HPC codes **modernization** using **vector** and threading **parallelism – part 2 (tools)**

Zakhar A. Matveev, PhD,

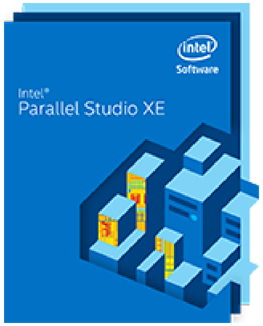
Intel Russia, Intel Software and Services Group

May‘ 2015

Code modernization:
Intel® Parallel Studio XE 2016 Beta

Intel® Parallel Studio XE

Faster code faster!



Vectorizing **Compiler**
Squeeze all the performance out of the latest instruction set

Threaded Performance **Libraries**
Pre-vectorized, pre-threaded, pre-optimized

Vectorization Optimization and Thread Prototyping
Data driven design tools help you vectorize & thread effectively

High Level **Parallel Models**
Productive solutions for thread, process & vector parallelism

Parallel Performance **Profilers**
Quickly discover bottlenecks and tune for high performance

Threading **Inspector**
Find and debug non-deterministic threading errors

Download Today

Google:
"Intel Parallel Studio 2016"

Or go directly to:
https://software.intel.com/en-us/articles/intel-parallel-studio-xe-2016-beta

Optimization Notice

Intel® Parallel Studio XE 2016 Suites

Vectorization – Boost Performance By Utilizing Vector Instructions / Units

- Intel® Advisor XE - ***Vectorization Advisor*** identifies new vectorization opportunities as well as improvements to existing vectorization and highlights them in your code. It makes actionable coding recommendations to boost performance and estimates the speedup.

Scalable MPI Analysis– Fast & Lightweight Analysis for 32K+ Ranks

- Intel® Trace Analyzer and Collector add ***MPI Performance Snapshot*** feature for easy to use, scalable MPI statistics collection and analysis of large MPI jobs to identify areas for improvement

Big Data Analytics – Easily Build IA Optimized Data Analytics Application

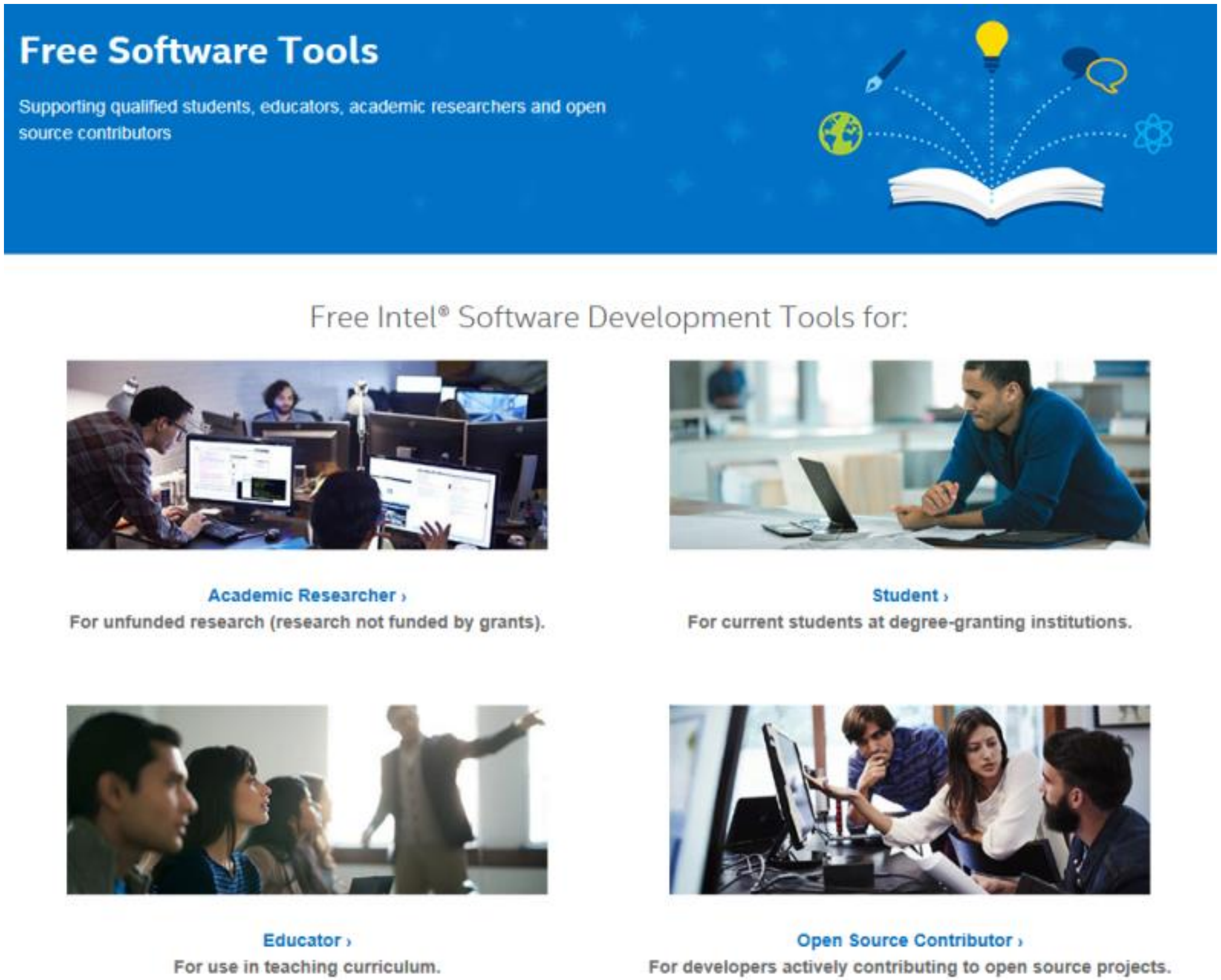
- Intel® Data Analytics Acceleration Library (Intel® DAAL) will help data scientists speed through big data challenges with optimized IA functions

Standards – Scaling Development Efforts Forward

- Supporting the evolution of industry standards of ***OpenMP*, MPI, Fortran and C++*** Intel® Compilers & performance libraries

Free Intel® Software Development Tools:

https://software.intel.com/en-us/qualify-for-free-software



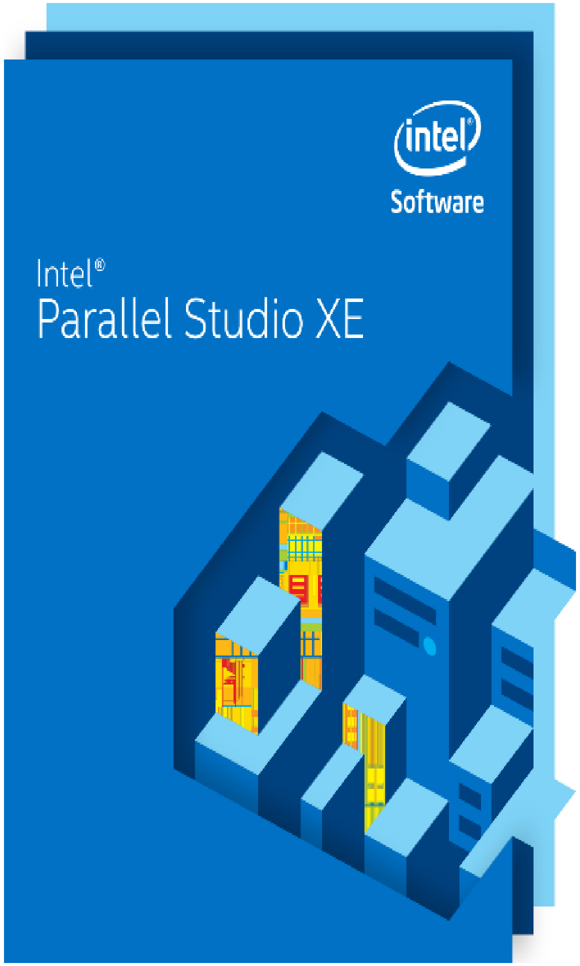
Beta 2016 program : Call to Action

Participate in the Beta Program today!

- Register at **bit.ly/psxe2016beta**
- Or simply send e-mail to **vector_advisor@intel.com**

Submit Feedback via Intel® Premier Support

Tell us about your experiences using the Intel® Parallel Studio XE 2016 Beta



Intel® Advisor XE

Vectorization Optimization and Thread Prototyping

Data Driven Code Modernization and Optimization For Vector and Threading Parallelism Intel® Advisor XE – Vectorization Advisor

In Beta Sign-up!

Have you:

- Recompiled with AVX2, but seen little benefit?
- Wondered where to start adding vectorization?
- Recoded intrinsics for each new architecture?
- Struggled with cryptic compiler vectorization messages?

Breakthrough for vectorization design

- What is blocking vectorization and why?
- Are my loops vector friendly?
- Will reorganizing data increase performance?
- Is it safe to just use pragma simd?

Vectorization challenges

LLNL (Hornung, Keasler, 2013):

”Typical codes get less than 5% of their FP instructions SIMD-ized… multi-physics codes - have thousands of small loops, which are all important”

Efficient vectorization is sometimes challenging:

- “Poor” baselines : solid gap
  - “Ninja gap” as well
- Industry code complexities
  - Vector productivity problem: “thousands of loops”
  - Need to know both static and dynamic code characteristics
- Some codes: demand for expensive data layout reorganization

“Vectorization Advisor”

1. *“All the data you need in one place”*.

Leverages Intel Compiler diagnostics, performance data and ISA statistics.

2. Detects “hot” un-vectorized or “under vectorized” loops. Identifies what is blocking efficient vectorization and why

3. Identify performance penalties and recommends fixes for them (including OpenMP4.x)

4. Memory layout analysis: explore alternative data reorganizations

5. Increase the confidence that vectorization is safe

Beta program : started April’15.
Release: end of August ‘15.

Vectorization Advisor.

Assist code modernization for x86 SIMD

1. Compiler diagnostics + Performance Data + SIMD efficiency information

Function Call Sites and Loops	Self Time
[loop in runCForallLambdaLoops]	0.094s
[loop in runCForallLambdaLoops]	0.140s
V [loop in std::_Complex_base<double,struct _C_double_complex>::i...	0.031s
Vectorized SSE; SSE2 loop processing Float32; Float64 data type	
Peeled loop; loop stmts were reordered	
[loop in std::basic_string<char,struct std::char_traits<char>,class std::allo...	0.000s
[loop in std::basic_string<char,struct std::char_traits<char>,class std::allo...	0.000s
[loop in std::num_put<char,class std::ostreambuf_iterator<char,struct st...	0.000s

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

2 Issue: Peeled/Remainder loop(s) present

...nel loop. Improve performance by moving ...nel loop. Read more at Vector Essentials.

...emory accesses in the source loop does not ...l the compiler your memory access is aligned.

...oat), 32);

3. "Accurate" Trip Counts: understand parallelism granularity and overheads

Trip Counts

Median	Min	Max	Call Count
101	101	101	12000000
3	3	3	1000000
101	101	101	2000000
1000000	1000000	1000000	1

4. Loop-Carried Dependency Analysis

Problems and Messages

ID		Type	Site Name	Sources	Modules	State
P1		Parallel site information	site2	dqtest2.cpp	dqtest2	✓ Not a problem
P2		Read after write dependency	site2	dqtest2.cpp	dqtest2	New
P3		Read after write dependency	site2	dqtest2.cpp	dqtest2	New
P4		Write after write dependency	site2	dqtest2.cpp	dqtest2	New
P5		Write after write dependency	site2	dqtest2.cpp	dqtest2	New
P6		Write after read dependency	site2	dqtest2.cpp	dqtest2	New
P7		Write after read dependency	site2	dqtest2.cpp; idle.h	dqtest2	New

5. Memory Access Patterns Analysis

Site Name	Site Function	Site Info	Loop-Carried Dependencies	Strides Distribution	Access Pattern
loop_site_203	runCRawLoops	runCRawLoops.cxx:1063	RAW:1	No information available	No information available
loop_site_139	runCRawLoops	runCRawLoops.cxx:622	No information available	39% / 36% / 25%	Mixed strides
loop_site_160	runCRawLoops	runCRawLoops.cxx:925	No information available	100% / 0% / 0%	All unit strides

Memory Access Patterns | Correctness Report

ID		Stride	Type	Source	Modules	Alignment
P22		0; 0; 1	Unit stride	runCRawLoops.cxx:637	lcals.exe	
635		j2 = (j2 & 64-1) ;				
636		p[ip][0] += y[i2+32];				
637		p[ip][1] += z[j2+32];				
638		i2 += e[i2+32];				
639		j2 += f[j2+32];				
P23		0; 0	Unit stride	runCRawLoops.cxx:638	lcals.exe	
P30		-1575; -63; -26; -25; -1; 0; 1; 25; 26; 63; 2164801	Variable stride	runCRawLoops.cxx:628	lcals.exe	
626		i1 &= 64-1;				
627		j1 &= 64-1;				
628		p[ip][2] += b[j1][i1];				

Optimization Notice

Data-Driven Threading Design

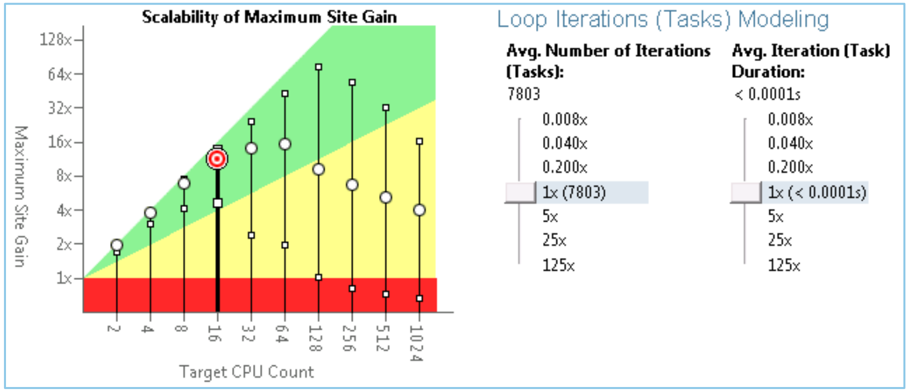
Intel® Advisor XE – Thread Prototyping

Have you:

- Tried threading an app, but seen little performance benefit?
- Hit a "scalability barrier"? Performance gains level off as you add cores?
- Delayed a release that adds threading because of synchronization errors?

