INTEL® ADVISOR

VECTORIZATION AND ROOFLINE ANALYSIS

Intel Software and Services, 2017

Agenda

Quick overview of the Intel® Parallel Studio 2018 Beta

Intel® Advisor overview

Intel® Advisor AVX-512 profiling

Intel® Advisor Roofline automation

Intel® Advisor new features

Summary/call to action

INTEL® ADVISOR

VECTORIZATION AND ROOFLINE ANALYSIS

What's new in the "2018" release

1. Compiler diagnostics + Performance Data + SIMD efficiency information

Modules exclusions,
MKL

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

More
ions

3. "Precise" Trip Counts & FLOPs. Roofline analysis.

cterize your application.

Call Counts, MKL, Instruction count, Hier. Roofline

Roofline is now a product feature!

4. Loop-Carried Dependency Analysis

Overhead decreased

5. Memory Access Patterns Analysis

Cache simulation (feature preview)

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

Focus and order **vectorized loops**

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)

Data Dependencies – Tough Problem #1

Is it safe to force the compiler to vectorize?

```
DO I = 1, N
   A(I) = A(I-1) * B(I)
ENDDO
```

or

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

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**

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]
```

Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern	Site Name
[loop in fPropagationSwap at lbpSUB.cpp:1247]	No information available	33% / 5% / 62%	Mixed strides	loop_site_60

blue color: fraction of unit stride accesses

yellow: "fixed" stride accesses ratio

red color: fraction of irregular (variable stride) accesses

Memory Access Patterns Report | Dependencies Report

ID		Stride	Type	Source	Site Name	Variable
P1		3	16% / 84% / 0%	Mixed strides		

```
1246 #endif
1247            for (int m=1; m<=half; m++) {
1248              nextx = fCppMod(i + lbv[3*m]
1249              nexty = fCppMod(j + lbv[3*m+
1250              nextz = fCppMod(k + lbv[3*m+
```

P11		0; 1
P12		-289559; -274359; -14477; -13717; -13679; 723; 302519;

```
1251              ilnext = (nextx * Ymax + nex
1252 #ifndef SWAP_OVERLAP
1253   fSwapPair (lbf[il*lbsitelength + l*lbsy.
```

16%: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

Stride 0 = Instruction accesses the same memory from iteration to iteration

84%: percentage of memory instructions with fixed or constant non-unit stride accesses

Constant stride (stride N) = Instruction accesses memory 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

0%: percentage of memory instructions with irregular (variable or random) stride accesses

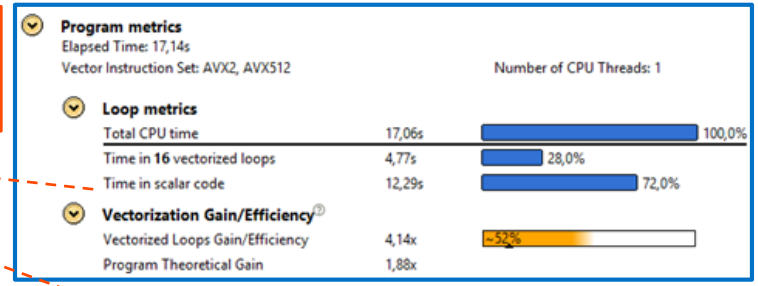
Irregular stride = Instruction accesses memory addresses that change by an unpredictable number of elements from iteration to iteration

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

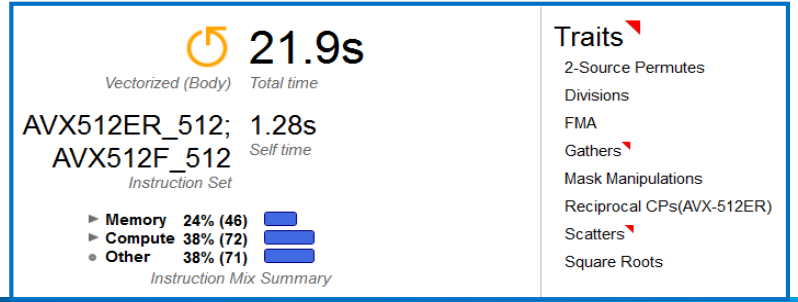
AVX-512 PROFILING WITH INTEL® ADVISOR

Intel®Advisor : **AVX-512** specific performance insights



- **Native AVX-512 profiling** on KNL
- Precise **FLOPs and Mask Utilization** profiler
- AVX-512 **Advices and "Traits"**
- And more..
  - Performance **Summary for** AVX-512 codes
  - **AVX-512 Gather/Scatter Profiler**
- No access to AVX-512 Hardware yet?
  - Explore AVX-512 code with –axcode flags and new Advisor Survey capability!

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); Expone...
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



Highlight “impactful” AVX-512 instructions

Survey Static Analysis - AVX-512 ***“Traits”***

Presence of remarkable

performance-impactful

(negative or positive impact)

instructions

Vectorization Advisor Trait and/or Recommendation	Theoretical Performance Impact Comments	Corresponding AVX-512 Instructions
Compress / Expand Trait and Recommendation	>> **4x** speedup	v(p)expand* v(p)compress*
Gather / Scatter Trait	**Up to 10x slower** than contiguous memory access **>2x faster** than scalar	v(p)gather* v(p)scatter*
Conflict Detection		v(p)conflict*
Approximate Reciprocals/Reciprocal SQRT; AVX-512ER	**>10x faster** than DIV/SQRT	vrcp* vrcsqrt* vdiv* vsqrt*
Exponent extraction Mantissa extraction Traits		vgetexp* vgetmant*
L1 (L2) Prefetch **L1 (L2) Sparse prefetch Trait**		prefetchw* vscatterpf* vgatherpf*

Gather/Scatter Analysis

Motivation

AVX-512 Gather/Scatter-based vectorization.

Much wider usage than before :

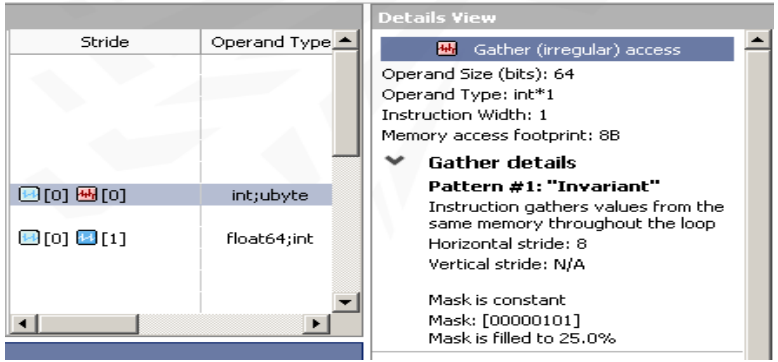
- Makes much more codes (profitably) vectorizable
- Gives good average performance, but often far from optimal.

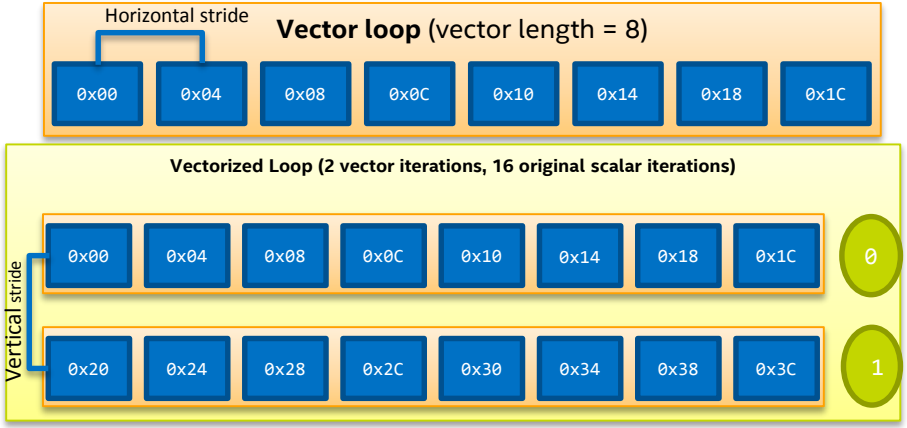
Could be 2x faster than scalar mov

Could be 10x slower than vmovp*

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

Gather/scatter issue improvements

Compiler may generate gather/scatter instructions despite regular access pattern. In this case, performance can be improved by refactoring the code.

- Detecting regular patterns taking into account masking instructions
- Added new access pattern for gather profiling – Constant (Non-Unit Stride) with adjusted recommendation to transform AOS to SOA

Recommendation: Refactor code with detected regular stride access patterns — Confidence: Low

The Memory Access Patterns Report shows the following regular stride access(es):

Variable	Pattern
block 0x7f049a6ff010	Constant (non-unit)

See details in the Memory Access Patterns Report Source Details view.

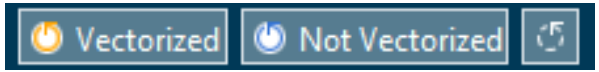
To improve memory access: Refactor your code to alert the compiler to a regular stride access. Sometimes, it might be beneficial to use the `ipo/Qipo` compiler option to enable interprocedural optimization (IPO) between files.

An array is the most common type of data structure containing a contiguous collection of data items that can be accessed by an ordinal index. You can organize this data as an array of structures (AoS) or as a structure of arrays (SoA). Detected constant stride might be the result of AoS implementation. While this organization is excellent for encapsulation, it can hinder effective vector processing. To fix: Rewrite code to organize data using SoA instead of AoS.
However, the cost of rewriting code to organize data using SoA instead of AoS may outweigh the benefit. To fix: Use Intel SIMD Data Layout Templates (Intel SDLT), introduced in version 16.1 of the Intel compiler, to mitigate the cost. Intel SDLT is a C++11 template library that may reduce code rewrites to just a few lines.

AVX-512-specific performance trade-offs

Advisor AVX-512 Recommendations

Increasing Vector Register Size ->



Increase fraction of time spent in Remainders

Function Call Sites and Loops		Vector Issues	Self Time	Total Time	Type	Vectori Vector I
[loop in fCollisionBGKShanChen$om ...		1 Ineffective peeled/remainder loop(s ...	0,110s	0,110s	Vectorized (Remainder; [Body])	AVX512
[loop in fCollisionBGKShanChen$o ...			0,110s	0,110s	Vectorized (Remainder)	AVX512
[loop in fCollisionBGKShanChen$o ...			n/a	n/a	Vectorized (Body) [Not Executed]	AVX512
[loop in fGetFracSite at IbpGET.cpp:19 ...		1 Ineffective peeled/remainder loop(s ...	0,060s	0,060s	Vectorized (Peeled; Remainder; [Body])	AVX512
[loop in fGetFracSite at IbpGET.cpp ...			0,040s	0,040s	Vectorized (Peeled)	AVX512
[loop in fGetFracSite at IbpGET.cpp ...			0,020s	0,020s	Vectorized (Remainder)	AVX512
[loop in fGetFracSite at IbpGET.cpp ...			n/a	n/a	Vectorized (Body) [Not Executed]	AVX512
[loop in fCalcInteraction_ShanChen a ...		1 Ineffective peeled/remainder loop(s ...	0,060s	0,060s	Vectorized (Remainder; [Body])	AVX512
[loop in fCalcInteraction_ShanChe ...			0,060s	0,060s	Vectorized (Remainder)	AVX512
[loop in fCalcInteraction_ShanChe ...			n/a	n/a	Vectorized (Body) [Not Executed]	AVX512
[loop in fGetOneMassSite at IbpGET.c ...		1 Ineffective peeled/remainder loop(s ...	0,050s	0,050s	Vectorized (Remainder; [Body])	AVX512
[loop in fGetTotMomentSite at Ibp ...		1 Ineffective peeled/remainder loo ...	0,040s	0,040s	Vectorized (Remainder)	AVX512
[loop in fGetOneDirecSpeedSite at Ibp ...		1 Ineffective peeled/remainder loop(s ...	0,030s	0,030s	Vectorized (Remainder)	AVX512
[loop in fGetOneMassSite at IbpGET.c ...		1 Ineffective peeled/remainder loop(s ...	0,030s	0,030s	Vectorized (Remainder)	AVX512
[loop in fGetOneDirecSpeedSite at Ibp ...		1 Ineffective peeled/remainder loop(s ...	0,020s	0,020s	Vectorized (Remainder)	AVX512

Ineffective masked remainder for AVX512 codes

- Compiler generates vector masked remainder due to the number of iterations (trip count) not being divisible by vector length. In case of executing a few iterations, it is ineffective comparing to scalar versions of the loop.

- Using AVX512 mask profiler and trip-counts data to prove the issue.

Recommendation: Force scalar remainder generation Confidence: Low

The compiler generated a masked vectorized remainder loop that contains too few iterations for efficient vector processing. A scalar loop may be more beneficial. To fix: Force scalar remainder generation using a directive: `#pragma simd novecremainder` or `#pragma vector novecremainder`.

Example: Force the compiler to not vectorize the remainder loop

