INTEL® ADVISOR

Part of Intel® Parallel Studio XE

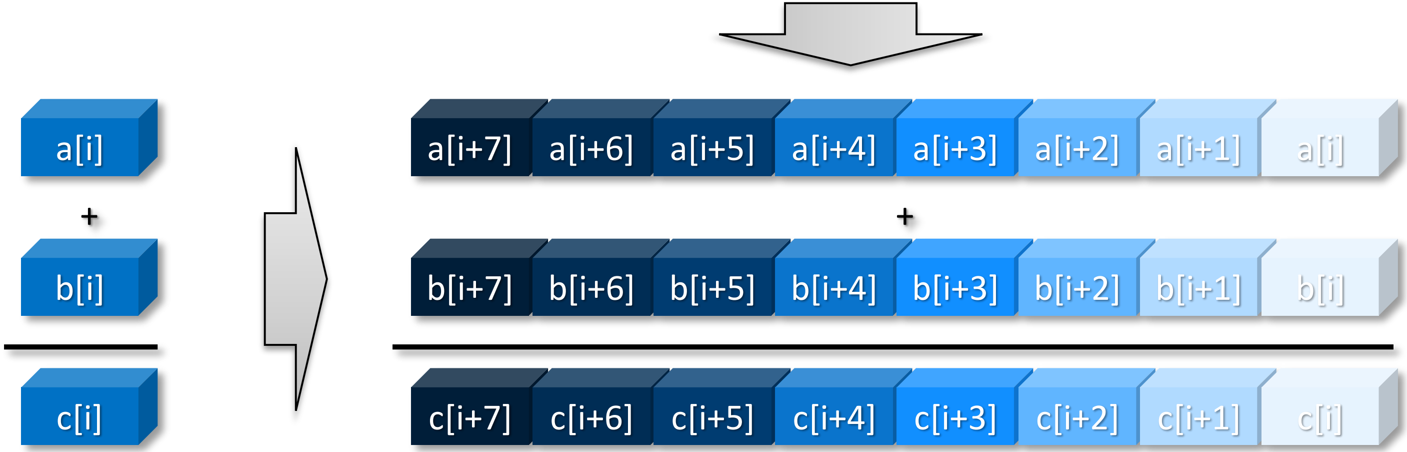
Roadmap Notice: All information provided here is subject to change without notice.
Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

VECTOR SIMD PARALLELISM, VECTORIZATION

VECTORIZATION OF CODE

- Transform sequential code to exploit vector processing capabilities

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

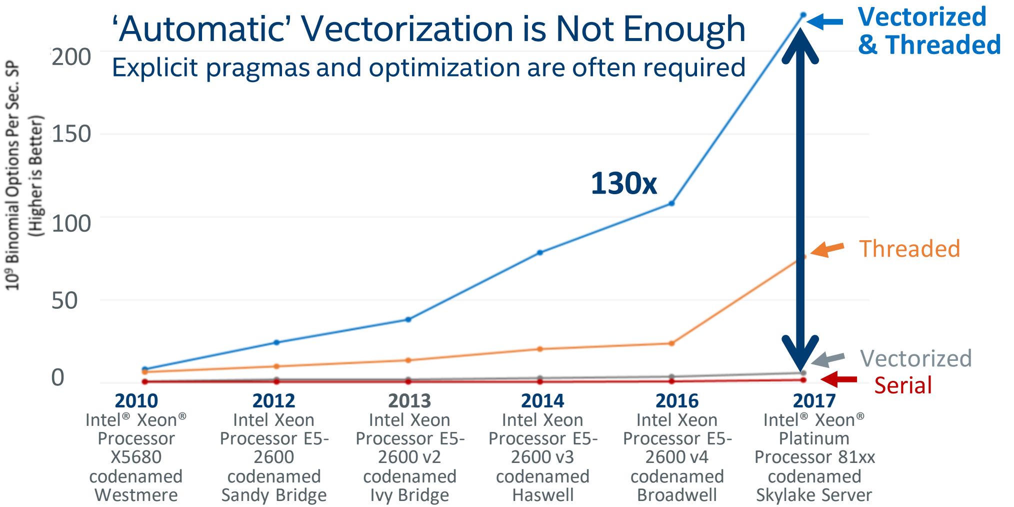

a[i]
+
b[i]
c[i]
a[i+7] a[i+6] a[i+5] a[i+4] a[i+3] a[i+2] a[i+1] a[i]
+
b[i+7] b[i+6] b[i+5] b[i+4] b[i+3] b[i+2] b[i+1] b[i]
c[i+7] c[i+6] c[i+5] c[i+4] c[i+3] c[i+2] c[i+1] c[i]

INTEL® ADVISOR

MODERNIZE YOUR CODE WITH INTEL® ADVISOR

OPTIMIZE VECTORIZATION, PROTOTYPE THREADING, CREATE & ANALYZE FLOW GRAPHS

The Difference Is Growing with Each New Hardware Generation



'Automatic' Vectorization is Not Enough

Explicit pragmas and optimization are often required

- Modern Performant Code
  - Vectorized (uses Intel® AVX-512/AVX2)
  - Efficient memory access
  - Threaded
- Capabilities
  - Adds & optimizes vectorization
  - Analyzes memory patterns
  - Quickly prototypes threading

Benchmark: Binomial Options Pricing Model https://software.intel.com/en-us/articles/binomial-options-pricing-model-code-for-intel-xeon-phi-coprocessor

Performance results are based on testing as of August 2017 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks See Vectorize & Thread or Performance Dies Configurations for 2010-2017 Benchmarks in Backup. Testing by Intel as of August 2017.

Learn More: http: intel.ly/advisor-xe

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, & SSSE3 instruction sets & 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

PERMISSION TO DESIGN FOR ALL LANES

THREADING AND VECTORIZATION NEEDED TO FULLY UTILIZE MODERN HARDWARE

For more complete information about compiler optimizations, see our Optimization Notice.

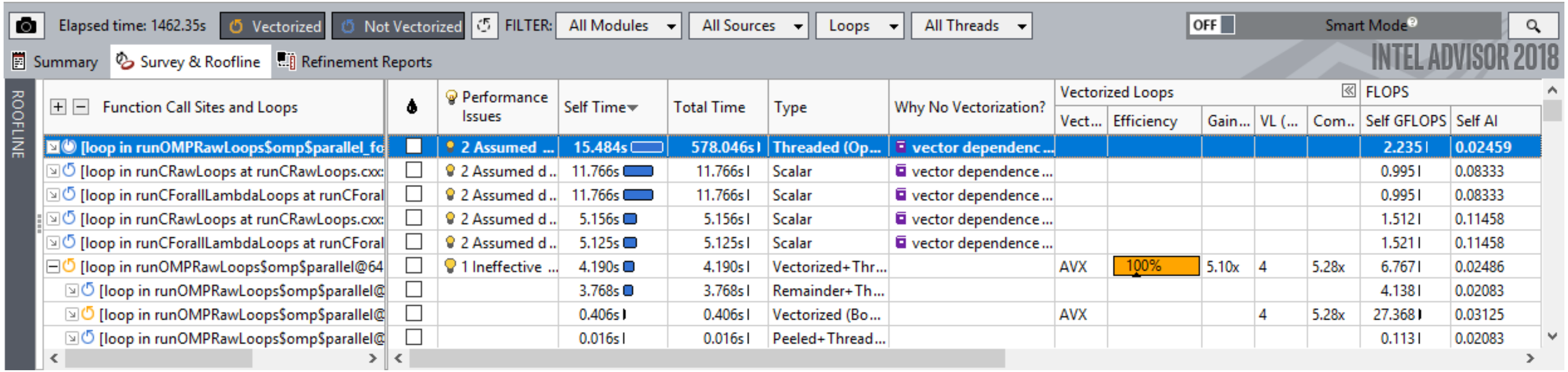
INTEL® ADVISOR: VECTORIZATION OPTIMIZATION

Have you:

- Recompiled for AVX2 with little gain?
- Wondered where to vectorize?
- Recoded intrinsics for new arch.?
- Struggled with compiler reports?

Data Driven Vectorization:

- What vectorization will pay off most?
- What's blocking vectorization? Why?
- Are my loops vector friendly?
- Will reorganizing data increase performance?
- Is it safe to just use pragma simd?

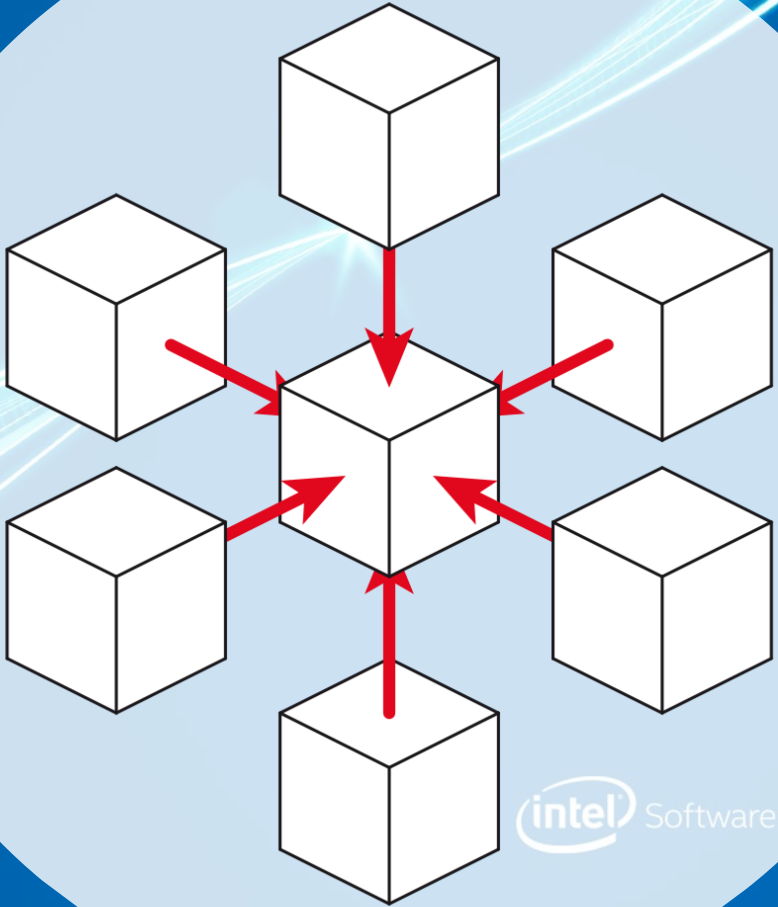


Function Call Sites and Loops		Performance Issues	Self Time	Total Time	Type	Why No Vectorization?	Vectorized Loops					FLOPS	
							Vect...	Efficiency	Gain...	VL (...	Com..	Self GFLOPS	Self AI
[loop in runOMPRawLoops$omp$parallel_fo	□	2 Assumed ...	15.484s	578.046s	Threaded (Op...	vector dependenc...						2.235	0.02459
[loop in runCRawLoops at runCRawLoops.cxx:	□	2 Assumed d..	11.766s	11.766s	Scalar	vector dependence...						0.995	0.08333
[loop in runCForallLambdaLoops at runCForal	□	2 Assumed d..	11.766s	11.766s	Scalar	vector dependence...						0.995	0.08333
[loop in runCRawLoops at runCRawLoops.cxx:	□	2 Assumed d..	5.156s	5.156s	Scalar	vector dependence...						1.512	0.11458
[loop in runCForallLambdaLoops at runCForal	□	2 Assumed d..	5.125s	5.125s	Scalar	vector dependence...						1.521	0.11458
[loop in runOMPRawLoops$omp$parallel@64	□	1 Ineffective ...	4.190s	4.190s	Vectorized+Thr...		AVX	100%	5.10x	4	5.28x	6.767	0.02486
[loop in runOMPRawLoops$omp$parallel@	□		3.768s	3.768s	Remainder+Th...							4.138	0.02083
[loop in runOMPRawLoops$omp$parallel@	□		0.406s	0.406s	Vectorized (Bo...		AVX			4	5.28x	27.368	0.03125
[loop in runOMPRawLoops$omp$parallel@	□		0.016s	0.016s	Peeled+Thread...							0.113	0.02083

