Contacts

Advisor Support Mail List vector.advisor@intel.com

Zakhar Matveev zakhar.a.matveev@intel.com
   Intel Advisor Product Architect

Kirill Rogozhin kirill.rogozhin@intel.com
   Intel Advisor Project Manager

Egor Kazachkov egor.kazachkov@intel.com
   Intel Advisor Senior Developer
What is Intel® Advisor

Vectorization analysis

<table>
<thead>
<tr>
<th>Function Call Sites and Loops</th>
<th>Why No Vectorization?</th>
<th>Self Time</th>
<th>Total Time</th>
<th>Type</th>
<th>Vectorized Loops</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mainCRTStartup</td>
<td></td>
<td>0.000s</td>
<td>1.66%</td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td>main</td>
<td></td>
<td>0.000s</td>
<td>1.66%</td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td>[loop in main at 3loops.d]</td>
<td></td>
<td>0.000s</td>
<td>1.373s</td>
<td>Vectorized AVX2</td>
<td>260% 6.43x</td>
</tr>
<tr>
<td>[loop in f at 3loops.cpp]</td>
<td></td>
<td>1.193s</td>
<td>1.373s</td>
<td>Vectorized AVX2</td>
<td>260% 6.43x</td>
</tr>
<tr>
<td>[loop in f at 3loops.cpp]</td>
<td></td>
<td>0.296s</td>
<td>0.296s</td>
<td>Vectorized AVX2</td>
<td>260% 6.43x</td>
</tr>
<tr>
<td>[loop in f at 3loops.cpp]</td>
<td></td>
<td>0.296s</td>
<td>0.296s</td>
<td>Inlined...</td>
<td></td>
</tr>
<tr>
<td>[loop in main at 3loops.d]</td>
<td></td>
<td>0.234s</td>
<td>0.234s</td>
<td>Inside...</td>
<td></td>
</tr>
</tbody>
</table>

Cache Simulator and MAP

<table>
<thead>
<tr>
<th>Site Location</th>
<th>Strides Distribution</th>
<th>Access Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>[loop in ComputeTimeStep]</td>
<td>80% / 0% / 20%</td>
<td>Mixed strides</td>
</tr>
<tr>
<td>[loop in pricePath_Core]</td>
<td>92% / 0% / 8%</td>
<td>Mixed strides</td>
</tr>
</tbody>
</table>

Memory Access Patterns Report

- P33: 1 Unit stride ch_4_v253.cpp:76
- P31: Gather stride ch_4_v253.cpp:76
- P32: 1 Unit stride ch_4_v253.cpp:218

Python API

```python
import advisor

project = advisor.open_project(sys.argv[1])
data = project.load(advisor.SURVEY)

roofs = data.get_roofs(4, advisor.RoofsStrategy.MULTI_THREAD)

for roof in roofs:
    # memory roofs
    if 'bandwidth' in roof.name.lower():
        bandwidth = roof.bandwidth / math.pow(10, 9) # convert print '{:.0f} GB/s'.format(roof.name, bandwidth)
    # compute roofs
    else:
        bandwidth = roof.bandwidth / math.pow(10, 9) # convert print '{:.0f} GFLOPS'.format(roof.name, bandwidth)
```

Threading prototyping

<table>
<thead>
<tr>
<th>Tasks Modeling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. Number of Tasks: 50000000</td>
</tr>
<tr>
<td>Avg. Task Duration:</td>
</tr>
<tr>
<td>Tasks: 25x</td>
</tr>
<tr>
<td>25x</td>
</tr>
<tr>
<td>25x</td>
</tr>
<tr>
<td>25x</td>
</tr>
</tbody>
</table>

Optimization Notice
VECTORIZATION
Get Faster Code Faster! 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 omp simd?
The Right Data At Your Fingertips
Get all the data you need for high impact vectorization

- Filter by which loops are vectorized!
- 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?

Get Faster Code Faster!
5 Steps to Efficient Vectorization

1. Compiler diagnostics + Performance Data + SIMD efficiency information

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

3. Trip Counts + FLOP: understand utilization, parallelism granularity & overheads

4. Memory Access Patterns Analysis

5. Loop-Carried Dependency Analysis
1. Compiler diagnostics + Performance Data + SIMD efficiency information + Binary Analysis
Vector Efficiency: All The Data In One Place

My “performance thermometer”

