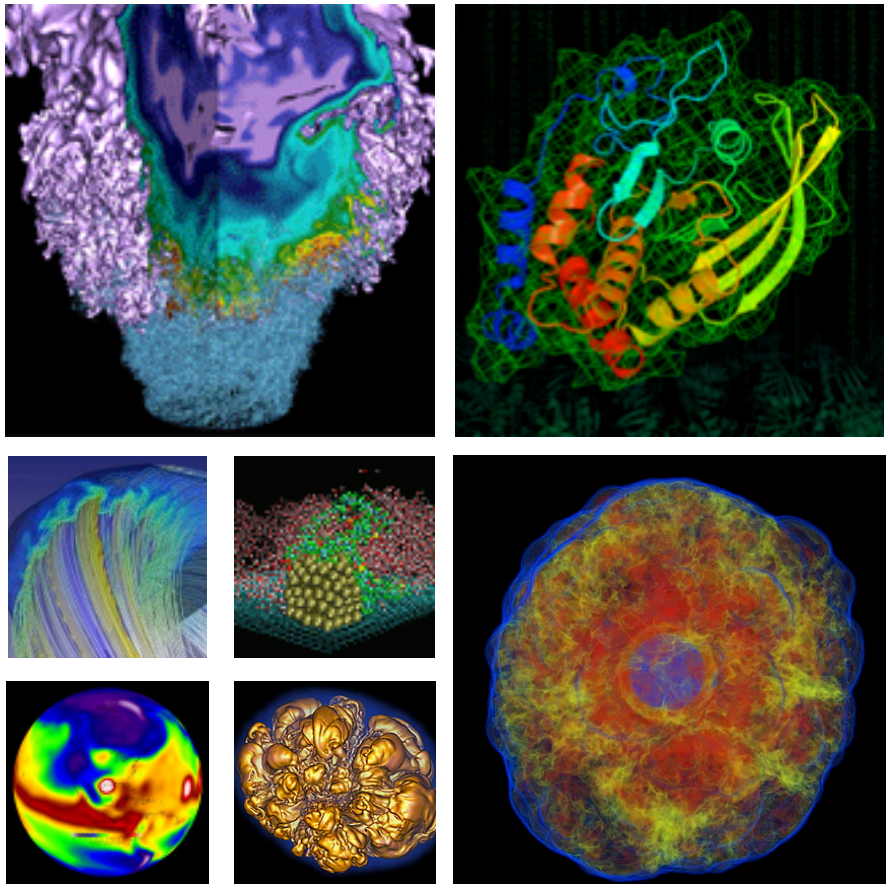
Vectorization



Woo-Sun Yang
NERSC User Engagement Group

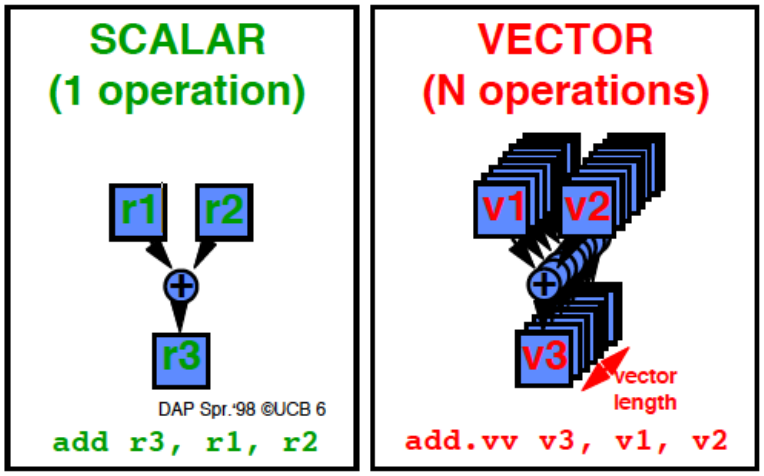
NERSC User Group Meeting 2016

March 23, 2016

What's All This About Vectorization?

- **Vectorization is an on-node, in-core way of exploiting data level parallelism in programs by applying the same operation to multiple data items in parallel.**

```
DO I= 1, N
     Z(I) = X(I) + Y(I)
ENDDO
```



- Requires *transforming* a program so that a single instruction can launch many operations on different data

- Applies most commonly to array operations in loops

What is Required for Vectorization?

- **Code transformation**

```
DO I = 1, N
   Z(I) = X(I) + Y(I)
ENDDO
```

Compiler →

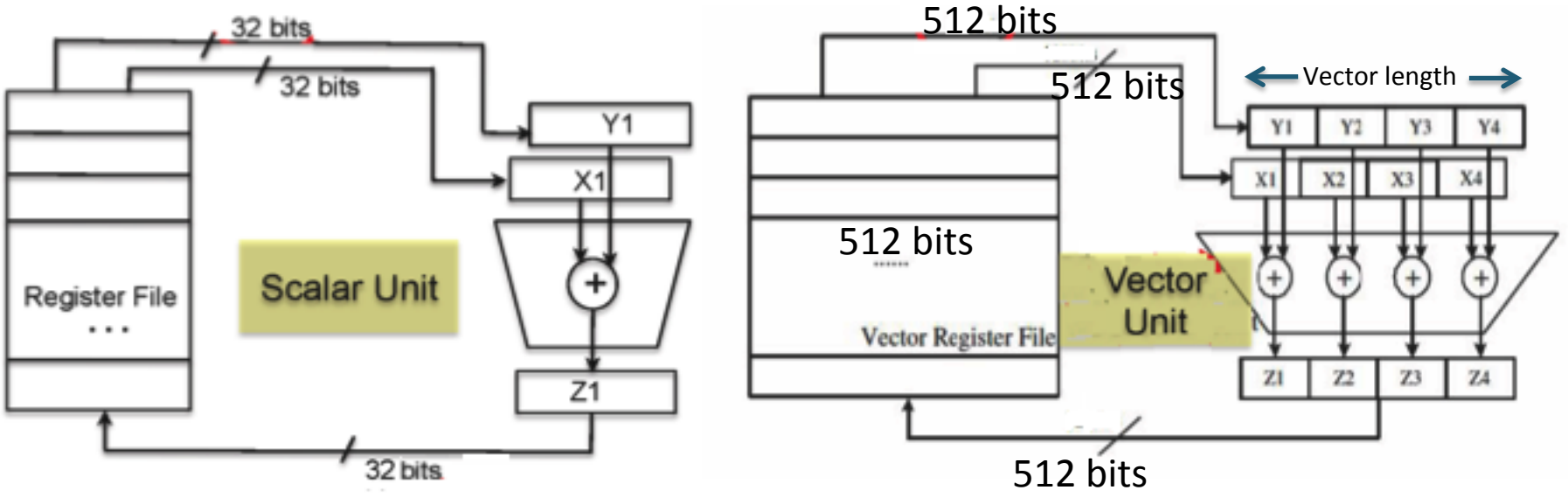
```
DO I = 1, N, 4
  Z(I)   =   X(I) +   Y(I)
  Z(I+1) = X(I+1) + Y(I+1)
  Z(I+2) = X(I+2) + Y(I+2)
  Z(I+3) = X(I+3) + Y(I+3)
ENDDO
```

- **Compiler generates vector instructions:**

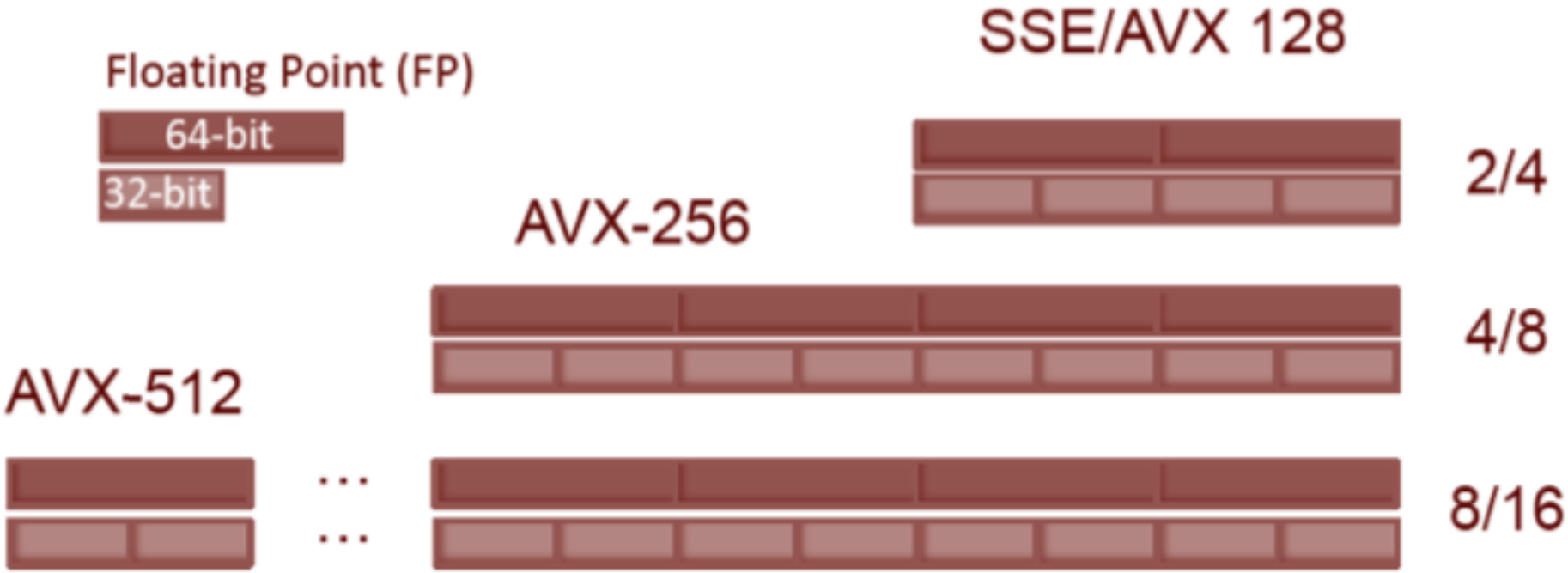
```
VLOAD  X(I), X(I+1), X(I+2), X(I+3)
VLOAD  Y(I), Y(I+1), Y(I+2), Y(I+3)
VADD   Z(I, ..., I+3)   X+Y(I, ..., I+3)
VSTORE Z(I), Z(I+1), Z(I+2), Z(I+3)
```