THE LAB ACTIVITIES

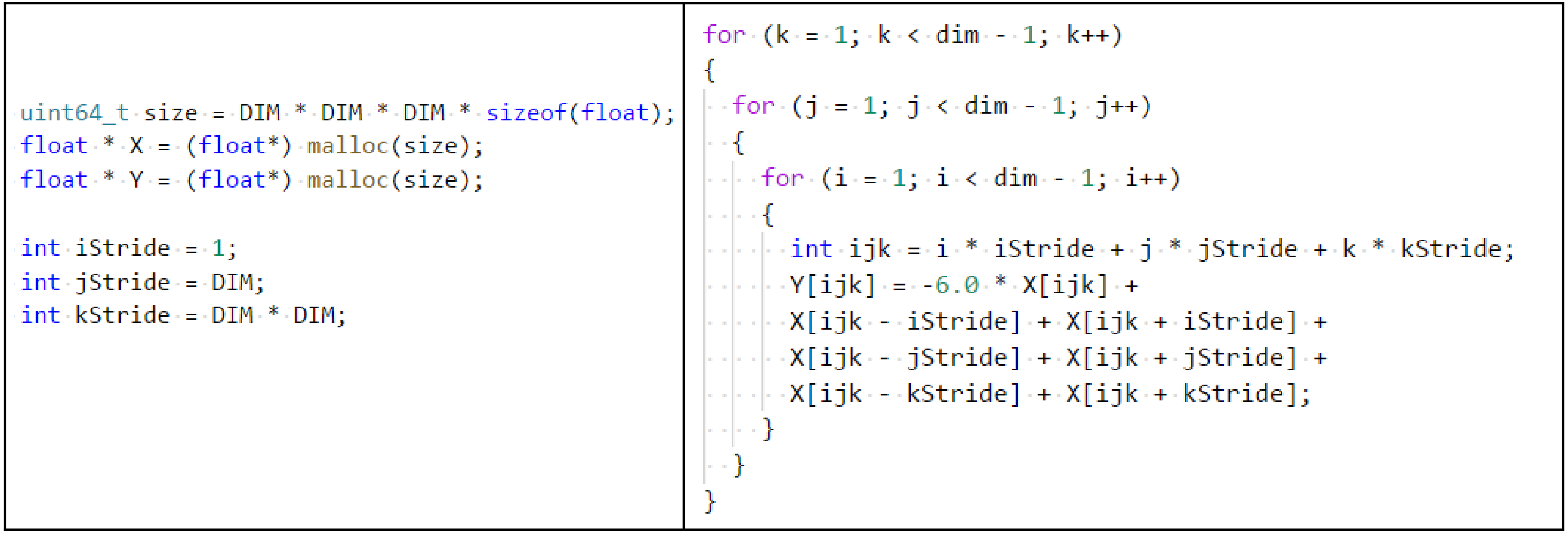
- Activity 0: Building Stencil
- Activity 1: Doing Survey
- Activity 2: Dealing with **data type conversions**
- Activity 3: Checking for dependencies
- Activity 4: Adding **threading and** trying to enable **vectorization**
- Activity 5: Checking Memory Access Patterns
- Activity 6: Making **unit stride** explicit
- Activity 7: Doing Roofline analysis
- Activity 8: Splitting task **to tiles**
- Activity 9: Enabling **AVX512**
- Activity 10: Comparing roofline charts

STENCIL



STENCIL CODE EXAMPLE

- Consider solving differential equation with finite-difference method on 3-dimensional grid
- Example: calculating Laplace operator of some field



```c
uint64_t size = DIM * DIM * DIM * sizeof(float);
float * X = (float*) malloc(size);
float * Y = (float*) malloc(size);

int iStride = 1;
int jStride = DIM;
int kStride = DIM * DIM;
```

```c
for (k = 1; k < dim - 1; k++)
{
  for (j = 1; j < dim - 1; j++)
  {
    for (i = 1; i < dim - 1; i++)
    {
      int ijk = i * iStride + j * jStride + k * kStride;
      Y[ijk] = -6.0 * X[ijk] +
      X[ijk - iStride] + X[ijk + iStride] +
      X[ijk - jStride] + X[ijk + jStride] +
      X[ijk - kStride] + X[ijk + kStride];
    }
  }
}
```

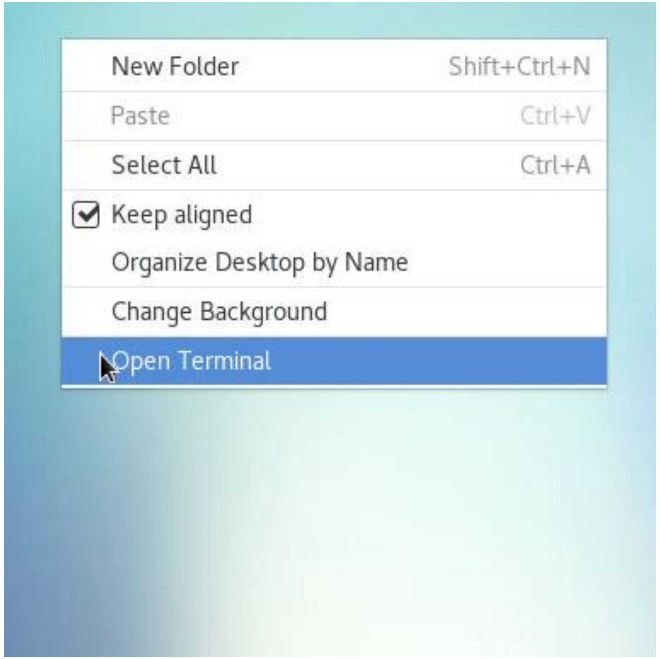
ACTIVITY 0: BUILDING STENCIL

BUILD & RUN

Purpose: Build an application, observe the performance

- Launch Terminal:

  Right click -> Open Terminal



BUILD & RUN

- Setup environment:

```
$ source /opt/intel/parallel_studio_xe_2019/psxevars.sh intel64
```

- Go to working directory

```
$ cd lab2
```

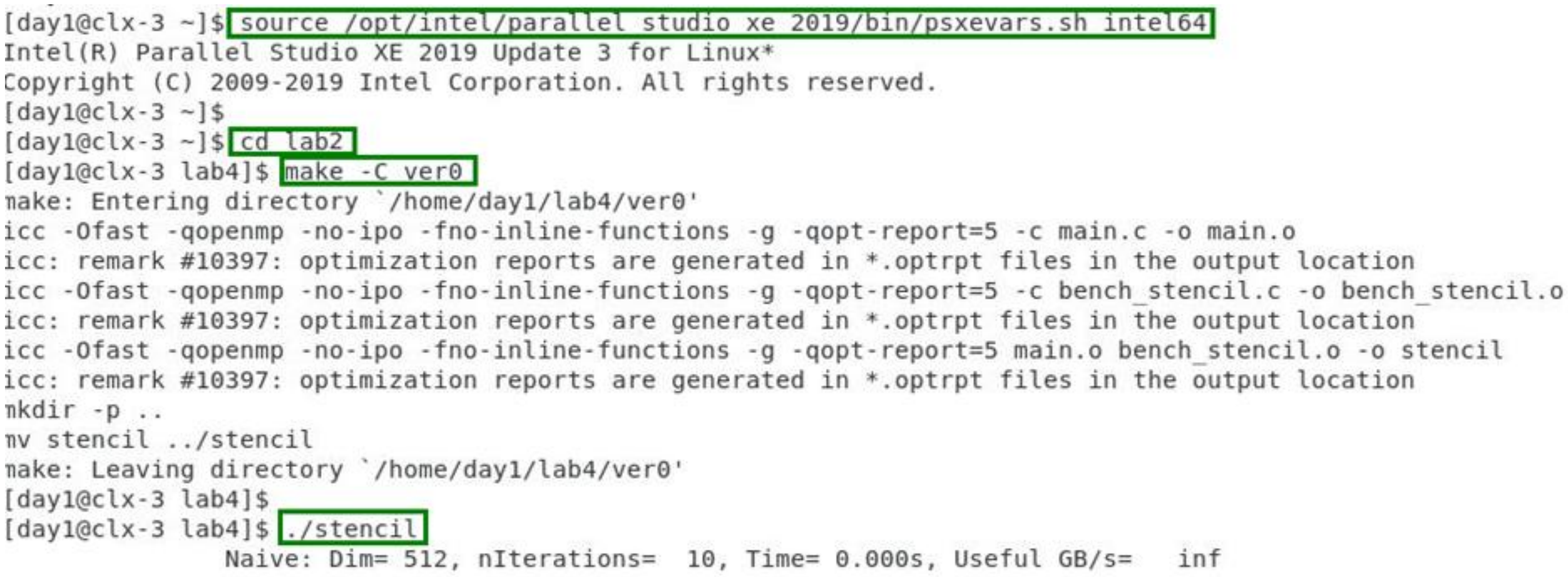
- Build application

```
$ make -C ver0
```

- Run application

```
$ ./stencil
```

ACTIVITY 0. SCREENSHOT



```
[day1@clx-3 ~]$ source /opt/intel/parallel studio xe 2019/bin/psxevars.sh intel64
Intel(R) Parallel Studio XE 2019 Update 3 for Linux*
Copyright (C) 2009-2019 Intel Corporation. All rights reserved.
[day1@clx-3 ~]$
[day1@clx-3 ~]$ cd lab2
[day1@clx-3 lab4]$ make -C ver0
nake: Entering directory `/home/day1/lab4/ver0'
icc -Ofast -qopenmp -no-ipo -fno-inline-functions -g -qopt-report=5 -c main.c -o main.o
icc: remark #10397: optimization reports are generated in *.optrpt files in the output location
icc -Ofast -qopenmp -no-ipo -fno-inline-functions -g -qopt-report=5 -c bench_stencil.c -o bench_stencil.o
icc: remark #10397: optimization reports are generated in *.optrpt files in the output location
icc -Ofast -qopenmp -no-ipo -fno-inline-functions -g -qopt-report=5 main.o bench_stencil.o -o stencil
icc: remark #10397: optimization reports are generated in *.optrpt files in the output location
nkdir -p ..
nv stencil ../stencil
nake: Leaving directory `/home/day1/lab4/ver0'
[day1@clx-3 lab4]$
[day1@clx-3 lab4]$ ./stencil
               Naive: Dim= 512, nIterations=  10, Time= 0.000s, Useful GB/s=   inf
```

ACTIVITY 1: DOING SURVEY

LAUNCH ADVISOR

Purpose: Run Survey analysis in Advisor to get the baseline version

- Open new terminal tab

  File -> New Tab

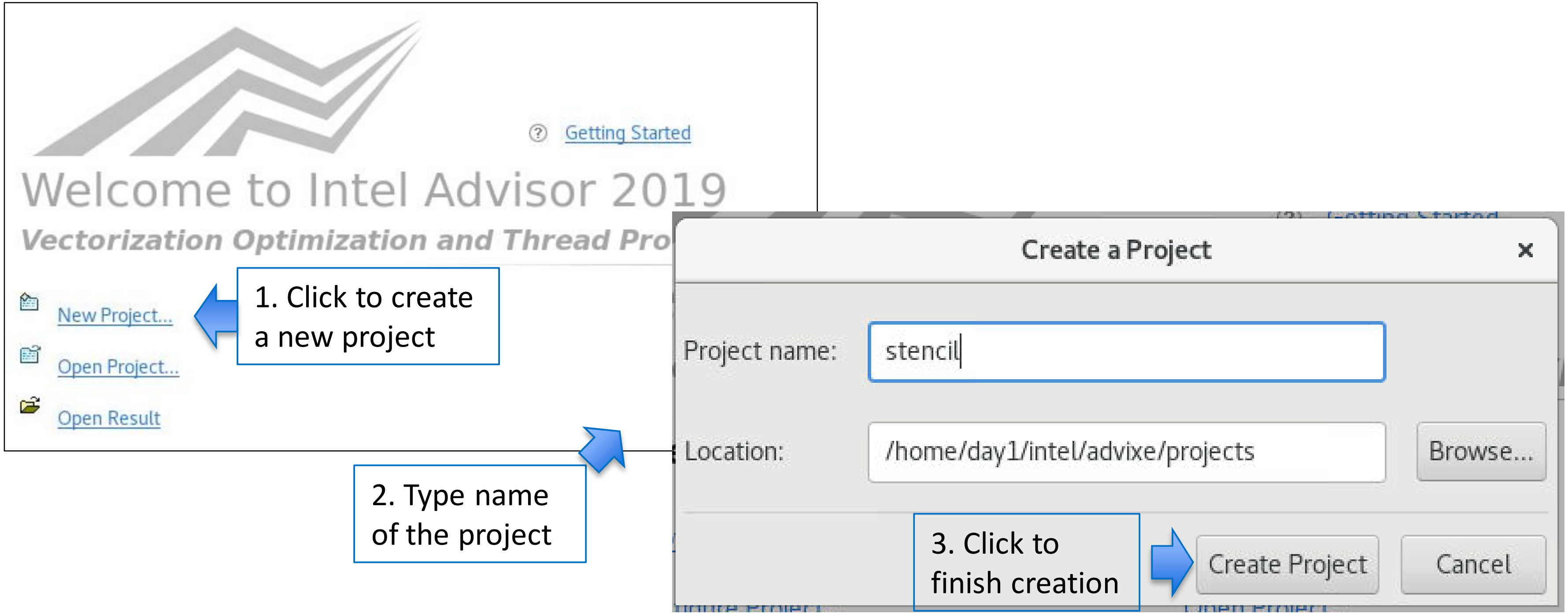
- Setup environment:

  ```
  $ source ./advixe_vars.sh
  ```

- Launch Advisor GUI:

  ```
  $ advixe-gui
  ```

CREATE ADVISOR PROJECT



Project name: stencil

Location: /home/day1/intel/advixe/projects

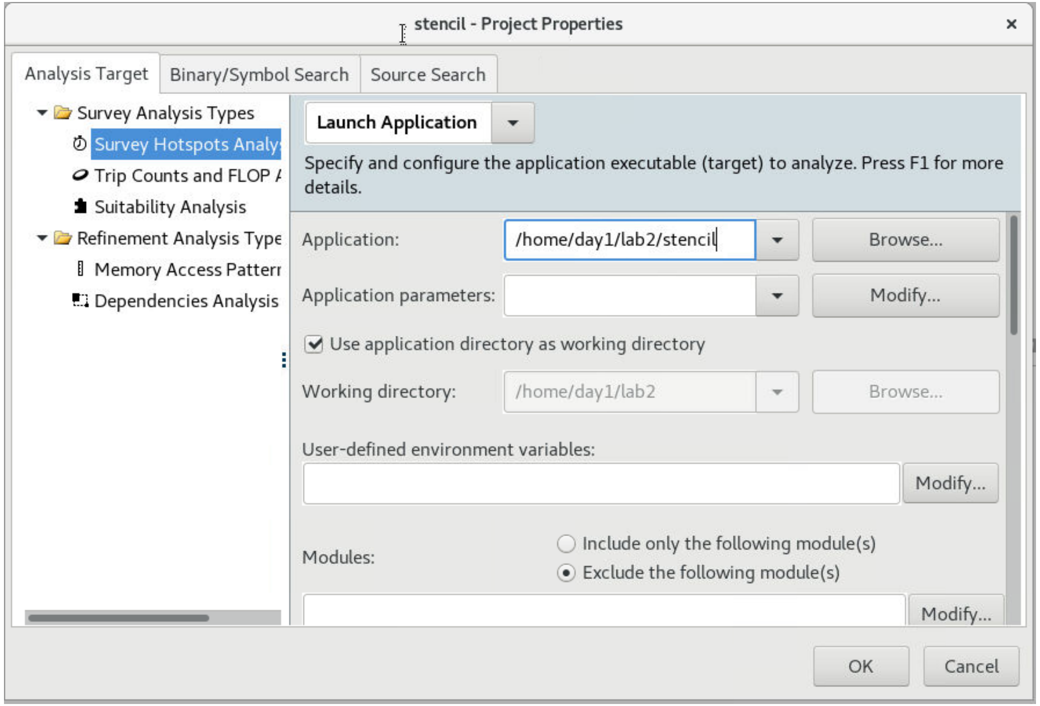
1. Click to create a new project

2. Type name of the project

3. Click to finish creation

SET UP PROJECT

- Set the application to launch:
  /home/day1/lab2/stencil
- Press OK button



Specify and configure the application executable (target) to analyze. Press F1 for more details.