Breakthrough for threading design:

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



Add Parallelism with Less Effort, Less Risk and More Impact

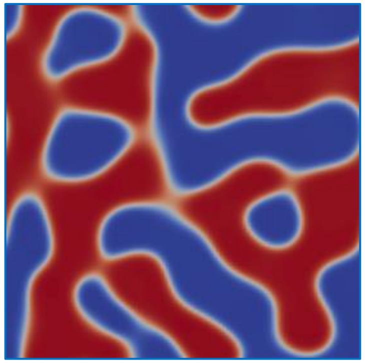
http://intel.ly/advisor-xe

Vectorization Advisor in action

Live Case Study with “DL-MESO”

DL-MESO

Hartree Centre
Science & Technology Facilities Council



Computational fluid dynamics engine

- New mesoscopic simulation engine
- Applicable for problems such as inkjet printing and steel production
- Lattice Boltzman Equation

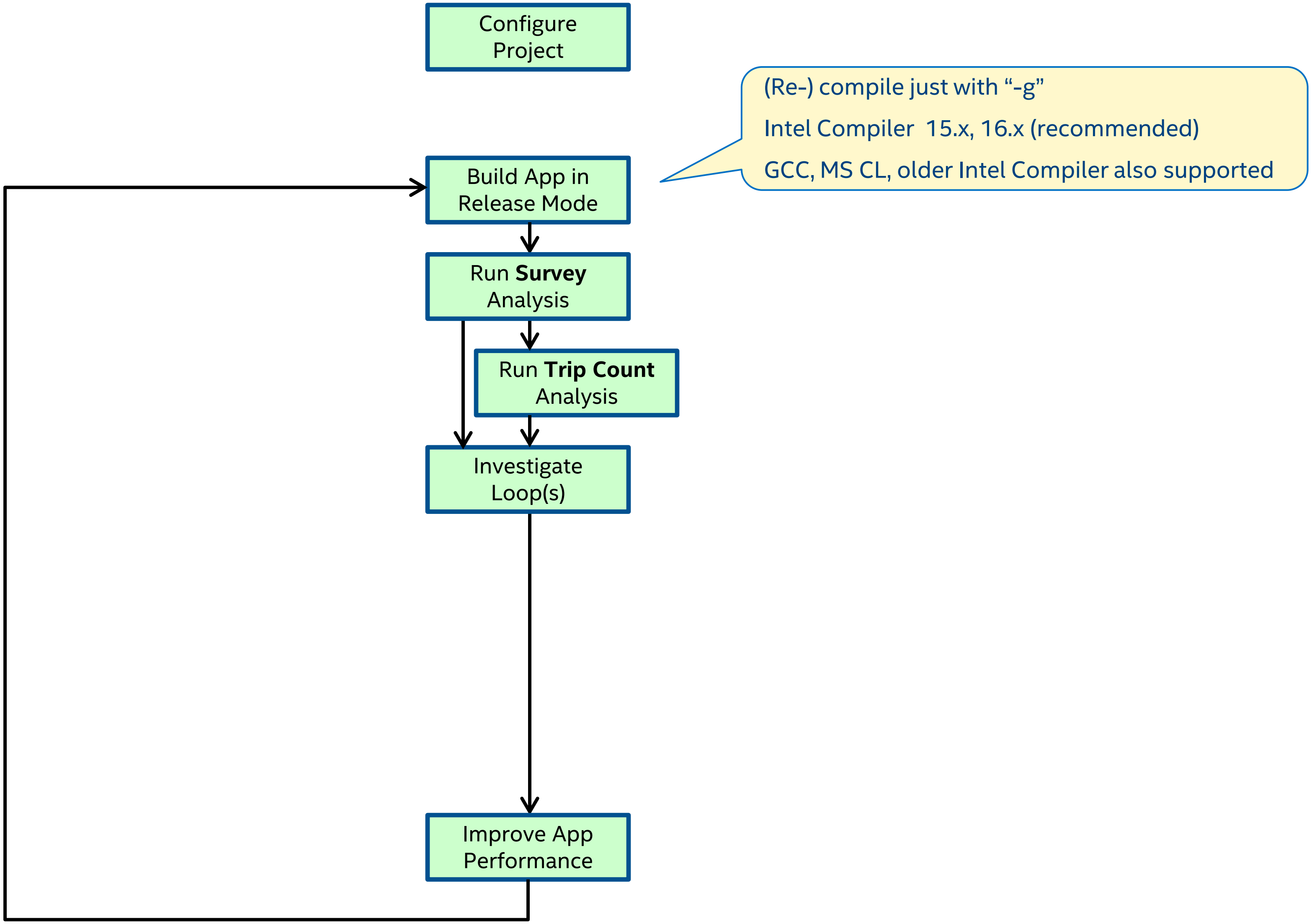
Developed by EPSRC CPP5

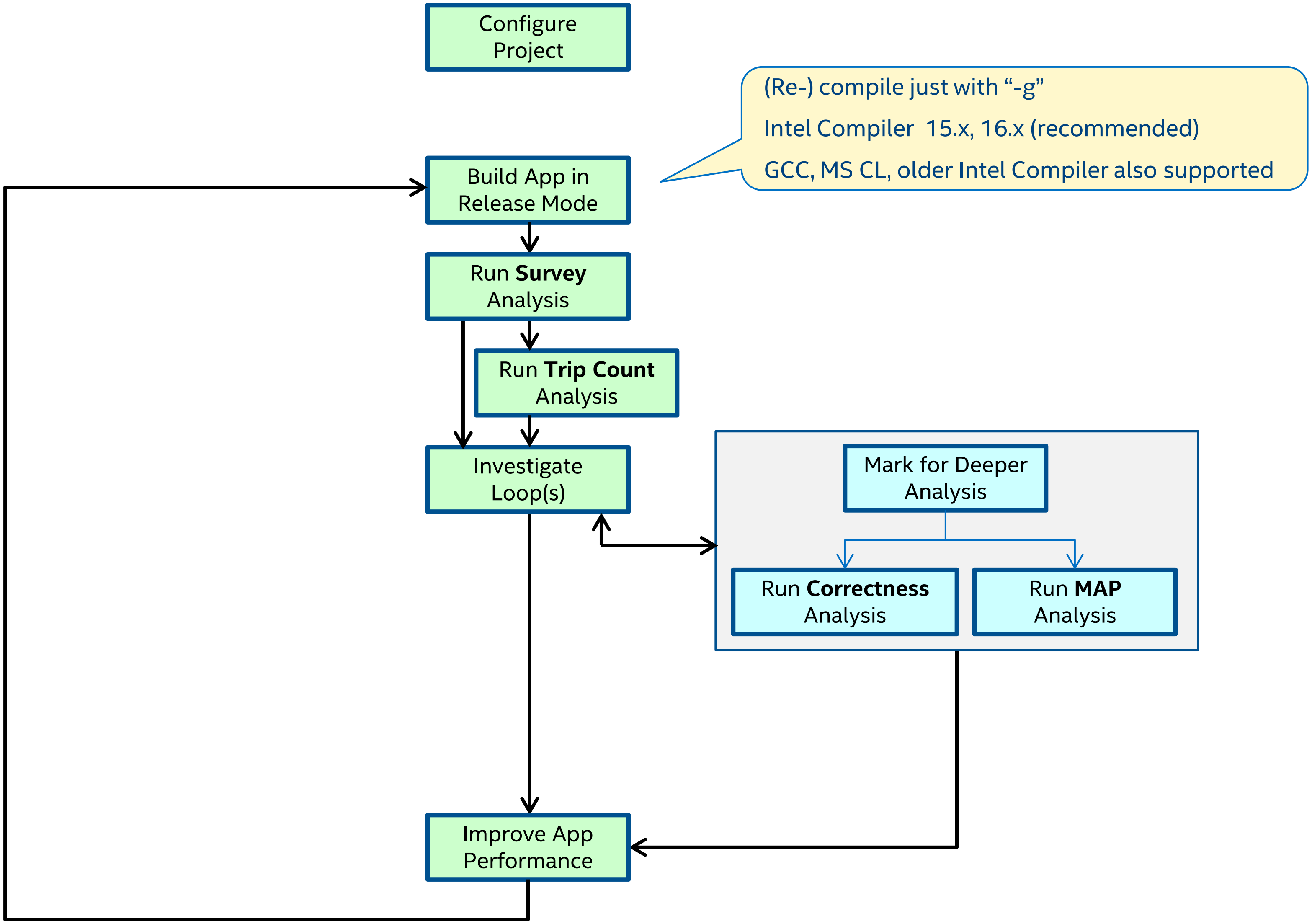
- including Hartree, Oxford, Imperial College
- Michael Seaton at Hartree as major contributor

Workload characteristics:

- "Flat profile", many small kernels
- Profiles are very diverse depending on input datasets

0.
Workflow


Configure Project
Build App in Release Mode
Run **Survey** Analysis
Run **Trip Count** Analysis
Investigate Loop(s)
Improve App Performance
(Re-) compile just with “-g”
Intel Compiler 15.x, 16.x (recommended)
GCC, MS CL, older Intel Compiler also supported



Configure Project
Build App in Release Mode
Run **Survey** Analysis
Run **Trip Count** Analysis
Investigate Loop(s)
Mark for Deeper Analysis
Run **Correctness** Analysis
Run **MAP** Analysis
Improve App Performance
(Re-) compile just with "-g"
Intel Compiler 15.x, 16.x (recommended)
GCC, MS CL, older Intel Compiler also supported

1.
The Right Data At Your Fingertips

Optimization Notice

1. Compiler diagnostics + Performance Data + SIMD efficiency information

Function Call Sites and Loops	Self Time	Total Time			Compiler Vectorization: Loop Type	Compiler Vectorization: Why No Vectorization?
[loop in runCForallLambdaLoops]	0.094s	0.094s			Scalar	vector dependence prevents vector...
[loop in runCForallLambdaLoops]	0.140s	3.744s			Scalar	inner loop was already vectorized
V [loop in std::_Complex_base<double,struct _C_double_complex>::i...	0.031s	0.031s			Vectorized (Body)	
Vectorized SSE; SSE2 loop processing Float32; Float64 data type(s) having Divisions; Square Roots operations Peeled loop; loop stmts were reordered						
[loop in std::basic_string<char,struct std::char_traits<char>,class std::allo...	0.000s	544.0...			Scalar	nonstandard loop is not a vectoriza...
[loop in std::basic_string<char,struct std::char_traits<char>,class std::allo...	0.000s	544.0...			Scalar	nonstandard loop is not a vectoriza...
[loop in std::num_put<char,class std::ostreambuf_iterator<char,struct st...	0.000s	0.234s			Scalar	nonstandard loop is not a vectoriza...