What is Required for Vectorization?

- **Vector Hardware: vector registers and vector functional units**



Evolution of Vector Hardware



- **Translates to (peak) speed: cores per processor X vector length X CPU Speed X 2 arith. ops per vector**

Data Dependencies

- **Examples:**

```
DO I=2,N-1
   A(I) = A(I-1) + B(I)
END DO
```

Compiler detects backward reference on `A(I-1)`

Read-after-write (also known as "flow dependency")

```
DO I=2,N-1
   A(I-1) = X(I) + DX
   A(I)   = 2.*DY
END DO
```

Compiler detects same location being written

Write-after-write (also known as ”output dependency")

```
ftn -qopt-report=2 -c mms.f90
 Report from: Loop nest, Vector & Auto-parallelization optimizations [loop, vec, par]
ftn -qopt-report=2 -c mms.f90
 remark #15346: vector dependence: assumed OUTPUT dependence between  line 12 and
line 11
```

How to Vectorize Your Code?

- Auto-Vectorization analysis by the compiler
- Auto-Vectorization analysis by the compiler enhanced with directives – code annotations that suggest what can be vectorized
- Code explicitly for vectorization using OpenMP 4.0 SIMD pragmas or SIMD intrinsics (not portable)
- Use assembly language
- Use vendor-supplied optimized libraries

Requirements for vectorization

- Loop trip count known at entry to the loop at runtime
- Single entry and single exit
- No function calls or I/O
- No data dependencies in the loop
- Uniform control flow (although conditional computation can be implemented using “masked” assignment)

Vectorization performance (speed-up)

- **Factors that affect vectorization performance**
  - Efficient loads and stores with vector registers
    - Data in caches
    - Data aligned to a certain byte boundary in memory
    - Unit stride access
  - Efficient vector operations
    - Certain arithmetic operations not at full speed
- **Good speed-up with vectorization when all the conditions are met**
- **Examples from https://www.nersc.gov/users/computational-systems/edison/programming/vectorization/**

How good is vectorization

- **Compiler vectorization of loops**
  - Enabled with default optimization levels for Intel and Cray compilers on Cori/Edison (and Intel on Babbage)
  - Use -qopt-report[=n] -qopt-report-phase=vec flag where (n is from 0 through 5; default: 2)

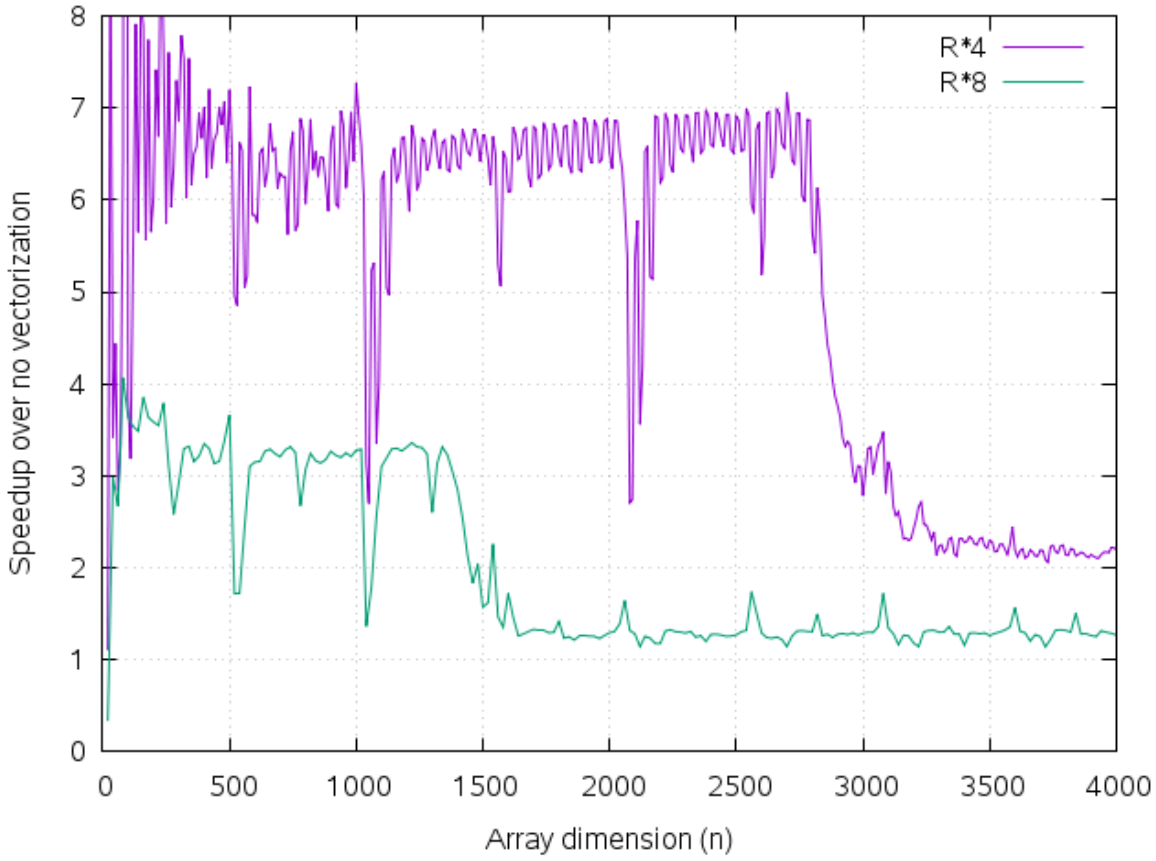
```
LOOP BEGIN at a1.F(35,10)
      remark #15300: LOOP WAS VECTORIZED
      remark #15450: unmasked unaligned unit stride loads: 2
      remark #15451: unmasked unaligned unit stride stores: 1
      remark #15475: --- begin vector loop cost summary ---
      remark #15476: scalar loop cost: 6
      remark #15477: vector loop cost: 2.000
      remark #15478: estimated potential speedup: 2.990
      remark #15488: --- end vector loop cost summary ---
      remark #25015: Estimate of max trip count of loop=249
   LOOP END

   LOOP BEGIN at a1.F(35,10)
   <Remainder loop for vectorization>
      remark #15301: REMAINDER LOOP WAS VECTORIZED
      remark #25015: Estimate of max trip count of loop=3
   LOOP END
```

How good is vectorization (Cont'd)

- **Intel Advisor**
  - Vectorization analysis tool that identifies loops for vectorization and reasons that blocks effective vectorization
- **Many web pages on useful info**
  - https://software.intel.com/en-us/intel-advisor-xe
  - https://software.intel.com/en-us/get-started-with-advisor-vectorization-linux
  - https://software.intel.com/en-us/intel-advisor-xe-support/training
  - https://software.intel.com/en-us/intel-advisor-2016-tutorial-vectorization-linux-cplusplus
  - …