<table>
<thead>
<tr>
<th>Function Call Sites and Loops</th>
<th>Self Time</th>
<th>Vectorized Loops</th>
<th>Instruction Set Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Vect...</td>
<td>Efficiency</td>
</tr>
<tr>
<td>Loop in runCforallLambdaLoops at runCforallLambdaLoops</td>
<td>0.734s</td>
<td>AVX; ...</td>
<td>26%</td>
</tr>
<tr>
<td>Loop in runCRawLoops at runCRawLoops.cxx:704</td>
<td>0.625s</td>
<td>AVX; ...</td>
<td>26%</td>
</tr>
<tr>
<td>Loop in runCforallLambdaLoops at runCforallLambdaLoops</td>
<td>2.703s</td>
<td>AVX2</td>
<td>31%</td>
</tr>
<tr>
<td>Loop in runCRawLoops at runCRawLoops.cxx:117</td>
<td>2.606s</td>
<td>AVX2</td>
<td>31%</td>
</tr>
<tr>
<td>Loop in runOMPRawLoopsSompSompParallel@135 at</td>
<td>0.453s</td>
<td>AVX2</td>
<td>45%</td>
</tr>
<tr>
<td>Loop in runCforallLambdaLoops at runCforallLambdaLoops</td>
<td>0.234s</td>
<td>AVX2</td>
<td>45%</td>
</tr>
</tbody>
</table>

• **Auto-vectorization**: affected <3% of code
  • With moderate speed-ups
• First attempt to **simply put #pragma omp 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 “under vectorized” and why
Vectorization tied to your code
Don’t Just Vectorize, Vectorize Efficiently
See detailed times for each part of your loops. Is it worth more effort?

<table>
<thead>
<tr>
<th>Function Call Sites and Loops</th>
<th>Self Time</th>
<th>Type</th>
<th>Vectorized Loops</th>
</tr>
</thead>
<tbody>
<tr>
<td>[loop in runCRawLoops at runCRawLoops.cxx:117]</td>
<td>2.609s</td>
<td>Vectorized (Body; Peeled; Remainder)</td>
<td>AVX2 31% 2.50x 4:8</td>
</tr>
<tr>
<td>[loop in runCRawLoops at runCRawLoops.cxx:2]</td>
<td>2.031s</td>
<td>Vectorized (Body)</td>
<td>AVX2</td>
</tr>
<tr>
<td>[loop in runCRawLoops at runCRawLoops.cxx:2]</td>
<td>0.516s</td>
<td>Vectorized (Remainder)</td>
<td>AVX2</td>
</tr>
<tr>
<td>[loop in runCRawLoops at runCRawLoops.cxx:2]</td>
<td>0.062s</td>
<td>Remainder</td>
<td></td>
</tr>
<tr>
<td>[loop in runCRawLoops at runCRawLoops.cxx:2]</td>
<td>0.000s</td>
<td>Peeled</td>
<td></td>
</tr>
<tr>
<td>[loop in runBRawLoops at runBRawLoops.cxx:55]</td>
<td>2.562s</td>
<td>Vectorized (Body; Remainder)</td>
<td>AVX 80% 3.19x 4</td>
</tr>
<tr>
<td>[loop in runBRawLoops at runBRawLoops.cxx:5]</td>
<td>2.500s</td>
<td>Vectorized (Body)</td>
<td>AVX</td>
</tr>
<tr>
<td>[loop in runBRawLoops at runBRawLoops.cxx:5]</td>
<td>0.062s</td>
<td>Remainder</td>
<td></td>
</tr>
<tr>
<td>[loop in runBRawLoops at runBRawLoops.cxx:5]</td>
<td>0.000s</td>
<td>Vectorized (Remainder)</td>
<td></td>
</tr>
</tbody>
</table>
### 1. Compiler diagnostics + Performance
Data + SIMD efficiency information

<table>
<thead>
<tr>
<th>Function Call Sites and Loops</th>
<th>Self Time</th>
<th>Vectorized Loops</th>
<th>Venderbilt Efficiency</th>
<th>Gem</th>
<th>VL</th>
<th>Traits</th>
<th>Instruction Set Analysis</th>
<th>Data T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop in loginit at gCALSuite.exe</td>
<td>0.01961 AX</td>
<td>100%</td>
<td>1.89%</td>
<td>4</td>
<td>Division: Type C: Float64</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Loop in loginit at LCALSuite.exe</td>
<td>0.01961 AX</td>
<td>100%</td>
<td>1.89%</td>
<td>4</td>
<td>Division: Type C: Float64</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Loop in runCForAllLoops a</td>
<td>0.07217 AX</td>
<td>100%</td>
<td>5.40%</td>
<td>4</td>
<td>Division: Type C: Float64</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Loop in runCForAllLoops b</td>
<td>0.05781 AX</td>
<td>100%</td>
<td>3.48%</td>
<td>4</td>
<td>Division: Type C: Float64</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Loop in runOMPForAllLoops AX</td>
<td>0.0514 AX</td>
<td>100%</td>
<td>2.73%</td>
<td>4</td>
<td>FMA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Loop in runOMPForAllLoops AX</td>
<td>0.0514 AX</td>
<td>100%</td>
<td>2.73%</td>
<td>4</td>
<td>FMA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Loop in runOMPForAllLoops AX</td>
<td>0.0514 AX</td>
<td>100%</td>
<td>2.73%</td>
<td>4</td>
<td>FMA</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

**Recommendation:** Add data padding

The loop count is not a multiple of `source length`. To fix, do one of the following:

- Increase the size of objects and add iterations so the loop counts a multiple of `source length`.
- Increase the size of static and automatic objects, and use a compiler option to add data padding.