Application: /home/day1/lab2/stencil

Application parameters:

☑ Use application directory as working directory

Working directory: /home/day1/lab2

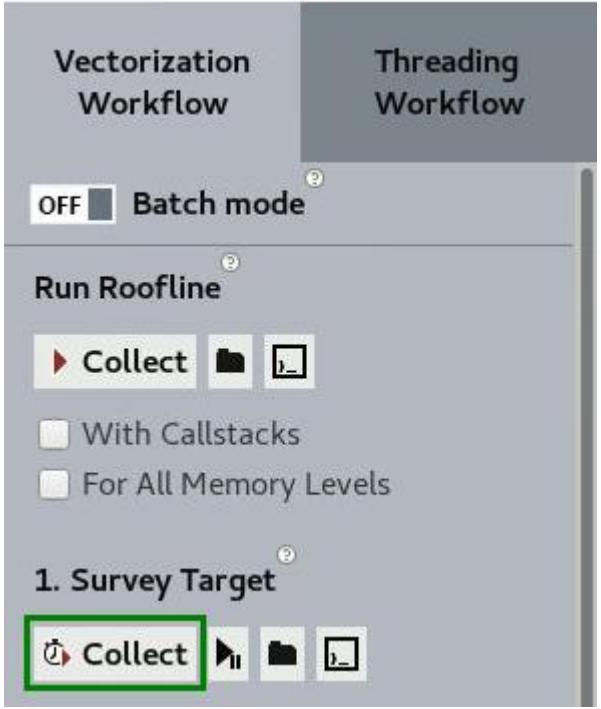
User-defined environment variables:

Modules:
○ Include only the following module(s)
● Exclude the following module(s)

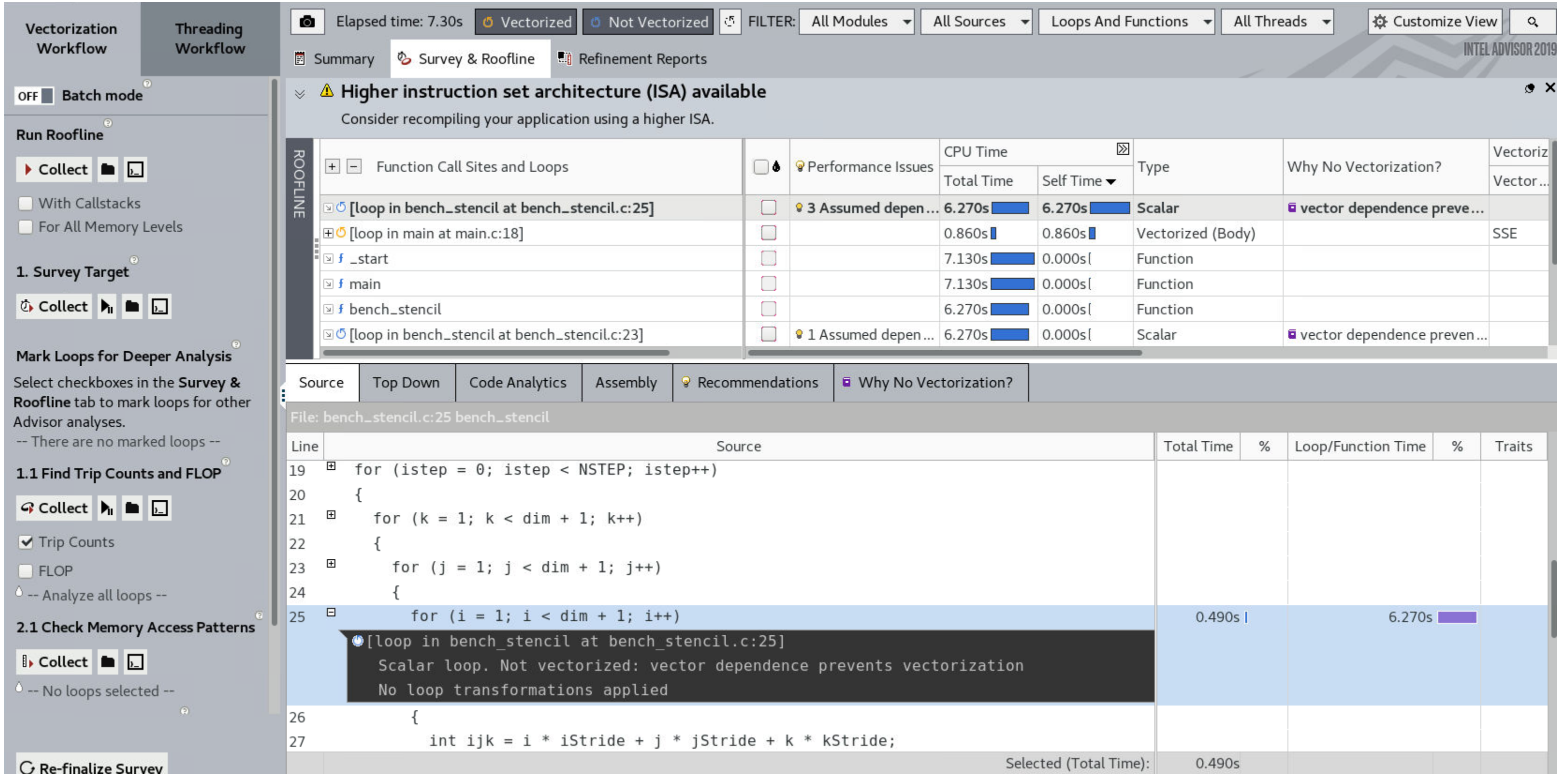
For more complete information about compiler optimizations, see our Optimization Notice.

START SURVEY ANALYSIS

- Press “Collect” button in “1. Survey Target” section



ACTIVITY 1. SCREENSHOT



Elapsed time: 7.30s

Select checkboxes in the **Survey & Roofline** tab to mark loops for other Advisor analyses.

-- There are no marked loops --

☑ Trip Counts

□ FLOP

-- Analyze all loops --

-- No loops selected --

Higher instruction set architecture (ISA) available

Consider recompiling your application using a higher ISA.

Function Call Sites and Loops	□	Performance Issues	CPU Time Total Time	 Self Time	Type	Why No Vectorization?	Vectoriz Vector...
[loop in bench_stencil at bench_stencil.c:25]	□	3 Assumed depen...	6.270s	6.270s	Scalar	vector dependence preve...	
[loop in main at main.c:18]	□		0.860s	0.860s	Vectorized (Body)		SSE
_start	□		7.130s	0.000s	Function		
main	□		7.130s	0.000s	Function		
bench_stencil	□		6.270s	0.000s	Function		
[loop in bench_stencil at bench_stencil.c:23]	□	1 Assumed depen...	6.270s	0.000s	Scalar	vector dependence preven...	

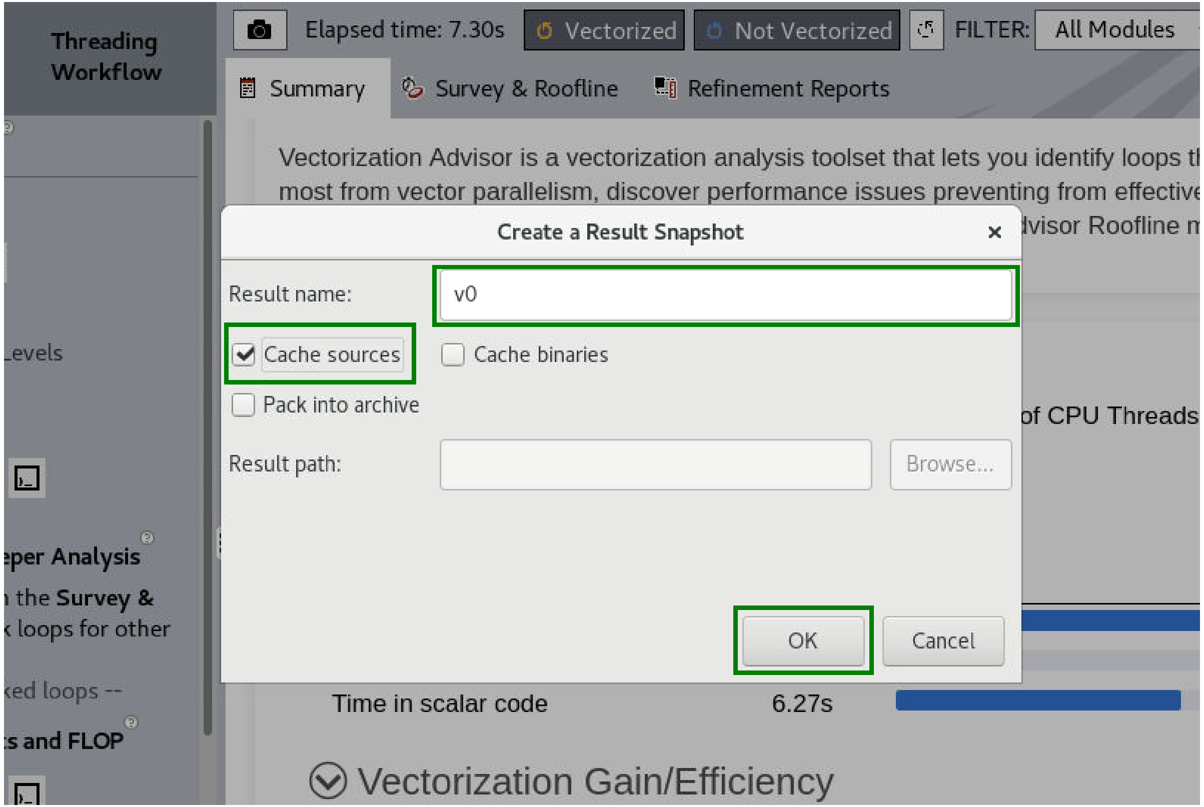
File: bench_stencil.c:25 bench_stencil

Line	Source	Total Time	%	Loop/Function Time	%	Traits

```
19   for (istep = 0; istep < NSTEP; istep++)
20   {
21     for (k = 1; k < dim + 1; k++)
22     {
23       for (j = 1; j < dim + 1; j++)
24       {
25         for (i = 1; i < dim + 1; i++)                0.490s    6.270s
            [loop in bench_stencil at bench_stencil.c:25]
             Scalar loop. Not vectorized: vector dependence prevents vectorization
             No loop transformations applied
26         {
27           int ijk = i * iStride + j * jStride + k * kStride;
```