Data in Caches

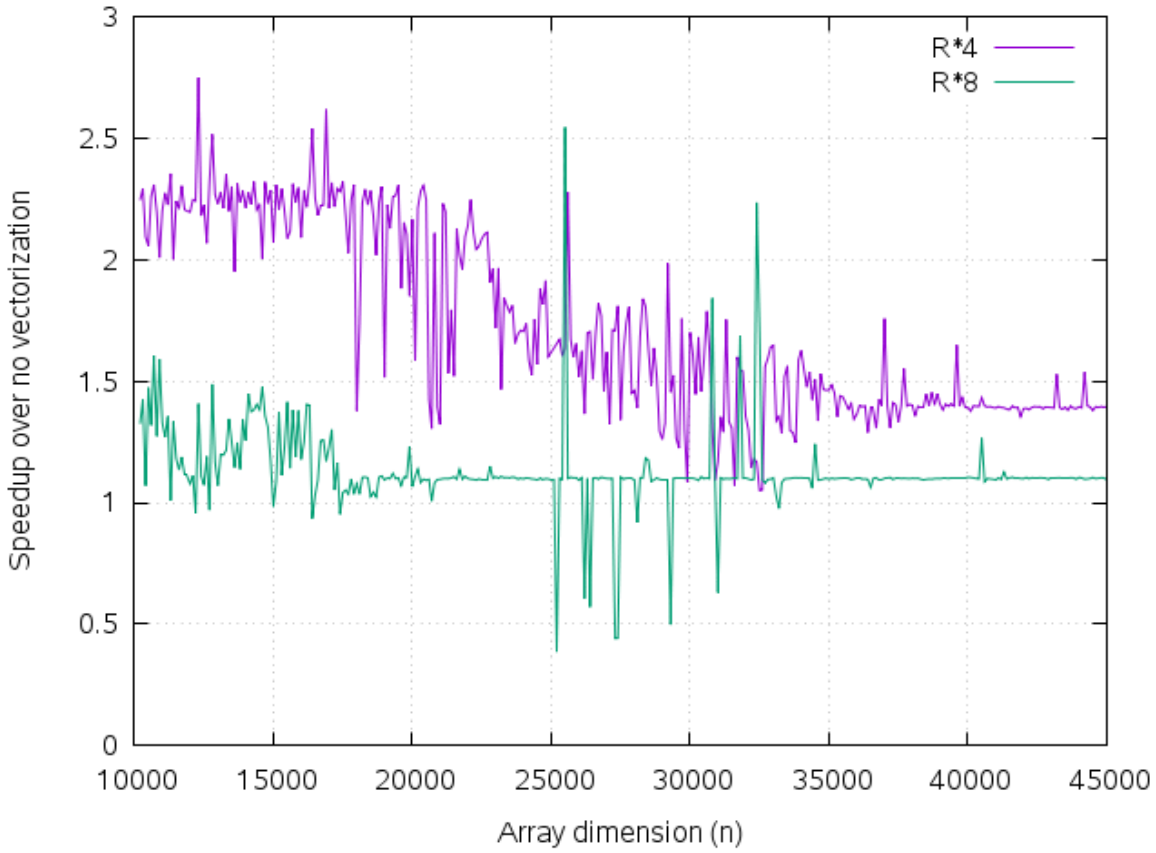
```
do i=1,n
    c(i) = a(i) + b(i)
end do
```



Speedup close the theoritical max below L1 Cache. Worse as array size passes L1 size.

Data in Caches (Cont'd)

```
do i=1,n
    c(i) = a(i) + b(i)
end do
```



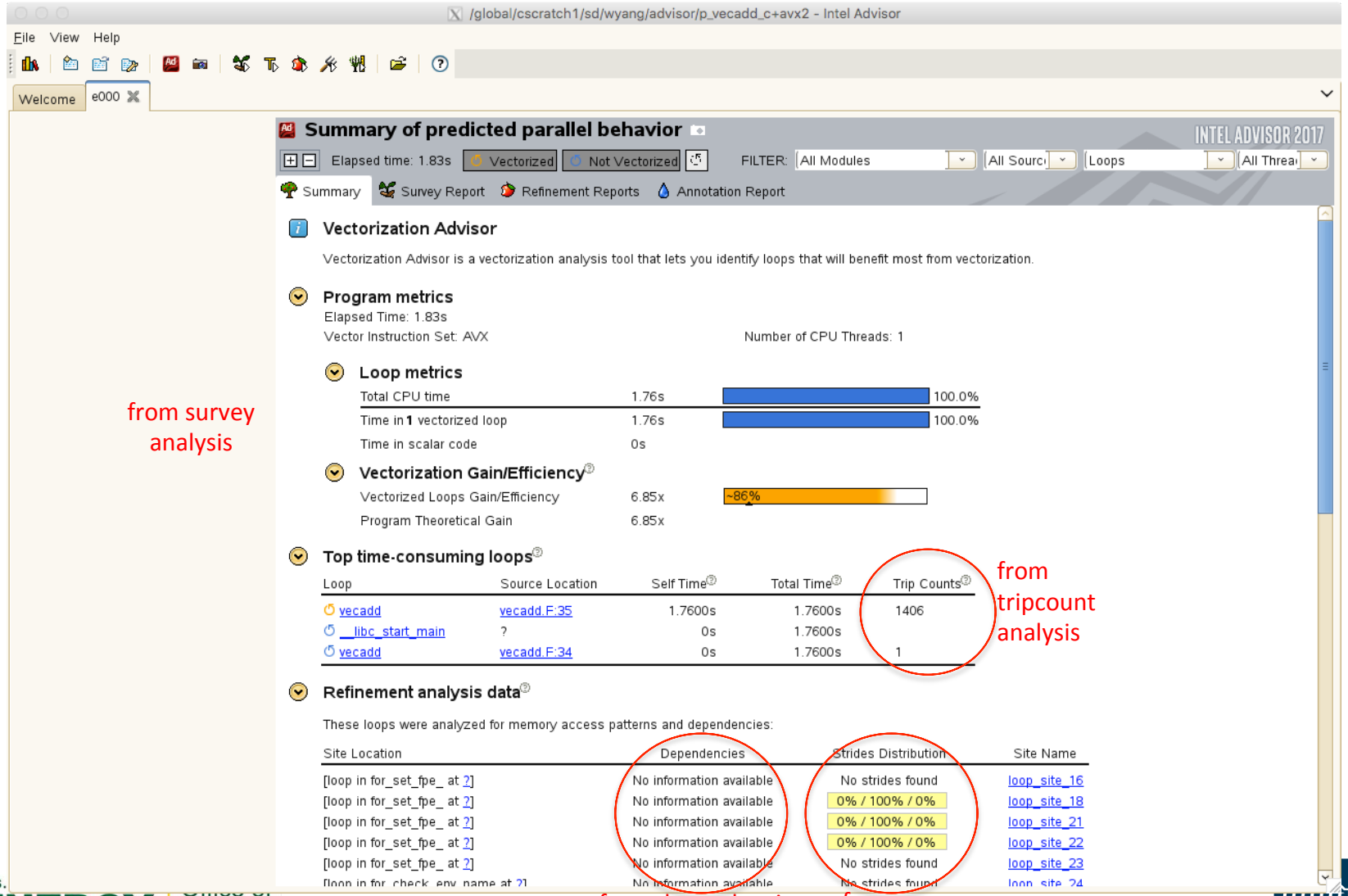
Speedup drops again as pass L2 cache size.

Let's try Intel Advisor on these runs

- **Analysis types that Intel Advisor provides**
  - **survey**: explore where to add efficient vectorization
  - **tripcounts**: iteration counts for loops
  - Refinement analysis
    - **map** (Memory access pattern): memory access strides for loops
    - **dependencies**: loop-carried dependencies
- **Analysis can be run in GUI or CLI**
  - **advixe-gui**: GUI command
  - **advixe-cl**: CLI command
- **Add -g to the usual optimization flag (e.g., -g -O3)**

- **A "project" is a physical directory where analyses can be carried out for a given executable**
  - Need to create the project directory and specify it so that analysis results are saved there
  - Can contain multiple analysis types (e.g., survey, tripcounts, map, …)

Intel Advisor



Some Advisor CLI commands

- From ‘advixe-cl -help’:

```
$ module load advisor

$ mkdir myproj                    project directory; contains all the
                                  following analysis results

$ advixe-cl -collect=survey -project-dir=./myproj -- ./a.out      results in
$ advixe-cl -report=survey -project-dir=./myproj -format=text \   myproj/e000/hs000
  -report-output=survey.txt

$ advixe-cl -collect=tripcounts -project-dir=./myproj -- ./a.out  results in
$ advixe-cl -report=tripcounts -project-dir=./myproj -format=text \ myproj/e000/trc000
  -report-output=survey.txt

$ advixe-cl -collect=map -project-dir=./myproj -- ./a.out         results in
$ advixe-cl -report=map -project-dir=./myproj -format=text \      myproj/e000/mp000
  -report-output=survey.txt

$ advixe-cl -collect=dependencies -project-dir=./myproj -- ./a.out
$ advixe-cl -report=dependencies -project-dir=./myproj -format=text \
  -report-output=survey.txt                                       results in
                                                                  myproj/e000/dp000;
                                                                  can take very long
```

Intel Advisor for c(:) = a(:) + b(:)

- **n=1500 (all data within L1 cache) using AVX2**

Where should I add vectorization and/or threading parallelism? INTEL ADVISOR 2017