Hot Loops: optimize one by one

Loops		Vector Issues	Self Time	Total Time	Loop Type
[loop at lbpSUB.cpp:1280 in fPropagationSwap]		1 Assumed depend ...	1,691s	1,691s	Scalar
[loop at lbpSUB.cpp:761 in fGetEquilibriumF]		2 Data type conver ...	1,384s	1,384s	Scalar
[loop at lbpGET.cpp:281 in fGetSpeedSite]		2 Data type conversi ...	1,027s	1,027s	Scalar
[loop at lbpBGK.cpp:30 in fSiteFluidCollisi nBGK]			0,599s	0,599s	Scalar
[loop in I10_OUTPUT]			0,150s	0,150s	Sca
[loop at lbpBGK.cpp:21 in fSiteFluidC onBGK]			0,119s	2,479s	Sc
[loop at lbpSUB.cpp:1264 in fProp Swap]		1 Assumed depend	0,090s	0,622s	S
[loop at lbpGET.cpp:42 in fGetC ite]		3 Ineffective peele ...	0,080s	0,080s	rized: Expand
[loop at lbpSUB.cpp:1276 in f nSwap]			0,060s	1,751s	r
[loop at lbpGET.cpp:152 in]		2 Ineffective ed ...	0,059s	0,059s	orized: Expand

Will only work through first 2 loops with max impact

Both loops were not auto-vectorized

Relatively “flat” profile

Quickly focus on loops which really matter

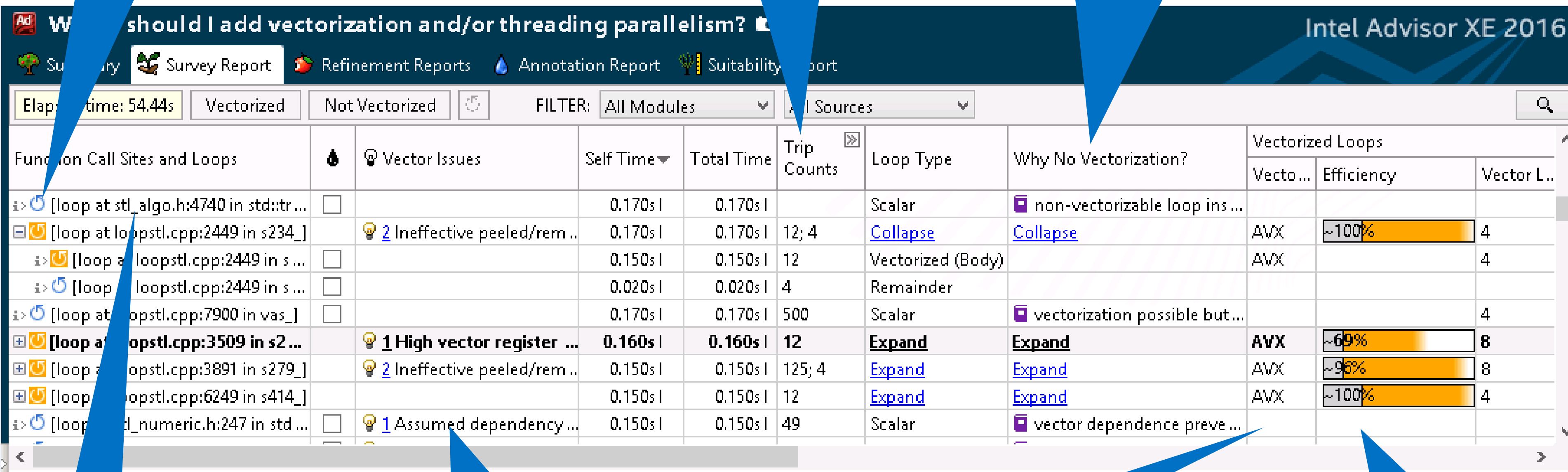
The Right Data At Your Fingertips

Get all the data you need for high impact vectorization

Am I vector?

Trip Counts

What prevents vectorization?



Focus on hot loops

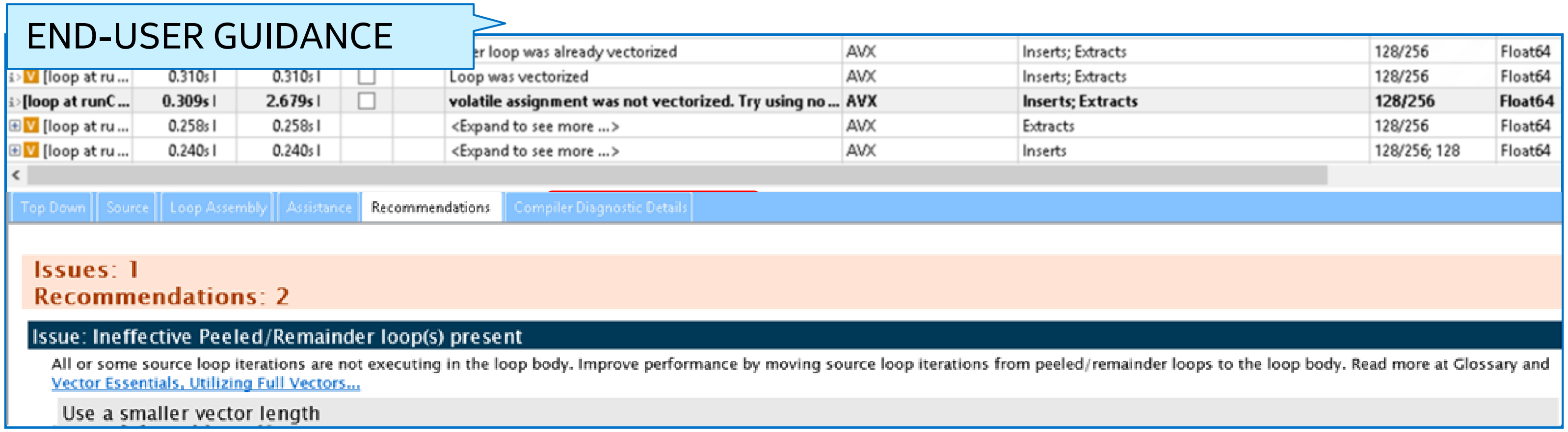
What vectorization issues do I have?

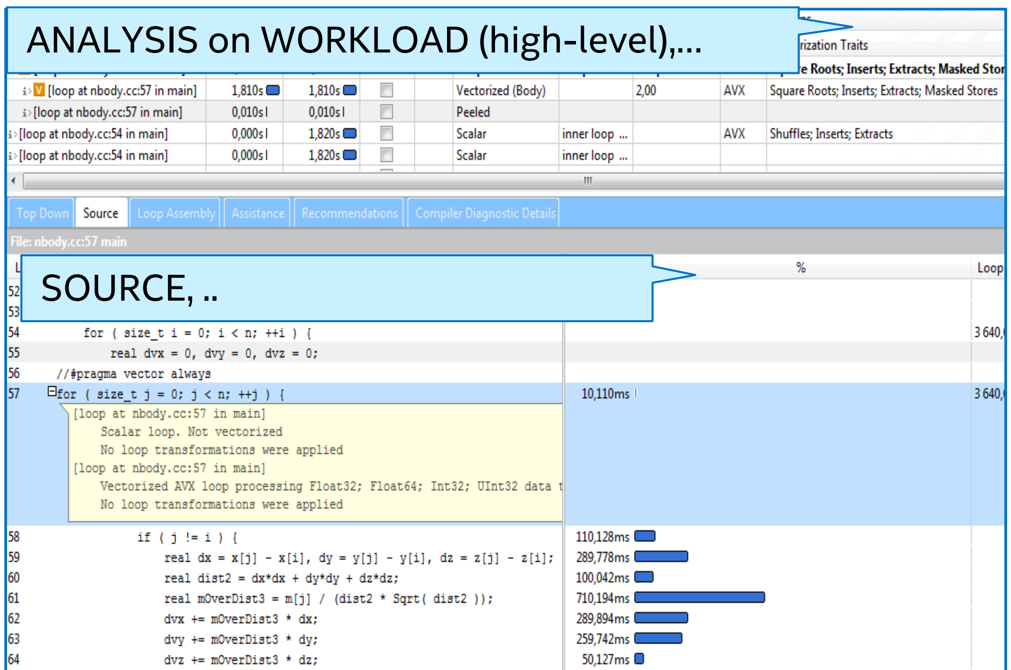
Which Vector instructions are being used?

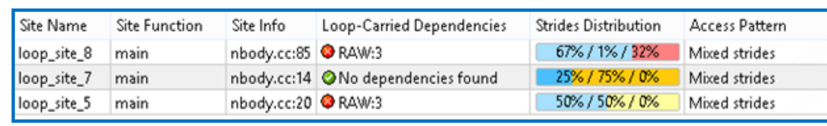
How efficient is the code?

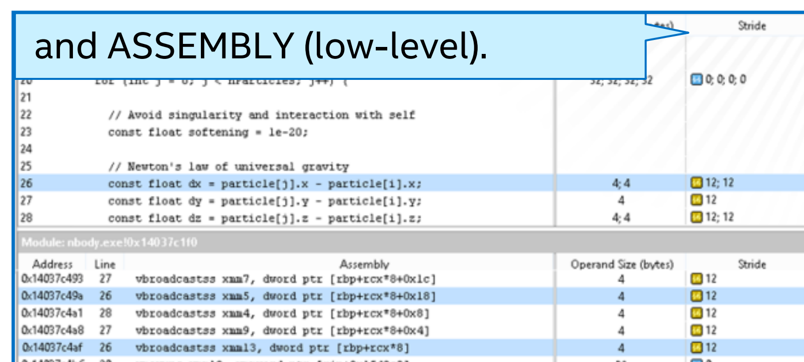
Optimization Notice

Assist user at different LoD and perspectives









2.
Is it safe to vectorize: Tough problem #1 for not yet vectorized codes.

1. Compiler diagnostics + Performance Data + SIMD efficiency information

Function Call Sites and Loops	Self Time	Total Time			Compiler Vectorization: Loop Type	Compiler Vectorization: Why No Vectorization?
[loop in runCForallLambdaLoops]	0.094s	0.094s			Scalar	vector dependence prevents vector...
[loop in runCForallLambdaLoops]	0.140s	3.744s			Scalar	inner loop was already vectorized
[loop in std::_Complex_base<double,struct _C_double_complex>::i...	0.031s	0.031s			Vectorized (Body)	
Vectorized SSE; SSE2 loop processing Float32; Float64 data type(s) having Divisions; Square Roots operations Peeled loop; loop stmts were reordered						
[loop in std::basic_string<char,struct std::char_traits<char>,class std::allo...	0.000s	544.0...			Scalar	nonstandard loop is not a vectoriza...
[loop in std::basic_string<char,struct std::char_traits<char>,class std::allo...	0.000s	544.0...			Scalar	nonstandard loop is not a vectoriza...
[loop in std::num_put<char,class std::ostreambuf_iterator<char,struct st...	0.000s	0.234s			Scalar	nonstandard loop is not a vectoriza...

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

2 Issue: Peeled/Remainder loop(s) present

8 All or some source loop iterations are not executing in the kernel loop. Improve performance by moving source loop iterations from peeled/remainder loops to the kernel loop. Read more at Vector Essentials, Utilizing Full Vectors...

Recommendation: Align memory access

Projected maximum performance gain: High

Projection confidence: Medium

The compiler created a peeled loop because one of the memory accesses in the source loop does not start at a data boundary. Align the memory access and tell the compiler your memory access is aligned. This example aligns memory using a 32-byte boundary:

```
float *array;
array = (float *)_mm_malloc(ARRAY_SIZE*sizeof(float), 32);

// Somewhere else
__assume_aligned(array, 32);
// Use array in loop
```

4. Loop-Carried Dependency Analysis

Problems and Messages

ID		Type	Site Name	Sources	Modules	State
P1		Parallel site information	site2	dqtest2.cpp	dqtest2	✓ Not a problem
P2		Read after write dependency	site2	dqtest2.cpp	dqtest2	New
P3		Read after write dependency	site2	dqtest2.cpp	dqtest2	New
P4		Write after write dependency	site2	dqtest2.cpp	dqtest2	New
P5		Write after write dependency	site2	dqtest2.cpp	dqtest2	New
P6		Write after read dependency	site2	dqtest2.cpp	dqtest2	New
P7		Write after read dependency	site2	dqtest2.cpp; idle.h	dqtest2	New

Data Dependencies – Tough Problem #1

Is it safe to force the compiler to vectorize?

Data dependencies

```
for (i=0;i<N;i++)       // Loop carried dependencies!
   A[i] = A[i-1]*C[i];// Need the ability to check if it
                      // it is safe to force the compiler
```

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**

Data Dependencies – Tough Problem #1

Dynamic check will ***know*** if indices overlap.

1)
```
fSwapPairM ( lbf[il*lbsitelength + l*lbsy.nq + m + half],
             lbf[ilnext*lbsitelength + l*lbsy.nq + m]);
```

Compiler Assumption:

[loop at lbpSUB.cpp:1280 in fPropagationSwap]	vector dependence prevents vectorization

2)
```
fSwapPairM ( lbf[il*lbsitelength + l*lbsy.nq + m + half],
             lbf[ilnext*lbsitelength + l*lbsy.nq + m]);
```

Compiler Assumption:

[loop at lbpSUB.cpp:1280 in fPropagationSwap]	vector dependence prevents vectorization

Both loops “equally bad” :
from static analysis/compilation “best knowledge”

Data Dependencies – Tough Problem #1

Dynamic check ***knows*** if memory accesses really overlap.