Selected (Total Time): 0.490s

CREATE A SNAPSHOT



Create a Result Snapshot

Result name: v0

☑ Cache sources

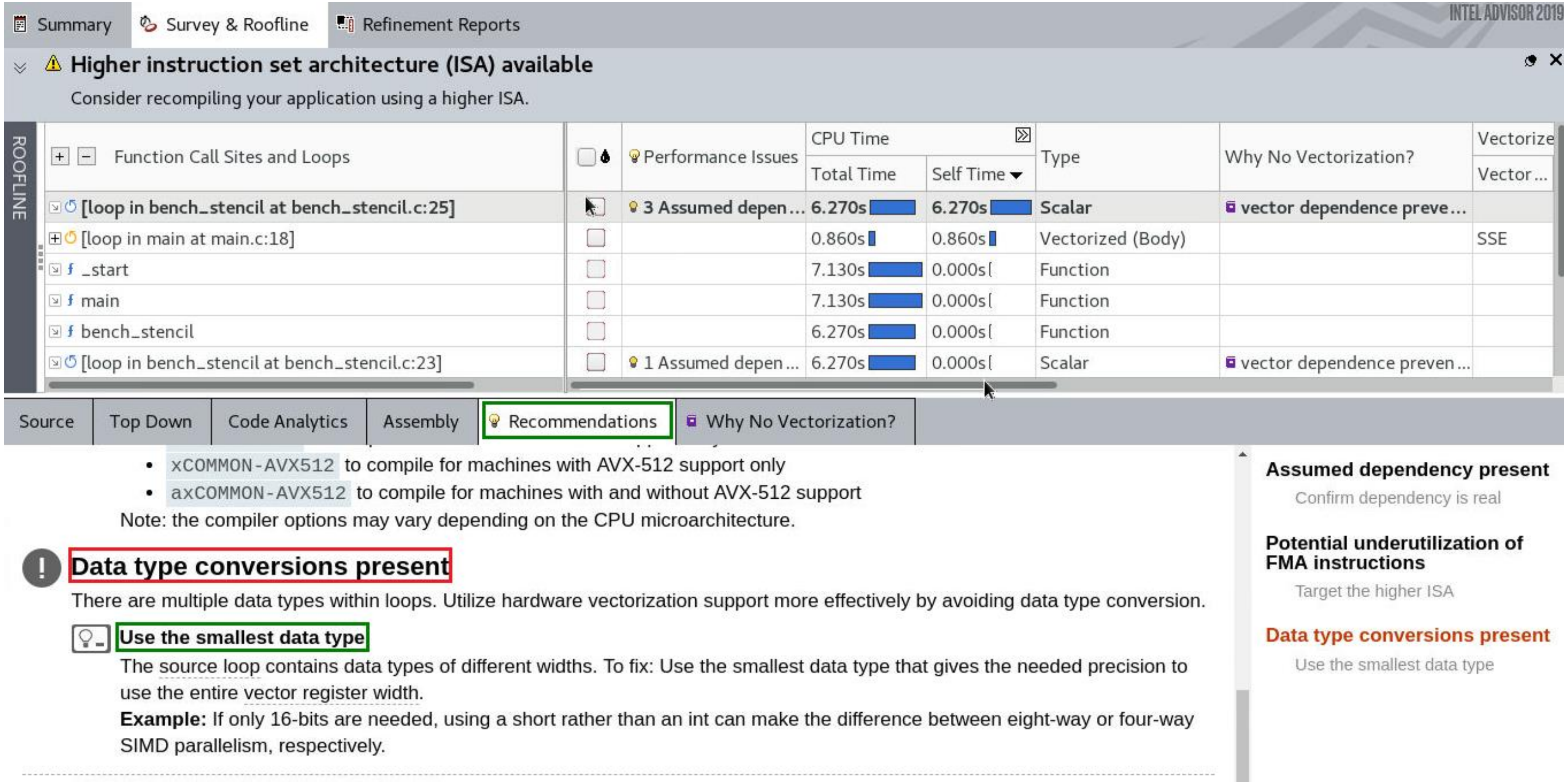
□ Cache binaries

□ Pack into archive

Result path:

ACTIVITY 2: DEALING WITH DATA TYPE CONVERSIONS

LOOK AT THE RECOMMENDATIONS



Higher instruction set architecture (ISA) available

Consider recompiling your application using a higher ISA.

Function Call Sites and Loops	□	Performance Issues	CPU Time Total Time	 Self Time	Type	Why No Vectorization?	Vectorize Vector...
[loop in bench_stencil at bench_stencil.c:25]	□	**3 Assumed depen...**	**6.270s**	**6.270s**	**Scalar**	**vector dependence preve...**	
[loop in main at main.c:18]	□		0.860s	0.860s	Vectorized (Body)		SSE
_start	□		7.130s	0.000s	Function		
main	□		7.130s	0.000s	Function		
bench_stencil	□		6.270s	0.000s	Function		
[loop in bench_stencil at bench_stencil.c:23]	□	1 Assumed depen...	6.270s	0.000s	Scalar	vector dependence preven...	

- `xCOMMON-AVX512` to compile for machines with AVX-512 support only
- `axCOMMON-AVX512` to compile for machines with and without AVX-512 support

Note: the compiler options may vary depending on the CPU microarchitecture.

Data type conversions present

There are multiple data types within loops. Utilize hardware vectorization support more effectively by avoiding data type conversion.

Use the smallest data type

The source loop contains data types of different widths. To fix: Use the smallest data type that gives the needed precision to use the entire vector register width.

Example: If only 16-bits are needed, using a short rather than an int can make the difference between eight-way or four-way SIMD parallelism, respectively.

Assumed dependency present

Confirm dependency is real

Potential underutilization of FMA instructions

Target the higher ISA

Data type conversions present

Use the smallest data type

ACTIVITY 2

Purpose: Identify and fix data type conversion issue

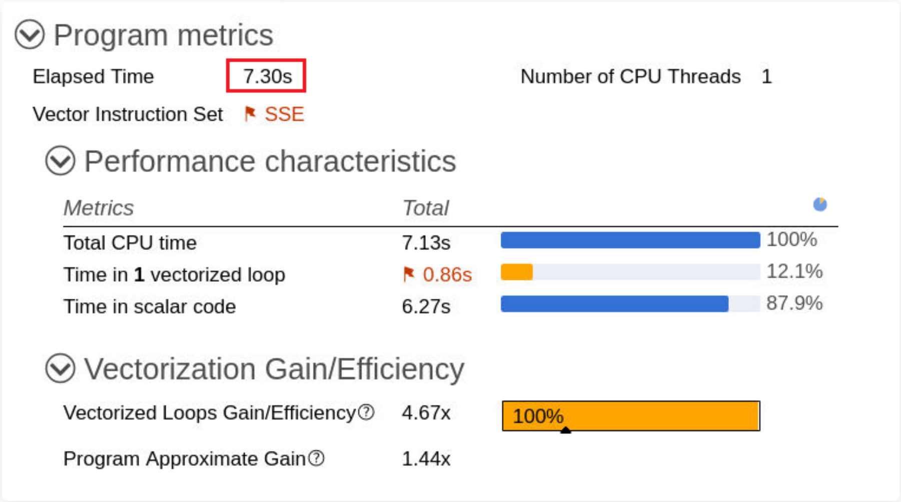
- Build version without data type conversions

```
$ make -C ver1
```

- Re-run Survey analysis
- Create a snapshot
- Compare with previous version

ACTIVITY 2. VERSION COMPARISON

1,414x ↑



Program metrics

Elapsed Time 7.30s

Number of CPU Threads 1

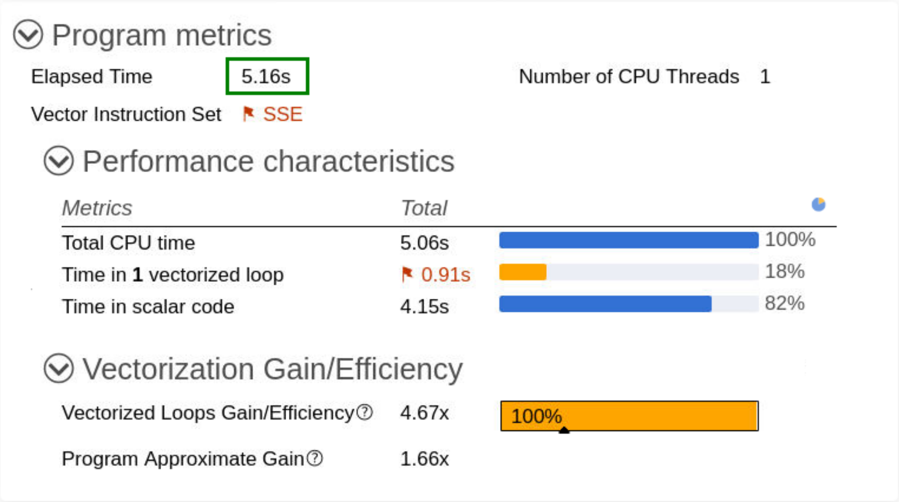
Vector Instruction Set ⚑ SSE

Performance characteristics

Metrics	*Total*		
Total CPU time	7.13s		100%
Time in **1** vectorized loop	⚑ 0.86s		12.1%
Time in scalar code	6.27s		87.9%

Vectorization Gain/Efficiency

Vectorized Loops Gain/Efficiency ⓘ	4.67x	100%
Program Approximate Gain ⓘ	1.44x	



Program metrics

Elapsed Time 5.16s

Number of CPU Threads 1

Vector Instruction Set ⚑ SSE

Performance characteristics

Metrics	*Total*		
Total CPU time	5.06s		100%
Time in **1** vectorized loop	⚑ 0.91s		18%
Time in scalar code	4.15s		82%

Vectorization Gain/Efficiency

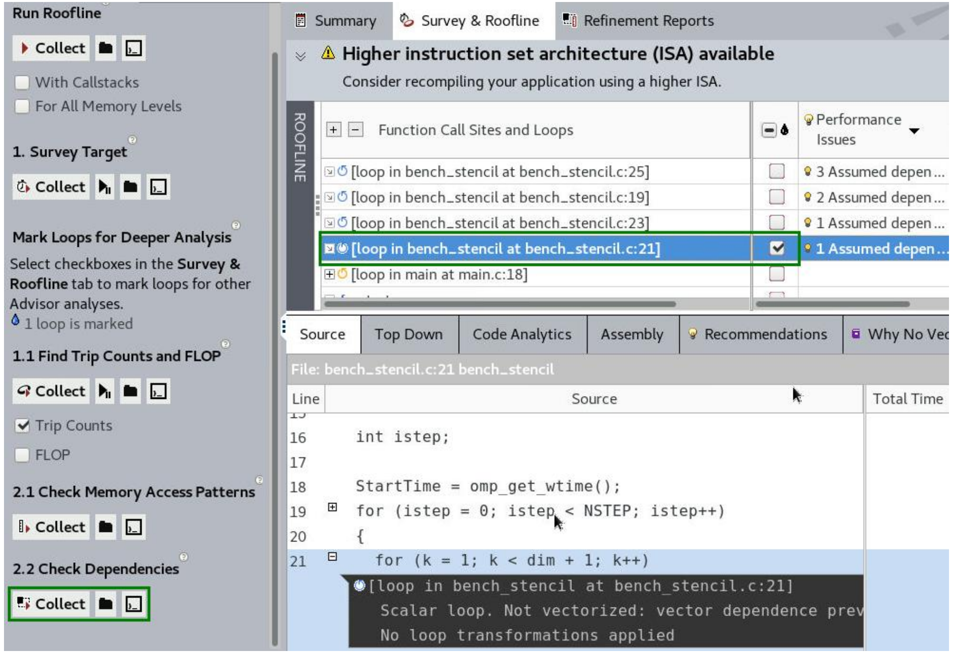
Vectorized Loops Gain/Efficiency ⓘ	4.67x	100%
Program Approximate Gain ⓘ	1.66x	

ACTIVITY 3: CHECKING FOR DEPENDENCIES

ACTIVITY 3. COLLECT DATA TO GET DEPENDENCIES

Purpose: Find loop-carried dependencies

- Select **[loop in bench_stencil at bench_stencil.c:21]**
- Press "Collect" button in "2.2 Check Dependencies" section
- Wait ~1 minute
- Create a snapshot



Run Roofline

Collect

□ With Callstacks
□ For All Memory Levels

1. Survey Target

Collect

Mark Loops for Deeper Analysis

Select checkboxes in the **Survey & Roofline** tab to mark loops for other Advisor analyses.
1 loop is marked

1.1 Find Trip Counts and FLOP

Collect

☑ Trip Counts
□ FLOP

2.1 Check Memory Access Patterns

Collect

2.2 Check Dependencies

Collect

Summary | Survey & Roofline | Refinement Reports

Higher instruction set architecture (ISA) available
Consider recompiling your application using a higher ISA.

