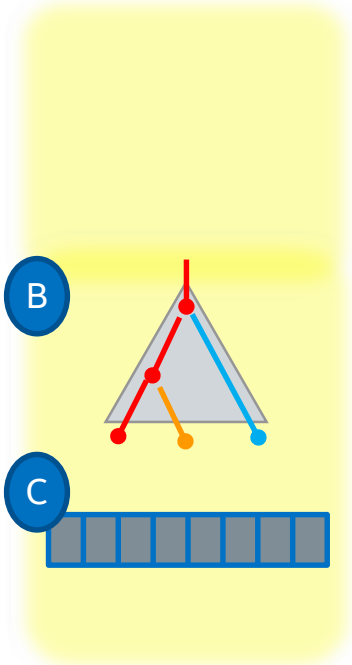
Two level parallelism decomposition with **OpenMP**: image processing example

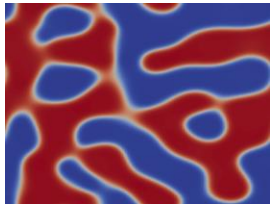




```
#pragma omp parallel for
for (int y = 0; y < ImageHeight; ++y){
#pragma omp simd
   for (int x = 0; x < ImageWidth; ++x){
      count[y][x] = mandel(in_vals[y][x]);
   }
}
```

Two level parallelism decomposition with **OpenMP**: fluid dynamics processing example





```
#pragma omp parallel for
for (int i = 0; i < X_Dim; ++i){
#pragma omp simd
   for (int m = 0; x < n_velocities; ++m){
      next_i = f(i, velocities(m));
      X[i] = next_i;
   }
}
```

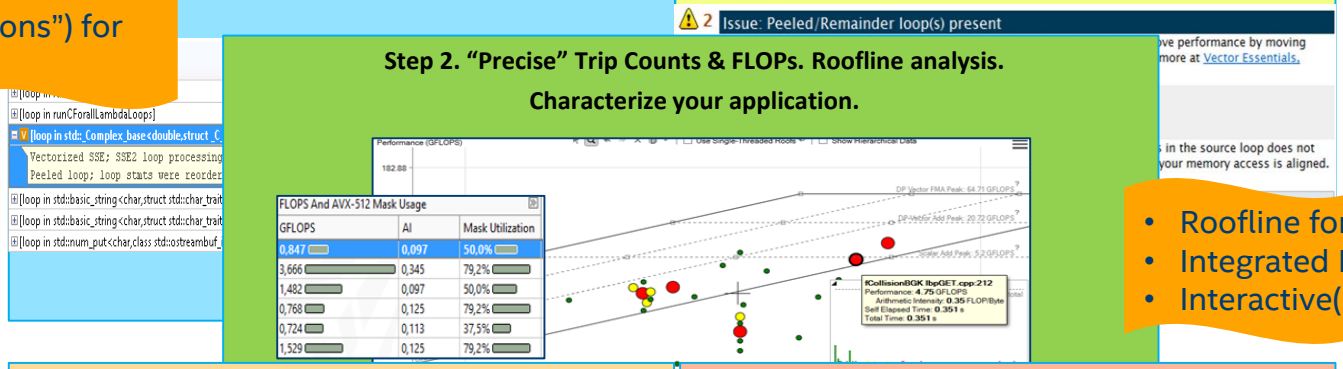
Key components of Intel® Advisor

What's new in "2019" release

More Advise ("Recommendations") for OpenMP SIMD

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

Guidance: detect problem and recommend how to fix it

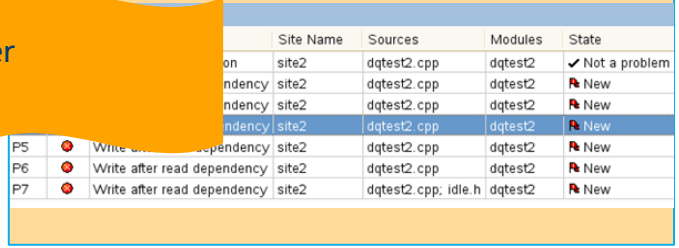


Step 2. "Precise" Trip Counts & FLOPs. Roofline analysis.

Characterize your application.

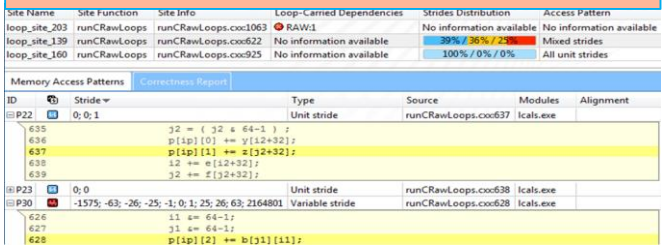
- Roofline for INT OP/S
- Integrated Roofline (exp)
- Interactive(!) HTML export

Step 3. Loop-Carried Dependency Analysis



- MAC OS viewer
- Python API..

Step 4. Memory Access Patterns Analysis



Faster Code Faster Using Intel® Advisor

Vectorization

"**Intel® Advisor's Vectorization Advisor** permitted me to focus my work where it really mattered. When you have only a limited amount of time to spend on optimization, it is invaluable."

Gilles Civario
Senior Software Architect
Irish Centre for High-End Computing

"**Intel® Advisor's Vectorization Advisor** fills a gap in code performance analysis. It can guide the informed user to better exploit the vector capabilities of modern processors and coprocessors."

Dr. Luigi Iapichino
Scientific Computing Expert
Leibniz Supercomputing Centre

Threading

"**Intel® Advisor** has been extremely helpful in identifying the best pieces of code for parallelization. We can save several days of manual work by targeting the right loops and we can use Advisor to find potential thread safety issues to help avoid problems later on."

Carlos Boneti
HPC software engineer,
Schlumberger

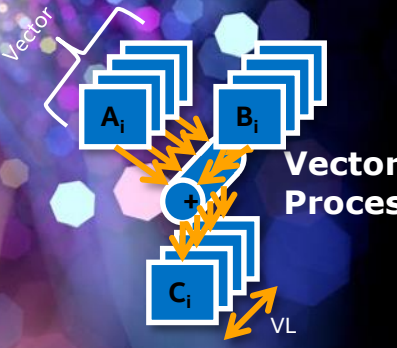
"**Intel® Advisor** has allowed us to quickly prototype ideas for parallelism, saving developer time and effort, and has already been used to highlight subtle parallel correctness issues in complex multi-file, multi-function algorithms."

Simon Hammond
Senior Technical Staff
Sandia National Laboratories

More Case Studies

intel experience what's inside™

TWO PILLARS OF OPENMP SIMD



Pillar 1: SIMD Pragma Notation

OpenMP 4.0: #pragma omp simd [clause [,clause] …]

- **Targets loops**
  - Can target inner or outer canonical loops
- **Developer asserts loop is suitable for SIMD**
  - The Intel Compiler will vectorize if possible (will ignore dependency or efficiency concerns)
  - Use when you **KNOW** that a given loop is safe to vectorize
  - Can choose from lexicon of clauses to modify behavior of SIMD directive
- **Developer should validate results (correctness)**
  - Just like for race conditions in OpenMP* threading loops
- **Minimizes source code changes needed to enforce vectorization**

OMP SIMD Pragma Clauses

`reduction(operator:v1, v2, …)`

- v1 etc are reduction variables for operation "operator"
- Examples include computing averages or sums of arrays into a single scalar value : *reduction (+:sum)*

`linear(v1:step1, v2:step2, …)`

- declares one or more list items to be private to a SIMD lane and to have a linear relationship with respect to the iteration space of a loop : *linear (i:2)*

`safelen (length)`

- no two iterations executed concurrently with SIMD instructions can have a greater distance in the logical iteration space than this value
- *Typical values are* 2, 4, 8, 16

`simdlen(length)`

OMP SIMD Pragma Clauses cont…

`aligned(v1:alignment, v2:alignment)`

- declares that the object to which each list item points is aligned to the number of bytes expressed in the optional parameter of the aligned clause.

`collapse(number of loops)`

- Nested loop iterations are collapsed into one loop with a larger iteration space.

`private(v1, v2, …), lastprivate (v1, v2, …)`

- declares one or more list items to be private to an implicit task or to a SIMD lane, lastprivate causes the corresponding original list item to be updated after the end of the region..

