

# USING INTEL<sup>®</sup> ADVISOR WITH THE ROOFLINE MODEL TO BOOST YOUR APPLICATIONS

Profiling and improving your vectorization

# THE MEMORY ACCESS PATTERN ANALYSIS

# MEMORY ACCESS PATTERN ANALYSIS

How should I access data ?

- Unit stride access are faster

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

- Constant stride are more complex

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

- Non predictable access are usually bad

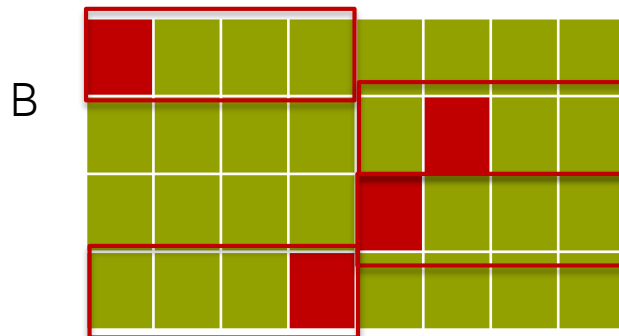
```
for (i=0; i<N; i++)  
  A[i] = B[C[i]]*d
```



For B, 1 cache line load computes 4 DP



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

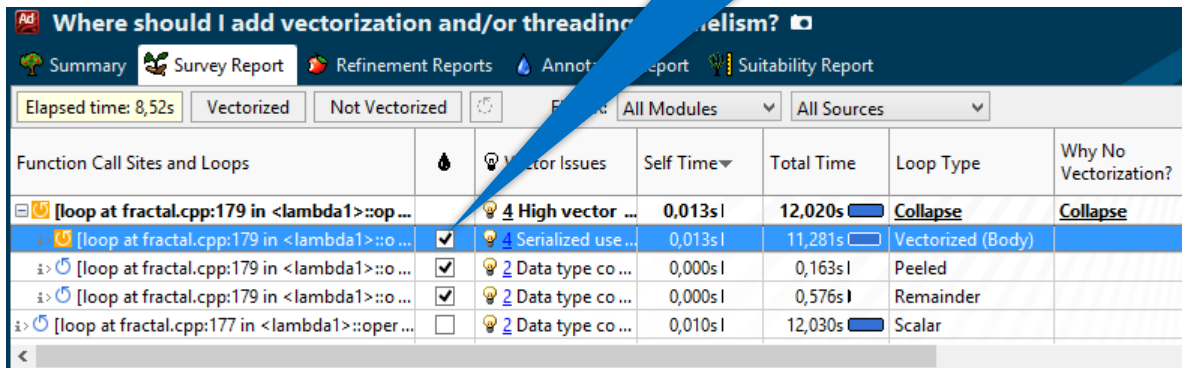


For B, 4 cache line loads compute 4 DP with reconstructions, prefetching might not work

# HOW TO RUN IT ?

Automatic or manual modes

Select loops of interest



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	

## 2.2 Check Memory Access Patterns

Identify and explore complex memory accesses for marked loops. Fix the reported problems.



Command Line

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

# QUICKLY FIND LOOP WITH NON-OPTIMAL STRIDE

Identify the problem first

- Quickly identify loops that are good, bad or mixed
- Find unaligned data
- `pragma omp declare simd uniform(...)`

Check memory access patterns in your application | Intel Advisor XE 2016

Summary | Survey Report | Refinement Reports | MAP Source: fractal.cpp | Annotation Report | Suitability Report

Site Name	Site Function	Site Info	Loop-Carried Dependencies	Strides Distribution	Access Pattern
loop_site_54	operator()	fractal.cpp:164	No dependencies found	No information available	No information available
loop_site_129	operator()	fractal.cpp:164	No information available	100% / 0% / 0%	All unit strides

Memory Access Patterns Report | Correctness Report

ID	Stride	Type	Source	Modules	Alignment
P1		Parallel site information	fractal.cpp:164	fractal.exe	
P3	0	Unit stride	fractal.cpp:100	fractal.exe	
P4	0	Unit stride	fractal.cpp:164	fractal.exe	
P5	0	Unit stride	fractal.cpp:164	fractal.exe	
P6	0, 1	Unit stride	fractal.cpp:165	fractal.exe	
P7	0, 1	Unit stride	fractal.cpp:165	fractal.exe	
P8	0	Unit stride	fractal.cpp:60	fractal.exe	

```
98     }
99     #endif
100    int b = (int) (256 * mu);
101    int g = (b / 8);
102    int r = (g / 16);
162
163    for (int x = x0; x < x1; ++x) {
164        for (int y = y0; y < y1; ++y) {
165            fractal_data_array[x - x0][y - y0] = calc_one_pixel(x, y, tmp_max_iterations, tmp_size_x, tmp_si
166        }
```

# IMPROVING DATA ACCESS

What can you do to improve vectorization ?

## Constant stride

- Verify loop order
- Move from AoS to SoA
  - Rewrite your code completely
  - Use **Intel® SIMD Data Layout Templates** (Intel® SDLT)

## Unpredictable memory accesses

- Very difficult case
- Algorithm changes might be required
- Reorganize the datalayout



Software