1)
```
fSwapPairM ( lbf[il*lbsitelength + l*lbsy.nq + m + half],
             lbf[ilnext*lbsitelength + l*lbsy.nq + m]);
```

[loop at lbpSUB.cpp:1280 in fPropagationSw ...	No dependencies found

2)
```
fSwapPairM ( lbf[il*lbsitelength + l*lbsy.nq + m + half],
             lbf[ilnext*lbsitelength + l*lbsy.nq + m]);
```

[loop at lbpSUB.cpp:1280 in fPropagationSw ...	RAW:1

	Read after write dependency

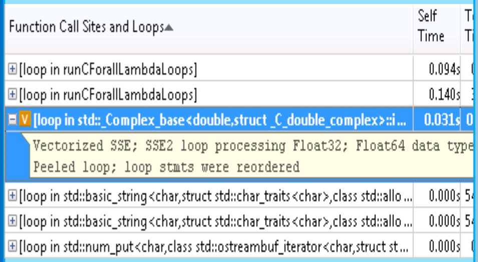
Correctness Analysis: confirm dependencies are **REAL**

2.

Any speed-up out of there?
Use SIMD to make your code faster, instead of slower.

Optimization Notice

1. Compiler diagnostics + Performance Data + SIMD efficiency information



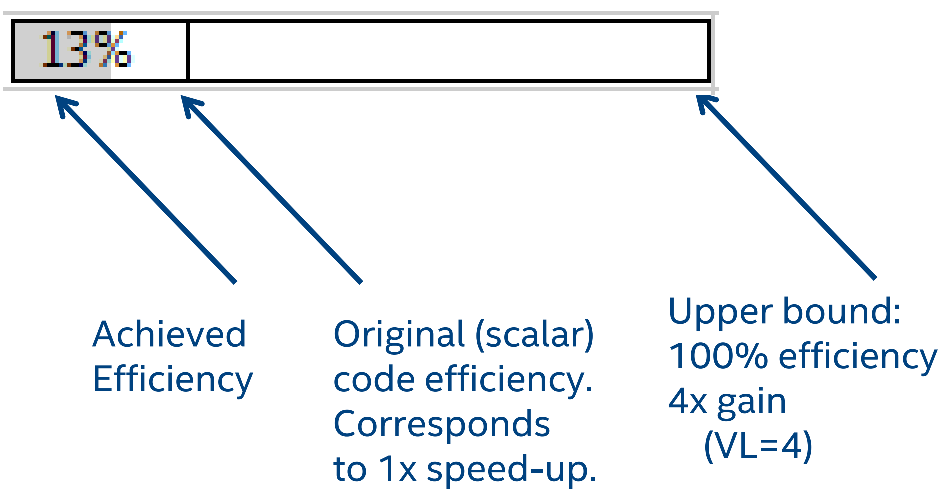
3. “Accurate” Trip Counts: understand parallelism granularity and overheads

Trip Counts			
Median	Min	Max	Call Count
101	101	101	12000000
3	3	3	1000000
101	101	101	2000000
1000000	1000000	1000000	1

Vector Efficiency: my performance thermometer
all the data in one place

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



- **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 “undervectorized” and why

Optimization Notice

3.
Tough problem #1 for already vectorized codes

Non-Contiguous Memory – Tough Problem #2

Potential to vectorize but may be inefficient

- Unit-Stride access to arrays

```
for (i=0;i<N;i++)
    A[i] = C[i]*D[i]; //Accessing array elements 1 by 1
```

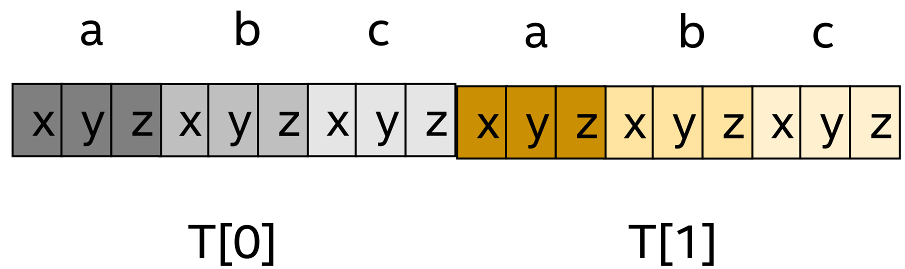
- Non-unit-stride (constant stride) access to arrays

```
for (i=0;i<N;i+=2)
    A[i] = C[i]*D[i]; //Incrementing "i" by 2: not unit stride
                      //Often indication of demand for AoS ->
                      // SoA conversion
```

- Indirect reference in a loop

```
for (i=0;i<N;i++)
    A[B[i]] = C[i]*D[i];//We have to decode B[i] to find out
                        //which element of A to reference
```

Object-oriented programming



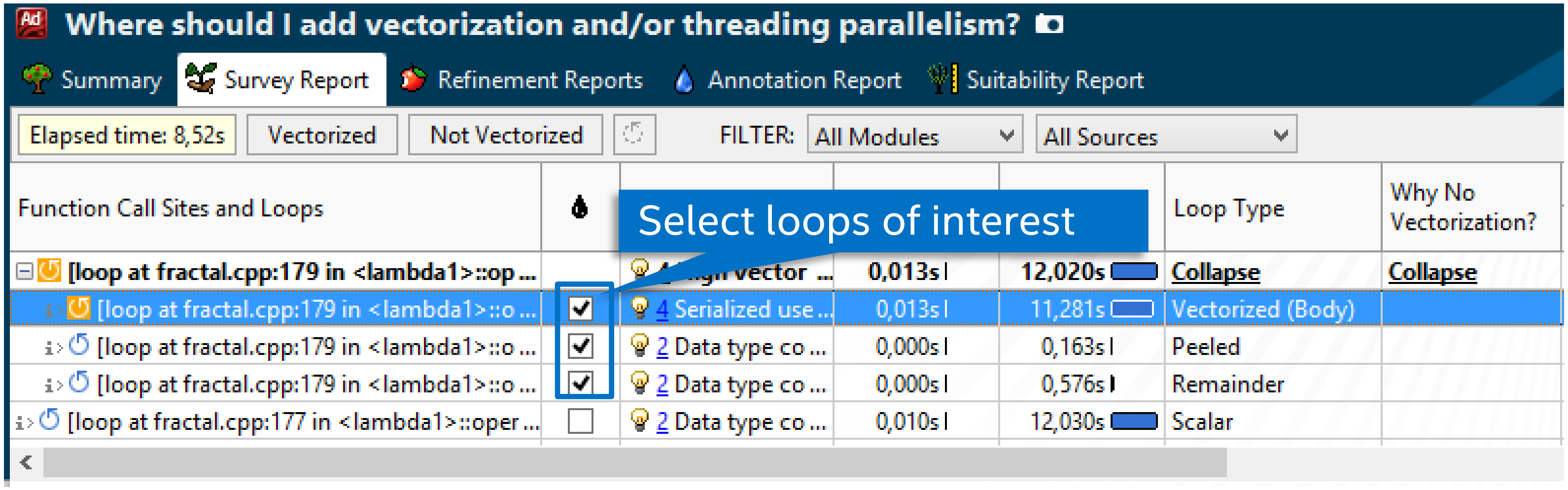

```
Class Point {float
x,y,z;}
Class Triangle {Point
a,b,c;}
Triangle T[100];
Point Cross( const Point& a, const Point& b ) {
    return Point( a.y*b.z-a.z*b.y, a.z*b.x-a.x*b.z,
a.x*a.y-a.y-b.x );
}

void ComputeNormals( Point normal[__restrict], const
Triangle p[], size_t n )
    for( size_t i=0; i<n; ++i )
        normal[i] = Cross( p[i].b-p[i].a, p[i].c-p[i].a );
}
```

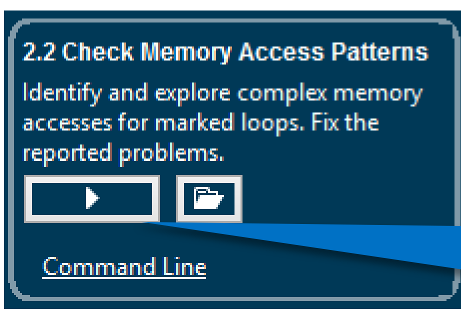
Object oriented programming may inhibit SIMD code generation

Improve Vectorization

Memory Access pattern analysis



Select loops of interest



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

1. Compiler diagnostics + Performance Data + SIMD efficiency information

Function Call Sites and Loops	Self Time	Total Time			Compiler Vectorization: Loop Type	Compiler Vectorization: Why No Vectorization?
[loop in runCForallLambdaLoops]	0.094s	0.094s			Scalar	vector dependence prevents vector...
[loop in runCForallLambdaLoops]	0.140s	3.744s			Scalar	inner loop was already vectorized
[loop in std::_Complex_base<double,struct _C_double_complex>::i...	0.031s	0.031s			Vectorized (Body)	
Vectorized SSE; SSE2 loop processing Float32; Float64 data type(s) having Divisions; Square Roots operations Peeled loop; loop stmts were reordered						
[loop in std::basic_string<char,struct std::char_traits<char>,class std::allo...	0.000s	544.0...			Scalar	nonstandard loop is not a vectoriza...
[loop in std::basic_string<char,struct std::char_traits<char>,class std::allo...	0.000s	544.0...			Scalar	nonstandard loop is not a vectoriza...
[loop in std::num_put<char,class std::ostreambuf_iterator<char,struct st...	0.000s	0.234s			Scalar	nonstandard loop is not a vectoriza...

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

2 | 8

Issue: Peeled/Remainder loop(s) present

All or some source loop iterations are not executing in the kernel loop. Improve performance by moving source loop iterations from peeled/remainder loops to the kernel loop. Read more at Vector Essentials, Utilizing Full Vectors...

Recommendation: Align memory access

Projected maximum performance gain: High
Projection confidence: Medium

The compiler created a peeled loop because one of the memory accesses in the source loop does not start at a data boundary. Align the memory access and tell the compiler your memory access is aligned. This example aligns memory using a 32-byte boundary:

```
float *array;
array = (float *)_mm_malloc(ARRAY_SIZE*sizeof(float), 32);

// Somewhere else
__assume_aligned(array, 32);
// Use array in loop
```

3. Loop-Carried Dependency Analysis

Problems and Messages

ID		Type	Site Name	Sources	Modules	State
P1		Parallel site information	site2	dqtest2.cpp	dqtest2	✓ Not a problem
P2		Read after write dependency	site2	dqtest2.cpp	dqtest2	New
P3		Read after write dependency	site2	dqtest2.cpp	dqtest2	New
P4		Write after write dependency	site2	dqtest2.cpp	dqtest2	New
P5		Write after write dependency	site2	dqtest2.cpp	dqtest2	New
P6		Write after read dependency	site2	dqtest2.cpp	dqtest2	New
P7		Write after read dependency	site2	dqtest2.cpp; idle.h	dqtest2	New

4. Memory Access Patterns Analysis

Site Name	Site Function	Site Info	Loop-Carried Dependencies	Strides Distribution	Access Pattern
loop_site_203	runCRawLoops	runCRawLoops.cxx:1063	RAW:1	No information available	No information available
loop_site_139	runCRawLoops	runCRawLoops.cxx:622	No information available	39% / 36% / 25%	Mixed strides
loop_site_160	runCRawLoops	runCRawLoops.cxx:925	No information available	100% / 0% / 0%	All unit strides

Memory Access Patterns | Correctness Report

ID		Stride	Type	Source	Modules	Alignment
P22		0; 0; 1	Unit stride	runCRawLoops.cxx:637	lcals.exe	

```
635    j2 = ( j2 & 64-1 ) ;
636    p[ip][0] += y[i2+32];
637    p[ip][1] += z[j2+32];
638    i2 += e[i2+32];
639    j2 += f[j2+32];
```

ID		Stride	Type	Source	Modules	Alignment
P23		0; 0	Unit stride	runCRawLoops.cxx:638	lcals.exe	
P30		-1575; -63; -26; -25; -1; 0; 1; 25; 26; 63; 2164801	Variable stride	runCRawLoops.cxx:628	lcals.exe	

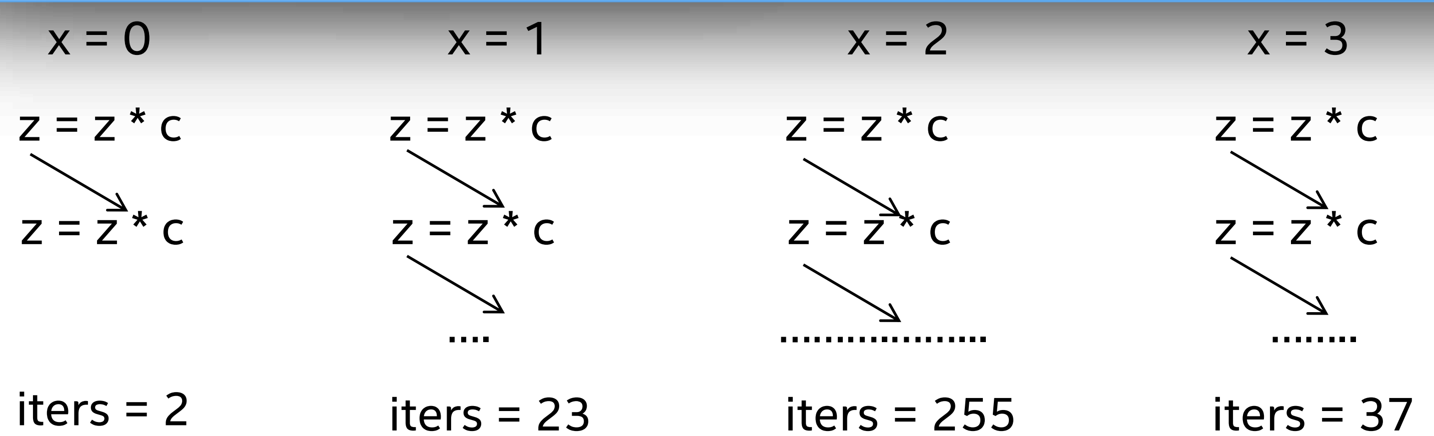
```
626    i1 &= 64-1;
627    j1 &= 64-1;
628    p[ip][2] += b[j1][i1];
```