Pillar 2: SIMD-enabled functions

Write a function for one element and add **pragma** as follows

```
#pragma omp declare simd
float foo(float a, float b, float c, float d)
{
  return a * b + c * d;
}
```

Call the scalar version:

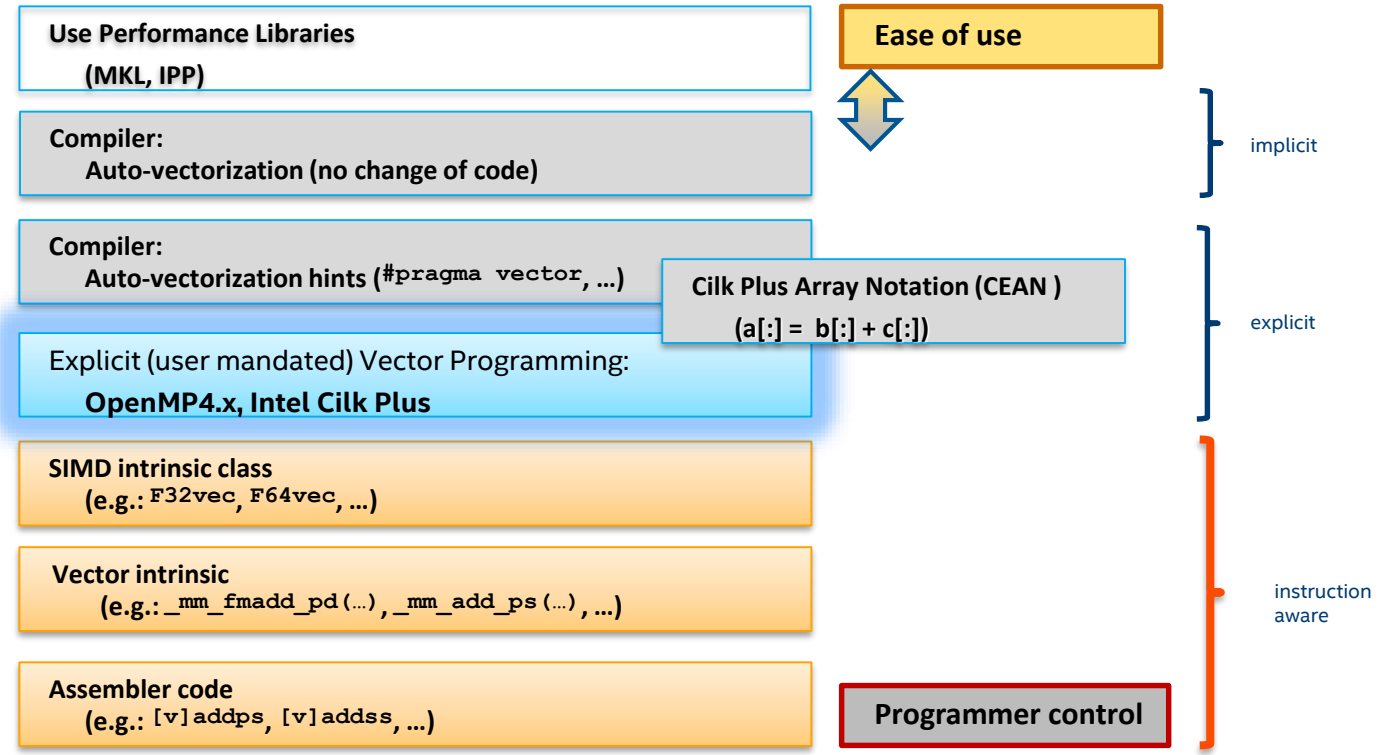
```
e = foo(a, b, c, d);
```

Call vector version via SIMD loop:

```
#pragma omp simd
for(i = 0; i < n; i++) {
  A[i] = foo(B[i], C[i], D[i], E[i]);
}
```

```
A[:] = foo(B[:], C[:], D[:], E[:]);
```

Many Ways to Vectorize



VECTORIZATION ANALYSIS WITH INTEL® ADVISOR

Factors that **prevent Vectorizing** your code

1. Loop-carried dependencies

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

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……

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

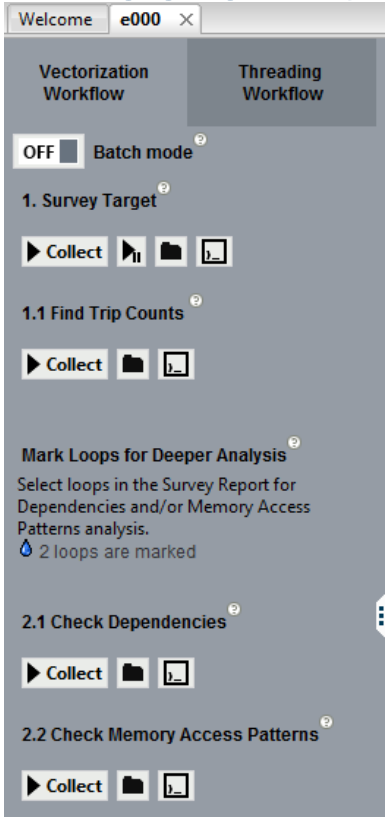
```
void doit(int *a, int *b, int
unknown_small_value)
{
  for(int i = 0; i <
unknown_small_value; i++)
      a[i] = z*b[i];
}
```