```
void add_floats(float *a, float *b, float *c, float *d, float *e, int n)
{
    int i;
    #pragma simd novecremainder
    for (i=0; i<n; i++)
    {
        a[i] = a[i] + b[i] + c[i] + d[i] + e[i];
    }
}
```

Read More:

- simd, vector
- Getting Started with Intel Compiler Pragmas and Directives and Vectorization Resources for Intel® Advisor Users

```
#pragma simd reduction(+:mean)
for(int j = 0; j < size; j++) {
        mean += data[order[j]] / N;
        data[order[j]] = 10.f / (j+1);
}
```

E.g. bad performance if ((size) % (loop_body_vl) == 1), in case of float number it results in 12.5% mask bits utilization only, in addition leads to gathers, scatters...

ROOFLINE PERFORMANCE MODEL

CASE STUDY

Roofline Analysis to Tune an MRI Image Reconstruction Benchmark

The 514.pomriq SPEC ACCEL Benchmark

An MRI image reconstruction kernel described in Stone et al. (2008). MRI image reconstruction is a conversion from sampled radio responses to magnetic field gradients. The sample coordinates are in the space of magnetic field gradients, or K-space.

The algorithm examines a large set of input, representing the intended MRI scanning trajectory and the points that will be sampled.

The input to 514.pomriq consists of one file containing the number of K-space values, the number of X-space values, and then the list of K-space coordinates, X-space coordinates, and Phi-field complex values for the K-space samples.

Hot loop is vectorized

Vectorization Advisor

Vectorization Advisor is a vectorization analysis tool that lets you identify loops that will benefit most from vectorization.

Program metrics

Elapsed Time: 36.93s

Vector Instruction Set: AVX512 | Number of CPU Threads: 136

Total GFLOP Count: 19293.90 | Total GFLOPS: 522.51

Loop metrics

Total CPU time	4267.88s	100.0%
Time in 1 vectorized loop	4206.25s	98.6%
Time in scalar code	61.62s	

Vectorization Gain/Efficiency (Not available)

Top time-consuming loops

Loop	Self Time	Total Time	Trip Counts
[loop in ComputeQCPU at computeQ.c:65]	1957.548s	4206.254s	12500
[loop in ComputeQCPU at computeQ.c:58]	6.963s	4213.216s	15420
[loop in outputData at file.c:70]	0.040s	4.160s	2097152
[loop in start_thread at ?]	0s	49.660s	
[loop in [OpenMP worker at z_Linux_util.c:769]	0s	49.660s	

Refinement analysis data

These loops were analyzed for memory access patterns and dependencies:

Site Location	Dependencies	Strides Distribution
[loop in ComputeQCPU at computeQ.c:66]	No information available	96% / 0% / 4%

Collection details

Platform information

CPU Name:	Intel(R) Xeon Phi(TM) CPU 7250 @000000 1.40GHz
Frequency:	1.40 GHz
Logical CPU Count:	272
Operating System:	Linux

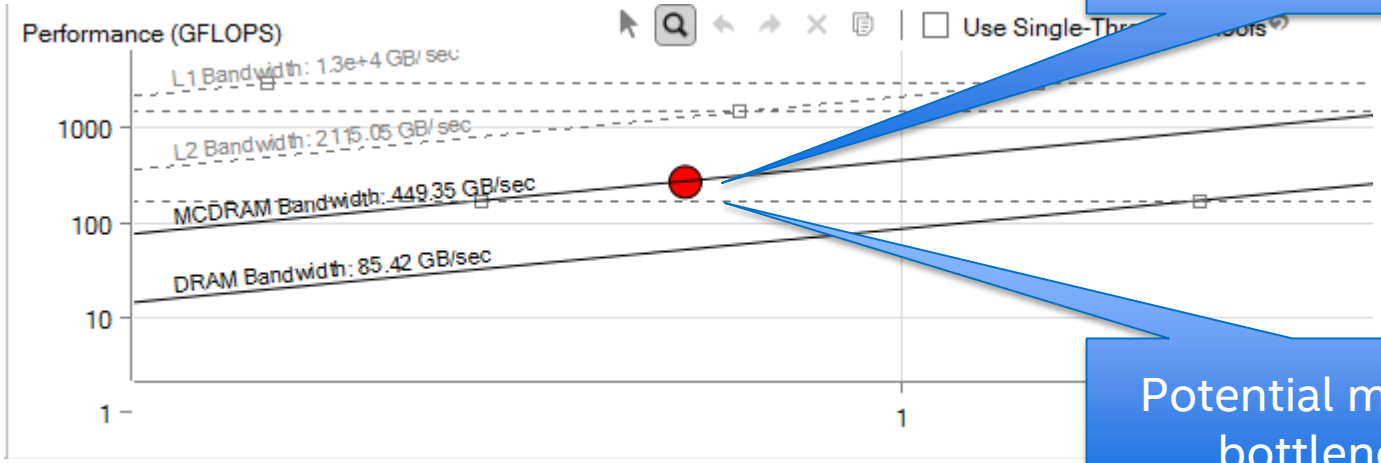
Intel Advisor summary view

1 vectorized loop that we spend 98.8% of our time in

Need more information to see if we can get more performance

What is our performance?

Relative to peak system performance



Our hot loop is below the MCDRAM roof

Potential memory bottleneck

Get detailed Advice from intel® Advisor

Intel® Advisor code analytics

Loop in ComputeQCPU at computeQ.c:65

4206.254s
Vectorized (Body) *Total time*

AVX512F_512 1957.548s
Instruction Set *Self time*

- Memory 16% (4)
- Compute 33% (8)
- Other 51% (12)

Instruction Mix Summary

Average Trip Counts: 12500

Instruction Mix

Memory: 4 *Compute:* 8 *Other:* 12 *Number of Vector Registers:* 13

Statistics for FLOPS And Data Transfers

GFLOPS	*266.242*	Giga Floating-point Operations Per Second Per-loop GFLOPS = Total FLOP / Elapsed Time. Elapsed time is the exclusive (self-time-based) wall time from the beginning to the end of loop/function execution. For single-threaded applications Elapsed time is equal to Self-Time.
AI	*0.606*	AI - Arithmetic Intesity - Ratio of Floating-point Operations to L1 Transferred Bytes
Mask Utilization	*100*	Ratio of Utilized Vector Elements to Total Vector Elements
GFLOP	*4194.304*	Giga Floating-point Operations
FLOP Per Iteration	*160*	Floating-point Operations Per Loop Iteration

Data transfers between CPU and memory sub-system (total traffic, including L1, L2, LLC and DRAM traffic)

Traits