ROOFLINE

Function Call Sites and Loops		Performance Issues
[loop in bench_stencil at bench_stencil.c:25]	□	3 Assumed depen...
[loop in bench_stencil at bench_stencil.c:19]	□	2 Assumed depen...
[loop in bench_stencil at bench_stencil.c:23]	□	1 Assumed depen...
[loop in bench_stencil at bench_stencil.c:21]	☑	1 Assumed depen...
[loop in main at main.c:18]	□	

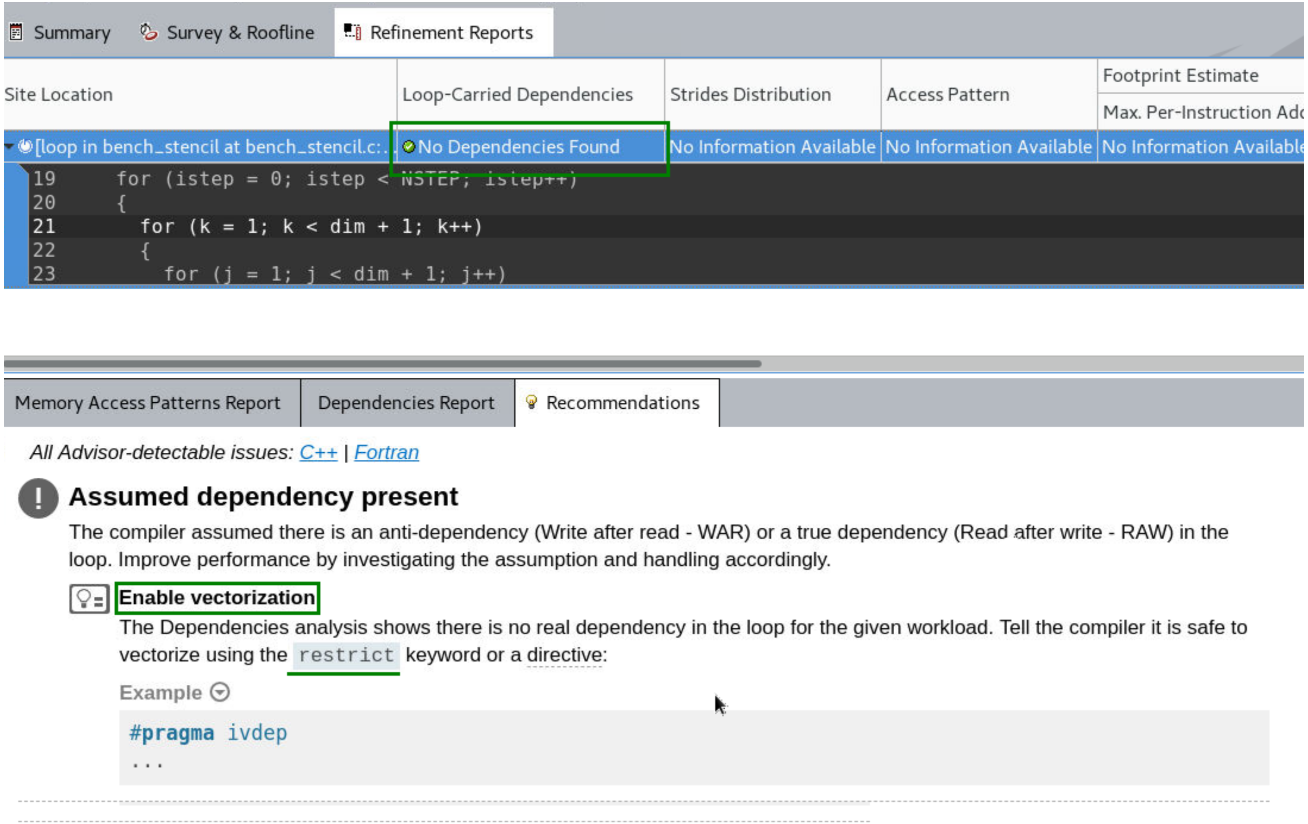
Source | Top Down | Code Analytics | Assembly | Recommendations | Why No Vec

File: bench_stencil.c:21 bench_stencil

Line | Source | Total Time

```
16      int istep;
17
18      StartTime = omp_get_wtime();
19      for (istep = 0; istep < NSTEP; istep++)
20      {
21        for (k = 1; k < dim + 1; k++)
         [loop in bench_stencil at bench_stencil.c:21]
           Scalar loop. Not vectorized: vector dependence prev
           No loop transformations applied
```

ACTIVITY 3. SNAPSHOT



Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern	Footprint Estimate
				Max. Per-Instruction Adc
[loop in bench_stencil at bench_stencil.c:...	✓ No Dependencies Found	No Information Available	No Information Available	No Information Available

```
    for (istep = 0; istep < NSTEP; istep++)
    {
      for (k = 1; k < dim + 1; k++)
      {
        for (j = 1; j < dim + 1; j++)
```

All Advisor-detectable issues: C++ | Fortran

Assumed dependency present

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

Enable vectorization

The Dependencies 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:

Example

```
#pragma ivdep
...
```

ACTIVITY 4: ADDING THREADING AND TRYING TO ENABLE VECTORIZATION

ACTIVITY 4

Purpose: Add threading and try to enable vectorization

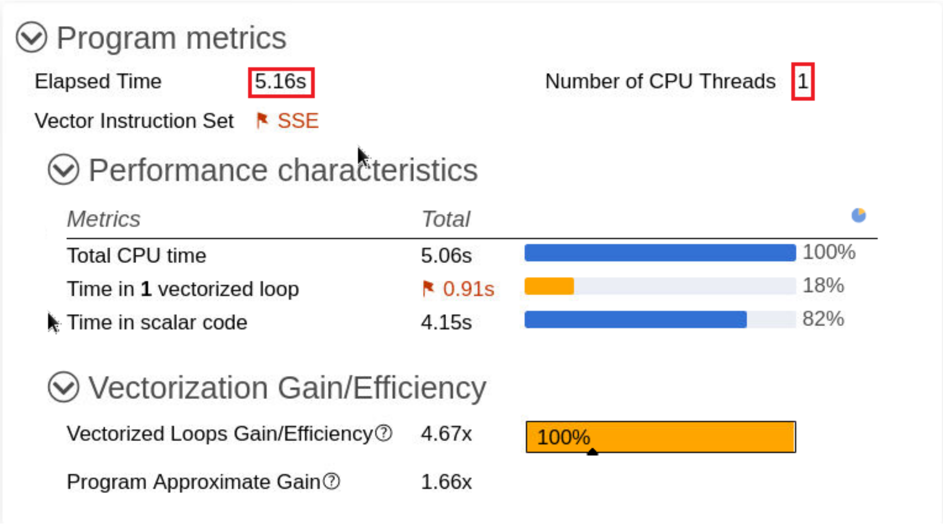
- Build a version with threading and vectorization

```
$ make -C ver2
```

- Re-run Survey analysis
- Create a snapshot
- Compare with previous version

ACTIVITY 4. VERSION COMPARISON

1,536x ↑



Program metrics

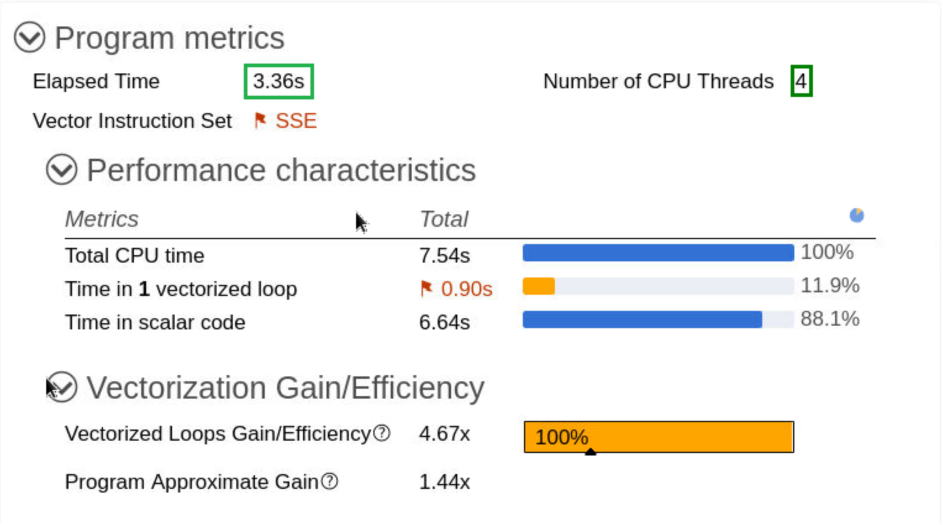
Elapsed Time	5.16s	Number of CPU Threads	1
Vector Instruction Set	⚑ SSE		

Performance characteristics

Metrics	Total		
Total CPU time	5.06s		100%
Time in **1** vectorized loop	⚑ 0.91s		18%
Time in scalar code	4.15s		82%

Vectorization Gain/Efficiency

Vectorized Loops Gain/Efficiency	4.67x	100%
Program Approximate Gain	1.66x	



Program metrics

Elapsed Time	3.36s	Number of CPU Threads	4
Vector Instruction Set	⚑ SSE		

Performance characteristics

Metrics	Total		
Total CPU time	7.54s		100%
Time in **1** vectorized loop	⚑ 0.90s		11.9%
Time in scalar code	6.64s		88.1%

Vectorization Gain/Efficiency

Vectorized Loops Gain/Efficiency	4.67x	100%
Program Approximate Gain	1.44x	

ACTIVITY 5: CHECKING MEMORY ACCESS PATTERNS

TYPES OF MEMORY ACCESS PATTERNS

Unit-Stride access

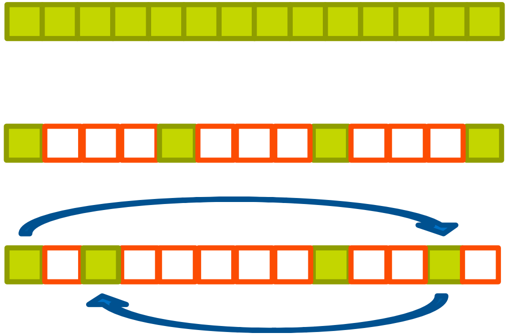
```
for (i=0; i<N; i++)
    A[i] = C[i]*D[i]
```

Constant stride access

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

Variable stride access

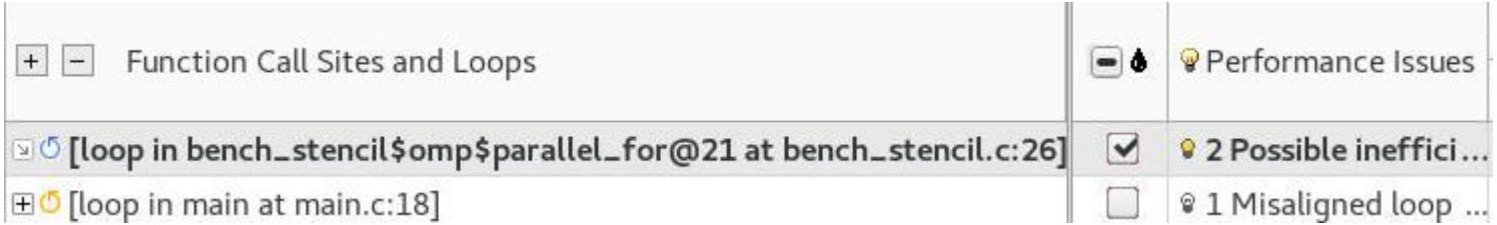
```
for (i=0; i<N; i++)
    A[B[i]] = C[i]*D[i]
```



ACTIVITY 5

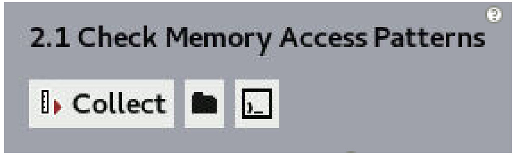
Purpose: Checking memory access patterns

- Select [loop in bench_stencil$omp$parallel_for@23 at bench_stencil.c:26]



Function Call Sites and Loops		Performance Issues
[loop in bench_stencil$omp$parallel_for@21 at bench_stencil.c:26]	☑	2 Possible ineffici...
[loop in main at main.c:18]	□	1 Misaligned loop ...

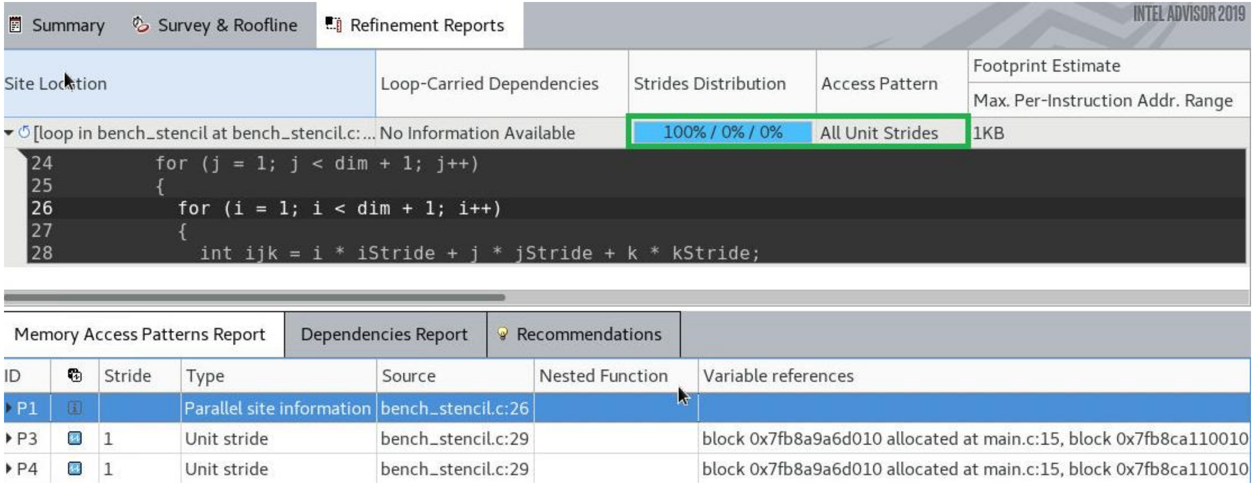
- Press “Collect” button in “2.1 Check Memory Access Patterns” section