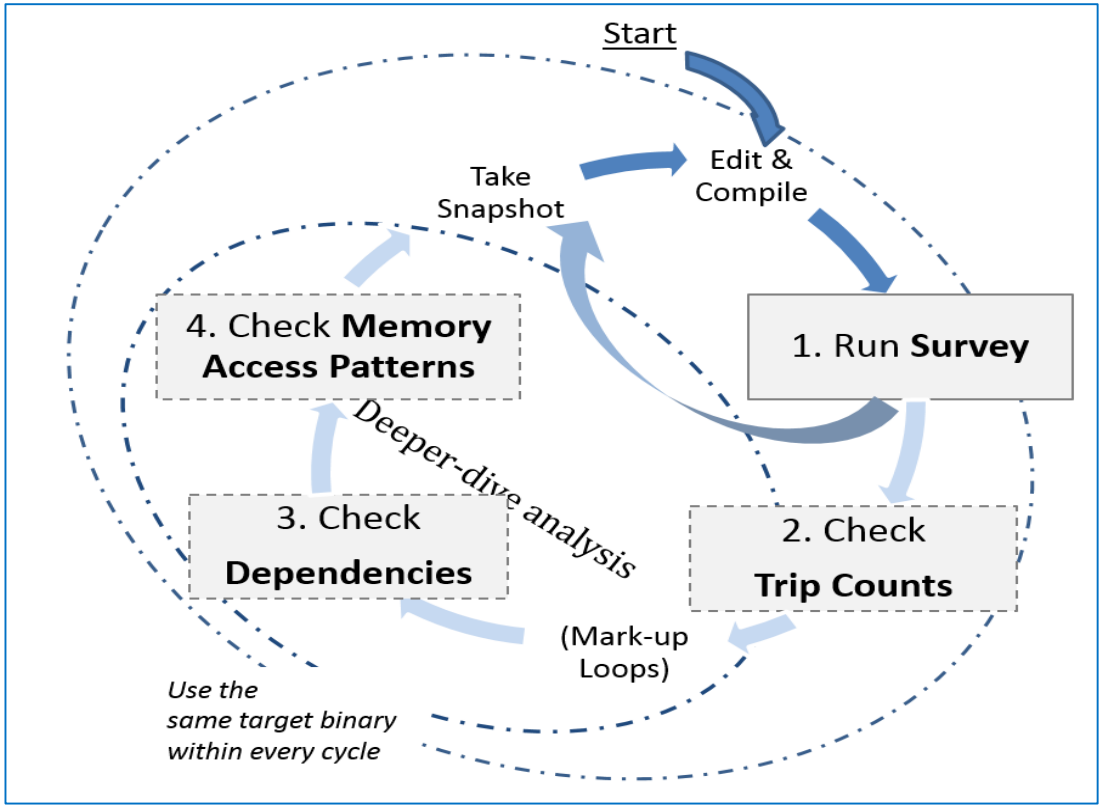
4. Branchy codes, *outer vs. inner loops*

```
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, even AVX throttling..

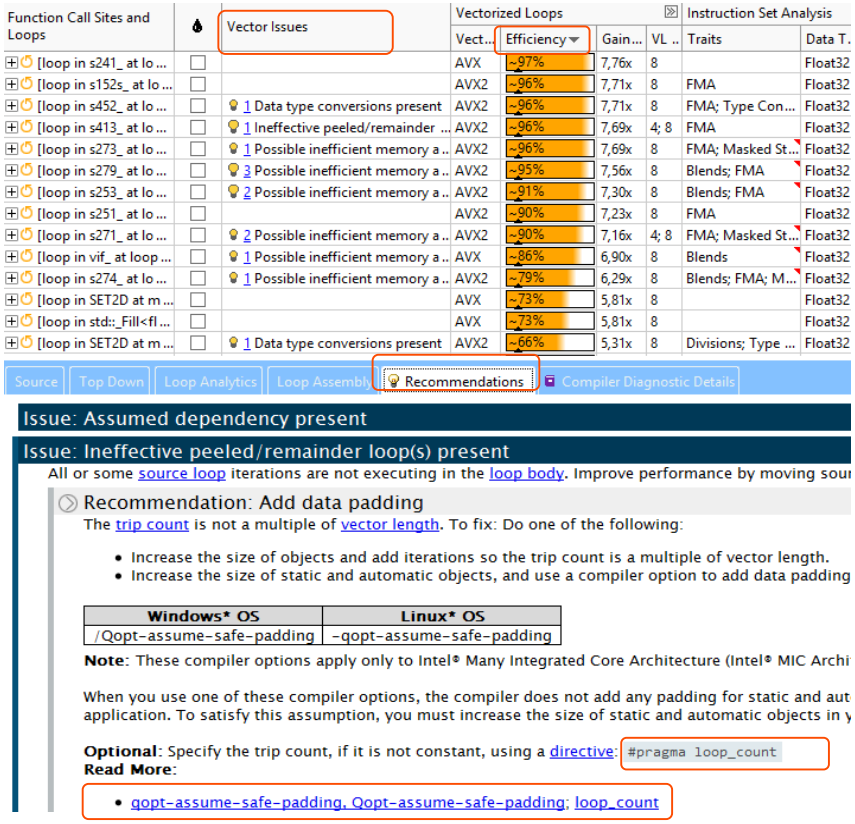
Vectorization Analysis Workflow





Intel® Advisor Survey

Why no vectorization? How to improve AVX performance?



Function Call Sites and Loops		Vector Issues	Vectorized Loops				Instruction Set Analysis	
			Vect...	Efficiency	Gain...	VL ..	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

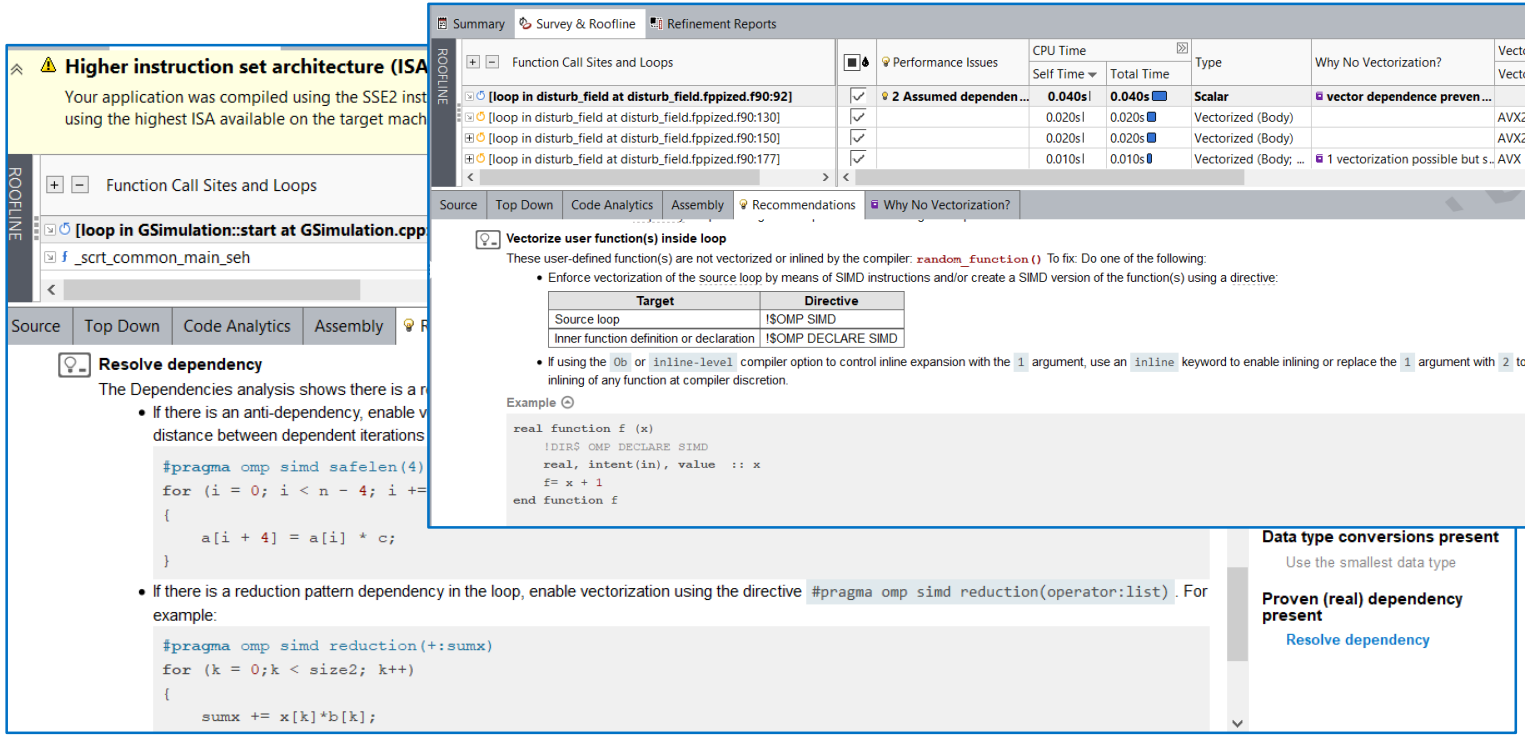
When you use one of these compiler options, the compiler does not add any padding for static and aut application. To satisfy this assumption, you must increase the size of static and automatic objects in y

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)

Actionable advice: introducing OpenMP SIMD



intel® experience what's inside™

INTEL® ADVISOR "MAP" AND DEPENDENCIES ANALYSIS

Vector Processing

Memory access pattern analysis

How should I access data ?

Unit stride access are faster

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

Constant stride are usually worse

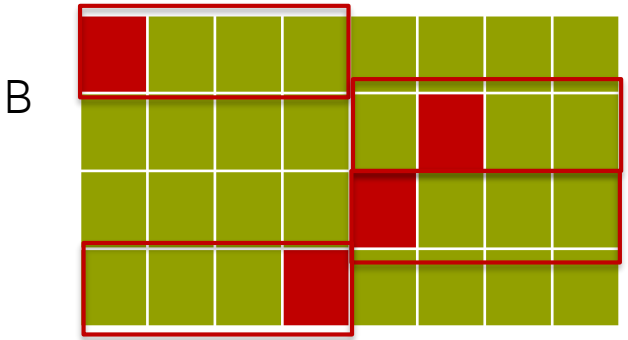
```
for (i=0; i<N; i+=2)
    A[i] = B[i]*d
```

Non predictable access are usually bad

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

For B, 1 cache line load computes 4 DP

For B, 2 cache line loads compute 4 DP with reconstructions



For B, 4 cache line loads compute 4 DP with reconstructions, prefetching might not work

Intel ® Advisor Memory Access Pattern (MAP)

know your access pattern

Unit-Stride access

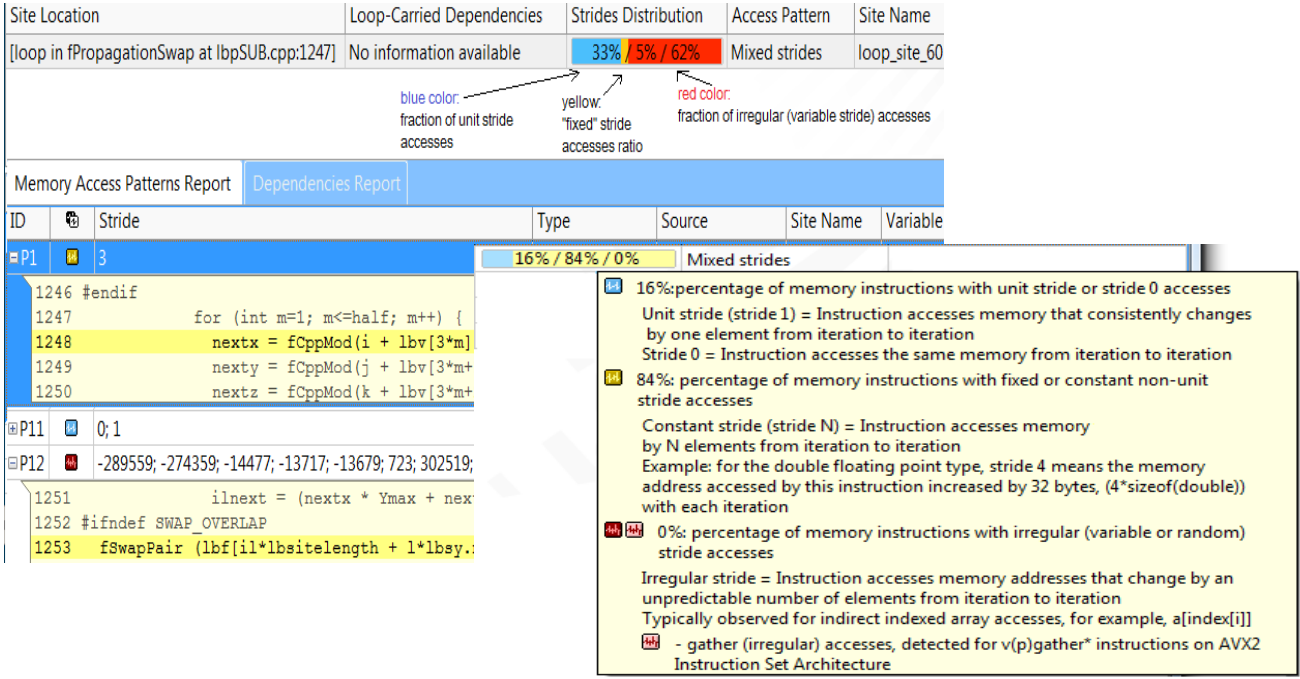
```
for (i=0; i<N; i++)
A[i] = C[i]*D[i]
```

Constant stride access