Gathers FMA, Mask Manipulations

Possible inefficient memory access. Gather stride.

Issue: Possible inefficient memory access patterns present

Inefficient memory access patterns may result in significant vector code execution slowdown or block automatic vectorization by the compiler. Improve performance by investigating.

Recommendation: Confirm inefficient memory access patterns — Confidence: Need More Data

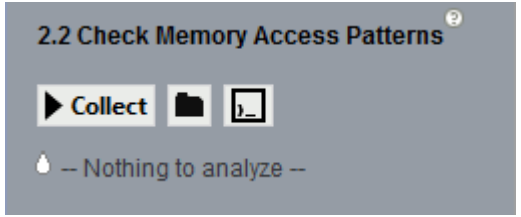
There is no confirmation inefficient memory access patterns are present. To confirm: Run a Memory Access Patterns analysis.

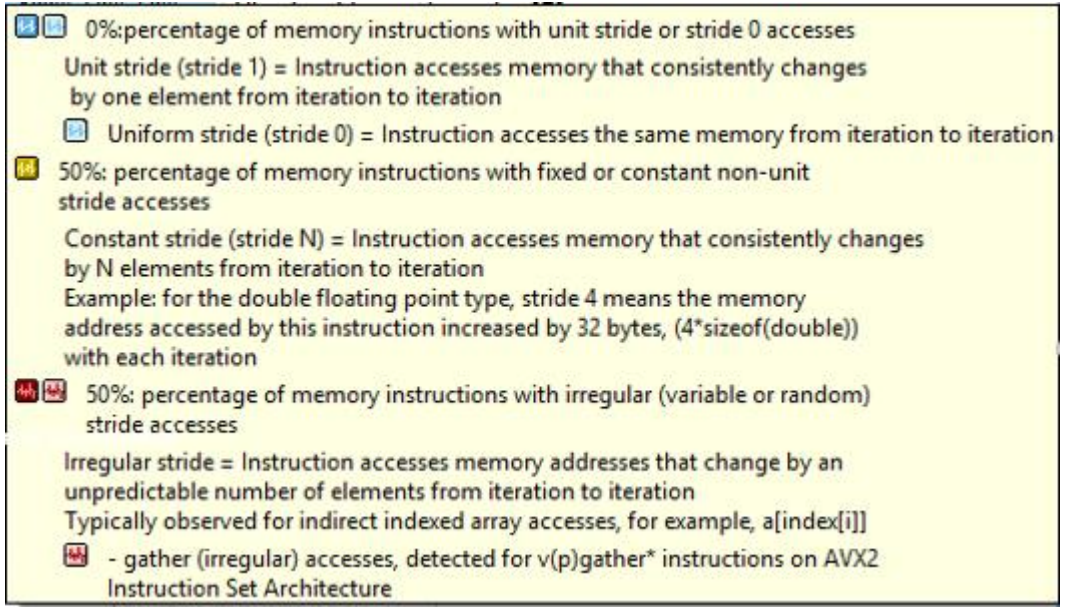
Recommendations – need more information, confirm inefficient memory access

Irregular access patterns decreases performance!

Gather profiling

Run Memory Access Pattern Analysis (MAP)





Irregular access patterns

Bad for vectorization performance

Details View

Gather (irregular) access

Operand Size (bits): 32

Operand Type: bit*16;float32*16

Vector Length: 16

Memory access footprint: 3MB

Gather/scatter details

Pattern: "Constant (non-unit)"

Instruction accesses values with constant offset from the base:

- stride within instruction = X
- stride between iterations = X*vector length

Horizontal stride (bytes): 16

Vertical stride (bytes): 256

Mask is constant

Mask: [1111111111111111]

Active elements in the mask: 100.0%

Variable references

Names: block 0x7f0045867010 allocated at main.c:99

Hint: use the Intel Advisor details!

Specific recommendation for your application

Issue: Inefficient gather/scatter instructions present

The compiler assumes indirect or irregular stride access to data used for vector operations. Improve memory access by alerting the compiler to detected regular stride access patterns, such as:

Pattern	Description
Invariant	The instruction accesses values in the same memory throughout the loop.
Uniform (Horizontal Invariant)	The instruction accesses values in the same memory within the vector iteration.
Vertical Invariant	The instruction accesses the memory locations using the same offset across all vector iterations.
Unit	The instruction accesses values in contiguous memory throughout the loop, and the stride between vector iterations = vector length.

Recommendation: Refactor code with detected regular stride access patterns — Confidence: Low

The Memory Access Patterns Report shows the following regular stride access(es):

Remove gather instructions

step #1 – use newer version of the intel compiler can recognize the access pattern

Gathers replacement is performed by the “Gather to Shuffle/Permutes” compiler transformation

Loop in ComputeQCPU at computeQ.c:65

3545.097s
Vectorized (Body) *Total time*

AVX512F_512 1444.120s
Instruction Set *Self time*

Average Trip Counts: 12500

Code Optimizations

Compiler: Intel(R) C Intel(R) 64 Compiler for applications running on Intel(R) 64,

Compiler estimated gain: 18.44x

Code Optimizations Applied By Compiler During Vectorization:

- Cost Model Was Ignored
- Dependency Analysis Was Ignored
- SIMD

Traits

2-Source Permutes
Blends
FMA

GFLOPS	*342.671*	Giga Floating-point Operations Per Second Per-loop GFLOPS = Total FLOP / Elapsed Time. Elapsed time is the exclusive (self-time-based) wall time from the beginning to the end of loop/function execution. For single-threaded applications Elapsed time is equal to Self-Time.
AI	*0.08*	AI - Arithmetic Intesity - Ratio of Floating-point Operations to L1 Transferred Bytes
Mask Utilization	*0*	Ratio of Utilized Vector Elements to Total Vector Elements
GFLOP	*4194*	Giga Floating-point Operations
FLOP Per Iteration		Floating-point Operations Per Loop Iteration

Data transfers between CPU and sub-system (total traffic, including L1, L2, LLC and DRAM traffic)

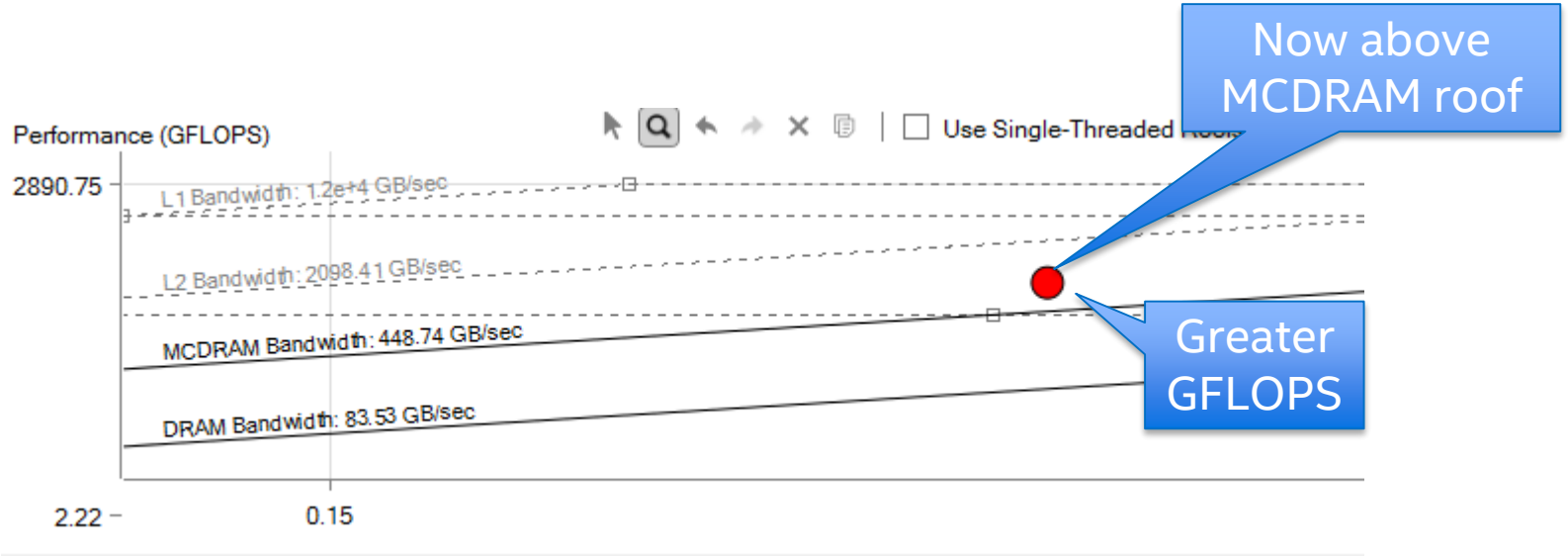
in Giga Bytes	*10276.*
in Giga Bytes Per Second	*839.*
in Bytes Per Loop Iteration	

Removed gathers

Increased GFLOPS (from 266.42 to 342.67)

Remove gather instructions

step #1 – newer version of the intel compiler can recognize the access pattern



Remove gather instructions

step #2 - Use structure of arrays instead of array of structures T