4.
It’s time for explicit parallelism choices to make your code faster, not slower.

Example of Outer Loop Vectorization

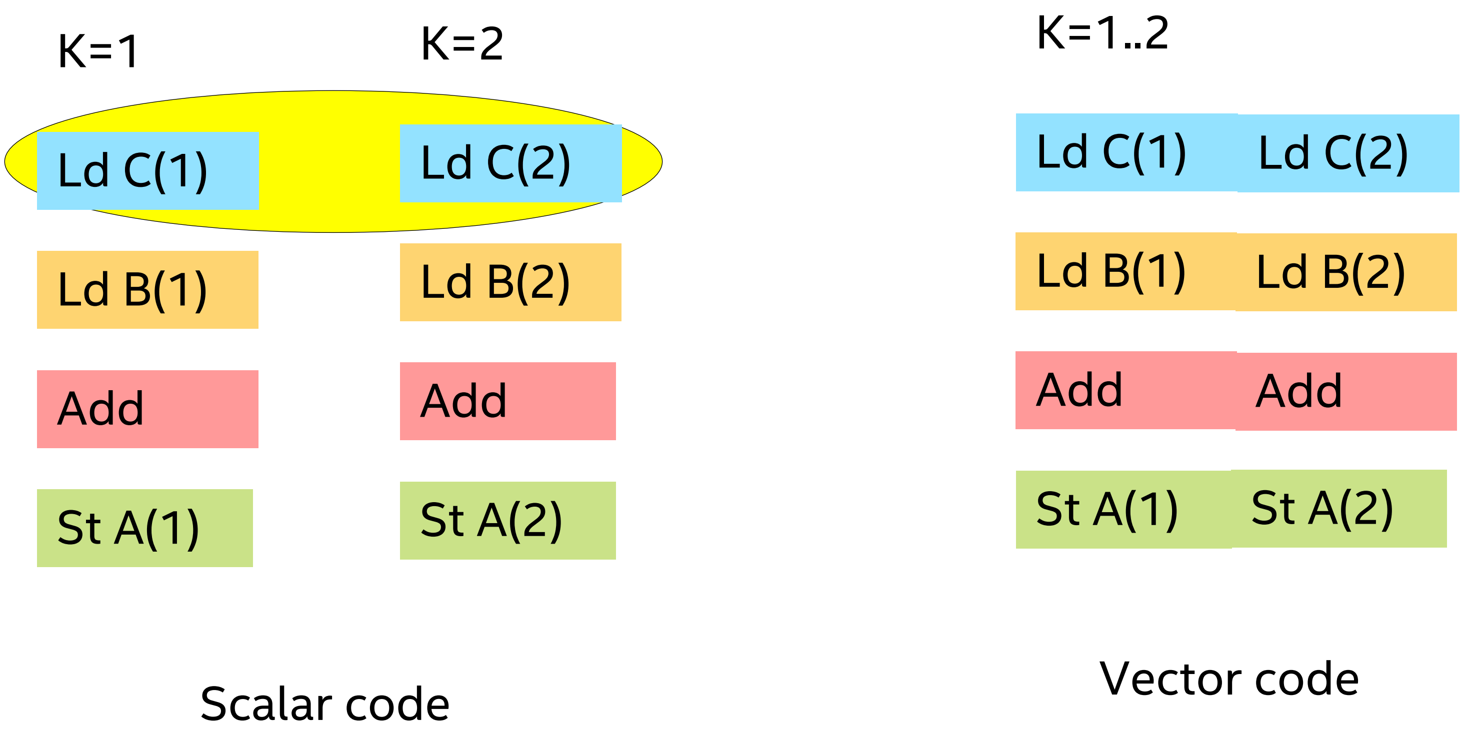
```
#pragma omp declare simd
int lednam(float c)
{   // Compute n >= 0 such that c^n > LIMIT
    float z = 1.0f; int iters = 0;
    while (z < LIMIT) {
        z = z * c; iters++;
    }
    return iters;
}
```

```
float in_vals[];
#pragma omp simd
for(int x = 0; x < Width; ++x) {
    count[x] = lednam(in_vals[x]);
}
```



Vectorization yesterday

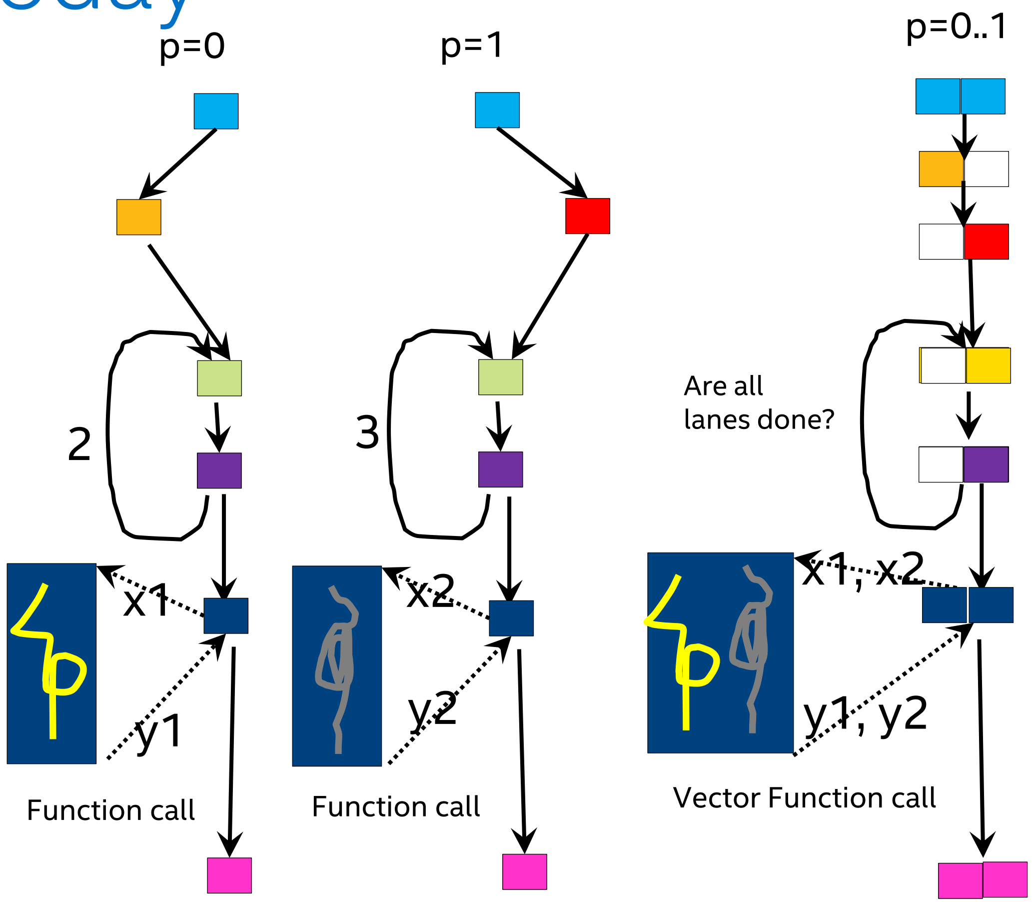
```
      DO 1 k = 1,n
1        A(k) = B(k) + C(k)
```



Vector code generation was straightforward
Emphasis on analysis and disambiguation

Vectorization today

```
#pragma omp simd reduction(+:....)
for(p=0; p<N; p++) {
  // Blue work
  if(...) {
     // Green work
  } else {
     // Red work
  }
  while(...) {
     // Gold work
     // Purple work
  }
  y = foo (x);
  Pink work
}
```



Two fundamental problems
- Data divergence
- Control divergence

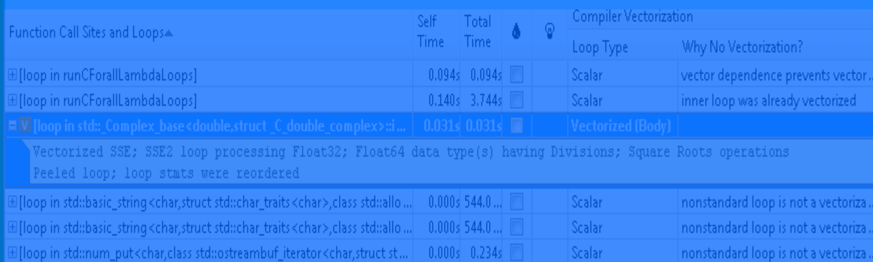
Vector code generation has become a more difficult problem
Increasing need for user guided explicit vectorization
Explicit vectorization maps threaded execution to simd hardware

Time for parallelism choices:
Where to introduce parallelism and how?

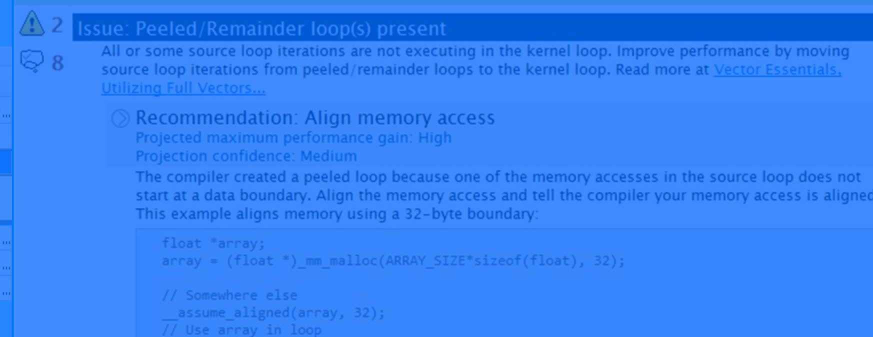
```
for(int i=0; i<Xmax; i++)                          ← Here?
    for(int j=0; j<Ymax; j++)
        for(int k=0; k<Zmax; k++) {                ← Here????
          //do some work
          for (int l=0; l<qdim; l++) {             ← Here???
            for (int m=1; m<=half; m++) {          ← Here??
                //...
                fSwapPairM (...);
            }
          }
        }
    }
}
```

No performance without “explicit parallelism” choices
(no performance “by default”)
No good choices without knowing “the DATA”

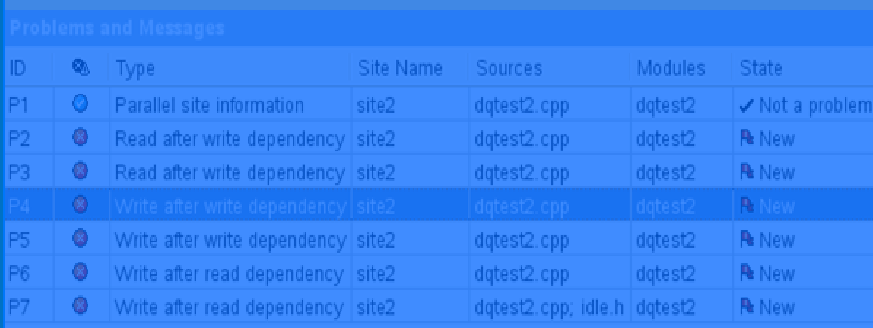
1. Compiler diagnostics + Performance Data + SIMD efficiency information



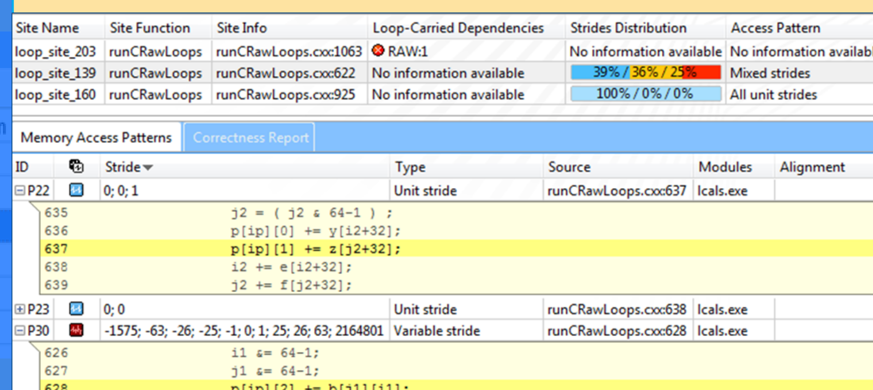
2. Guidance: detect problem and recommend how to fix it



3. Loop-Carried Dependency Analysis



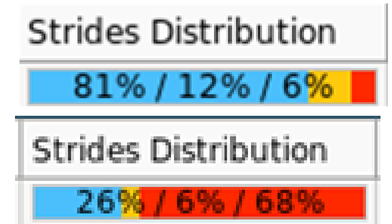
4. Memory Access Patterns Analysis



Optimization Notice

Time for parallelism choices: Advisor MAP to make informed optimal decision!