**Source Issue:** Ineffective padding/remainder

All or some `source loop` iterations are not including the `source loop` improve performance by removing some loop iterations from `source loop`. Keeps to the loop body.

**Windows OS**
- `-O2 -arch-amd64 -m64 -msvcrt -m32`

**Linux OS**
- `-O2 -arch-amd64 -m64 -msvcrt -m32`

To add data padding:
- `--opt-assume-safe-padding`

Get Specific Advice For Improving Vectorization

Advisor shows hints to move iterations to vector body.

All Advisor-detectable issues: C++ | Fortran

Issue: Ineffective peeled/remainder loop(s) present

All or some source loop iterations are not executing in the loop body. Improve performance by moving source loop iterations from peeled/remainder loops to the loop body.

Add data padding

The trip count is not a multiple of vector length. To fix: Do one of the following:

- Increase the size of objects and add iterations so the trip count is a multiple of vector length.
- Increase the size of static and automatic objects, and use a compiler option to add data padding.
1. Compiler diagnostics + Performance Data + SIMD efficiency information

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

3. Trip Counts + FLOP: understand utilization, parallelism granularity & overheads
Critical Data Made Easy
Loop Trip Counts

Knowing the time spent in a loop is not enough!

Check actual trip counts
Find trip counts for each part of a loop
Precise Repeatable FLOP Metrics

- FLOPS by loop and function
- All recent Intel processors
- Instrumentation (count FLOP) plus sampling (time with low overhead)
- Adjusted for masking with AVX-512 processors
1. Compiler diagnostics + Performance Data + SIMD efficiency information

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

3. Trip Counts + FLOP: understand utilization, parallelism granularity & overheads

3. Memory Access Patterns Analysis
Improve Vectorization

Memory Access pattern analysis

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

Select loops of interest
### Advisor Memory Access Pattern (MAP): know your access pattern

#### Unit-Stride access

for (i=0; i<N; i++)

\[ A[i] = C[i] \times D[i] \]

#### Constant stride access

for (i=0; i<N; i++)

\[ \text{point}[i].x = x[i] \]

#### Variable stride access

for (i=0; i<N; i++)

\[ A[B[i]] = C[i] \times D[i] \]

---

<table>
<thead>
<tr>
<th>ID</th>
<th>Stride</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>3</td>
</tr>
</tbody>
</table>

**Memory Access Patterns Report**

<table>
<thead>
<tr>
<th>ID</th>
<th>Stride</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>3</td>
</tr>
</tbody>
</table>

---

**Optimization Notice**

Copyright © 2017, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.*
Find vector optimization opportunities
Memory Access pattern analysis

Site Location | Strides Distribution | Access Pattern | Max. Site Footprint
--- | --- | --- | ---
[loop in ComputeTimeStepKer... | 80% / 0% / 20% | Mixed strides | 2KB
[loop in pricePath_Core at ch_3... | 92% / 0% / 8% | Mixed strides | 1KB

Memory Access Patterns Report | Dependencies Report | Recommendations

All Advisor-detectable issues: **C++ | Fortran**

**Recommendation:** Refactor code with detected regular stride access patterns

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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>block 0x2e23c404b80 allocated at cache_aligned_allocator.cpp:196</td>
<td>Invariant</td>
</tr>
</tbody>
</table>

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

To improve memory access: Refactor your code to alert the compiler to a regular stride access. Sometimes, it might be beneficial to use the **ipo/Qipo** compiler option to enable interprocedural optimization (IPO) between files.
1. Compiler diagnostics + Performance Data + SIMD efficiency information

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

3. Trip Counts + FLOP: understand utilization, parallelism granularity & overheads

4. Memory Access Patterns Analysis

5. Loop-Carried Dependency Analysis
# ENABLING VECTORIZATION

<table>
<thead>
<tr>
<th>Vector Issues</th>
<th>Self Time</th>
<th>Total Time</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Assumed dependency present</td>
<td>20.030s</td>
<td>20.030s</td>
<td>Scalar Versions</td>
</tr>
<tr>
<td></td>
<td>13.508s</td>
<td>13.508s</td>
<td>Scalar</td>
</tr>
<tr>
<td></td>
<td>6.895s</td>
<td>27.750s</td>
<td>Scalar</td>
</tr>
</tbody>
</table>