```
struct kValues {
  float Kx;
  float Ky;
  float Kz;
  float PhiMag;
};

SDLT_PRIMITIVE(kValues, Kx, Ky, Kz, PhiMag)


sdlt::soa1d_container<kValues> inputKValues(numK);
auto kValues = inputKValues.access();

  for (k = 0; k < numK; k++) {
    kValues [k].Kx() = kx[k];
    kValues [k].Ky() = ky[k];
    kValues [k].Kz() = kz[k];
    kValues [k].PhiMag() = phiMag[k];
  }

auto kVals = inputKValues.const_access();
#pragma omp simd private(expArg, cosArg, sinArg) reduction(+:QrSum, QiSum)
     for (indexK = 0; indexK < numK; indexK++) {
       expArg = PIx2 * (kVals[indexK].Kx() * x[indexX] +
       kVals[indexK].Ky() * y[indexX] +
       kVals[indexK].Kz() * z[indexX]);

       cosArg = cosf(expArg);
       sinArg = sinf(expArg);

       float phi = kVals[indexK].PhiMag();
       QrSum += phi * cosArg;
       QiSum += phi * sinArg;
     }
```

This is a classic vectorization efficiency strategy

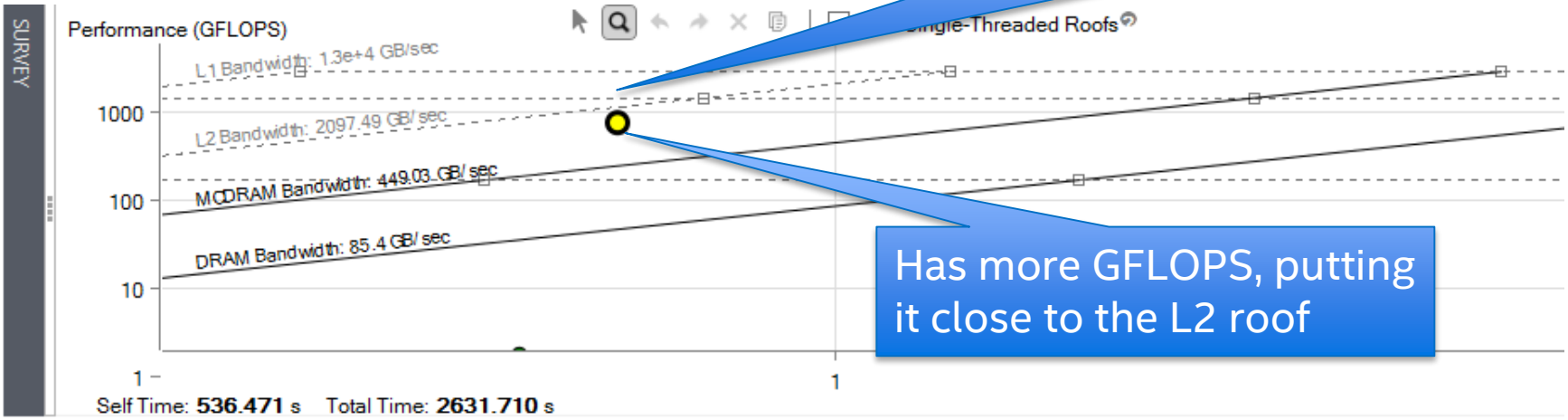
But it can yield poorly designed code

Intel® SIMD Data Layout Templates makes this transformation easy and painless!

Remove gather instructions

step #2 - Transform code using the Intel® SIMD Data Layout Templates

The loop is no longer red. This means it takes less time now



Has more GFLOPS, putting it close to the L2 roof

The total performance improvement is almost 3x for the kernel and 50% for the entire application.

ROOFLINE PERFORMANCE MODEL

AUTOMATION

Acknowledgments/References

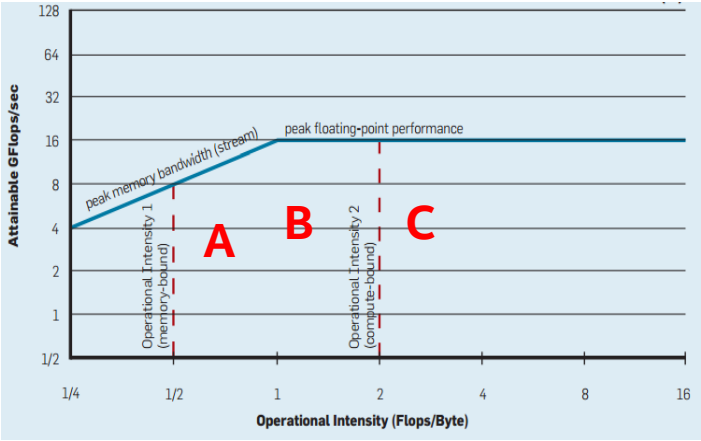
Roofline model proposed by Williams, Waterman, Patterson:
http://www.eecs.berkeley.edu/~waterman/papers/roofline.pdf

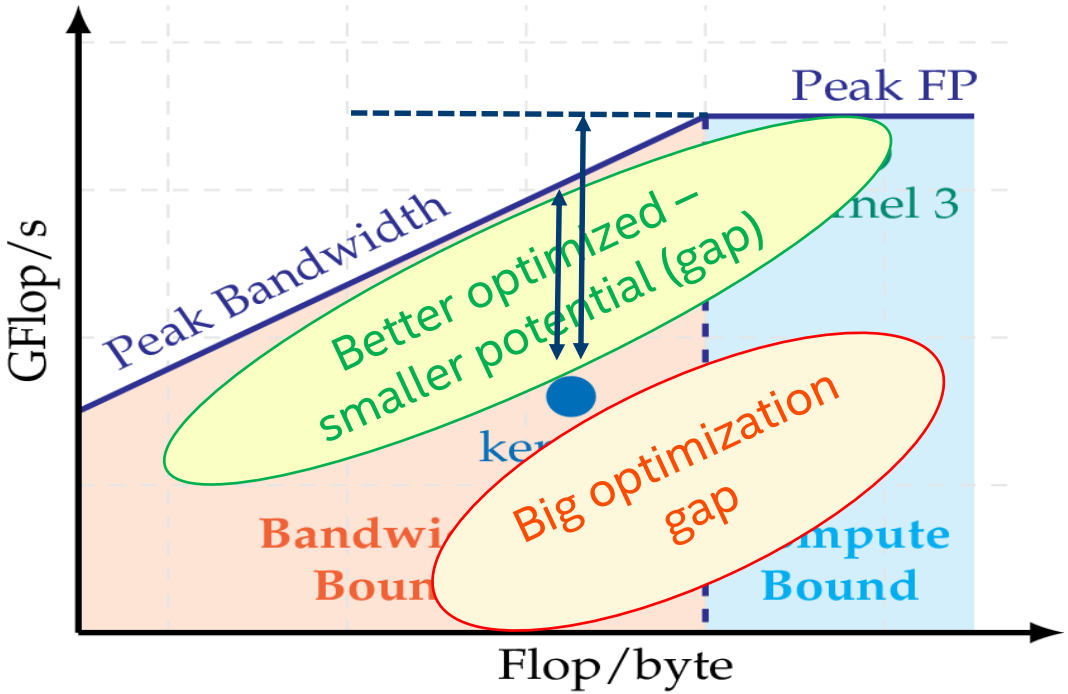
"Cache-aware Roofline model: Upgrading the loft" (Ilic, Pratas, Sousa, INESC-ID/IST, Thec Uni of Lisbon) http://www.inesc-id.pt/ficheiros/publicacoes/9068.pdf

At Intel:

Roman Belenov, **Zakhar Matveev**, Julia Fedorova
SSG product teams, Hugh Caffey,