2.1 Check Memory Access Patterns

- Wait ~1 minute

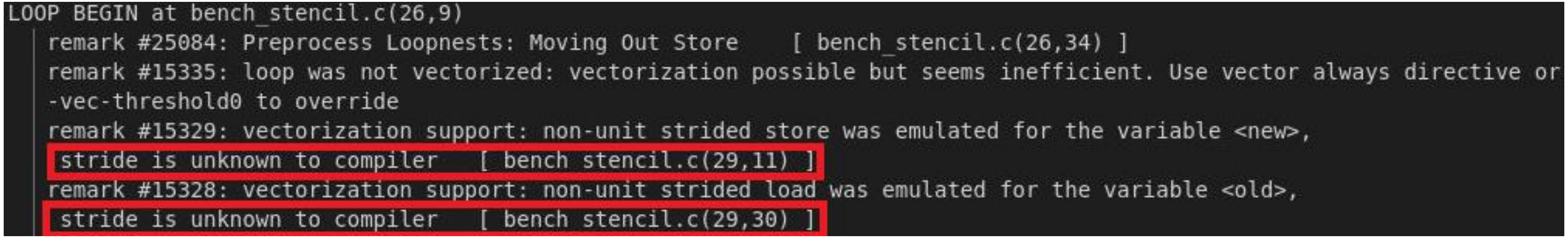
ACTIVITY 5. SCREENSHOTS



Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern	Footprint Estimate Max. Per-Instruction Addr. Range
[loop in bench_stencil at bench_stencil.c:...	No Information Available	100% / 0% / 0%	All Unit Strides	1KB

```
24          for (j = 1; j < dim + 1; j++)
25          {
26            for (i = 1; i < dim + 1; i++)
27            {
28              int ijk = i * iStride + j * jStride + k * kStride;
```

ID		Stride	Type	Source	Nested Function	Variable references
P1			Parallel site information	bench_stencil.c:26		
P3		1	Unit stride	bench_stencil.c:29		block 0x7fb8a9a6d010 allocated at main.c:15, block 0x7fb8ca110010
P4		1	Unit stride	bench_stencil.c:29		block 0x7fb8a9a6d010 allocated at main.c:15, block 0x7fb8ca110010



```
LOOP BEGIN at bench_stencil.c(26,9)
   remark #25084: Preprocess Loopnests: Moving Out Store    [ bench_stencil.c(26,34) ]
   remark #15335: loop was not vectorized: vectorization possible but seems inefficient. Use vector always directive or
   -vec-threshold0 to override
   remark #15329: vectorization support: non-unit strided store was emulated for the variable <new>,
    stride is unknown to compiler   [ bench_stencil.c(29,11) ]
   remark #15328: vectorization support: non-unit strided load was emulated for the variable <old>,
    stride is unknown to compiler   [ bench_stencil.c(29,30) ]
```

ACTIVITY 6: MAKING UNIT STRIDE EXPLICIT

ACTIVITY 6

Purpose: Making unit stride explicit to improve memory access pattern

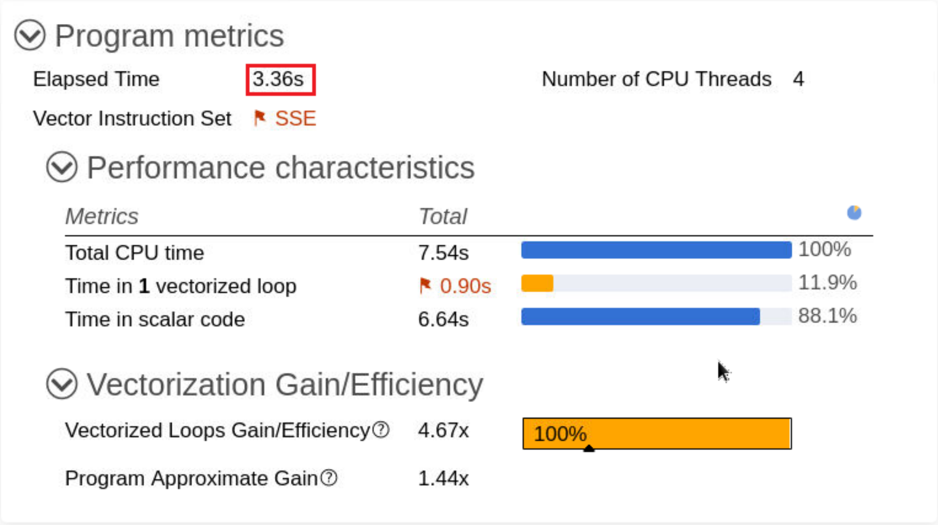
- Build a version with explicit unit stride

```
$ make -C ver3
```

- Re-run Survey analysis
- Create a snapshot
- Compare with previous version

ACTIVITY 6. VERSION COMPARISON

1,592x ↑



Program metrics

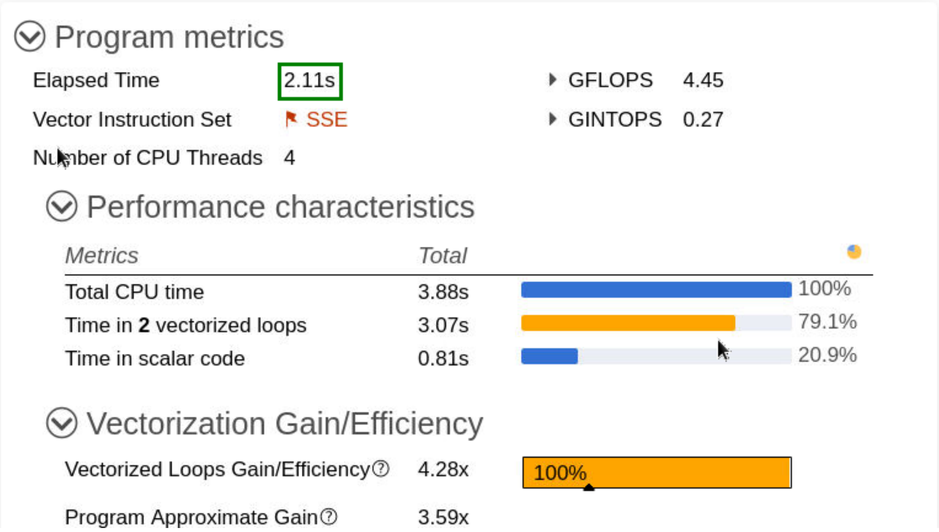
Elapsed Time	3.36s	Number of CPU Threads	4
Vector Instruction Set	⚑ SSE		

Performance characteristics

Metrics	*Total*	
Total CPU time	7.54s	100%
Time in **1** vectorized loop	⚑ 0.90s	11.9%
Time in scalar code	6.64s	88.1%

Vectorization Gain/Efficiency

Vectorized Loops Gain/Efficiency ⓘ	4.67x	100%
Program Approximate Gain ⓘ	1.44x	



Program metrics

Elapsed Time	2.11s	▸ GFLOPS	4.45
Vector Instruction Set	⚑ SSE	▸ GINTOPS	0.27
Number of CPU Threads	4		

Performance characteristics

Metrics	*Total*	
Total CPU time	3.88s	100%
Time in **2** vectorized loops	3.07s	79.1%
Time in scalar code	0.81s	20.9%

Vectorization Gain/Efficiency

Vectorized Loops Gain/Efficiency ⓘ	4.28x	100%
Program Approximate Gain ⓘ	3.59x	

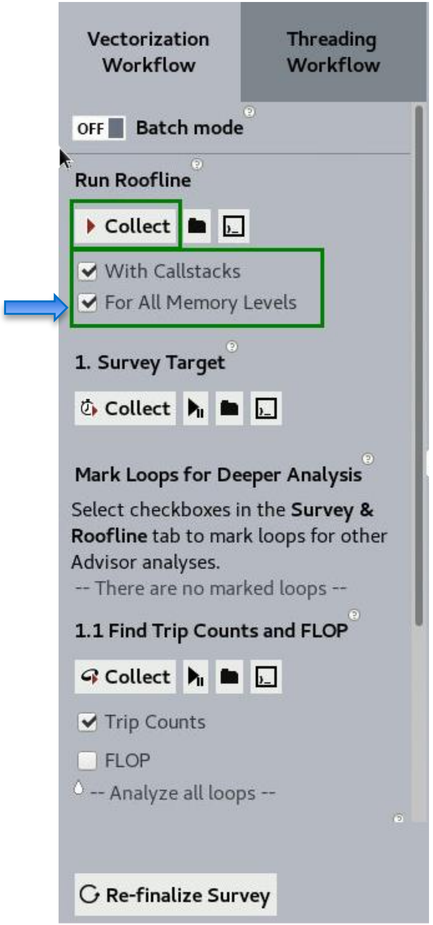
ACTIVITY 7: DOING ROOFLINE ANALYSIS

ACTIVITY 7. COLLECT DATA TO GET ROOFLINE CHART

Purpose: Characterize the application using roofline model

- Select “With Callstacks” and “For all memory levels”
- Press “Collect” button in “Run Roofline” section
- Wait ~4 minutes
- Create a snapshot

For Integrated Roofline (NEW!)



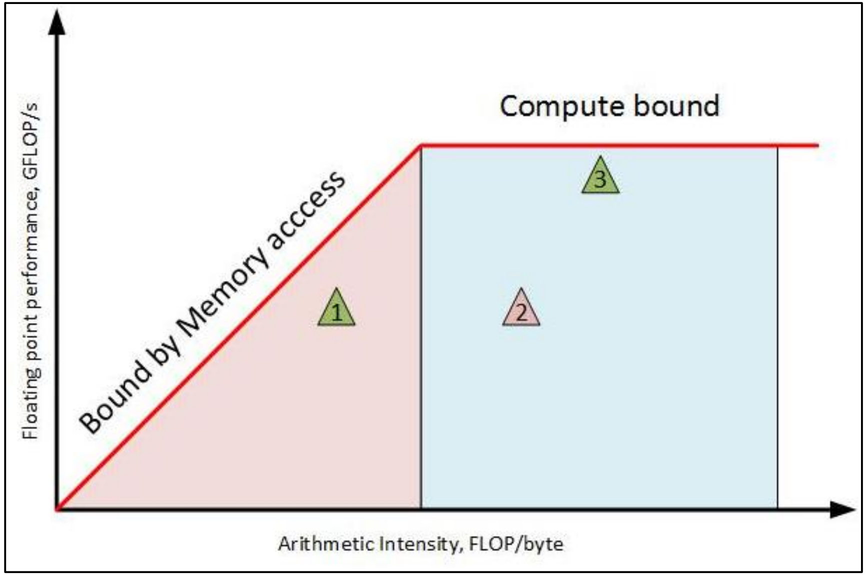
Select checkboxes in the **Survey & Roofline** tab to mark loops for other Advisor analyses.

-- There are no marked loops --

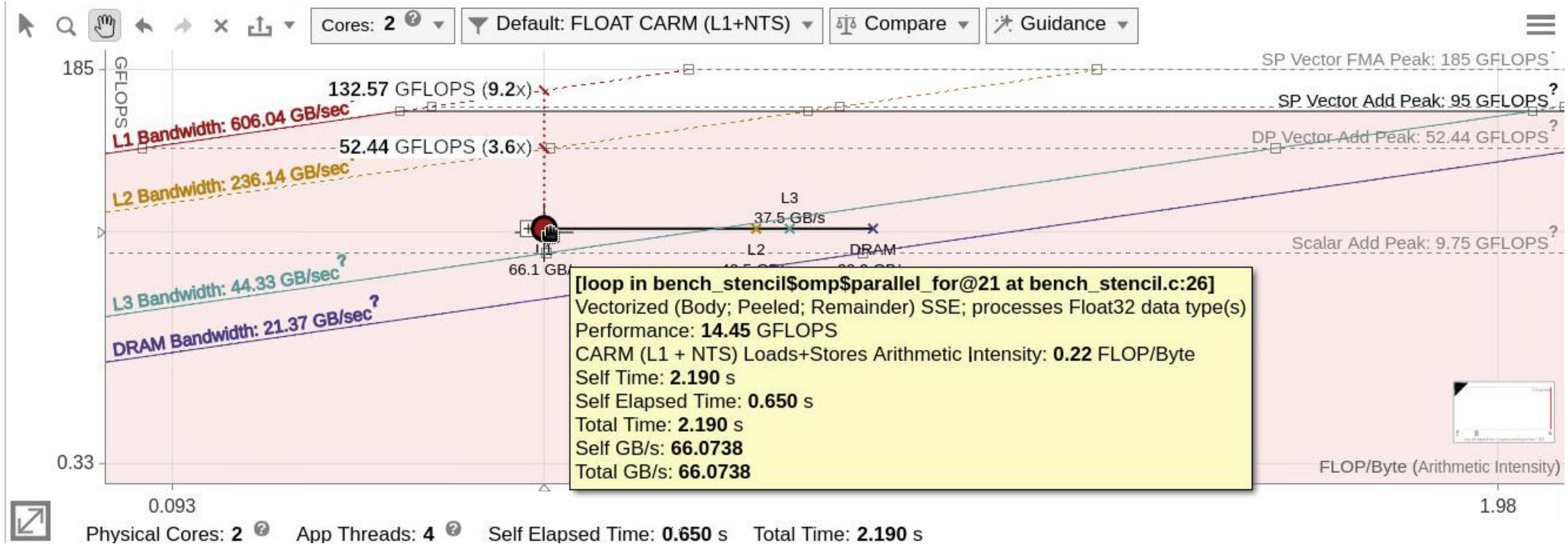
ROOFLINE MODEL

A roofline model helping you answer these questions:

- Does my application work optimally on the current hardware? If not, what is the most underutilized hardware resource?
- What limits performance? Is my application workload memory or compute bound?
- What is the right strategy to improve application performance?