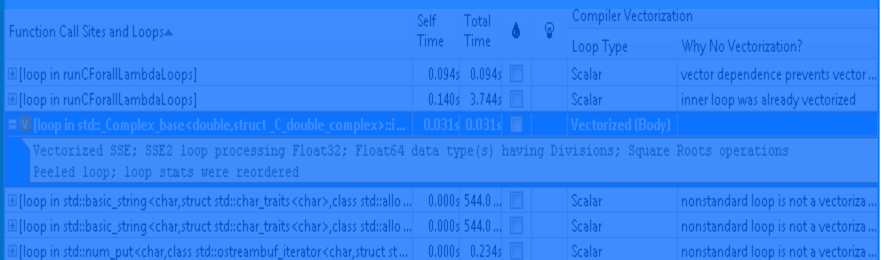
```
for(int i=0; i<Xmax; i++)
   for(int j=0; j<Ymax; j++)
      for(int k=0; k<Zmax; k++) {
        //do some work
        for (int l=0; l<qdim; l++) {
          for (int m=1; m<=half; m++) {
              //...
              fSwapPairM (...);
          }
        }
      }
```



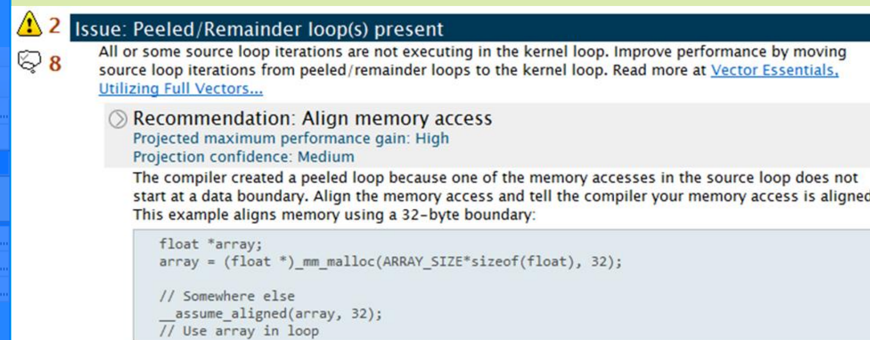
Memory Access Patterns analysis (+ also Trip Counts)
to drive decision
wrt most appropriate parallelism level

Optimization Notice

1. Compiler diagnostics + Performance Data + SIMD efficiency information



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



Background on loop vectorization

A typical vectorized loop consists of

Main vector body

- Fastest among the three!

This is where we want our loops to be executing!

Optional peel part

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

Remainder part

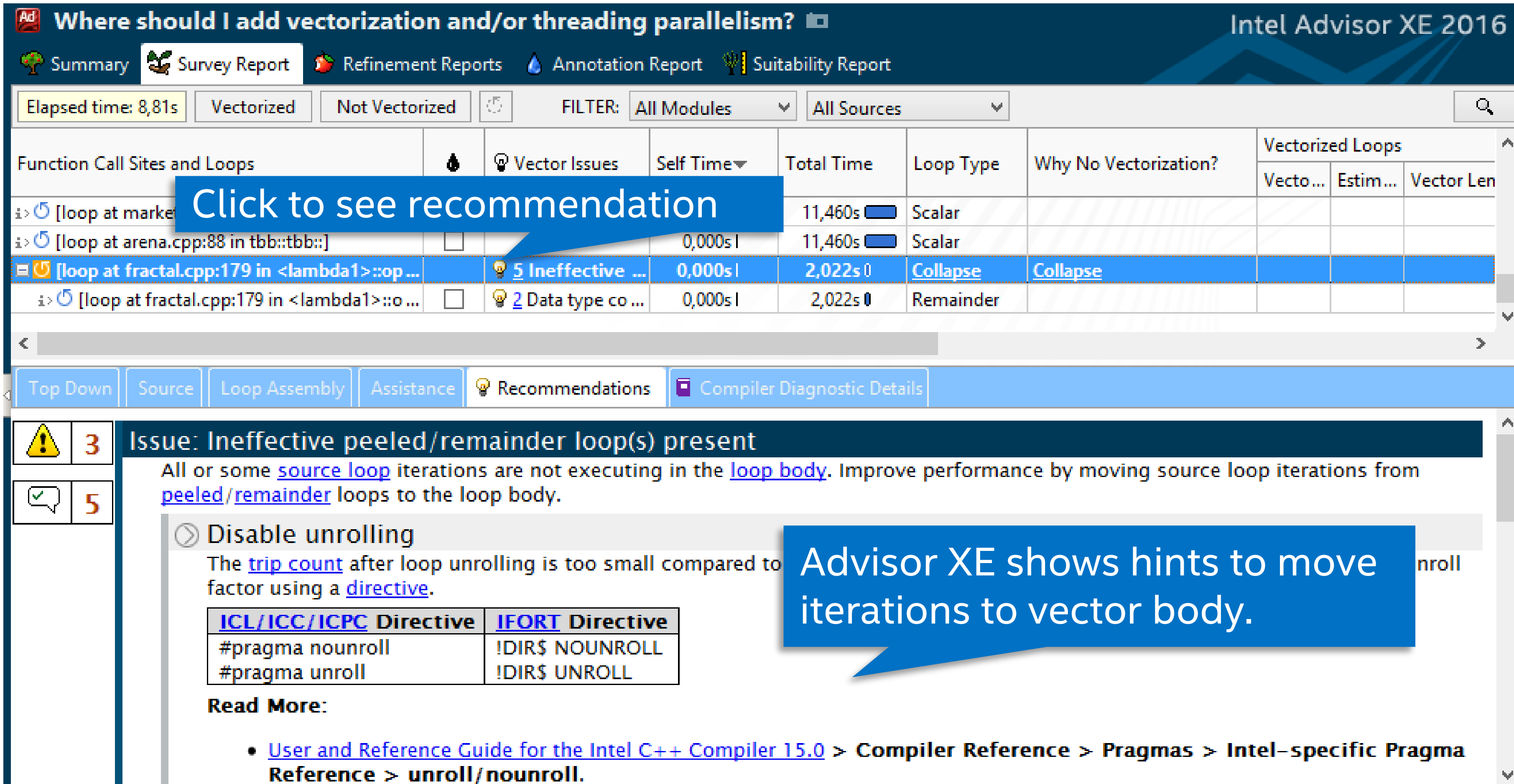
- Due to the number of iterations (trip count) not being divisible by vector length. Uses Scalar or slower vector.

Larger vector register means more iterations in peel/remainder

- Make sure you Align your data!
- Make the number of iterations divisible by the vector length!

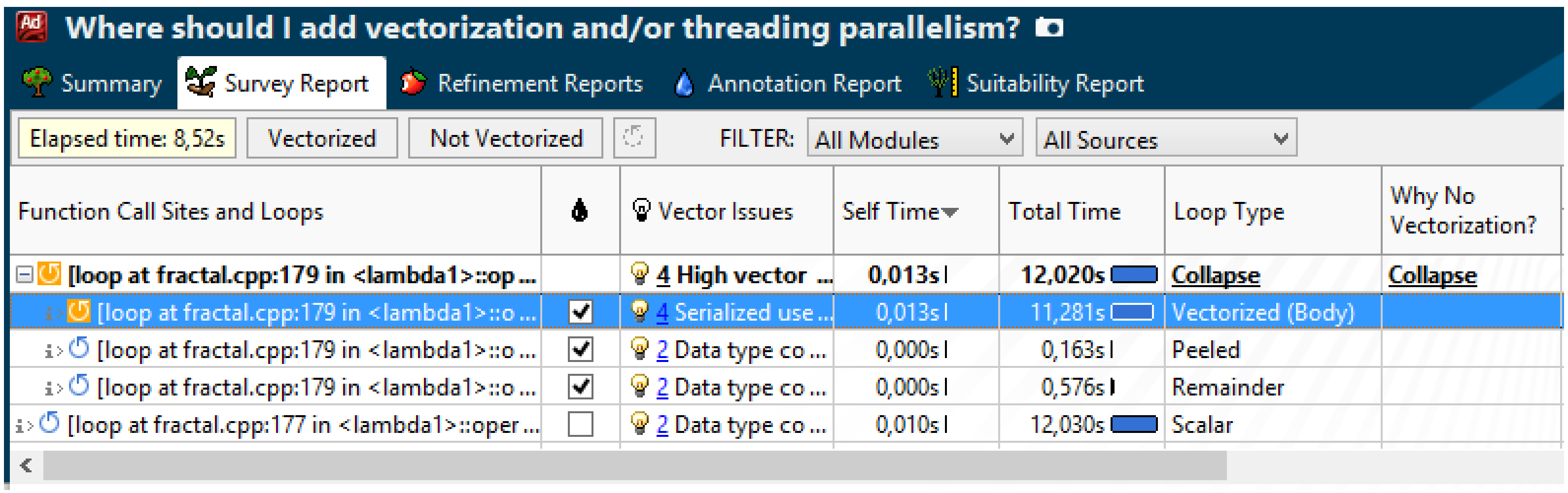
Get Specific Advice For Improving Vectorization

Intel® Advisor XE – Vectorization Advisor



Don’t Just Vectorize, Vectorize Efficiently

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

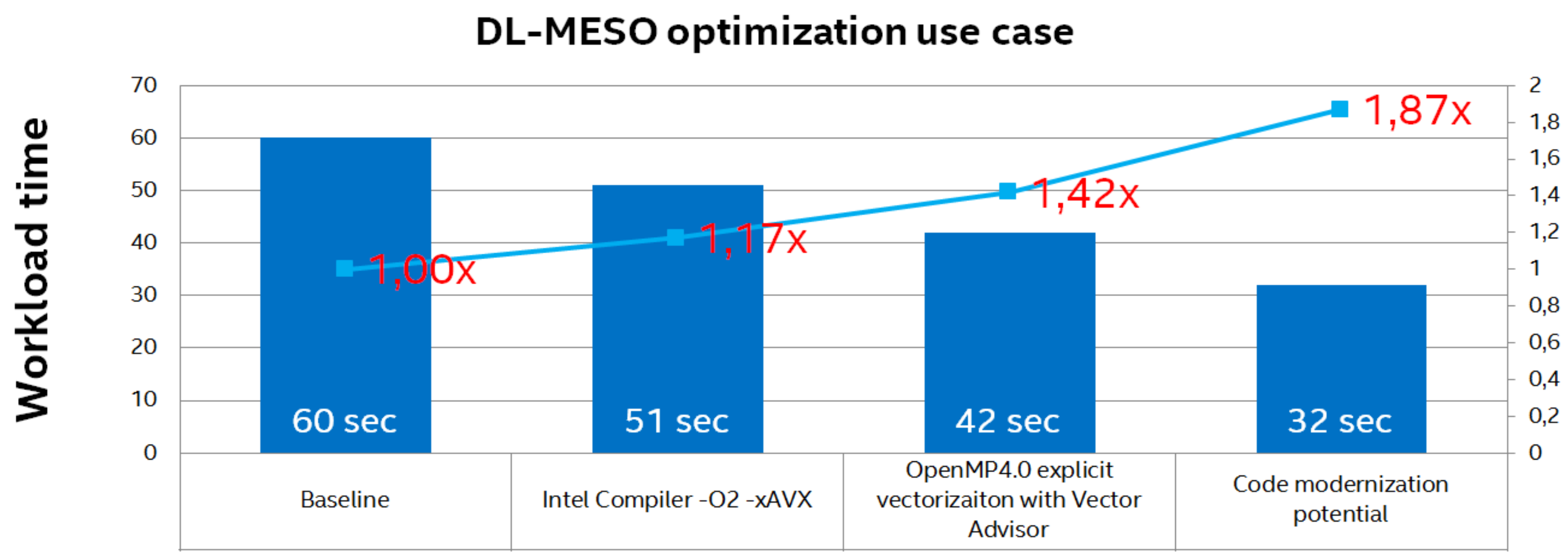


Function Call Sites and Loops		Vector Issues	Self Time	Total Time	Loop Type	Why No Vectorization?
[loop at fractal.cpp:179 in <lambda1>::op ...		4 High vector ...	0,013s	12,020s	Collapse	Collapse
[loop at fractal.cpp:179 in <lambda1>::o ...	☑	4 Serialized use ...	0,013s	11,281s	Vectorized (Body)	
[loop at fractal.cpp:179 in <lambda1>::o ...	☑	2 Data type co ...	0,000s	0,163s	Peeled	
[loop at fractal.cpp:179 in <lambda1>::o ...	☑	2 Data type co ...	0,000s	0,576s	Remainder	
[loop at fractal.cpp:177 in <lambda1>::oper ...	☐	2 Data type co ...	0,010s	12,030s	Scalar	