Elapsed time: 0.29s | Vectorized | Not Vectorized | FILTER: All Modules | All Sourc | Loops | All Threads

Summary | Survey Report | Refinement Reports | Annotation Report

Function Call Sites and Loops	Vector Issues	Self Time	Total Time	Type	Why No Vectorization?	Vectorized Loops: Vec...	Efficiency	Gai...	VL ...	Trip ...: Median	Instruction Set Anal...: Traits	Data ...	Num..	Advanced	Loca...
[loop in vecadd at vecadd....	1 Ineffecti...	**0.230s**	**0.230s**	**Vectorized (...**		**AVX**	**~76%**	**6.12x**	**8**	**46; 3**		**Float32**	**2; 8**	**Unrolled by 4**	**vecadd.F:**
[loop in vecadd at vecadd. ...		0.198s	0.198s	Vectorized (B...		AVX			8	46		Float32	8	Unrolled by 4	vecadd.F:3
[loop in vecadd at vecadd. ...		0.032s	0.032s	Vectorized (R...		AVX			8	3		Float32	2		vecadd.F:3
[loop in __libc_start_main]		0.000s	0.230s	Scalar									0		
[loop in vecadd at vecadd.F:34]		0.000s	0.230s	Scalar	inner loop ...					2500000		Float32	10		vecadd.F:3

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

0.23s

Vectorized (Body; Remainder) *Total time*

AVX 0.23s

Instruction Set *Self time*

- Memory 53% (10)
- Compute 37% (7)
- Other 11% (2)

Insruction Mix Summary

Median Trip Counts: 46; 3

Instruction Mix

Memory: 10 *Compute:* 7 *Other:* 2

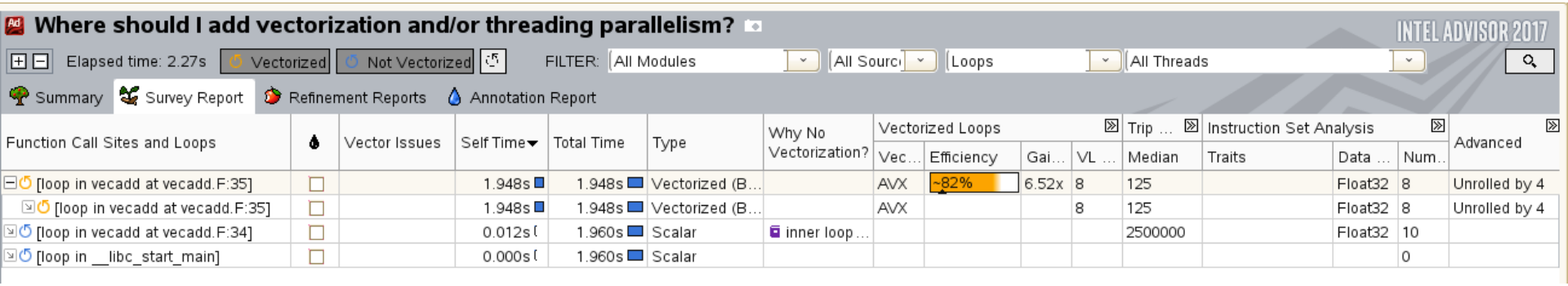
Memory: 52.63% | Vector: 52.63% | AVX: 52.63%

Compute: 36.84% | Vector: 26.3... | AVX: 26.32% | x...

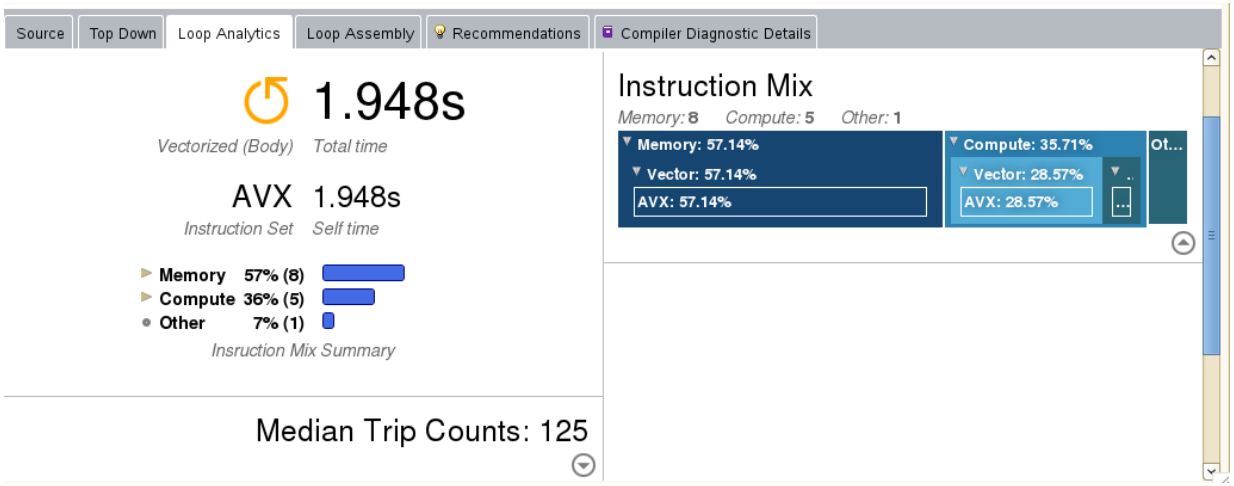
Othe...

Intel Advisor for c(:) = a(:) + b(:)

- **n=4000 (data cannot fit L1 cache) using AVX2**



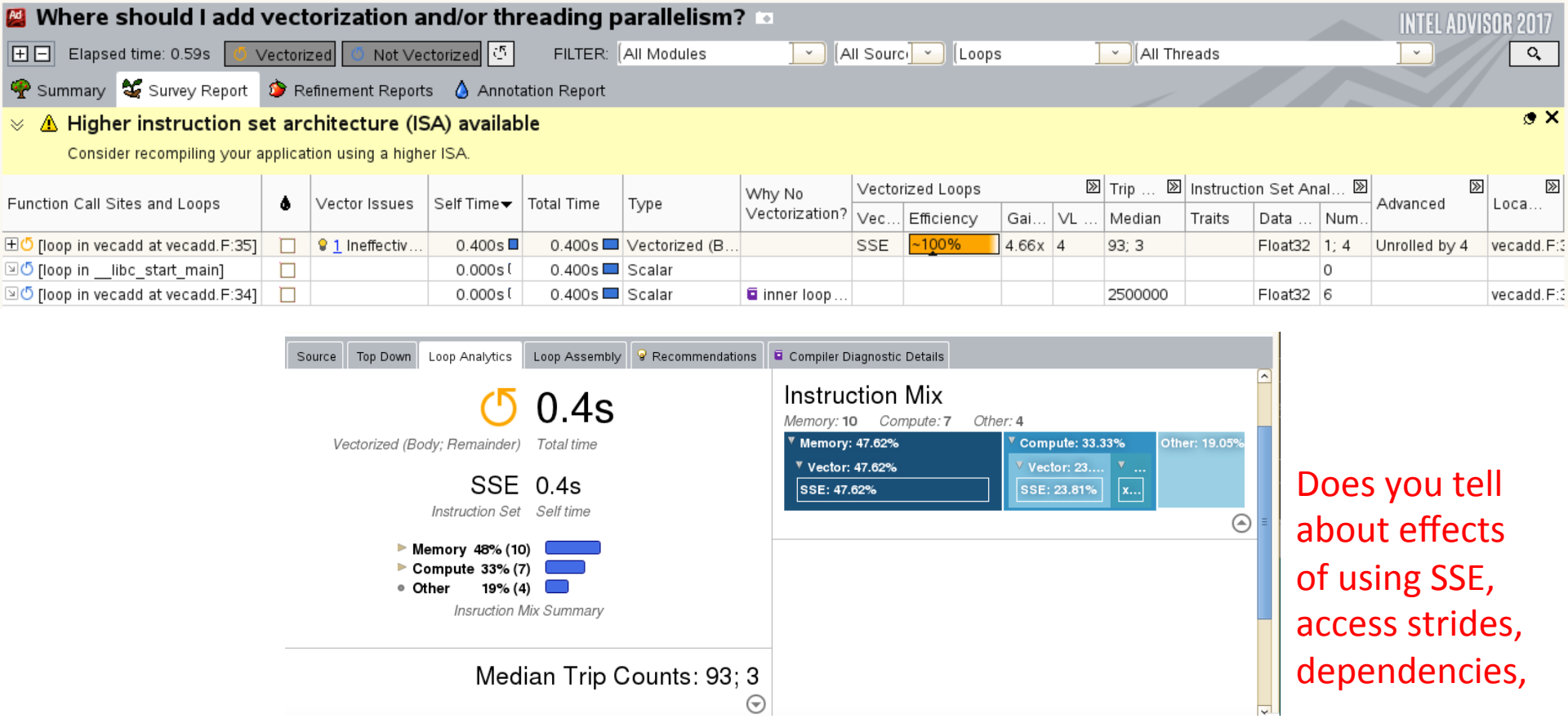
Function Call Sites and Loops	Vector Issues	Self Time	Total Time	Type	Why No Vectorization?	Vectorized Loops: Vec...	Efficiency	Gai...	VL ...	Trip ...: Median	Instruction Set Analysis: Traits	Data ...	Num..	Advanced
[loop in vecadd at vecadd.F:35]		1.948s	1.948s	Vectorized (B...		AVX	~82%	6.52x	8	125		Float32	8	Unrolled by 4
[loop in vecadd at vecadd.F:35]		1.948s	1.948s	Vectorized (B...		AVX			8	125		Float32	8	Unrolled by 4
[loop in vecadd at vecadd.F:34]		0.012s	1.960s	Scalar	inner loop ...					2500000		Float32	10	
[loop in __libc_start_main]		0.000s	1.960s	Scalar									0	