Compute bound
Bound by Memory acccess
1
2
3
Floating point performance, GFLOP/s
Arithmetic Intensity, FLOP/byte

ACTIVITY 7. SCREENSHOT


Compare
Guidance
185
0.33
GFLOPS
0.093
1.98
FLOP/Byte (Arithmetic Intensity)
SP Vector FMA Peak: 185 GFLOPS
SP Vector Add Peak: 95 GFLOPS
DP Vector Add Peak: 52.44 GFLOPS
Scalar Add Peak: 9.75 GFLOPS
132.57 GFLOPS (9.2x)
52.44 GFLOPS (3.6x)
L1 Bandwidth: 606.04 GB/sec
L2 Bandwidth: 236.14 GB/sec
L3 Bandwidth: 44.33 GB/sec
DRAM Bandwidth: 21.37 GB/sec
L3
37.5 GB/s
L2
DRAM
L1
66.1 GB/

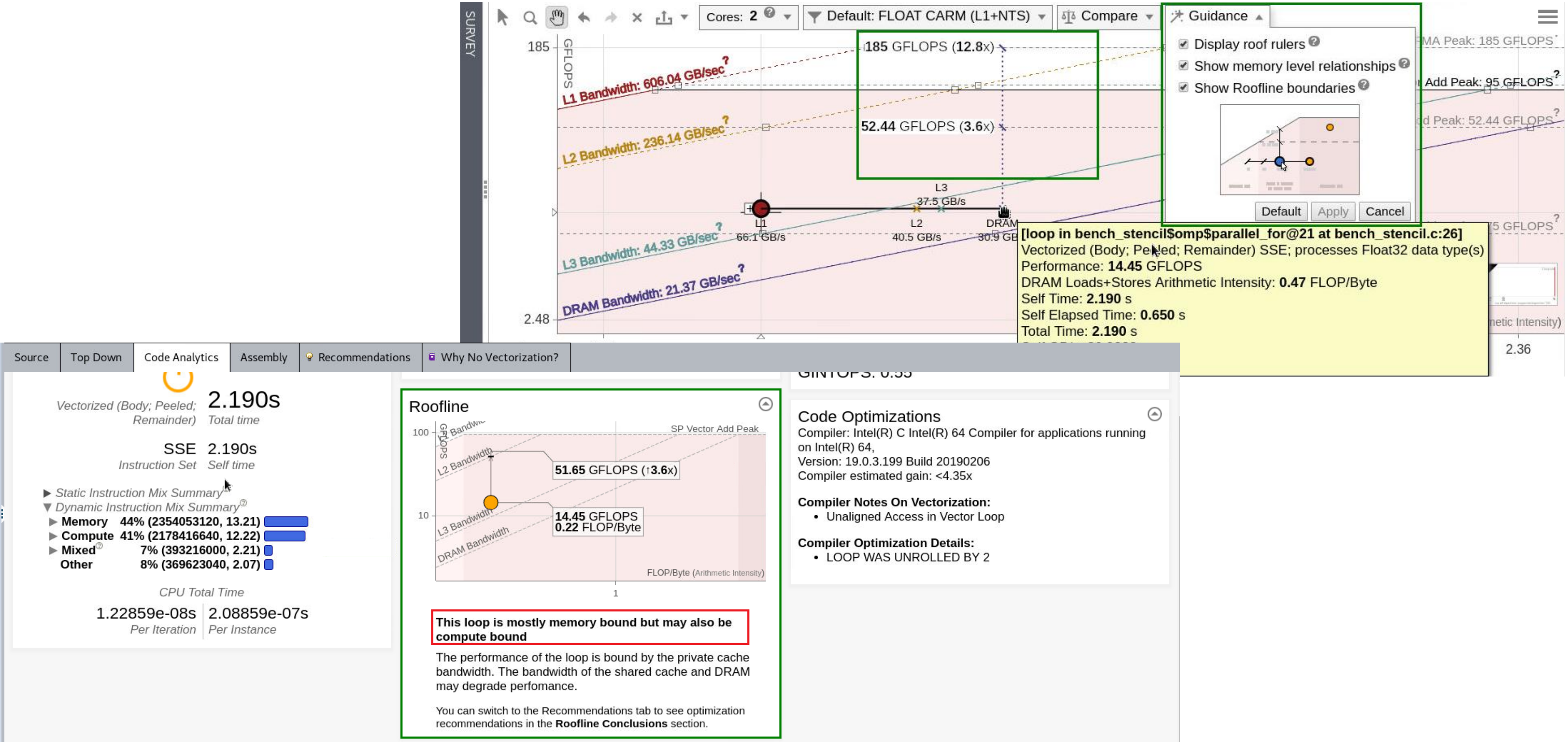
Cores: 2

Default: FLOAT CARM (L1+NTS)

[loop in bench_stencil$omp$parallel_for@21 at bench_stencil.c:26]
Vectorized (Body; Peeled; Remainder) SSE; processes Float32 data type(s)
Performance: **14.45** GFLOPS
CARM (L1 + NTS) Loads+Stores Arithmetic Intensity: **0.22** FLOP/Byte
Self Time: **2.190** s
Self Elapsed Time: **0.650** s
Total Time: **2.190** s
Self GB/s: **66.0738**
Total GB/s: **66.0738**

Physical Cores: **2** App Threads: **4** Self Elapsed Time: **0.650** s Total Time: **2.190** s

ACTIVITY 7. ROOFLINE GUIDANCE



☑ Display roof rulers

☑ Show memory level relationships

☑ Show Roofline boundaries

[loop in bench_stencil$omp$parallel_for@21 at bench_stencil.c:26]
Vectorized (Body; Peeled; Remainder) SSE; processes Float32 data type(s)
Performance: **14.45** GFLOPS
DRAM Loads+Stores Arithmetic Intensity: **0.47** FLOP/Byte
Self Time: **2.190** s
Self Elapsed Time: **0.650** s
Total Time: **2.190** s

Vectorized (Body; Peeled; Remainder) — 2.190s Total time

SSE Instruction Set — 2.190s Self time

▶ Static Instruction Mix Summary

▼ Dynamic Instruction Mix Summary

▶ **Memory**	**44% (2354053120, 13.21)**
▶ **Compute**	**41% (2178416640, 12.22)**
▶ **Mixed**	**7% (393216000, 2.21)**
Other	**8% (369623040, 2.07)**

CPU Total Time

1.22859e-08s Per Iteration | 2.08859e-07s Per Instance

GINTOPS: 0.55

This loop is mostly memory bound but may also be compute bound

The performance of the loop is bound by the private cache bandwidth. The bandwidth of the shared cache and DRAM may degrade perfomance.

You can switch to the Recommendations tab to see optimization recommendations in the **Roofline Conclusions** section.

Code Optimizations

Compiler: Intel(R) C Intel(R) 64 Compiler for applications running on Intel(R) 64,
Version: 19.0.3.199 Build 20190206
Compiler estimated gain: <4.35x

Compiler Notes On Vectorization:
- Unaligned Access in Vector Loop

Compiler Optimization Details:
- LOOP WAS UNROLLED BY 2

ACTIVITY 8: SPLITTING TASK TO TILES

ACTIVITY 8

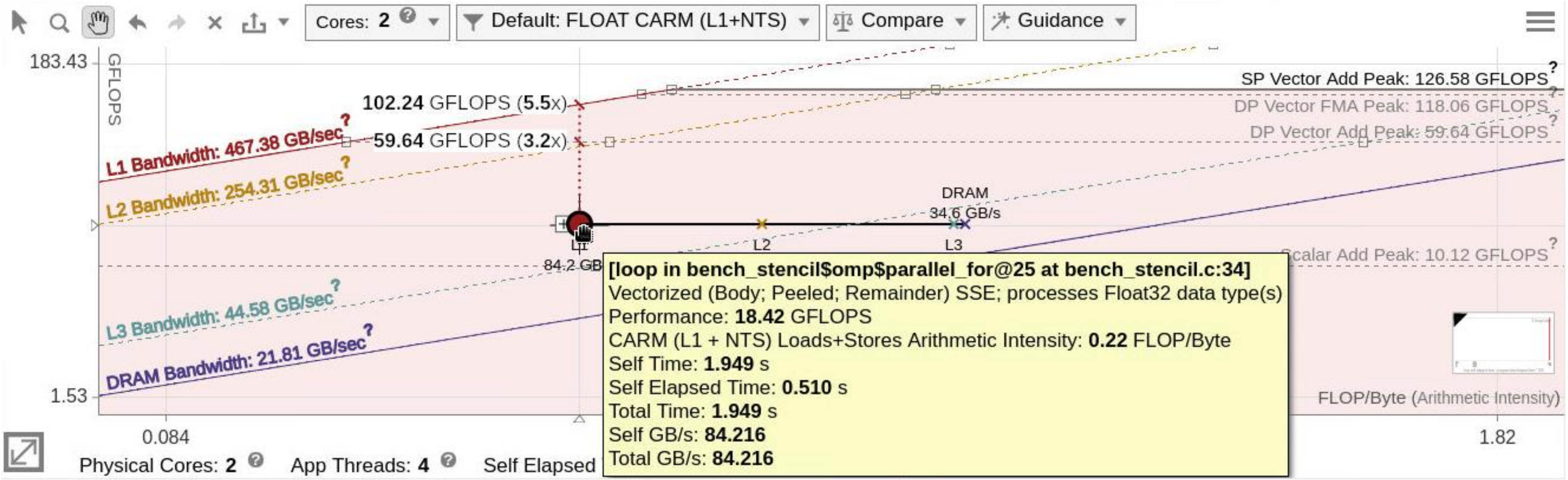
Purpose: Splitting task to tiles to reduce cache working set

- Build a version with splitting task to tiles

```
$ make -C ver4
```

- Re-run Roofline analysis
- Create a snapshot
- Compare with previous version

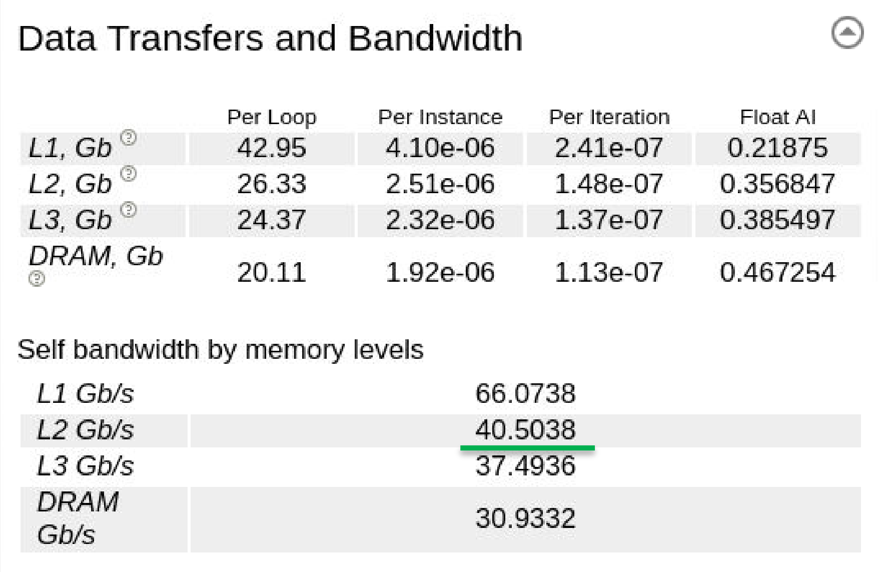
ACTIVITY 8. SCREENSHOT


Cores: 2
Default: FLOAT CARM (L1+NTS)
Compare
Guidance
183.43
GFLOPS
1.53
0.084
1.82
102.24 GFLOPS (5.5x)
59.64 GFLOPS (3.2x)
L1 Bandwidth: 467.38 GB/sec
L2 Bandwidth: 254.31 GB/sec
L3 Bandwidth: 44.58 GB/sec
DRAM Bandwidth: 21.81 GB/sec
SP Vector Add Peak: 126.58 GFLOPS
DP Vector FMA Peak: 118.06 GFLOPS
DP Vector Add Peak: 59.64 GFLOPS
Scalar Add Peak: 10.12 GFLOPS
DRAM
34.6 GB/s
L1
L2
L3
84.2 GB
FLOP/Byte (Arithmetic Intensity)
Physical Cores: 2
App Threads: 4
Self Elapsed

[loop in bench_stencil$omp$parallel_for@25 at bench_stencil.c:34]
Vectorized (Body; Peeled; Remainder) SSE; processes Float32 data type(s)
Performance: **18.42** GFLOPS
CARM (L1 + NTS) Loads+Stores Arithmetic Intensity: **0.22** FLOP/Byte
Self Time: **1.949** s
Self Elapsed Time: **0.510** s
Total Time: **1.949** s
Self GB/s: **84.216**
Total GB/s: **84.216**