Vectorization Advisor: use cases

Hartree Centre
Science & Technology Facilities Council

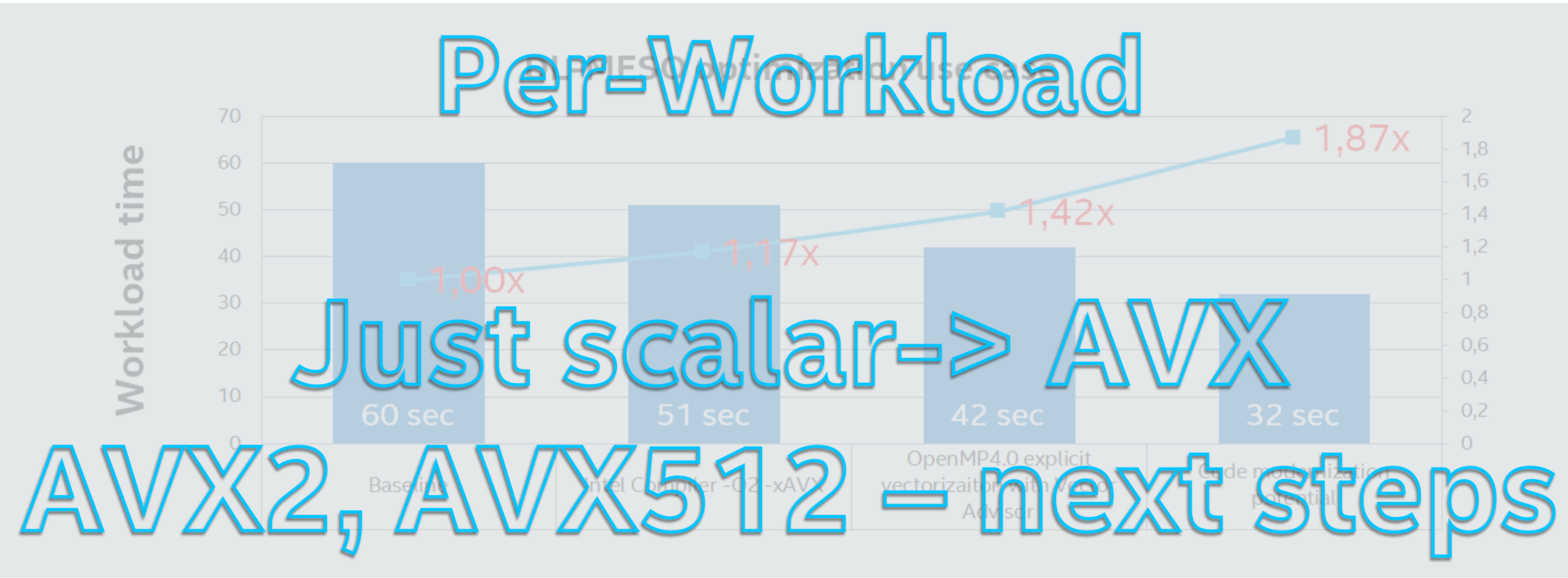
DL-MESO : major CFD package for cross-UK industrial consortium : new chemical products development (led by STFC **Hartree/Daresbury**)



Vectorization Advisor: use cases

Hartree Centre
Science & Technology Facilities Council

DL-MESO : major CFD package for cross-UK industrial consortium : new chemical products development (led by STFC **Hartree/Daresbury**)



Optimization Notice

Intel® Parallel Studio XE 2016 Beta
What's New

Intel® Parallel Studio XE 2016 Suites

Vectorization – Boost Performance By Utilizing Vector Instructions / Units

- Intel® Advisor XE - ***Vectorization Advisor*** identifies new vectorization opportunities as well as improvements to existing vectorization and highlights them in your code. It makes actionable coding recommendations to boost performance and estimates the speedup.

Scalable MPI Analysis– Fast & Lightweight Analysis for 32K+ Ranks

- Intel® Trace Analyzer and Collector add ***MPI Performance Snapshot*** feature for easy to use, scalable MPI statistics collection and analysis of large MPI jobs to identify areas for improvement

Big Data Analytics – Easily Build IA Optimized Data Analytics Application

- Intel® Data Analytics Acceleration Library (Intel® DAAL) will help data scientists speed through big data challenges with optimized IA functions

Standards – Scaling Development Efforts Forward

- Supporting the evolution of industry standards of ***OpenMP*, MPI, Fortran and C++*** Intel® Compilers & performance libraries

Intel® Advisor XE – New! Vectorization Advisor

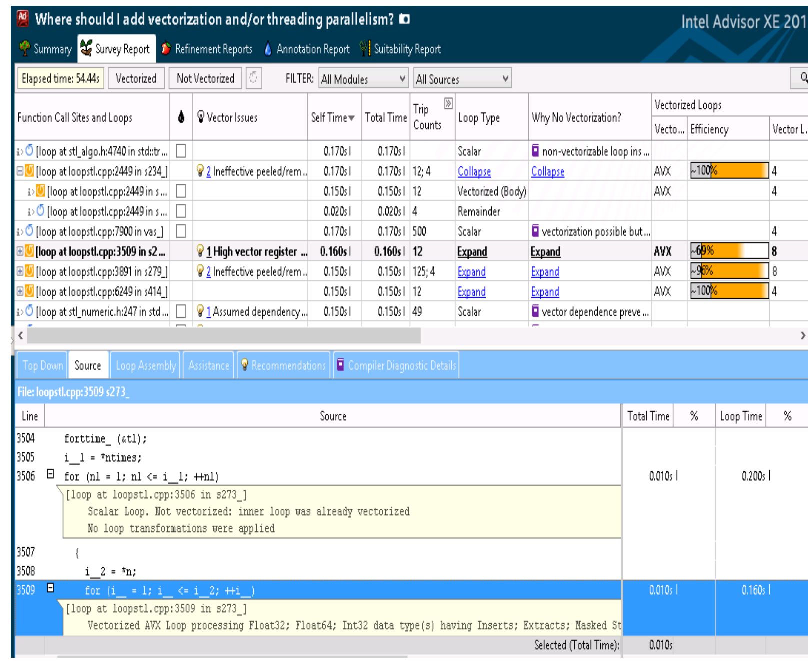
Data Driven Vectorization Design TO REWORK

Have you:

- Recompiled with AVX2, but seen little benefit?
- Wondered where to start adding vectorization?
- Recoded intrinsics for each new architecture?
- Struggled with cryptic compiler vectorization messages?

Breakthrough for vectorization design

- What vectorization will pay off the most?
- What is blocking vectorization and why?
- Are my loops vector friendly?
- Will reorganizing data increase performance?
- Is it safe to just use pragma simd?



More Performance
Fewer Machine Dependencies

Intel® Advisor XE – Vectorization Advisor

Provides the data you need for high impact vectorization

Compiler diagnostics + Performance Data = All the data you need in one place

- Find "hot" un-vectorized or "under vectorized" loops.
- Trip counts

Recommendations – How do I fix it?

Correctness via dependency analysis

- Is it safe to vectorize?

Memory Access Patterns analysis

- Unit stride vs Non-unit stride access, Unaligned memory access, etc.

Intel® C/C++ and Fortran Compilers 16.0
Get best performance with latest standards

Standards:

- More of C++14, generic lambdas, member initializers and aggregates
- More of C11, _Static_assert, _Generic, _Noreturn, and more
- OpenMP 4.0 C++ User Defined Reductions, Fortran Array Reductions

Vectorization:

- OpenMP 4.1 asynchronous offloading, simdlen, simd ordered
- Significant improvement in alignment analysis, vectorization robustness
- Much improved Neighboring Gather optimization

Fortran:

- F2008 Submodules, Impure Elemental Functions
- F2015 TYPE(*), DIMENSION(..), RANK intrinsic, attributes for args with BIND

Intel® Math Kernel Library (Intel® MKL) 11.3

Better performance with new two-stage API for Sparse BLAS routines

Additional Sparse Matrix Vector Multiplication API

- new two-stage API for Sparse BLAS level 2 and 3 routines

MKL MPI wrappers

- all MPI implementations are API-compatible but MPI implementations are **not ABI-compatible**
- MKL MPI wrapper solves this problem by providing an MPI-independent ABI to MKL

Support For Batched Small Matrix multiplication

- a single call executes multiple independent ?GEMM operation simultaneously

Support for Philox4x35 and ARS5 RNG

- two new pseudorandom number generators with a period of 2^128 are highly optimized for multithreaded environment

Sparse Solver SMP improvements

- significantly improved overall scalability for Intel Xeon Phi coprocessors and Intel Xeon processors

Intel® Data Analytics Acceleration Library 2016

A C++ and Java API library of optimized analytics building blocks for all data analysis stages, from data acquisition to data mining and machine learning. Essential for engineering high performance Big Data applications.

New library targeting data analytics market

- **Customers:** analytics solution providers, system integrators, and application developers (FSI, Telco, Retail, Grid, etc.)
- **Key benefits:** improved time-to-value, forward-scaling performance and parallelism on IA, advanced analytics building blocks

Key features

- Building blocks highly optimized for IA to support all data analysis stages
- Support batch, streaming, and distributed processing with easy connectors to popular platforms (Hadoop, Spark) and tools (R, Python, Matlab)
- Flexible interfaces for handling different data sources (CSV, MySQL, HDFS, RDD (Spark))
- Rich set of operations to handle sparse and noisy data
- C++ and Java APIs

Data Management

Interfaces for data representation and access. Connectors to a variety of data sources and data formats, such HDFS, SQL, CSV, ARFF, and user-defined data source/format.

- Data Sources
- Numeric Tables
- Outliers Detection
- Compression / Decompression
- Serialization / Deserialization

Data Processing

Optimized analytics building blocks for all data analysis stages, from data acquisition to data mining and machine learning.

Data Modeling

Data structures for model representation, and operations to derive model-based predictions and conclusions.

Important features offered in the initial Beta

Analysis

- PCA
- Variance-Covariance Matrix
- Distances
- Matrix decompositions (SVD, QR, Cholesky)
- EM for GMM
- Uni-/multi-variate outlier detection
- Statistical moments

Machine learning

- Linear regression
- Apriori
- K-Means clustering
- Naïve Bayes
- LogitBoost, BrownBoost, AdaBoost
- SVM

- Data layouts: AOS, SOA, homogeneous, CSR
- Data sources: csv, MySQL, HDFS/RDD
- Compression/decompression: ZLIB, LZO, RLE, BZIP2
- Serialization/deserialization

Intel® VTune™ Amplifier XE 2016 Beta
Enhanced GPU and Microarchitecture Profiling

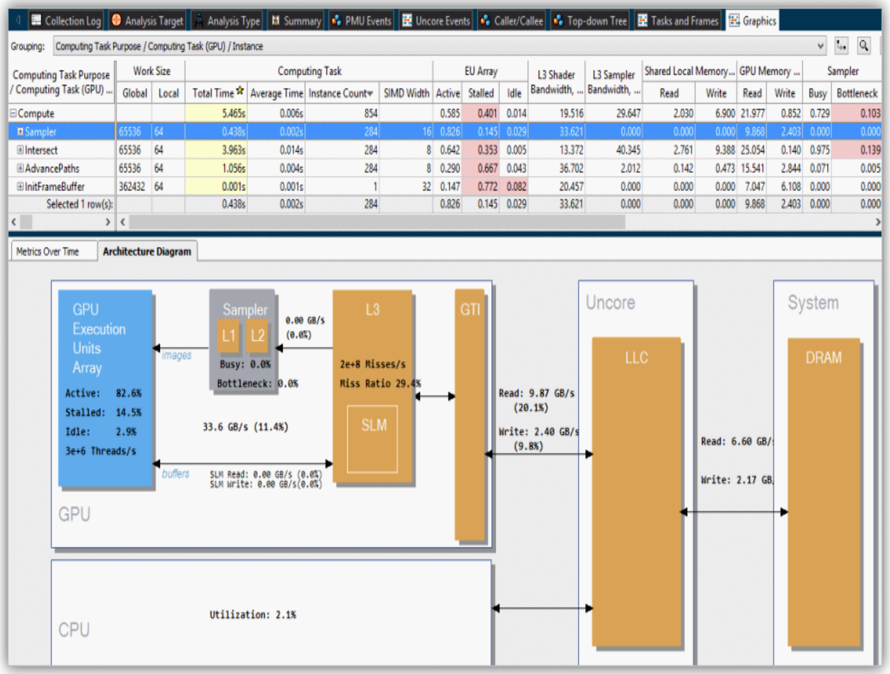
New OS and IDE support: Visual Studio* 2015 & Windows* 10 Threshold