in collaboration with **Philippe Thierry**

Roofline model: **Am I bound by VPU/CPU or by Memory?**





What makes loops
A, B, C different?

Roofline in Intel® Advisor

Switch between the roofline and the grid

Loop data hint

Source for the selected loop

SURVEY

Performance (GFLOPS)

Use Single-Threaded Roofs | Show Hierarchical Data

DoubleVectorFMAPeak: 351.1 GFLOPS

ScalarAddPeak: 28.08 GFLOPS

L1 Bandwidth: 2580.24 GB/ sec

L2 Bandwidth: 740.66 GB/ sec

L3 Bandwidth: 363.04 GB/ sec

DRAM Bandwidth: 22.93 GB/ sec

100

10

0.1

0.01

0.1

1

10

Arithmetic Intensity (FLOP/By

COMPUTE_RHSompparallel@17 rhs.f:301
Performance: **7.06** GFLOPS
L1 Arithmetic Intensity: **0.21** FLOP/Byte
Self Time: **4.647** s
Total Time: **4.647** s

Self Time: **4.647** s Total Time: **4.647** s

Source | Top Down | Code Analytics | Assembly | Recommendations | Why No Vectorization?

File: rhs.f:301 COMPUTE_RHSompparallel@17

	Source	Total Time	%	Loop/Function Time	%	Traits
	c----------					
	!$omp do schedule(static)					
	do k = 1, grid_points(3)-2					
300	do j = 1, grid_points(2)-2					
301	do i = 1, grid_points(1)-2	0.048s		4.647s		
	[loop in COMPUTE_RHSompparallel@17 at rhs.f:301]					
	Vectorized AVX; FMA loop processes Float64 data type(s) and in					
	Loop stmts were reordered; loop was blocked by 16					

Automatic and integrated – first class citizen in Intel® Advisor

Find Effective Optimization Strategies

Intel Advisor: Cache-aware roofline analysis

Roofs Show Platform Limits

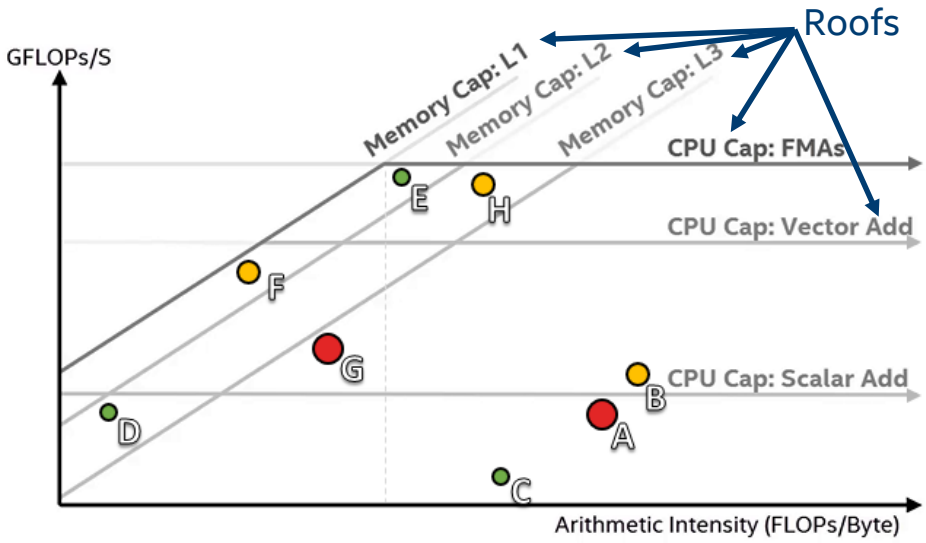
- Memory, cache & compute limits

Dots Are Loops

- Bigger, red dots take more time so optimization has a bigger impact
- Dots farther from a roof have more room for improvement

Higher Dot = Higher GFLOPs/sec

- Optimization moves dots up
- Algorithmic changes move dots horizontally

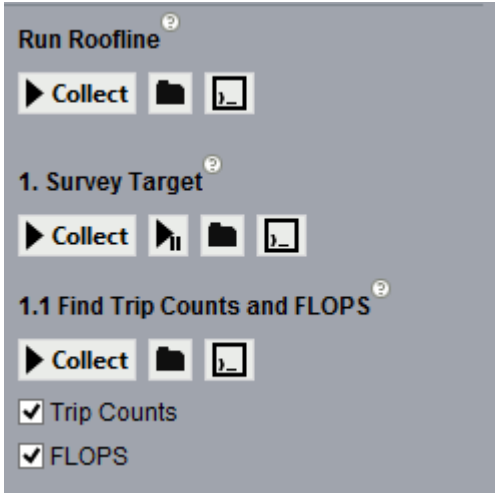


Which loops should we optimize?

- A and G have the biggest impact & biggest gap