```
for (i=0; i<N; i++)
point[i].x = x[i]
```

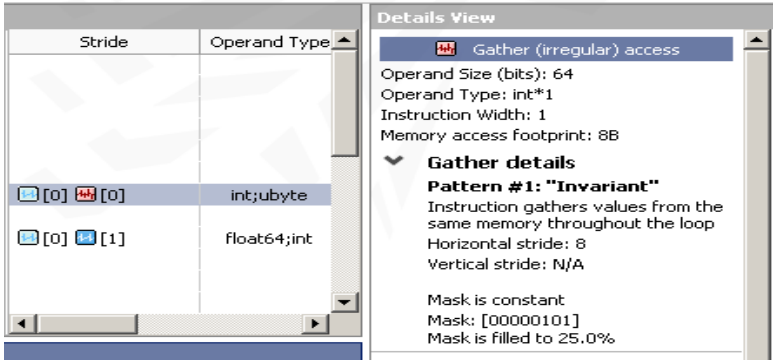
Variable stride access

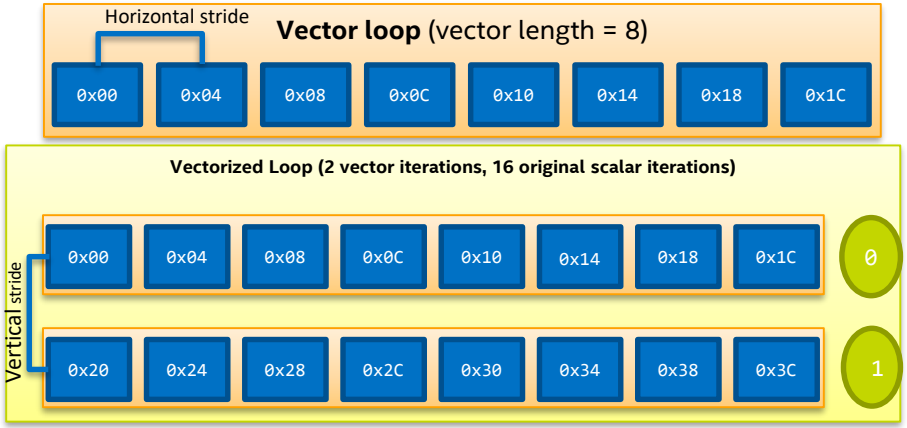
```
for (i=0; i<N; i++)
A[B[i]] = C[i]*D[i]
```



Gather/Scatter Analysis

Advisor MAP detects gather “offset patterns”.

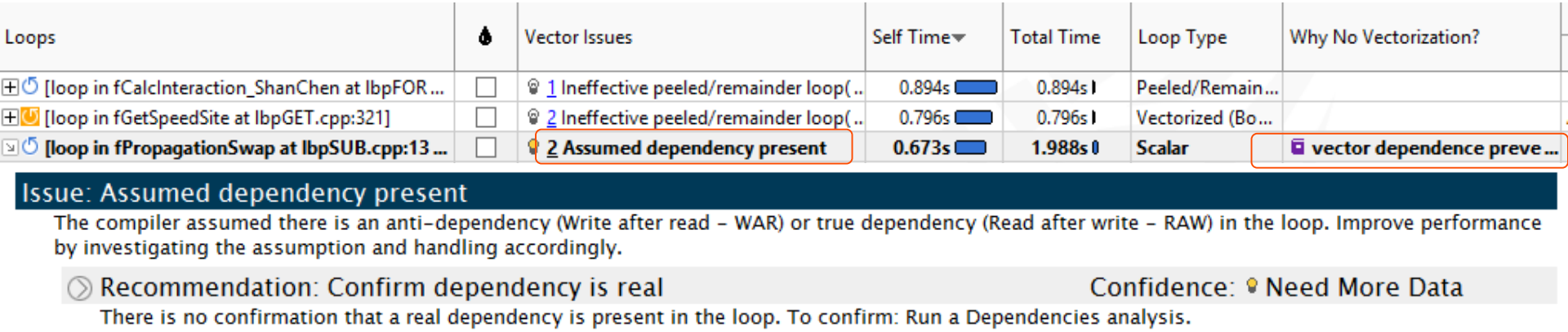




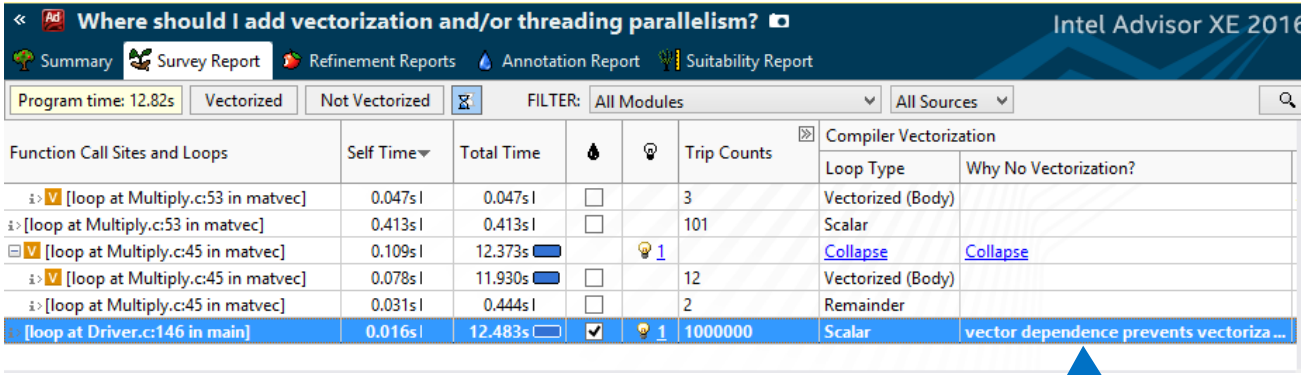
Pattern #	Pattern Name	Horizontal Stride Value	Vertical Stride Value	Example of Corresponding *Fix(es)*
1	Invariant	0	0	OpenMP uniform clause, simd pragma/directive, refactoring
2	Uniform (horizontal invariant)	0	Arbitrary	OpenMP uniform clause, simd pragma/directive
3	Vertical Invariant	Constant	0	OpenMP private clause, simd pragma/directive
4	Unit	1 or -1	|Vertical Stride| = Vector Length	OpenMP linear clause, simd pragma/directive
5	Constant	Constant = X	Constant = X*VectorLength	Subject for AoS -> SoA transformation

Enable vectorization of your scalar code: Assumed Dependency

- More data is needed to confirm if the loop can be vectorized
- Select the given loop and run dependency analysis to see if it can be vectorized using an OpenMP* 4.0 simd pragma.



Check if it is safe to vectorize : Advisor Dependencies



2.1 Check Dependencies

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

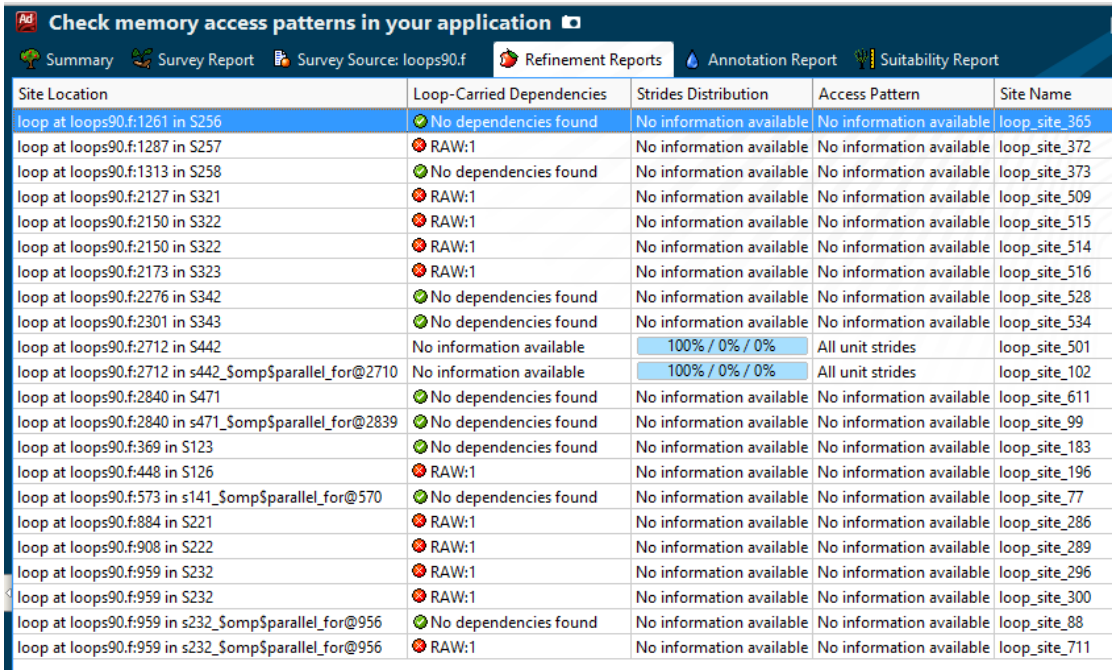
Collect

Command Line

Select loop for Dependency Analysis and press play!

Vector Dependence prevents Vectorization!

Almost half loops checked did not have actual dependencies

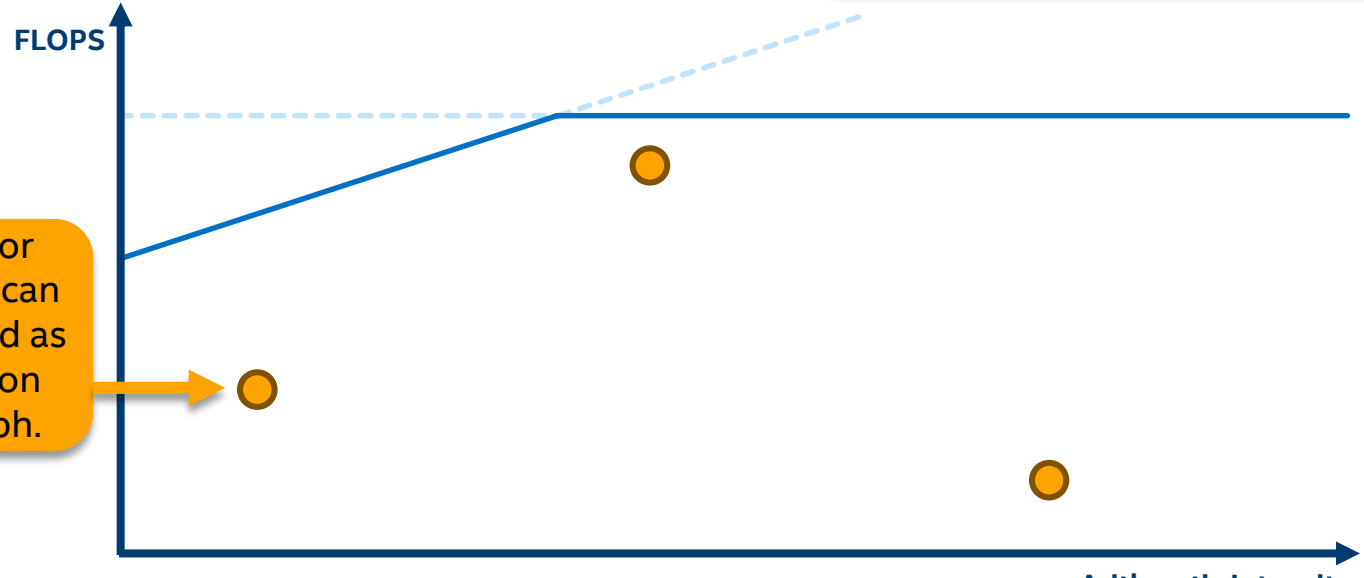


Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern	Site Name
loop at loops90.f:1261 in S256	No dependencies found	No information available	No information available	loop_site_365
loop at loops90.f:1287 in S257	RAW:1	No information available	No information available	loop_site_372
loop at loops90.f:1313 in S258	No dependencies found	No information available	No information available	loop_site_373
loop at loops90.f:2127 in S321	RAW:1	No information available	No information available	loop_site_509
loop at loops90.f:2150 in S322	RAW:1	No information available	No information available	loop_site_515
loop at loops90.f:2150 in S322	RAW:1	No information available	No information available	loop_site_514
loop at loops90.f:2173 in S323	RAW:1	No information available	No information available	loop_site_516
loop at loops90.f:2276 in S342	No dependencies found	No information available	No information available	loop_site_528
loop at loops90.f:2301 in S343	No dependencies found	No information available	No information available	loop_site_534
loop at loops90.f:2712 in S442	No information available	100% / 0% / 0%	All unit strides	loop_site_501
loop at loops90.f:2712 in s442_ompparallel_for@2710	No information available	100% / 0% / 0%	All unit strides	loop_site_102
loop at loops90.f:2840 in S471	No dependencies found	No information available	No information available	loop_site_611
loop at loops90.f:2840 in s471_ompparallel_for@2839	No dependencies found	No information available	No information available	loop_site_99
loop at loops90.f:369 in S123	No dependencies found	No information available	No information available	loop_site_183
loop at loops90.f:448 in S126	RAW:1	No information available	No information available	loop_site_196
loop at loops90.f:573 in s141_ompparallel_for@570	No dependencies found	No information available	No information available	loop_site_77
loop at loops90.f:884 in S221	RAW:1	No information available	No information available	loop_site_286
loop at loops90.f:908 in S222	RAW:1	No information available	No information available	loop_site_289
loop at loops90.f:959 in S232	RAW:1	No information available	No information available	loop_site_296
loop at loops90.f:959 in S232	RAW:1	No information available	No information available	loop_site_300
loop at loops90.f:959 in s232_ompparallel_for@956	No dependencies found	No information available	No information available	loop_site_88
loop at loops90.f:959 in s232_ompparallel_for@956	RAW:1	No information available	No information available	loop_site_711


intel
experience
what's inside™
INTEL® ADVISOR ROOFLINE AUTOMATION

Plotting a Roofline Chart

The maximum FLOPS as a product of ops/byte (AI) and maximum bytes supplied per second is a diagonal line.

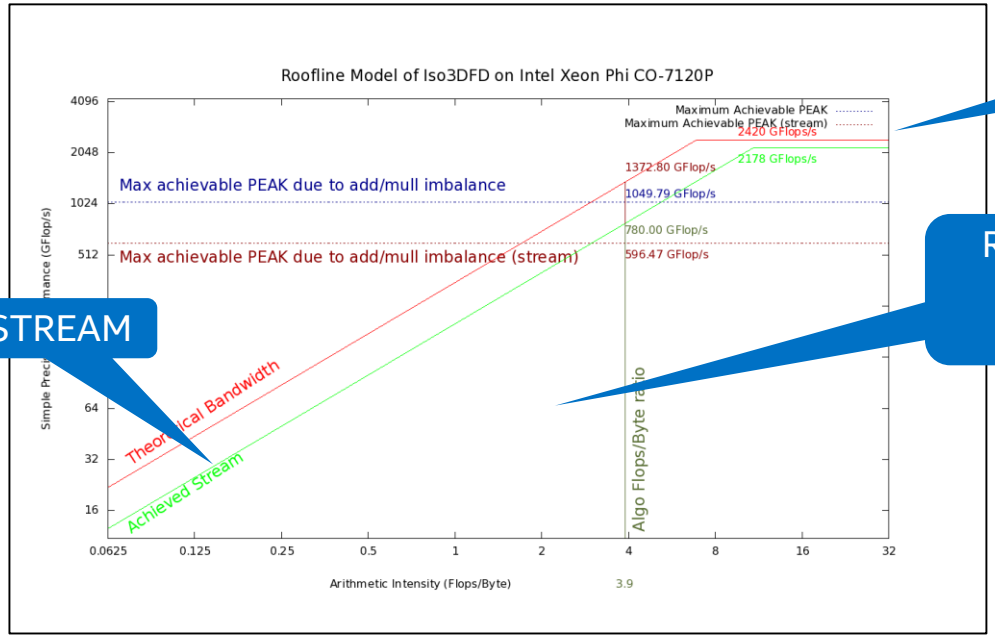


The CPU's maximum FLOPS can be plotted as a horizontal line.

A loop or function can be plotted as a point on the graph.

A Roofline Chart uses AI as its X axis and FLOPS as its Y axis.

Old approach – pen and paper



Run DGEMM

Run STREAM

Read the source, count FP ops, loads&stores