Does *not* tell you about effects of cache misses

Intel Advisor for c(:) = a(:) + b(:)

- **n=1500 (all data within L1 cache) using SSE**



Does you tell about effects of using SSE, access strides, dependencies, …

Memory alignment

- **More instructions are needed to collect and organize in registers if data is not optimally laid out in memory**
- **Data movement is optimal if the address of data starts at certain byte boundaries**
  - SSE: 16 bytes (128 bits)
  - AVX: 32 bytes (256 bits)
  - AVX-512 on KNL: 64 bytes (512 bits)

Memory alignment to assist vectorization

- **From https://software.intel.com/en-us/articles/data-alignment-to-assist-vectorization**
- **Alignment of data (Intel)**
  - Fortran compiler flag -align
    - '-align array<n>bytes', where n=8,16,32,64,128,256, as in '–align array64byte'
    - Entities of COMMON blocks: '-align commons' (4-byte); '-align dcommons' (8-byte); '-align qcommons' (16-byte); '-align zcommons' (32-byte); none for 64-byte
    - '-align rec<n>byte', where n=1,2,4,8,16,32,64: for derived-data-type components
  - Alignment directive/pragmas in source code
    - Fortran
      - !dir$ attributes align: 64::A – when A is declared
      - !dir$ assume_aligned A:64 – informs that A has been aligned
      - !dir$ vector aligned – vectorize a loop using aligned loads for all arrays
    - C or C++
      - 'float A[1000] __attribute__((align(64));' or '__declspec(align(64)) float A[1000];' when declaring a static array
      - _alligned_malloc()/_aligned_free() or _mm_malloc()/_mm_free() to allocate heap memory
      - __assume_aligned(A,64)
      - #pragma vector aligned – vectorize a loop using aligned loads for all arrays

Memory alignment for multidimensional arrays

- **Multi-dimensional arrays need to be padded in the fastest-moving dimension, to ensure array sections to be aligned at the desired byte boundaries**
  - Fortran: first array dimension
  - C/C++: last array dimension
- **npadded = ((n + veclen – 1) / veclen) * veclen**
  - No alignment requested: veclen = 1
  - 16-byte alignment (SSE): veclen = 4 (sp) or 2 (dp)
  - 32-byte alignment (AVX2): veclen = 8 (sp) or 4 (dp)
  - 64-byte alignment (AVX-512): veclen = 16 (sp) or 8 (dp)

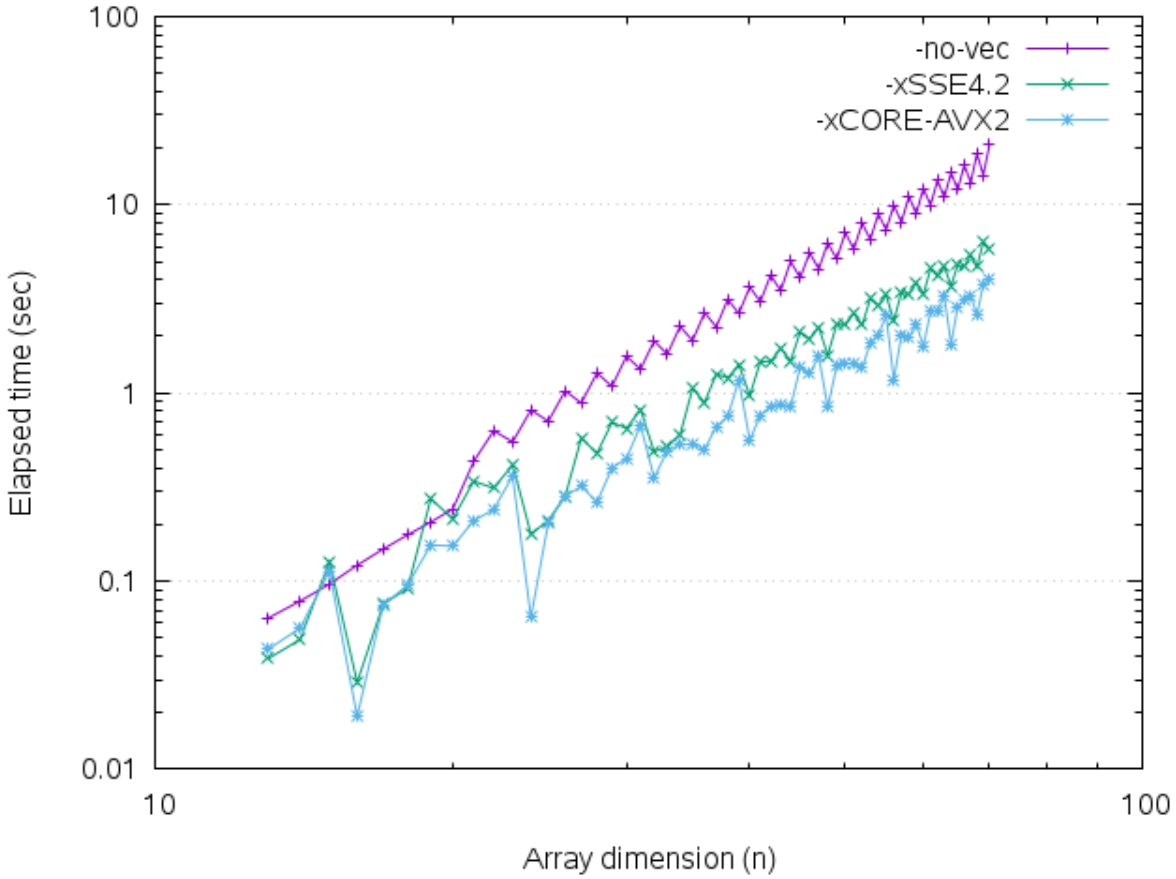
Memory alignment example

- **Naïve matrix-matrix multiplication on Edison**

```
real, allocatable :: a(:,:), b(:,:), c(:,:)
!dir$ attributes align : 32 :: a,b,c
...
allocate (a(npadded,n))
allocate (b(npadded,n))
allocate (c(npadded,n))
...
do j=1,n
   do k=1,n
!dir$ vector aligned
      do i=1,npadded
         c(i,j) = c(i,j) &
            + a(i,k) * b(k,j)
      end do
   end do
end do

!... Ignore c(n+1:npadded,:)
```