- B has room to improve, but will have less impact
- E and H are perfectly optimized already

Roofline tutorial video

Getting Roofline data in Intel®Advisor



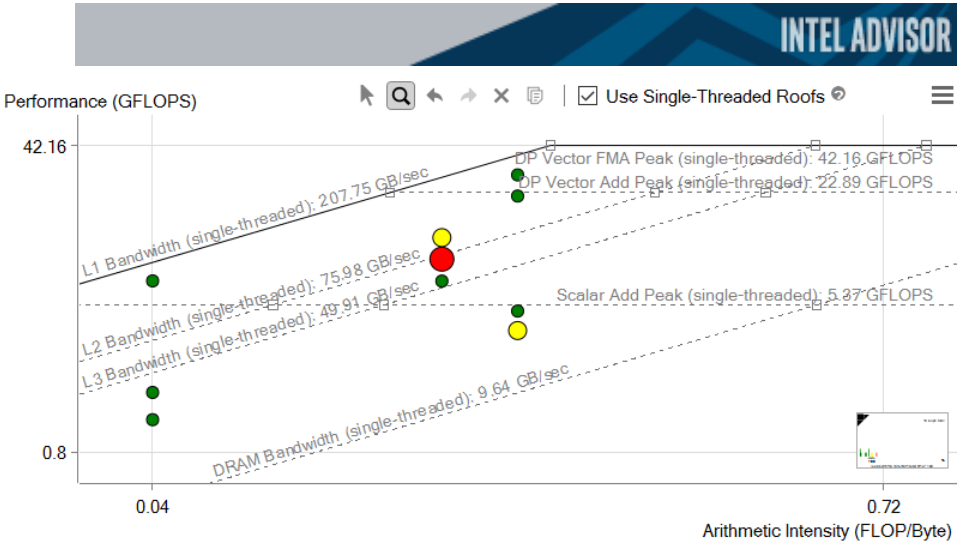
FLOP/S **= #FLOP/Seconds**	**Seconds**	**#FLOP** **- Mask Utilization** **- #Bytes**
Step 1: Survey - Non intrusive. *Representative* - Output: Seconds (+much more)	✓	
Step 2: Trip counts+FLOPS - Precise, instrumentation based - Physically count Num-Instructions - Output: #FLOP, #Bytes		✓

Find Effective Optimization Strategies

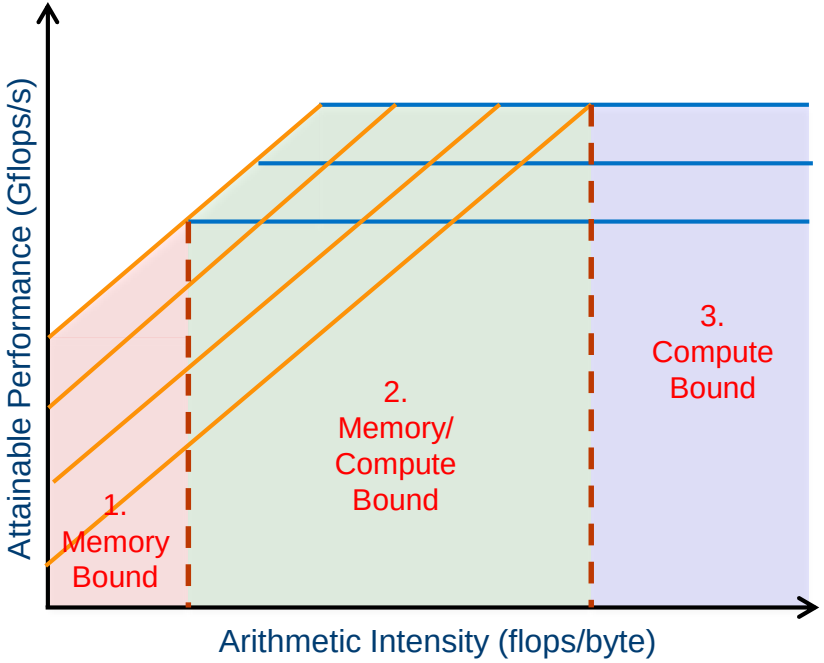
Intel Advisor: Cache-aware roofline analysis

Roofline Performance Insights

- Highlights poor performing loops
- Shows performance "headroom" for each loop
  - Which can be improved
  - Which are worth improving
- Shows likely causes of bottlenecks
- Suggests next optimization steps



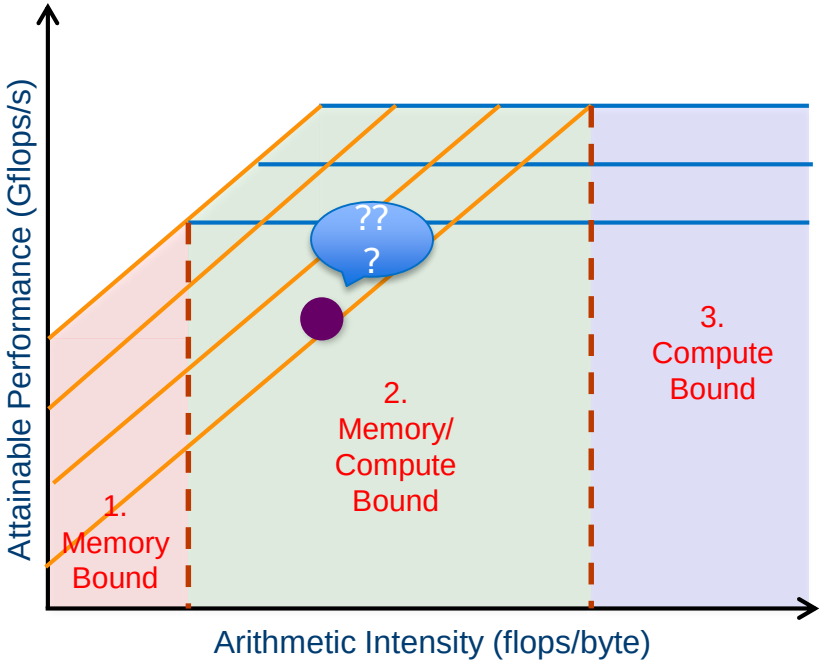
Is My Application Bound by a Memory Bandwidth or a Compute Peak?



Often it's a combination of the two

- Applications in area 1 are purely memory bandwidth bound
- Applications in area 3 are purely compute bound
- In area 2 we need more information

Ask Yourself “Why am I Here?” and “Where am I going?”



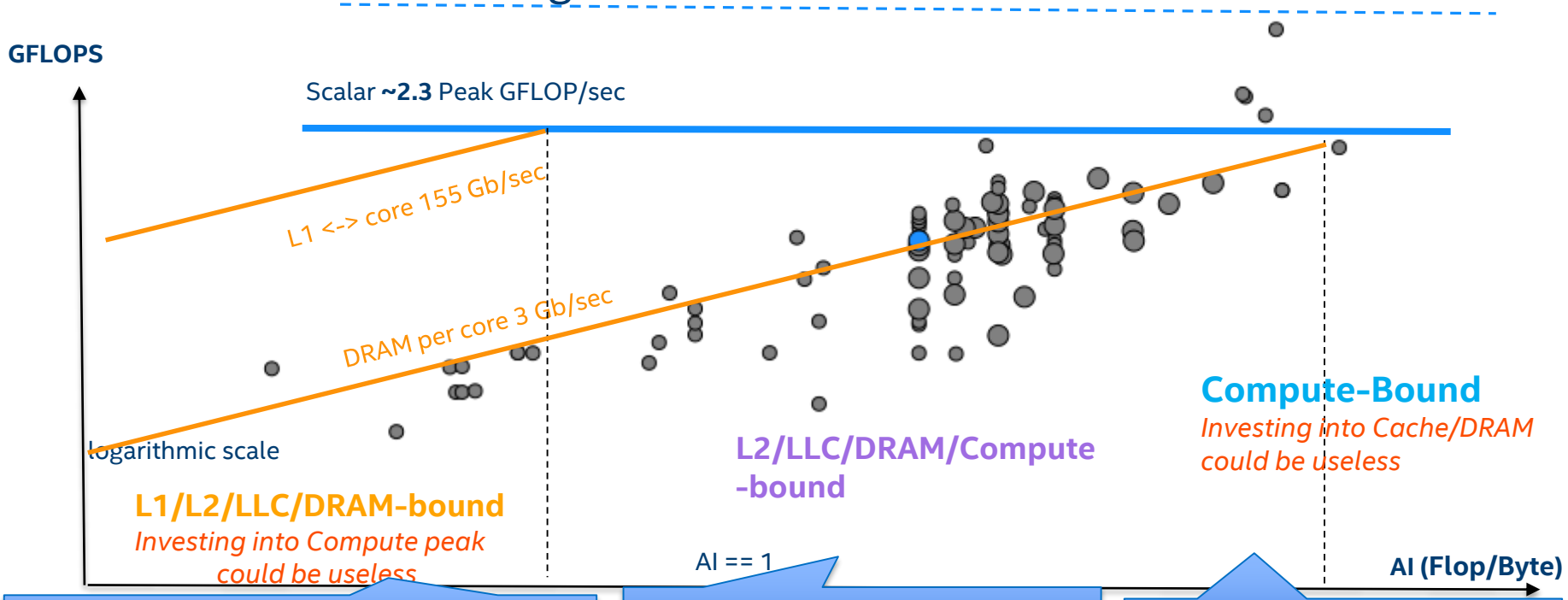
Usually, it is more complicated…

You won’t be on any ceiling. Or if you are, it is kind of coincidence.

BUT - asking the questions
“why am I not on a higher ceiling?”
and “what should I do to reach it?”
is always productive.

Perform the right optimization for your region

Roofline: characterization regions



Optimize memory (cache blocking, etc)

Gray area (need more data to determine right strategy)

Optimize compute (threading, vectorization)

WHAT'S NEW 2018

Intel® Advisor 2018 What's New

Hierarchical Roofline (Experimental)

MKL Summary

Python API

Cache simulation (Experimental)

Hierarchical (top-down) Roofline: new in 2018 release