```
for(int bz=HALF_LENGTH; bz<n3; bz+=n3_Tblock)
    ALF_LENGTH; by<n2; by+=n2_Tblock)
    LF_LENGTH; bx<n1; bx+=n1_Tblock) {
    MIN(bz+n3_Tblock, n3);
    MIN(by+n2_Tblock, n2);
    MIN(n1_Tblock, n1-bx);

  bz; iz<izEnd; iz++) {
  by; iy<iyEnd; iy++) {
  xt = ptr_next_base + iz*n1n2 + iy*n1 + bx;
  rev = ptr_prev_base + iz*n1n2 + iy*n1 + bx;
float* vel = ptr_vel_base + iz*n1n2 + i
for(int ix=0; ix<ixEnd; ix++) {
  value = 0.0;
  += prev[ix]*coeff[0];
t ir=1; ir<=HALF_LENGTH; ir++) {
e += coeff[ir] * (prev[ix + ir] + prev[ix - ir])
  ;
value += coeff[ir] * (prev[ix + ir*n1] + prev[ix -
    ir*n1]);
value += coeff[
    ir*n1n2]) + prev[ix -
}
next[ix] = 2.0f* prev[ix] - next[ix] + value*vel[ix];
}
}}}
```

4 loads

51 adds

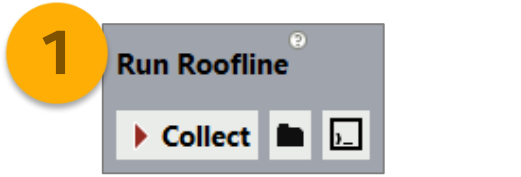
27 muls

1 store

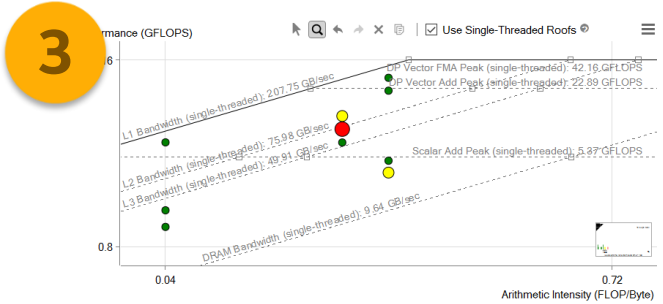
"3D stencil performance evaluation and auto-tuning on multi and many-core computers", C.Andreolli et.al.

Cumbersome – but people still did it!

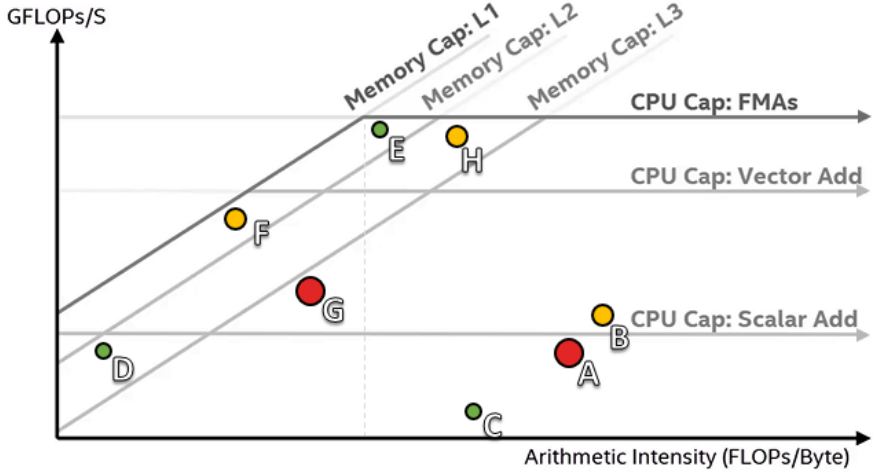
The Roofline Chart in Intel® Advisor



A single button or CLI command runs the Survey and FLOPS analyses to generate the Roofline chart.



- Focus on big dots with headroom.
  - Intel® Advisor sizes and color-codes dots by relative time taken.
  - Space between dots and their uppermost limits is room to improve.
- Roofs above a dot indicate potential sources of bottlenecks.

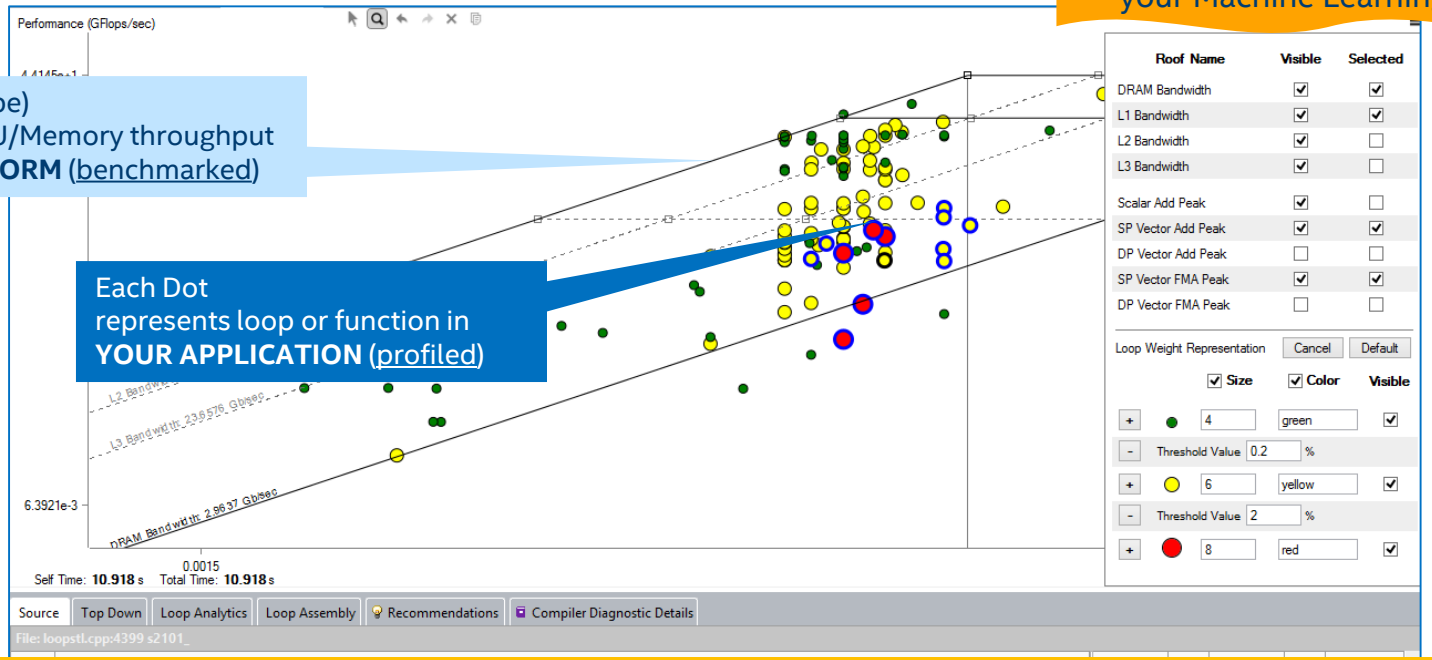


Roofline Automation in Intel® Advisor

NEW: INTOP/S based Roofline for your Machine Learning codes

Each Roof (slope)
Gives peak CPU/Memory throughput of your **PLATFORM** (benchmarked)

Each Dot
represents loop or function in **YOUR APPLICATION** (profiled)



Automatic and integrated – first class citizen in Intel® Advisor

Questions to answer with Roofline: for your loops / functions

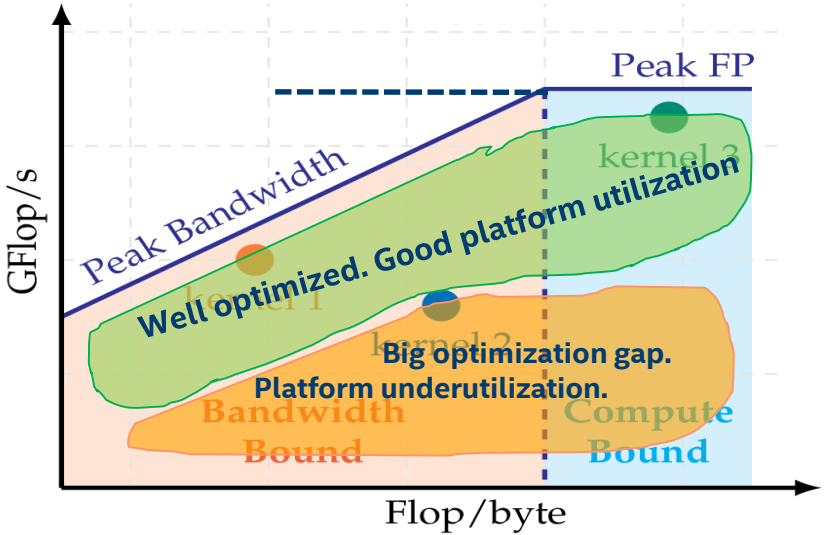
1 **Am I doing well? How far am I from the pick?**

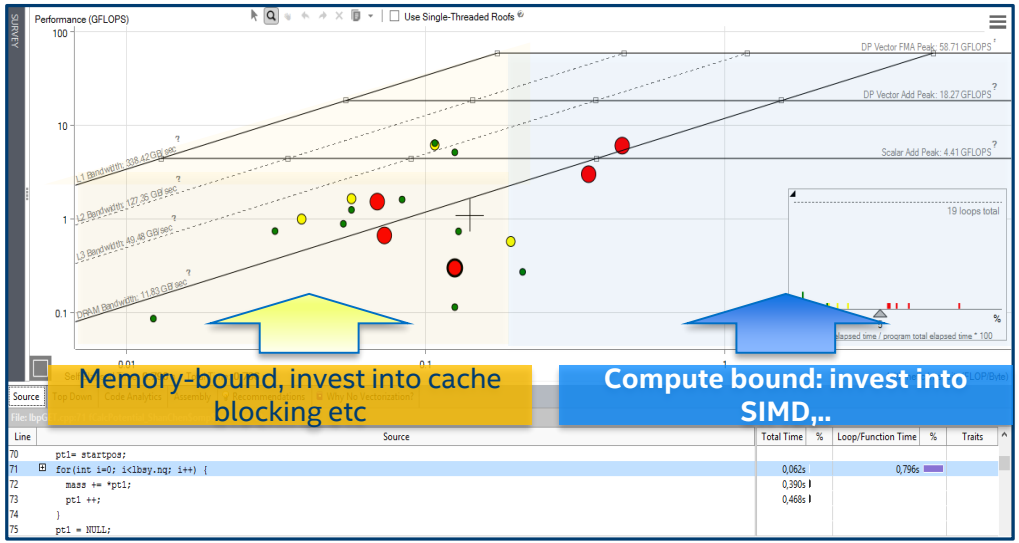
(do I utilize hardware well or not?)

2 **Final** Bottleneck?

(where will be my limit after I done all optimizations?)

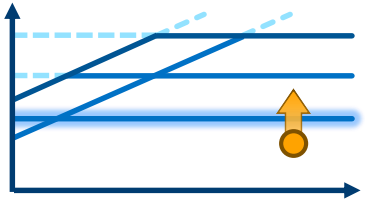
Long-term ROI, optimization strategy





Feature Synergy

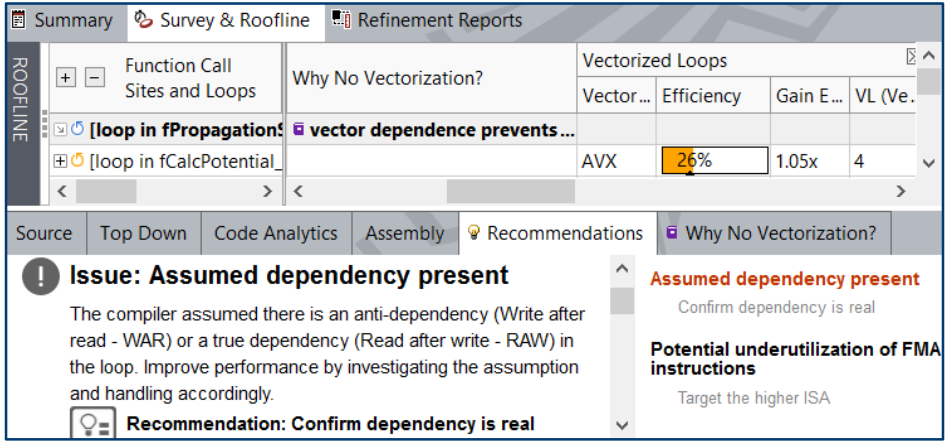
Overcoming the Scalar Add Peak

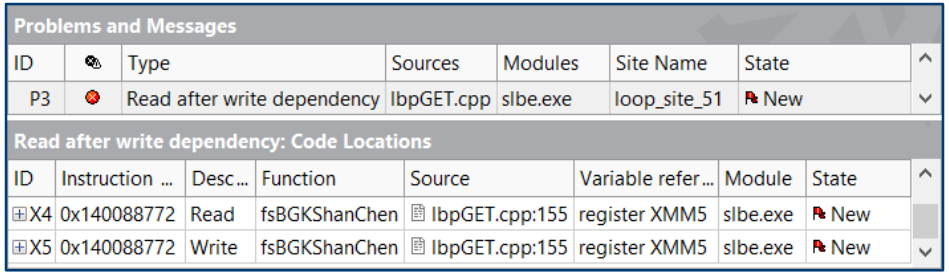


- Survey and Code Analytics tabs indicate vectorization status with colored icons.

  = Scalar = Vectorized

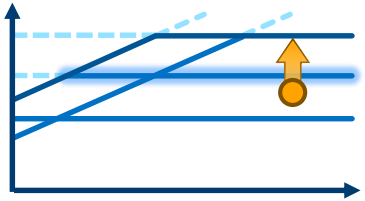
- "Why No Vectorization" tab and column in Survey explain what prevented vectorization.
- Recommendations tab may help you vectorize the loop.
- Dependencies determines if it's safe to force vectorization.





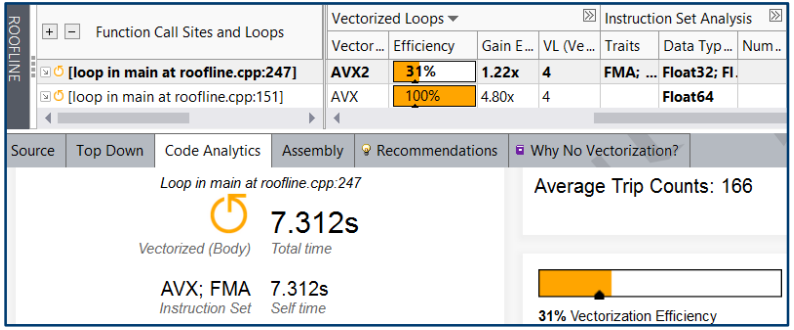
Feature Synergy

Overcoming the Vector Add Peak



Survey and Code Analytics display the vector efficiency and presence of FMAs.

- Recommendations may help improve efficiency or induce FMA usage.



Address	Line	Assembly
0x140001550		Block 1: 1660000000
0x140001550	262	vmovupd ymm3, ymmword ptr [rsi+rcx*8+0x26400]
0x140001559	262	vmovdqa ymm1, ymm0