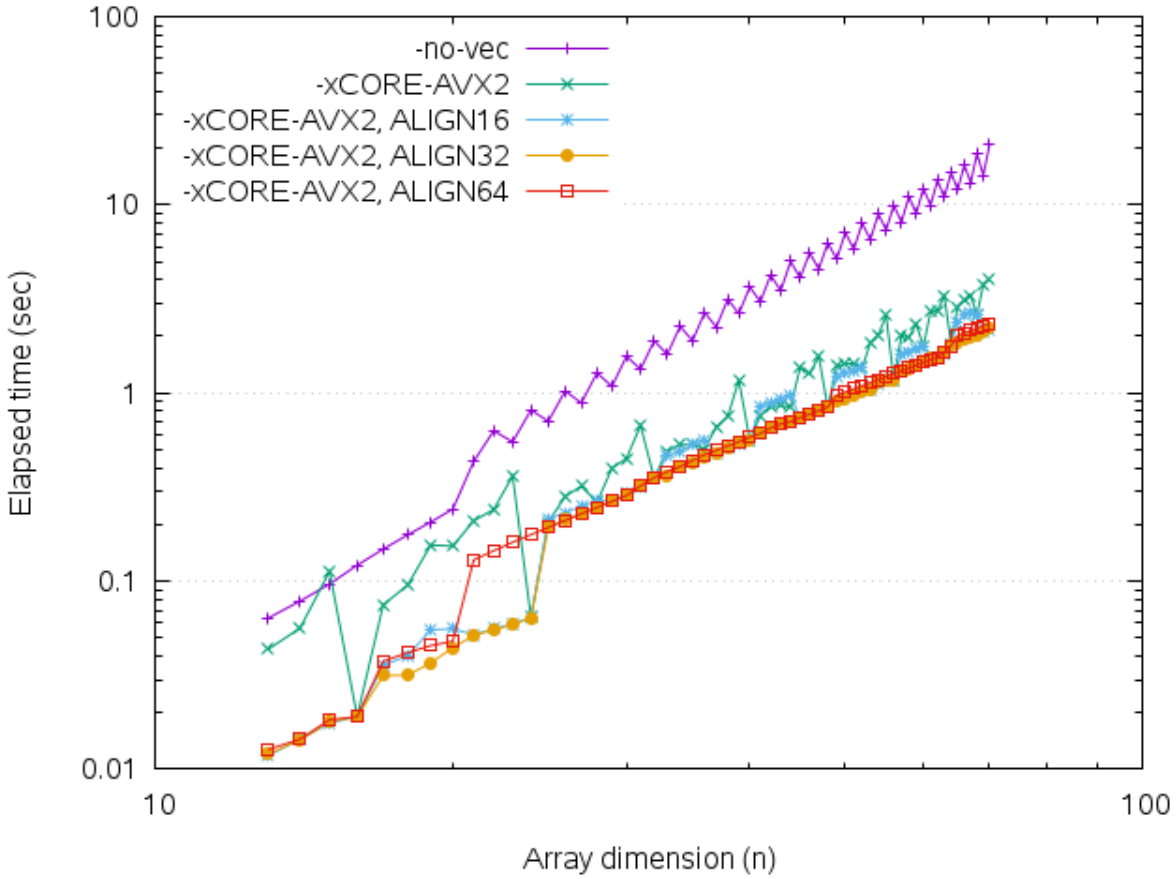
Effect of vectorization (no alignment case)

- **Runtime: -no-vec > -xSSE4.2 > -xCORE-AVX2 (similarly for aligned cases)**
- **Runtime drops when n is a multiple of 4**



Effect of memory alignment

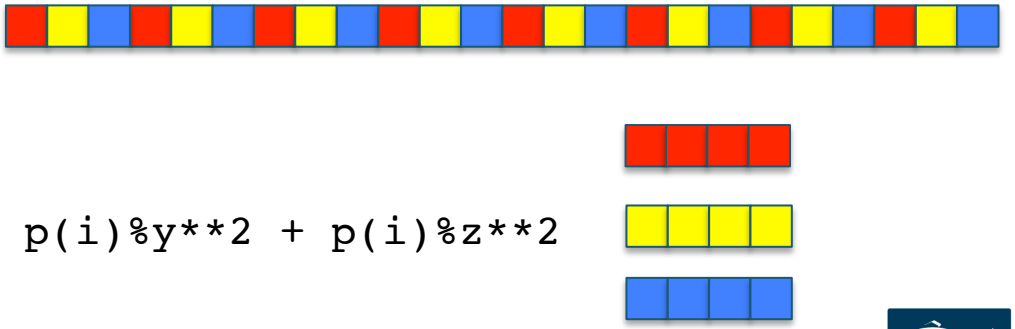
- Effect of padding rows (Fortran)
- Bumps get smoothened (toward better performance)
- Little improvement with ALIGN64 over ALIGN32



AoS vs. SoA

- **Data objects with component elements or attributes**
- **Array of a structure (AoS)**
  - The natural order in arranging such objects
  - But it gives non-unit strided access when loading into vector registers

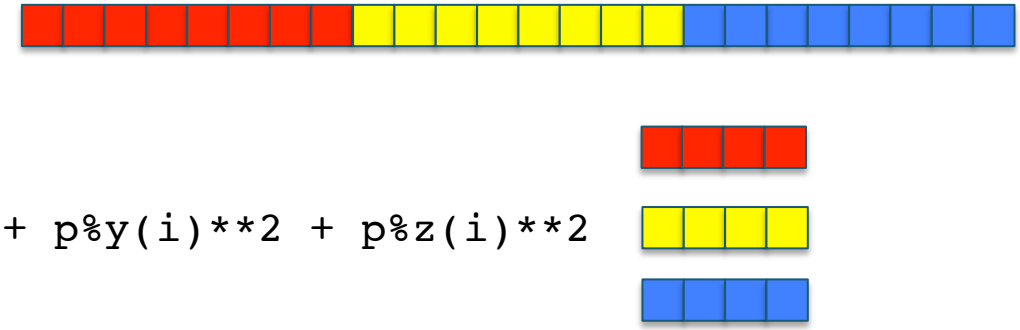
```
type coords
  real :: x, y, z
end type
type (coords) :: p(1024)
real dsquared(1024)

do i=1,1024
  dsquared(i) = p(i)%x**2 + p(i)%y**2 + p(i)%z**2
end do
```



AoS vs. SoA

- **Structure of arrays (SoA)**
  - Unit strided access when loading into vector registers
  - More efficient with loading into vector registers

```
type coords
  real :: x(1024), y(1024), z(1024)
end type
type (coords) :: p
real dsquared(1024)

do i=1,1024
  dsquared(i) = p%x(i)**2 + p%y(i)**2 + p%z(i)**2
end do
```



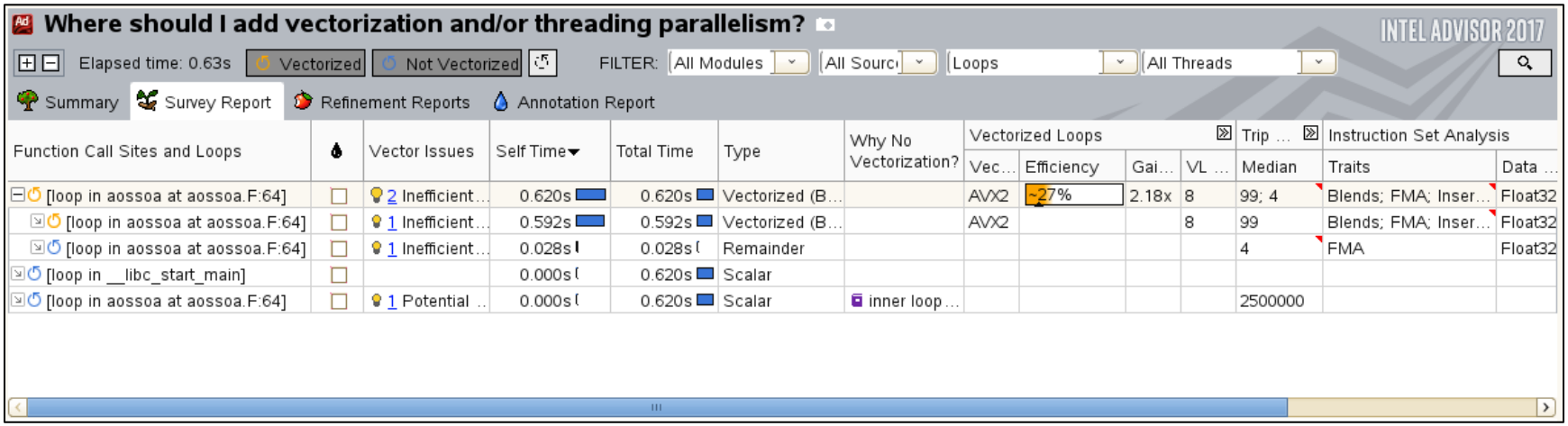
AoSoA

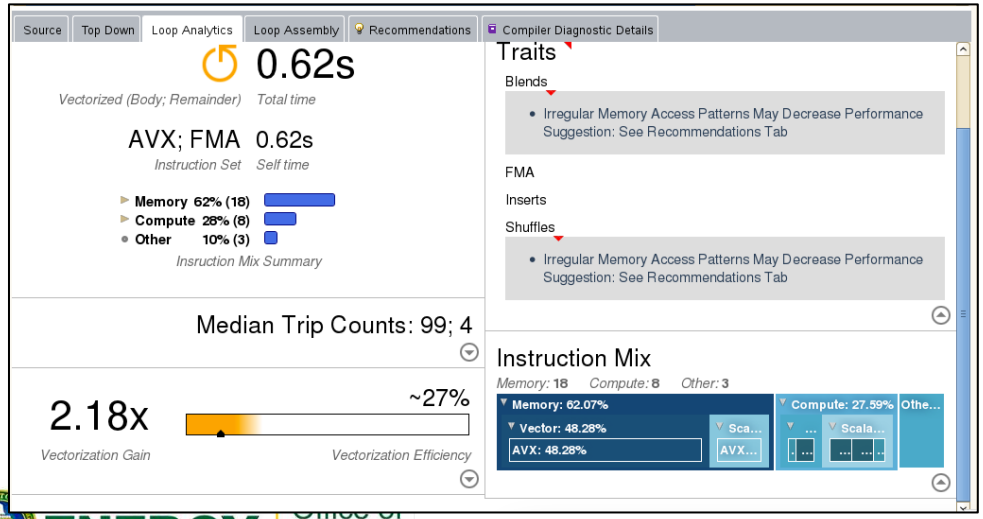
- With SoA, locality of multiple fields was reduced
- With Array of Structures of Arrays (Tiled Array of Structures), we have locality over multiple fields at the outer-level and unit-stride at the innermost-level