```
export ADVIXE_EXPERIMENTAL=roofline_ex
```

Hierarchical Roofline (based on stacks w/ FLOPS)

```
> source advixe-vars.sh
> export ADVIXE_EXPERIMENTAL=roofline_ex
```

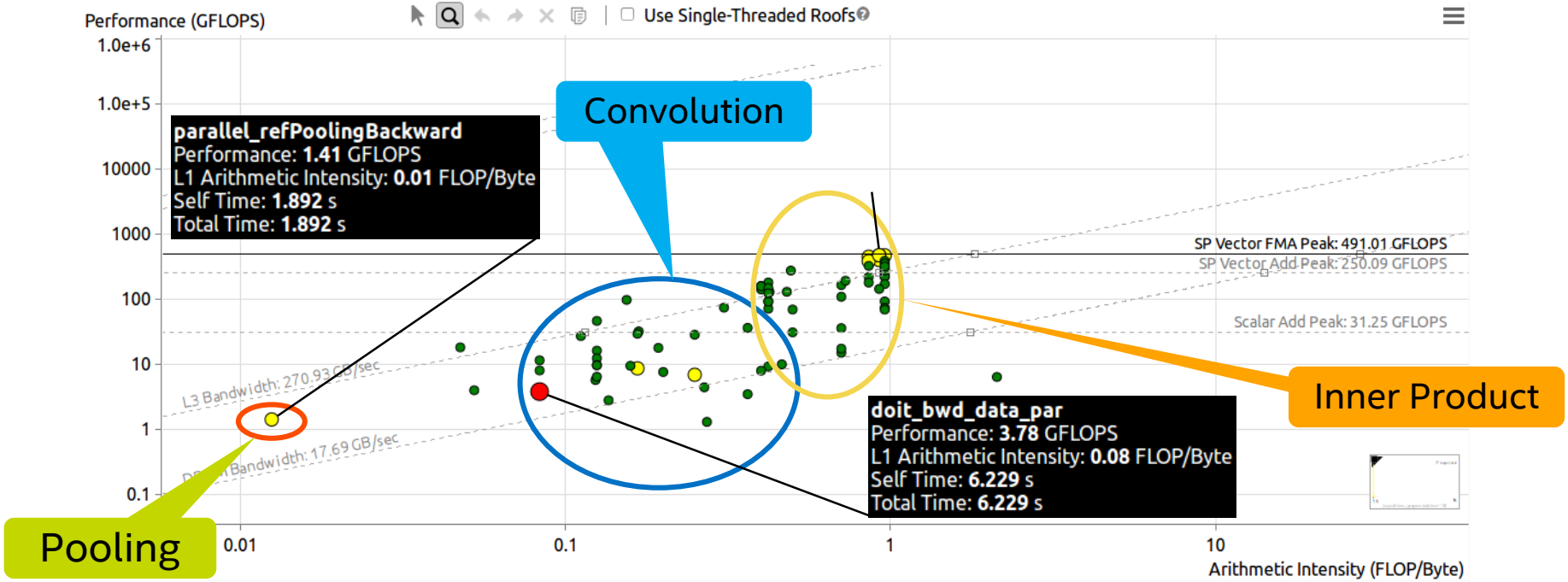
```
> advixe-cl --collect survey --project-dir ./your_project -- <your-executable-with-parameters>
```

2nd pass
Obtain #FLOP count:
>>5x overhead

```
> advixe-cl --collect tripcounts -flops-and-masks -callstack-flops --project-dir ./your_project -- <your-executable-with-parameters>
```