0x14000155d	262	vfmadd132pd ymm1, ymm3, ymmword ptr [rsi+rcx*8+0x23a80]
0x140001567	262	vaddpd ymm2, ymm1, ymm3
0x14000156b	262	vmovupd ymm1, ymmword ptr [rsi+rcx*8+0x26420]
0x140001574	262	vaddpd ymm4, ymm2, ymm3
0x140001578	262	vmovdqa ymm5, ymm0
0x14000157c	262	vfmadd132pd ymm5, ymm1, ymmword ptr [rsi+rcx*8+0x23aa0]
0x140001586	262	vmovupd ymmword ptr [rsi+rcx*8+0x21100], ymm4
0x14000158f	262	vaddpd ymm5, ymm5, ymm1
0x140001593	262	vaddpd ymm2, ymm5, ymm1
0x140001597	260	add rcx, 0x8
0x14000159b	262	vmovupd ymmword ptr [rsi+rdx*8+0x21100], ymm2
0x1400015a4	260	add rdx, 0x8
0x1400015a8	260	cmp rcx, 0x530
0x1400015af	260	jb 0x140001550 <Block 1>

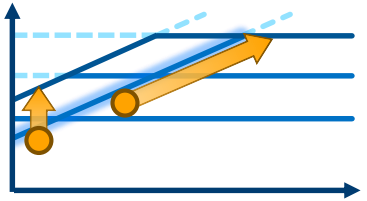
The Assembly tab* is useful for determining how well you are making use of FMAs.

*Color coding added for clarity.

Address	Line	Assembly
0x1400015f0		Block 1: 1660000000
0x1400015f0	275	vmovupd ymm1, ymmword ptr [rsi+rcx*8+0x26400]
0x1400015f9	275	vmovupd ymm2, ymmword ptr [rsi+rcx*8+0x26420]
0x140001602	275	vfmadd231pd ymm1, ymm1, ymm0
0x140001607	275	vfmadd231pd ymm2, ymm2, ymm0
0x14000160c	275	vfmadd231pd ymm1, ymm0, ymmword ptr [rsi+rcx*8+0x23a80]
0x140001616	275	vfmadd231pd ymm2, ymm0, ymmword ptr [rsi+rcx*8+0x23aa0]
0x140001620	275	vmovupd ymmword ptr [rsi+rcx*8+0x21100], ymm1
0x140001629	275	vmovupd ymmword ptr [rsi+rdx*8+0x21100], ymm2
0x140001632	273	add rcx, 0x8
0x140001636	273	add rdx, 0x8
0x14000163a	273	cmp rcx, 0x530
0x140001641	273	jb 0x1400015f0 <Block 1>

Feature Synergy

Overcoming the Memory Bandwidth Roofs



- Memory Access Patterns (MAP) identifies inefficient access patterns.
- Intel® SIMD Data Layout Templates (Intel® SDLT) allows code written as AOS to be stored as efficient SOA.
- Intel® VTune™ Amplifier can be used to further optimize cache usage.
- If cache usage cannot be improved, try re-working the algorithm to increase the AI (and slide up the roof)

Summary | Survey & Roofline | Refinement Reports

Site Location	Strides Distribution	Access Pattern	Max. Site Footprint	Recommendations
[loop in main at roofline.cpp:1 ...	0% / 100% / 0%	All const strides	38KB	1 Inefficient me...
[loop in main at roofline.cpp:1 ...	50% / 50% / 0%	Mixed strides	10KB	1 Inefficient me...
[loop in main at roofline.cpp:1 ...	100% / 0% / 0%	All unit strides	9KB	

Memory Access Patterns Report | Dependencies Report | Recommendations

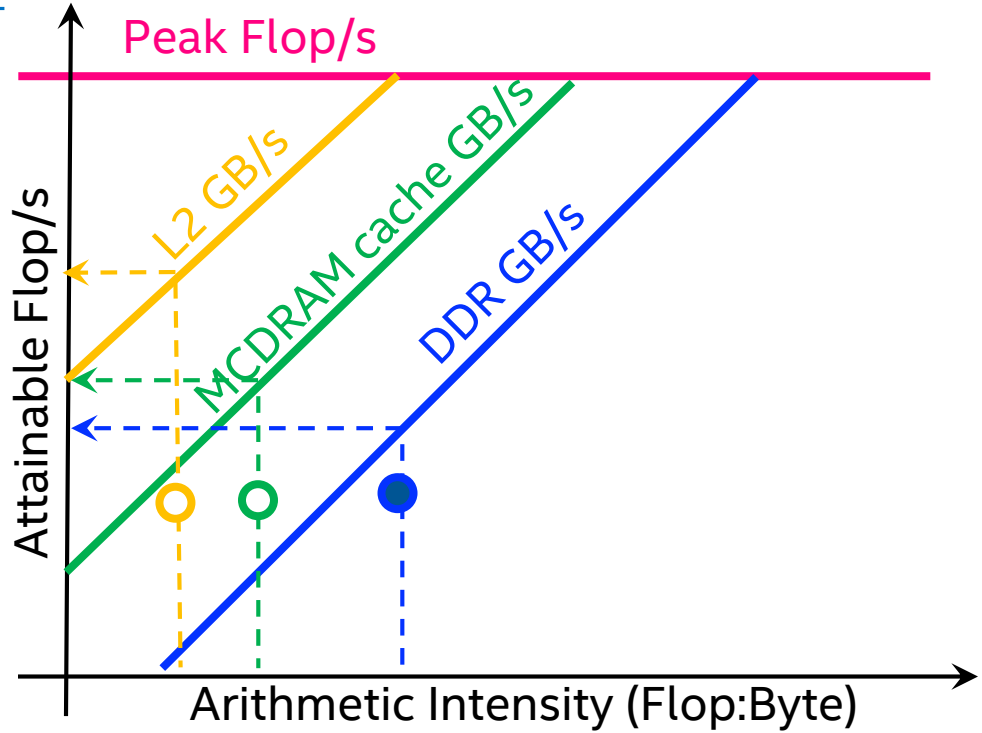
ID	Stride	Type	Source	Variable references	Max. Site Footprint	Access Type
P1	8	Constant stride	roofline.cpp:127	AoS1_Y	38KB	Read
P2	2	Constant stride	roofline.cpp:127	AoS1_X	10KB	Write

PLACEHOLDER FOR VTUNE SCREENSHOT

3 Find 1st bottleneck: NEW "Integrated Roofline", learn more at: https://https://software.intel.com/en-us/articles/integrated-roofline-model-with-intel-advisor

Performance is limited by minimum of intercepts (L2, LLC, DRAM, CPU)

In this example:
by DRAM



How to generate **CARM** Roofline profile?* (using Advisor from USB stick/goto link)

We will **NOT do this during hands-on*

As simple as: `$ advixe-cl -collect roofline -- <your-executable-with-parameters>`