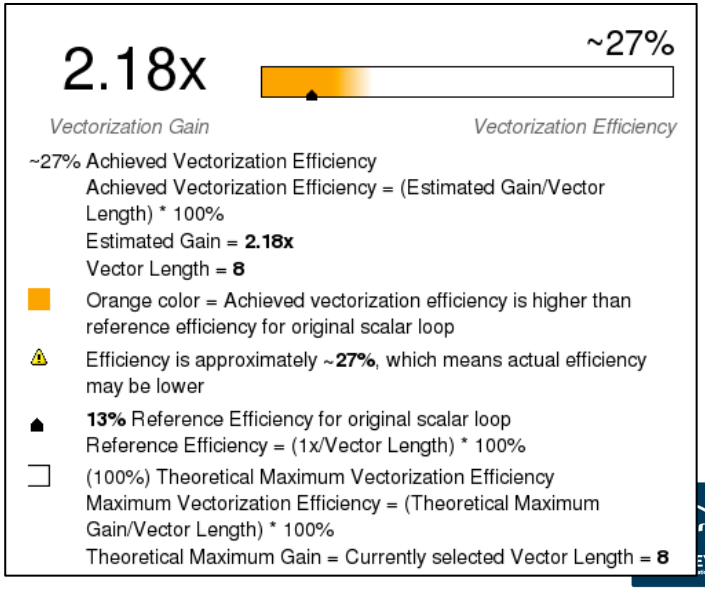
```
type coords
  real :: x(16), y(16), z(16)
end type
type (coords) :: p(64)
real dsquared(16,64)

do i=1,64
  do j=1,16
    dsquared(j,i) = p(i)%x(j)**2 + p(i)%y(j)**2 + p(i)%z(j)**2
  end do
end do
```

AoS

- **aossoa.F from http://www.nersc.gov/users/computational-systems/edison/programming/vectorization/**

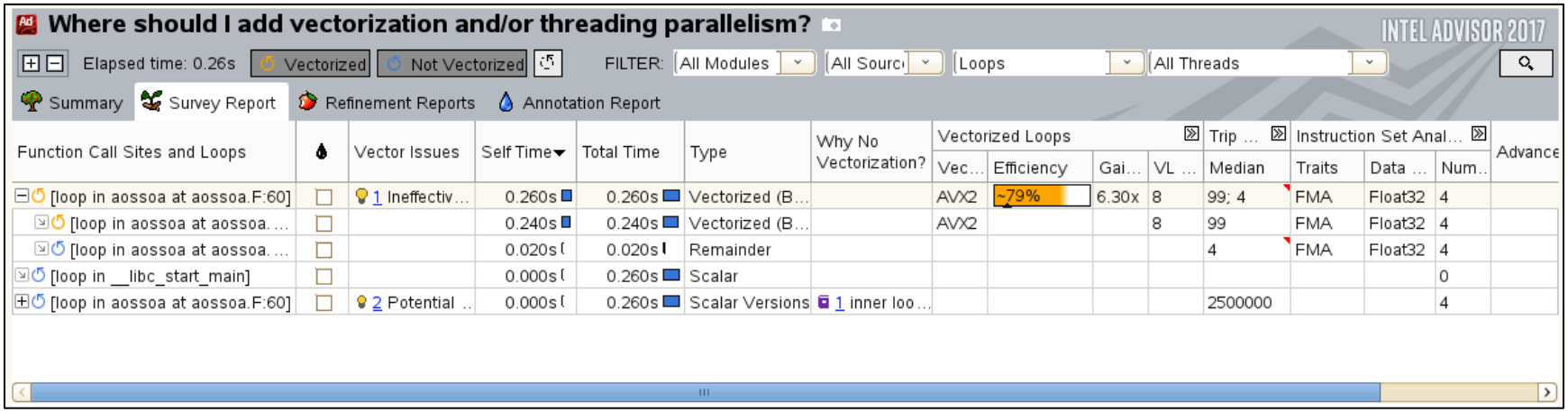


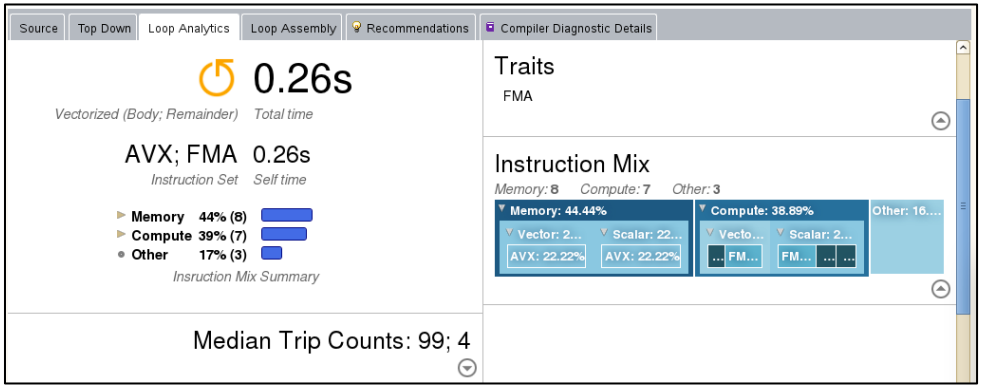


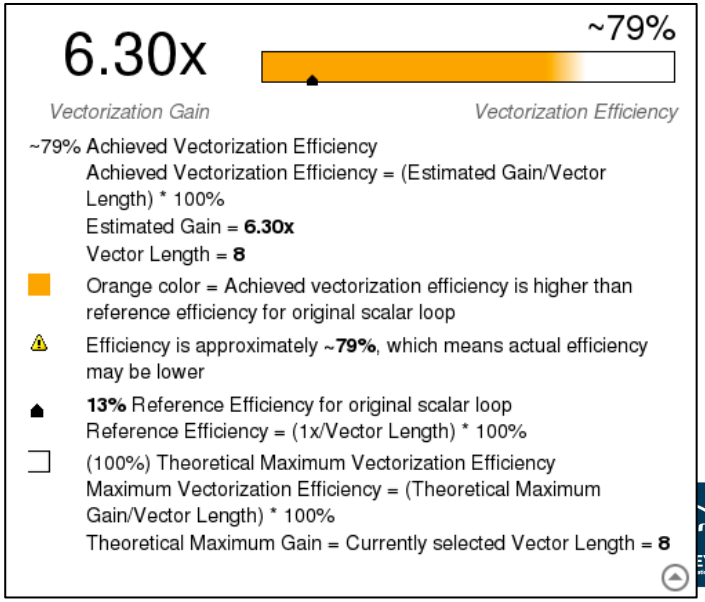


SoA

- **aossoa.F from http://www.nersc.gov/users/computational-systems/edison/programming/vectorization/**







SIMD-Enabled ("Elemental") function

- **An elemental function operates element-wise and returns an array with the same shape as the input parameter**
  - Widely used in Fortran intrinsic functions (but not in a vectorization sense)
- **When declared, the Intel compiler generates a vector version and a scalar version of the function**
- **A function call within a loop generally inhibits vectorization. But a loop containing a call to an elemental function can be vectorized. In that case, the vector version is used**

SIMD-Enabled function example

```
module fofx
contains
  function f(x)
!dir$ attributes vector :: f
    real, intent(in) :: x
    real f
    f = cos(x * x + 1.) / (x * x + 1.)
  end function
end module

program main
  use fofx
  real a(100), x(100)
  ...
  do i=1,100
    a(i) = f(x(i))
  end do
  ...
end program
```

Line 7

Line 50

```
$ ifort -qopt-report=3 elemental.F
...
 LOOP BEGIN at elemental.F(50,11)
      remark #15300: LOOP WAS VECTORIZED
...
```

OpenMP 4.0 SIMD constructs

- **SIMD constructs for execution of a loop in vectorization mode**

```
#pragma omp simd [clauses…]
```

```
!$omp simd [clauses…]
```

- **Optional clauses**
  - safelen(length)
  - aligned(list[:alignment])
  - reduction(reduction-identifier:list)
  - collapse(n)

OpenMP 4.0 SIMD constructs (Cont'd)

- **Example**

```
...
do j=1,n
  do k=1,n
!$omp simd aligned(a,b,c:32)
    do i=1,nr
      c(i,j) = c(i,j) + a(i,k) * b(k,j)
    end do
  end do
end do
...
```

OpenMP 4.0 SIMD constructs (Cont'd)

- SIMD-enabled function ("elemental function")

```
#pragma omp declare simd [clauses…]
  function definition or declaration
```

```
!$omp declare simd(proc-name) [clauses…]
  function definition
```