```
> export ADVIXE_EXPERIMENTAL=roofline_ex
> advixe-gui ./your_project
```

Roofline in Action: neural networks profiling



Inefficient implementation for pooling and convolutional layers

Get insights on how effectively you are using MKL

MKL summary

Elapsed time: 62.49s | Vectorized | Not Vectorized | MKL | FILTER: All Modules | All Sources

Summary | Survey & Roofline | Refinement Reports

Program metrics

Elapsed Time	62.49s		
Vector Instruction Set	AVX2, AVX, SSE2, SSE	Number of CPU Threads	72
Total GFLOP Count	549.89	Total GFLOPS	8.80
Total Arithmetic Intensity	0.10		

Loop metrics

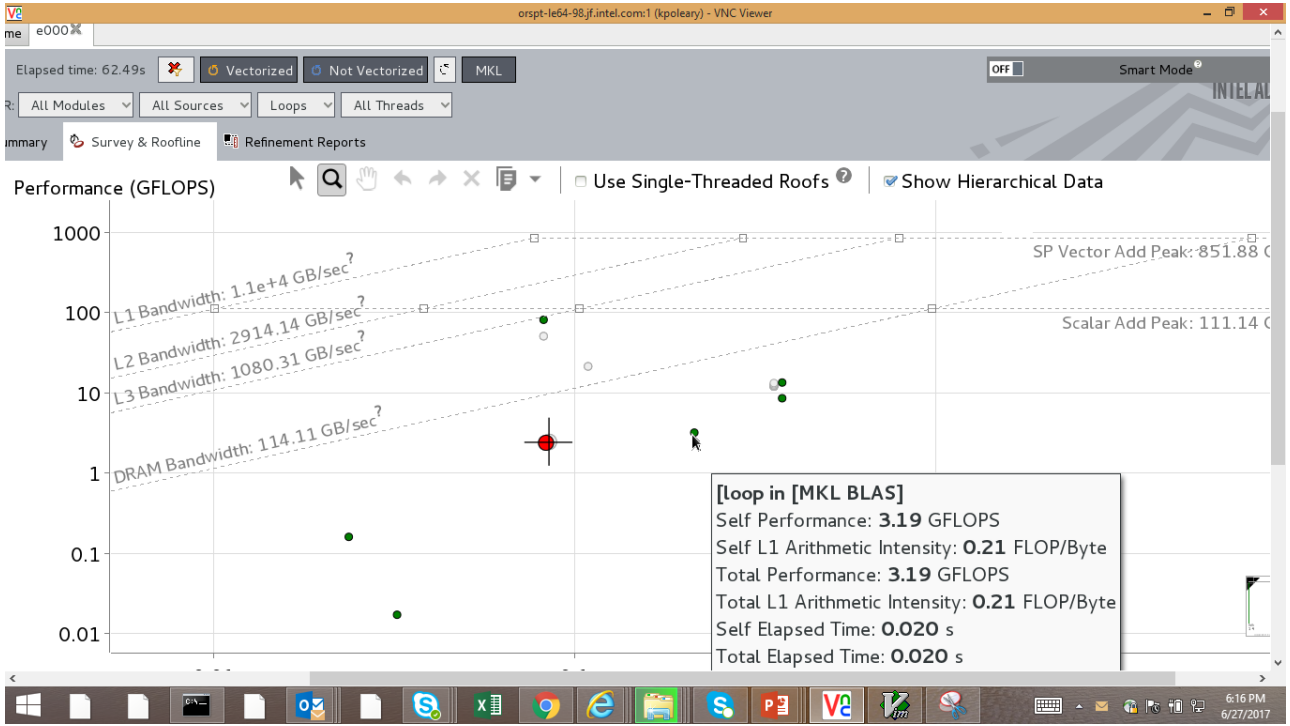
Metrics	Total		User		MKL
Total CPU time	3604.58s	100.0%	3593.13s	99.7%	11.45
Time in **10** vectorized loops	162.29s		156.18s		6.11s
Time in scalar code	3442.29s	95.5%	3436.95s	95.3%	5.34s
Total GFLOP Count	549.89	100.0%	412.32	75.0%	137.5
Total GFLOPS	8.80		6.66		245.6

See how effectively you are using MKL!

See GFLOPS for user and MKL code

Also see how much time you are spending in MKL relative to total program time.

See MKL code on the Roofline chart



Drill down into MKL loop details

Verify vectorization and memory access patterns

Elapsed time: 62.49s | Vectorized | Not Vectorized | MKL | OFF Smart Mode

FILTER: All Modules | All Sources | Loops | All Threads

Summary | Survey & Roofline | Refinement Reports

ROOFLINE

Function Call Sites and Loops	Total GFLOPS	Total AI	Trip Counts: Average	Trip Counts: Call Count	Instruction Set Analysis: Traits	Instruction Set Analysis: Data Types	Instruction Set Analysis: Numb...	Advanced
[loop in matmult_naive at matmult.cpp:18]	2.384	0.083	2048	16777216		Float64	3	Unrolled by 2
[loop in matmult_transposed at matmult.cpp:30]			512	16777216		Float32; Fl.	5	Unrolled by 4
[loop in matmult_blocked at matmult.cpp:48]			16	536870912		Float32; Fl.	5	Unrolled by 4
[loop in [MKL BLAS]	13.430	0.375	34	7203966	FMA	Float64	16	
[loop in matmult_blocked at matmult.cpp:44]			129	4194304	Unpacks	Float64; ...	5	
[loop in [MKL BLAS]	3.556	0.375	14	6860920	FMA	Float64	16	
[loop in [MKL BLAS]	12.373	0.361	28	257796		Float64; ...	16	

Source | Top Down | Code Analytics | Assembly | Recommendations | Why No Vectorization?

Function Call Sites and Loops	Total Time %	Total Time	Self Time	Type	Why No Vectorization?	Vectorized Lo...	
						Vecto...	VL (V
[MKL BLAS]@avx2_dgemm_k...	0.2%	6.212s	0.000s	Function			
[MKL BLAS]@avx2_dgem...	0.2%	6.019s	0.000s	Function			

Access the Intel® Advisor database using Python

Python api

- You can now access the Intel® Advisor database using our new Python API
- We have provided several reference examples on how to use this new functionality.

```
> source advixe-vars.sh

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

> advixe-cl --collect tripcounts -flops-and-masks -callstack-flops --project-dir ./your_project -- <your-executable-with-parameters>

> python /opt/intel/advisor_2018/pythonapi/joined.py ./your_project >& report.txt

>
```

Flexible way to report on useful program metrics

Over 500 metric elements can be displayed. (See column.txt)

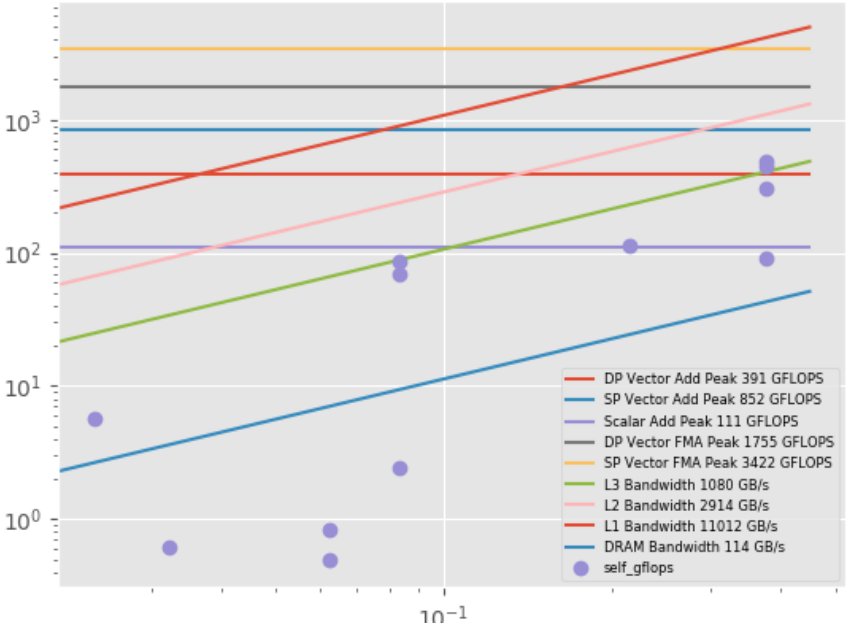
We also provide a way to generate customizable html reports

python ./to_html.py ./adv

№	access_pattern	address_distanc	architecture	average_trip_co	cache_line_utiliz	call_count	cfg_index_modif	cfg_ju
8			2				0	0
9			2	34		7.20397e+06	0	0
10			2	129		4.1943e+06	0	1
11			2	14		6.86092e+06	0	0

Generate a Roofline chart using the Python api

- **python /opt/intel/advisor_2018/pythonapi/roofline.py ./your_project**



Experiment with how you using utilizing cache

Experimental cache simulation feature

```
> source advixe-vars.sh
> export ADVIXE_EXPERIMENTAL=cachesim
```

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

Select your loops of interest using the mark-up-list (you can generate this using the Advisor GUI)

```
> advixe-cl --collect map -mark-up-list=4 --project-dir ./your_project -- <your-executable-with-parameters>
```

```
> python cache.py ./your_project
```

Cache simulation setting in Project properties

Advanced

Suppression mode: Show problems in system modules
Suppress problems in system modules

Report stack variables

Report heap allocated variables

Enable CPU cache simulation (experimental)

Cache associativity: 8

Cache sets: 4096

Cache simulation

Model how effectively you are utilizing cache

Site: loop_site_9

Location: loop_site_9

File Line: 69

Cache model settings:

Associativity = 8

Sets = 4096

Cache model results:

Writes = 46

Reads = 92

Read misses = 50

Evicted cache lines utilization:

Average utilization = 6.25%

Bytes used | Evicted lines

4 | 38

SUMMARY

Call to Action

Modernize your Code

- To get the most out of your hardware, you need to modernize your code with vectorization and threading.
- Taking a methodical approach such as the one outlined in this presentation, and taking advantage of the powerful tools in Intel® Parallel Studio XE, can make the modernization task dramatically easier.
  - Download the latest here: https://software.intel.com/en-us/intel-parallel-studio-xe
  - The Professional and Cluster Edition both include Advisor
- Join the 2018 beta of Intel Parallel Studio XE to get the latest version
- Send e-mail to vector_advisor@intel.com to get the latest information on some exciting new capabilities that are currently under development.

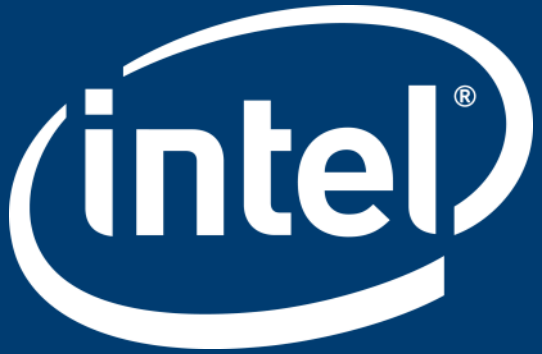
Legal Disclaimer & Optimization Notice

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.

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