```
$ source advixe-vars.sh
```

1st method. Not compatible with MPI applications :

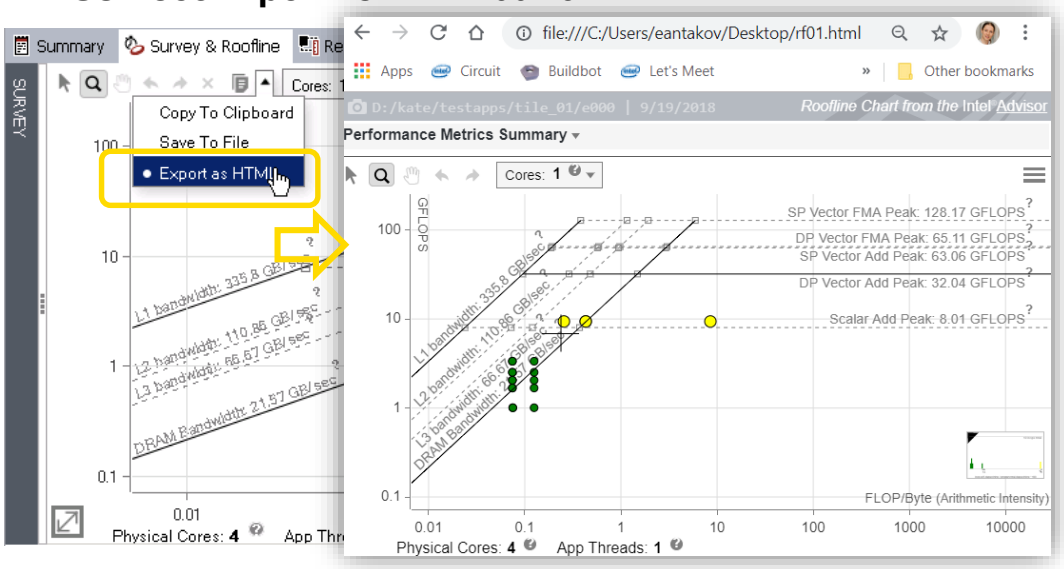
```
$ advixe-cl -collect roofline --project-dir ./your_project -- <your-executable-with-parameters>
```

2nd method (compatible with MPI, 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>

(optional) copy data to your UI desktop system

$ advixe-gui ./your_project

$ advixe-cl -report roofline --project-dir ./your_project > roofline.html
```