Intel® HD Graphics (GPU) profiling

- GPU Architecture Annotation Diagram
- GPU profiling on Linux (OpenCL, Media SDK)

Microarchitecture tuning

- General Exploration analysis with confidence indication
- Driverless 'perf' EBS with stacks

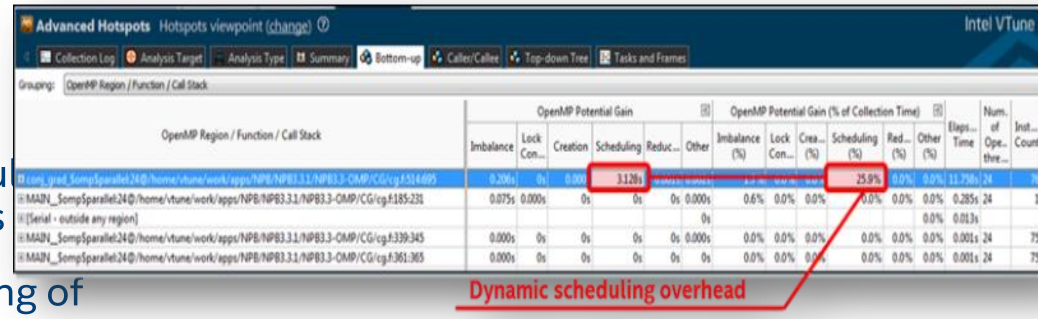


Optimization Notice

Intel® VTune™ Amplifier XE 2016 Beta
Improved OpenMP* and Hybrid Support

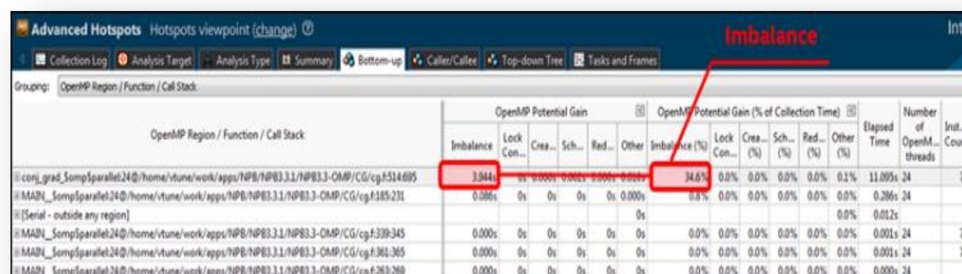
Intel OpenMP analysis enhancements

- Precise trace-based imbalance calculation that is especially useful profiling of small region instances
- Classification and issue highlighting of potential gains, e.g., imbalance, lock contention, creation overhead, etc.
- Detailed analysis of barrier-to-barrier region segments



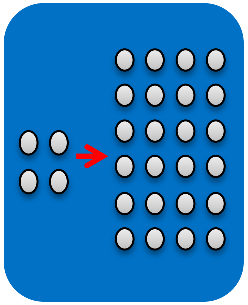
MPI+OpenMP: multi-rank analysis on a compute node

- Per-rank OpenMP potential gain and serial time metrics
- Per-rank Intel MPI communication busy wait time detection

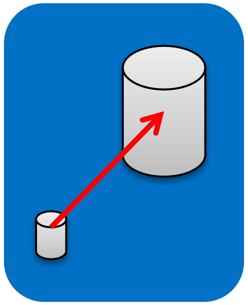


MPI Performance Snapshot

Scalable profiling for MPI and Hybrid

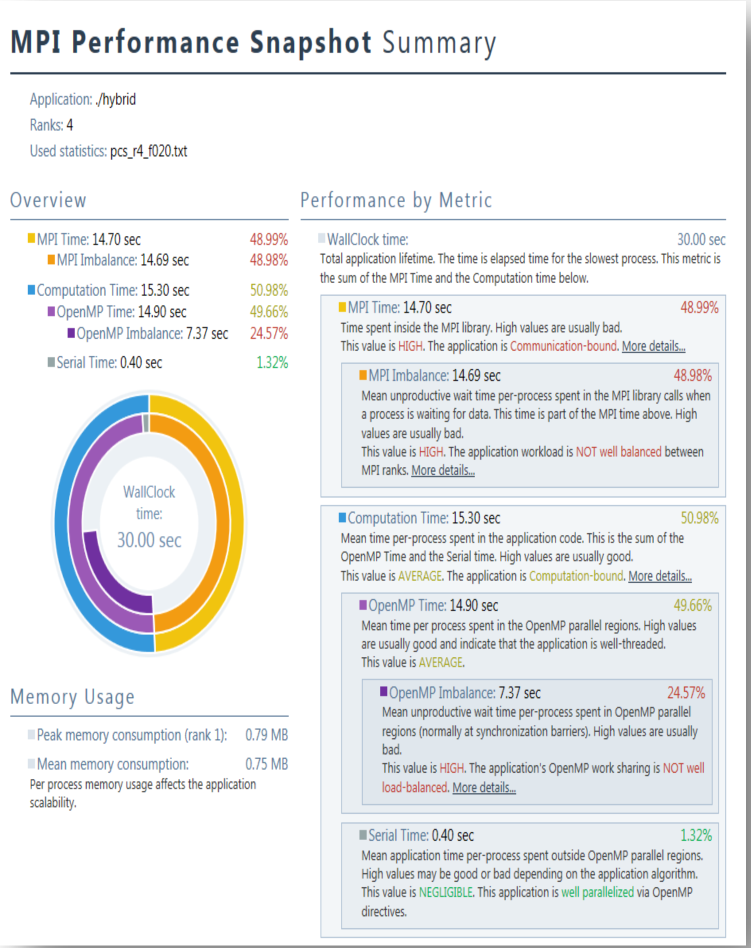


Lightweight – Low overhead profiling for 32K+ Ranks



Scalability- Performance variation at scale can be detected sooner

Identifying Key Metrics – Shows PAPI counters and MPI/OpenMP imbalances



MPI Performance Snapshot Summary

Application: ./hybrid
Ranks: 4
Used statistics: pcs_r4_f020.txt

Overview

MPI Time: 14.70 sec 48.99%
MPI Imbalance: 14.69 sec 48.98%
Computation Time: 15.30 sec 50.98%
OpenMP Time: 14.90 sec 49.66%
OpenMP Imbalance: 7.37 sec 24.57%
Serial Time: 0.40 sec 1.32%

WallClock time: 30.00 sec

Memory Usage

Peak memory consumption (rank 1): 0.79 MB
Mean memory consumption: 0.75 MB
Per process memory usage affects the application scalability.

Performance by Metric

WallClock time: 30.00 sec
Total application lifetime. The time is elapsed time for the slowest process. This metric is the sum of the MPI Time and the Computation time below.

MPI Time: 14.70 sec 48.99%
Time spent inside the MPI library. High values are usually bad.
This value is HIGH. The application is Communication-bound. More details...

MPI Imbalance: 14.69 sec 48.98%
Mean unproductive wait time per-process spent in the MPI library calls when a process is waiting for data. This time is part of the MPI time above. High values are usually bad.
This value is HIGH. The application workload is NOT well balanced between MPI ranks. More details...

Computation Time: 15.30 sec 50.98%
Mean time per-process spent in the application code. This is the sum of the OpenMP Time and the Serial time. High values are usually good.
This value is AVERAGE. The application is Computation-bound. More details...

OpenMP Time: 14.90 sec 49.66%
Mean time per process spent in the OpenMP parallel regions. High values are usually good and indicate that the application is well-threaded.
This value is AVERAGE.

OpenMP Imbalance: 7.37 sec 24.57%
Mean unproductive wait time per-process spent in OpenMP parallel regions (normally at synchronization barriers). High values are usually bad.
This value is HIGH. The application's OpenMP work sharing is NOT well load-balanced. More details...

Serial Time: 0.40 sec 1.32%
Mean application time per-process spent outside OpenMP parallel regions. High values may be good or bad depending on the application algorithm.
This value is NEGLIGIBLE. This application is well parallelized via OpenMP directives.

Optimization Notice

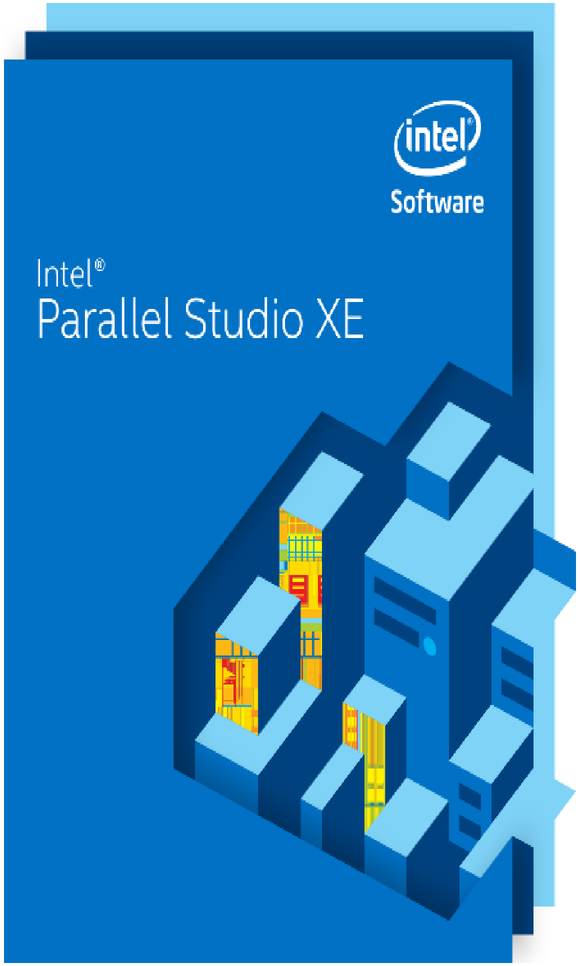
Summary/Call to Action

Participate in the Beta Program today!

- Register at **bit.ly/psxe2016beta**
- Or simply send e-mail to **vector_advisor@intel.com**

Submit Feedback via Intel® Premier Support

Tell us about your experiences using the Intel® Parallel Studio XE 2016 Beta



Additional Resources

All links start with: **https://software.intel.com/**

Learn more about Vectorization Advisor:

https://software.intel.com/en-us/articles/vectorization-advisor-faq

https://software.intel.com/en-us/intel-advisor-xe

Vectorization Guide:

https://software.intel.com/articles/a-guide-to-auto-vectorization-with-intel-c-compilers/

Explicit Vector Programming in Fortran:
https://software.intel.com/articles/explicit-vector-programming-in-fortran

Optimization Reports:

https://software.intel.com/videos/getting-the-most-out-of-the-intel-compiler-with-new-optimization-reports

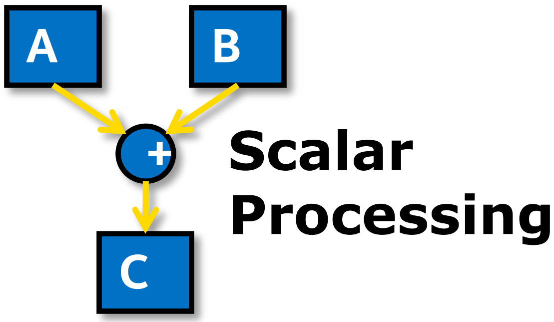
Beta Registration & Download:

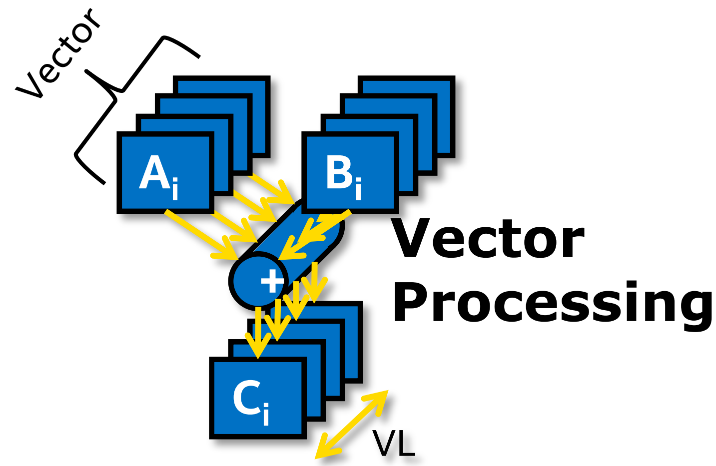
https://software.intel.com/en-us/articles/intel-parallel-studio-xe-2016-beta

For Intel® Xeon Phi™ coprocessors, but also applicable:
https://software.intel.com/en-us/articles/vectorization-essential
https://software.intel.com/en-us/articles/fortran-array-data-and-arguments-and-vectorization

intel®

Recap





AVX: Adding 2 vectors (SP)

+	4.4	1.1	3.1	-8.5	-1.3	1.7	7.5	5.6
	-0.3	-0.5	0.5	0	0.1	0.8	0.9	0.7
=	4.1	0.6	3.6	-8.5	-1.2	2.5	8.4	6.3

Legal Disclaimer & Optimization Notice

Optimization Notice

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

Notice revision #20110804