- Example

```
!$omp declare simd(f)
  function f(x)
    real f, x
    f = cos(x * x + 1.e0) / (x * x + 1.e0)
  end function f
...
  do i=1,n
    a(i) = f(x(i))
  end do
```

More Intel Advisor usage tips

- **Tool design and functionalities can change over time, but here are some tips for using advisor/2016.1.40.455986 (default on cori)**

- **To run a MPI code**
  - Via advixe-cl only (not GUI)
  - Dynamically-linked executable
  - Only one rank run through advixe-cl
  - Run in MPMD mode if # of tasks > 1

```
$ salloc -N 1 -t 30:00 -p debug
...
$ ftn -dynamic -g -O3 myprog.f
$ module load advisor

$ cat mpmd.conf                    Rank 0 with advixe-cl; ranks 1-9, without
0 advixe-cl --collect survey --project-dir ./myproj -- ./a.out
1-9 ./a.out

$ srun --multi-prog ./mpmd.conf
```

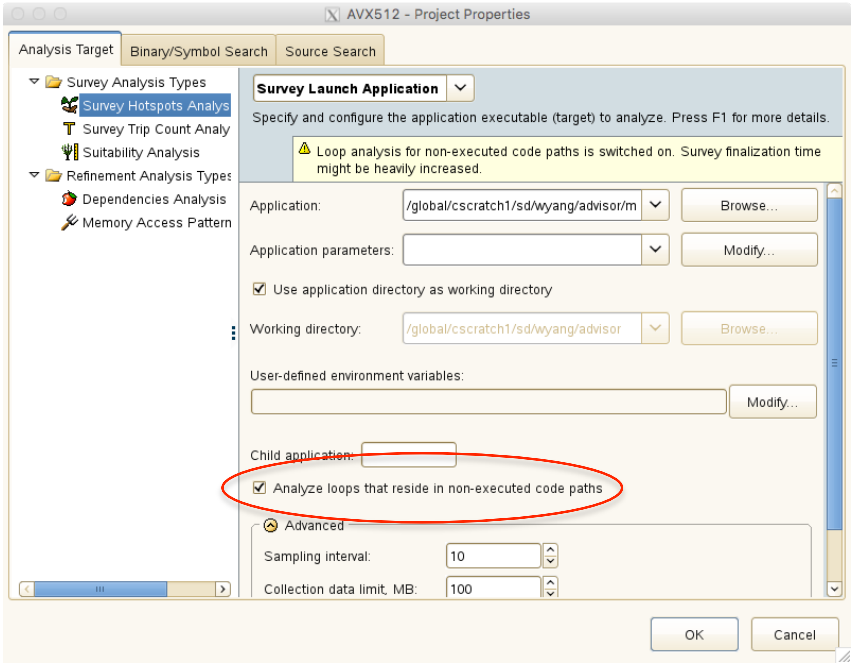
More Intel Advisor usage tips (Cont'd)

- Intel compiler can generate multiple instruction sets although they may not be executable on your machine
- Below is to generate an executable on Haswell nodes (-xCORE-AVX2) that also contains AVX-512 instructions (-axCORE-AVX512), to get a glimpse of its expected performance

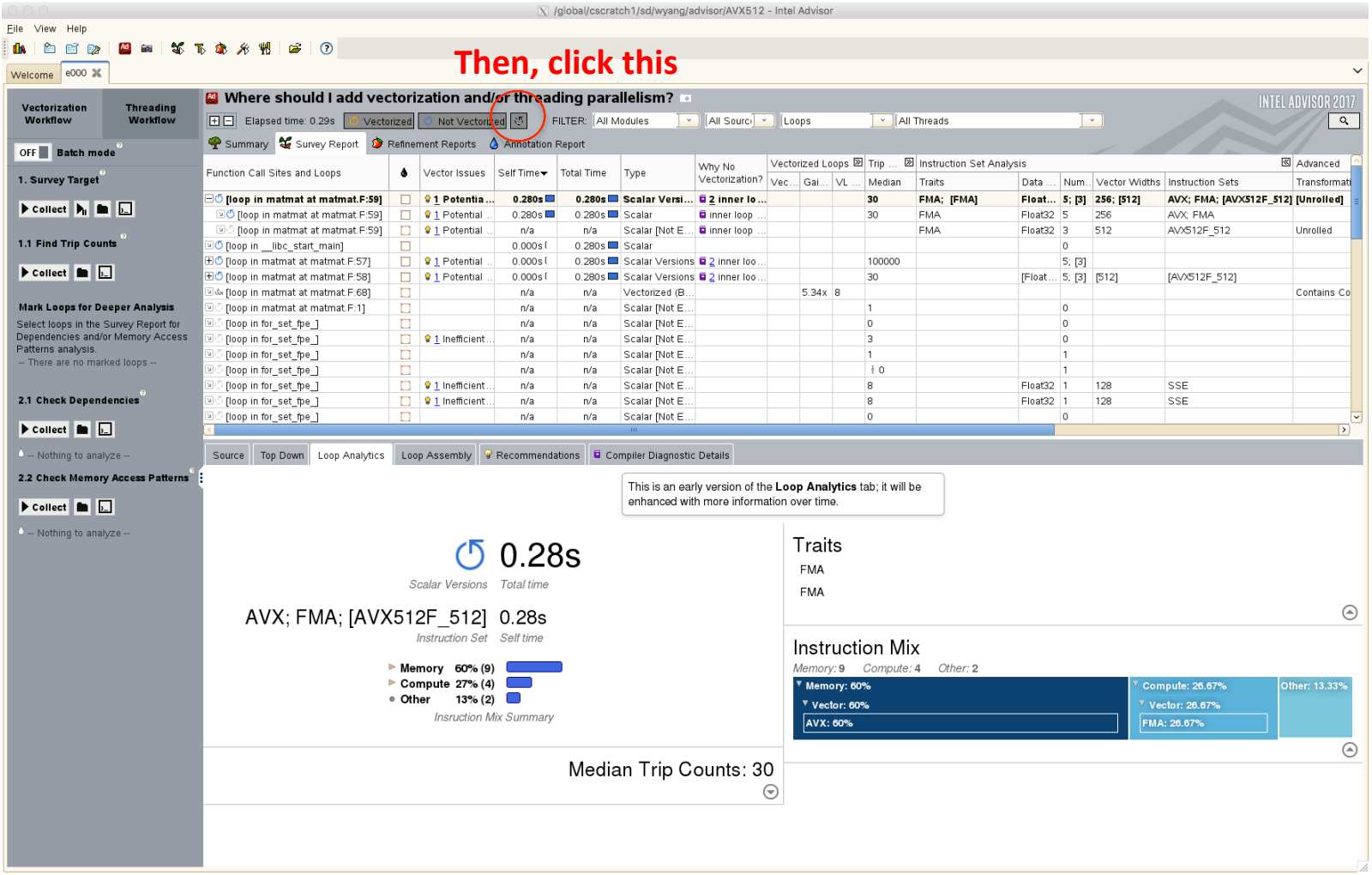
```
$ salloc -N 1 -t 30:00 -p debug
...
$ ftn -dynamic -g -O3 -xCORE-AVX2 -axCORE-AVX512 myprog.f
$ module load advisor
$ advixe-cl --collect survey --support-multi-isa-binaries \
  --project-dir ./myproj -- ./a.out
```

More Intel Advisor usage tips (Cont'd)

- Select 'Analyze loops that reside in non-executed code paths' in the Project Properties window (reached thru GUI's 'File -> Project Properties')
- See https://software.intel.com/en-us/blogs/2016/02/02/explore-intel-avx-512-code-paths-while-not-having-compatible-hardware



More Intel Advisor usage tips (Cont'd)



Vectorization sample codes

- http://www.nersc.gov/users/computational-systems/edison/programming/vectorization/
- Intel-provided samples in $ADVISOR_XE_2016_DIR/samples/en

```
$ module load advisor

$ ls $ADVISOR_XE_2016_DIR/samples/en/C++
Vector_Tutorial_Data_Alignment.tgz
Vector_Tutorial_Introduction.tgz
Vector_Tutorial_Memory_Access_101.tgz
Vector_Tutorial_Stride_and_MAP.tgz
Vector_Tutorial_Vectorization_and_Data_Size.tgz
mmult_Advisor.tgz
mpi_sample.tgz
nqueens_Advisor.tgz
tachyon_Advisor.tgz
vec_samples.tgz

$ ls $ADVISOR_XE_2016_DIR/samples/en/Fortran
mmult.tgz
nqueens.tgz
```

Intel Advisor 2016 tutorial

- https://software.intel.com/en-us/intel-advisor-2016-tutorial-vectorization-linux-cplusplus
- **Uses $ADVISOR_XE_2016_DIR/samples/en/C++/vec_samples.tgz**

National Energy Research Scientific Computing Center