Exporting Integer and Integrated Roofline as HTML

GUI: Use **Export as HTML** button



Command line:

Set ADVIXE_EXPERIMENTAL=int_roofline

advixe-cl –report roofline
-data-type=float
-memory-level=L2
-memory-operation-type=load
-project-dir /path/to/project/dir

Possible
- data types: float, int, mixed
- memory levels: L1, L2, L3, DRAM
- memory operation types: load, store, all

- Export Roofline from command line does not need GUI sub-system on clusters
- Useful for rooflines quick exchange

Python API

- Fairly new Advisor Extensibility/customization mechanism. Actively used internally in Intel
- Up to 500 metrics for each loop/function. Really easy to use:
  - <advisor_install_dir>/pythonapi/examples
  - $python ***survey_bottomup.py*** *<project_dir>*
  - Generate your own customized roofline charts

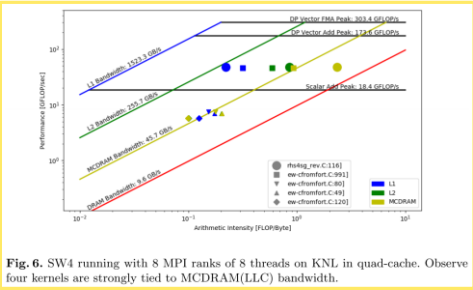


Fig. 6. SW4 running with 8 MPI ranks of 8 threads on KNL in quad-cache. Observe four kernels are strongly tied to MCDRAM(LLC) bandwidth.

From ISC'18 paper (cudos Tuomas)

```
import sys

try:
    import advisor

project = advisor.open_project(sys.argv[1])

survey = project.load(advisor.SURVEY)

for row in survey.bottomup:
    # row as string
    print(row)
    # row as iterator
    for key in row:
        # row as dictionary
        print('{}: {}'.format(key, row[key]))
```

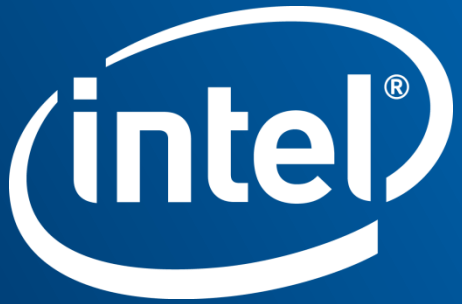
Legal Disclaimer & Optimization Notice

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


intel®