ACTIVITY 8. VERSION COMPARISON

L2 bandwidth: 1,364x ↑

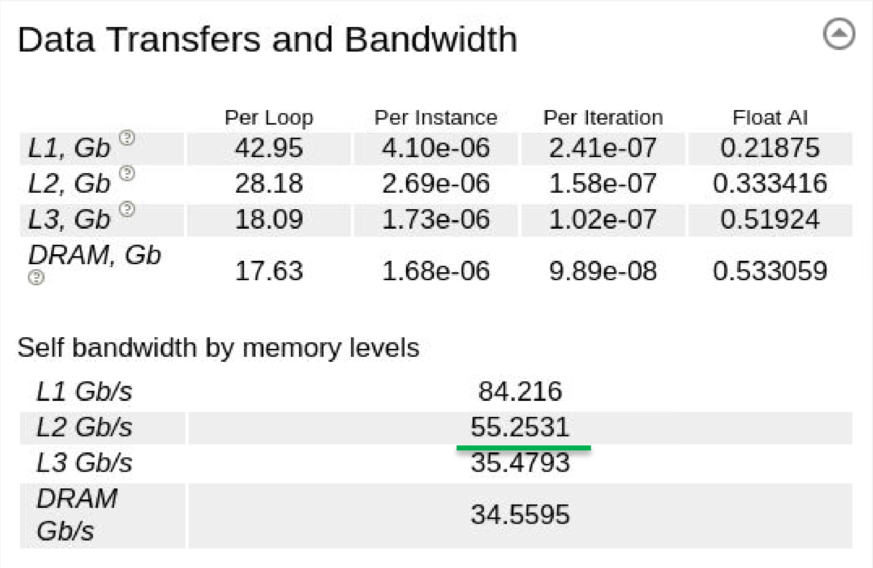


Data Transfers and Bandwidth

	Per Loop	Per Instance	Per Iteration	Float AI
L1, Gb	42.95	4.10e-06	2.41e-07	0.21875
L2, Gb	26.33	2.51e-06	1.48e-07	0.356847
L3, Gb	24.37	2.32e-06	1.37e-07	0.385497
DRAM, Gb	20.11	1.92e-06	1.13e-07	0.467254

Self bandwidth by memory levels

L1 Gb/s	66.0738
L2 Gb/s	40.5038
L3 Gb/s	37.4936
DRAM Gb/s	30.9332



Data Transfers and Bandwidth

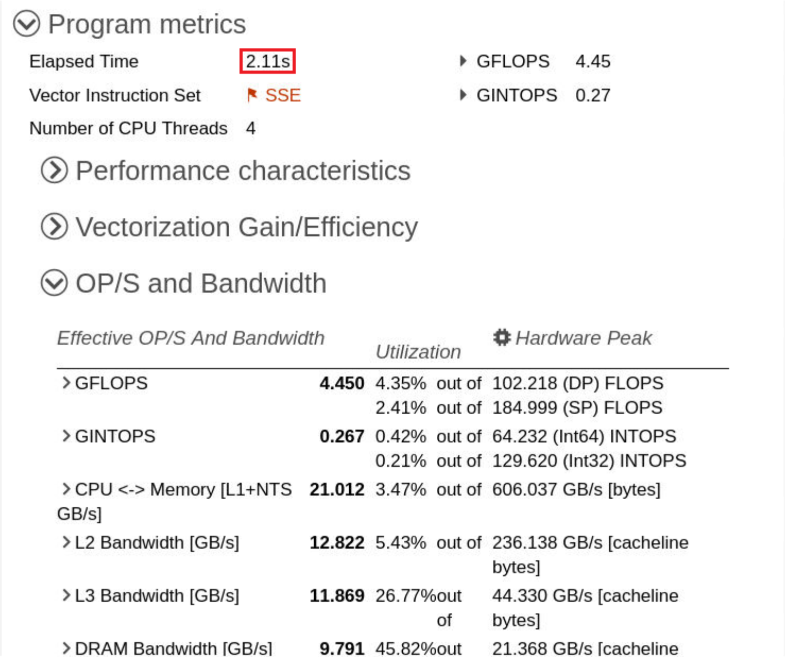
	Per Loop	Per Instance	Per Iteration	Float AI
L1, Gb	42.95	4.10e-06	2.41e-07	0.21875
L2, Gb	28.18	2.69e-06	1.58e-07	0.333416
L3, Gb	18.09	1.73e-06	1.02e-07	0.51924
DRAM, Gb	17.63	1.68e-06	9.89e-08	0.533059

Self bandwidth by memory levels

L1 Gb/s	84.216
L2 Gb/s	55.2531
L3 Gb/s	35.4793
DRAM Gb/s	34.5595

ACTIVITY 8. VERSION COMPARISON

1,185x ↑



Program metrics

Elapsed Time 2.11s

Vector Instruction Set SSE

Number of CPU Threads 4

GFLOPS 4.45

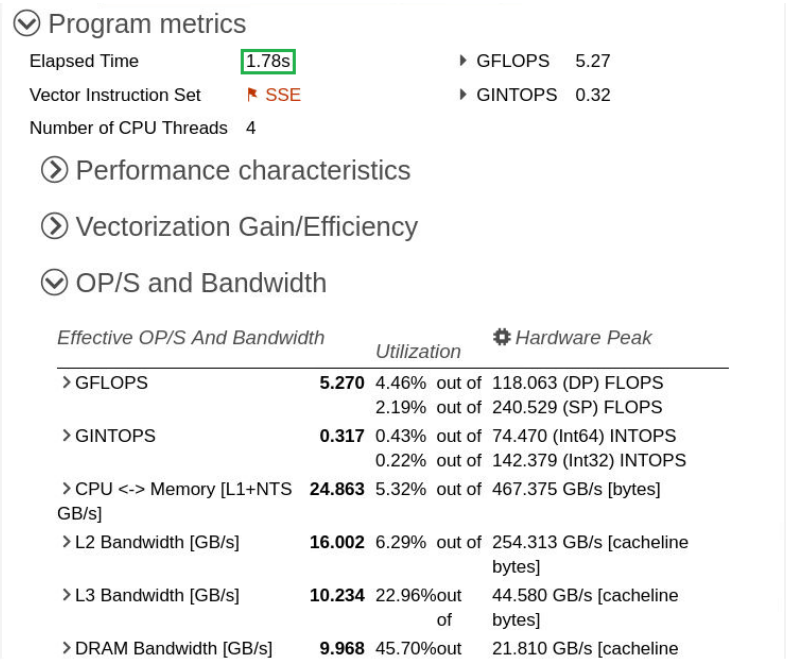
GINTOPS 0.27

Performance characteristics

Vectorization Gain/Efficiency

OP/S and Bandwidth

Effective OP/S And Bandwidth		*Utilization*		*Hardware Peak*
GFLOPS	**4.450**	4.35% 2.41%	out of out of	102.218 (DP) FLOPS 184.999 (SP) FLOPS
GINTOPS	**0.267**	0.42% 0.21%	out of out of	64.232 (Int64) INTOPS 129.620 (Int32) INTOPS
CPU <-> Memory [L1+NTS GB/s]	**21.012**	3.47%	out of	606.037 GB/s [bytes]
L2 Bandwidth [GB/s]	**12.822**	5.43%	out of	236.138 GB/s [cacheline bytes]
L3 Bandwidth [GB/s]	**11.869**	26.77%	out of	44.330 GB/s [cacheline bytes]
DRAM Bandwidth [GB/s]	**9.791**	45.82%	out	21.368 GB/s [cacheline



Program metrics

Elapsed Time 1.78s

Vector Instruction Set SSE

Number of CPU Threads 4

GFLOPS 5.27

GINTOPS 0.32

Performance characteristics

Vectorization Gain/Efficiency

OP/S and Bandwidth

Effective OP/S And Bandwidth		*Utilization*		*Hardware Peak*
GFLOPS	**5.270**	4.46% 2.19%	out of out of	118.063 (DP) FLOPS 240.529 (SP) FLOPS
GINTOPS	**0.317**	0.43% 0.22%	out of out of	74.470 (Int64) INTOPS 142.379 (Int32) INTOPS
CPU <-> Memory [L1+NTS GB/s]	**24.863**	5.32%	out of	467.375 GB/s [bytes]
L2 Bandwidth [GB/s]	**16.002**	6.29%	out of	254.313 GB/s [cacheline bytes]
L3 Bandwidth [GB/s]	**10.234**	22.96%	out of	44.580 GB/s [cacheline bytes]
DRAM Bandwidth [GB/s]	**9.968**	45.70%	out	21.810 GB/s [cacheline

ACTIVITY 9: ENABLING AVX512

ACTIVITY 9

Purpose: Set compilation options to use the highest available ISA

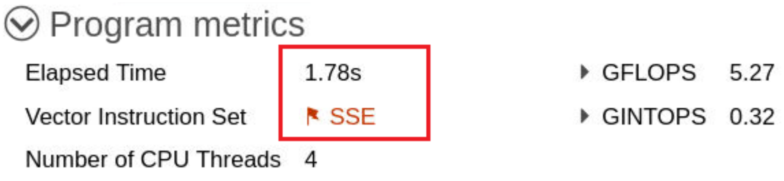
- Build a version with new compilation flags

```
$ make -C ver5
```

- Re-run Survey analysis
- Create a snapshot
- Compare with previous version

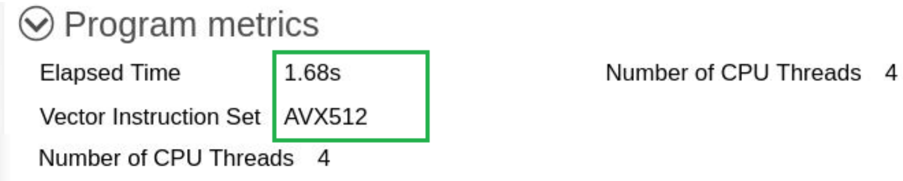
ACTIVITY 9. VERSION COMPARISON

1,059x ↑



Program metrics

Elapsed Time	1.78s	GFLOPS	5.27
Vector Instruction Set	⚑ SSE	GINTOPS	0.32
Number of CPU Threads	4		



Program metrics

Elapsed Time	1.68s	Number of CPU Threads	4
Vector Instruction Set	AVX512		
Number of CPU Threads	4		

ACTIVITY 10: COMPARING ROOFLINE CHARTS

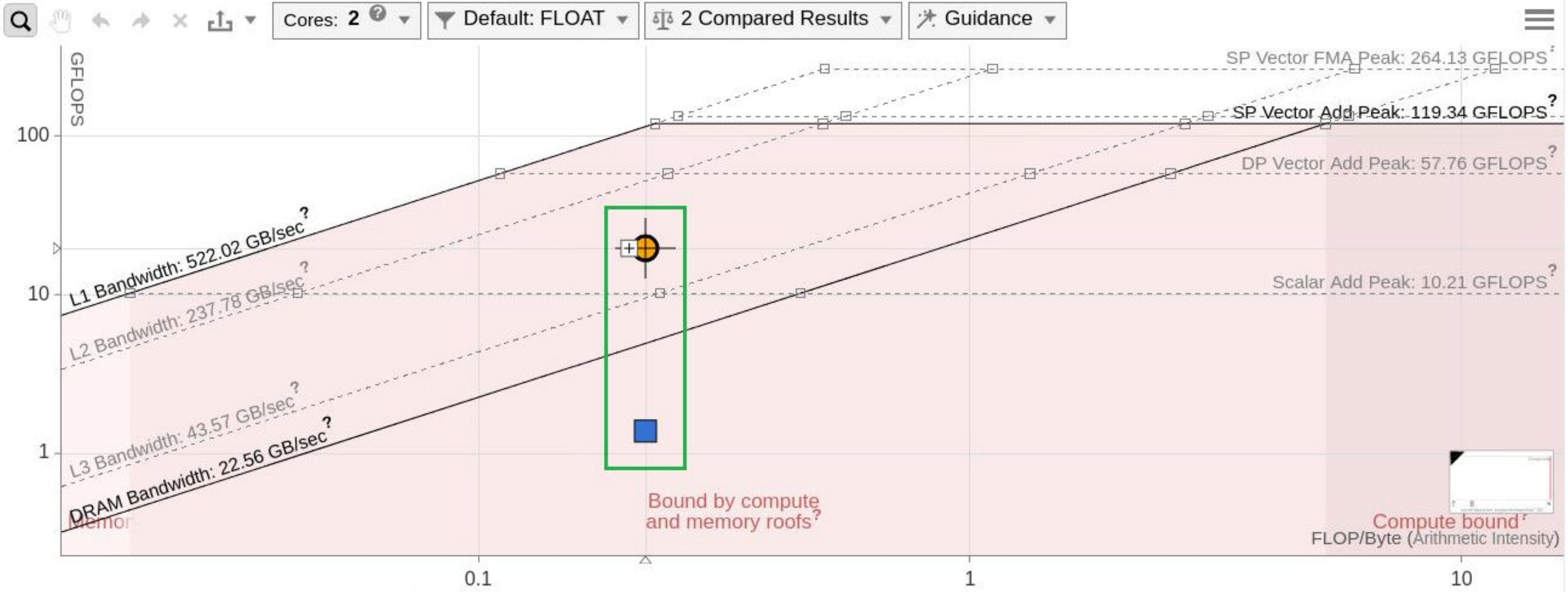
ACTIVITY 10

Purpose: See the performance difference for non-optimized and optimized versions.

- Run Roofline analysis w/o additional options for ver0 and ver5
- Compare profiles

ACTIVITY 9. ROOFLINE COMPARISON

Hotspot elapsed time speedup: ~14x ↑
Program elapsed time speedup: ~5x ↑



Cores: 2
Default: FLOAT
2 Compared Results
Guidance
GFLOPS
SP Vector FMA Peak: 264.13 GFLOPS
SP Vector Add Peak: 119.34 GFLOPS
DP Vector Add Peak: 57.76 GFLOPS
Scalar Add Peak: 10.21 GFLOPS
L1 Bandwidth: 522.02 GB/sec
L2 Bandwidth: 237.78 GB/sec
L3 Bandwidth: 43.57 GB/sec
DRAM Bandwidth: 22.56 GB/sec
Memor
Bound by compute and memory roofs
Compute bound
FLOP/Byte (Arithmetic Intensity)
100
10
1
0.1
1
10

For more complete information about compiler optimizations, see our Optimization Notice.

intel®

Software

Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

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