**Check dependencies**

**Use #pragma simd**

<table>
<thead>
<tr>
<th>Vector Issues</th>
<th>Self Time</th>
<th>Total Time</th>
<th>Type</th>
<th>Vectorized Loops</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10.507s</td>
<td>22.989s</td>
<td>Scalar</td>
<td>Vector ISA</td>
</tr>
<tr>
<td>2 Possible inef...</td>
<td>1.762s</td>
<td>3.190s</td>
<td>Vectorized Ver...</td>
<td>AVX512</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Efficiency</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Gain</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>VL (V...</td>
</tr>
</tbody>
</table>

**Optimization Notice**

Copyright © 2017, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.*
Is It Safe to Vectorize?
Loop-carried dependencies analysis verifies correctness

Select loop for Correct Analysis and press play!

Vector Dependence prevents Vectorization!
Correctness – Is It Safe to Vectorize?

Loop-carried dependencies analysis

Received recommendations to force vectorization of a loop:

1. Mark-up loop and check for REAL dependencies
2. Explore dependencies with code snippets

In this example 3 dependencies were detected:

- **RAW** – Read After Write
- **WAR** – Write After Read
- **WAW** – Write After Write

This is NOT a good candidate to force vectorization!
Data Dependencies – Tough Problem #1
Is it safe to force the compiler to vectorize?

**Recommendation:** 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:

<table>
<thead>
<tr>
<th>Directive</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>#pragma simd or #pragma omp simd</td>
<td>Ignores all dependencies in the loop</td>
</tr>
<tr>
<td>#pragma ivdep</td>
<td>Ignores only vector dependencies (which is safest)</td>
</tr>
</tbody>
</table>
ROOFLINE
Questions to answer with Roofline: for your loops / functions

1. Am I doing well? How far am I from the peak? (do I utilize hardware well or not?)

2. Where is the final bottleneck? (where will be my limit after all optimizations?)
   Long-term ROI, optimization strategy

- Memory-bound, invest into cache blocking etc
- Compute bound: invest into SIMD...
Automated Roofline Chart Generation in Advisor - CARM

Each Roof (slope) gives peak CPU/Memory throughput of your PLATFORM (benchmarked)

Legend:
- Green dot: Takes less time
- Yellow dot: Takes considerable time
- Red dot: Takes much time

Each Dot represents loop or function in YOUR APPLICATION (profiled)

Summarized memory-compute efficiency picture for the application
Roofline picture

Performance headroom

Chart configuration

Tooltip with more data for dots

Switch to grid representation
Chart configuration

- Aggregate data over calltree
- Select which operations are counted
- Select memory levels
- Select only loads or stores

Options:
- Operations: FLOAT, INT, INT+FLOAT
- Callstacks: With Callstacks, CARM (L1+NTS), L2, L3, DRAM
- Memory Operation Type: Loads, Stores, Loads+Stores

[Diagram with options selected]
Integrated Roofline Memory Traffic Data in Survey Grid

Review memory level and loads/stores distribution to see memory traffic for specific memory level.
Integrated Roofline. What is my current limit?

Performance is limited by minimum of intercepts (L2, LLC, DRAM, CPU)

In this case: by DRAM
NEW*: Selecting Integer, Float or Mixed operations

Operations:
- FLOAT
- IN
- IN + FLOAT

Callstacks:
- With Callstacks

Memory Level:
- CARM (L1 + NTS)
- L2
- L3
- DRAM

Memory Operation Type:
- Loads
- Stores
- Loads + Stores

Physical Cores: 4
App Threads: 1
Self Elapsed Time: 7.524 s
Total Time: 7.524 s
Integer Operations in Survey Grid and Loop Analytics

Use settings button on the “Compute Performance” column

Use type selector in expanded Loop Analytics section
Compare results

Loaded results for two versions

Current version (highlighted)

Previous version

Easy to check optimization progress
Share with others

Snapshot (full-featured, opens in Advisor)

Standalone interactive HTML
(limited functionality)
Share roofline by email! - with colleagues or your manager
Use the rest of the Advisor

Select loops for deeper analysis

Examine source or assembly

See additional info in grid
A few words about callstacks

Same function, same loop...

But different FLOPS and AI on different callpaths!

Collecting callstacks we can

- See aggregated dots
- Callees and inner loops included
- See different dots for different callchains
Exporting Integer and Integrated Roofline as HTML

**GUI:** Use **Export as HTML** button

**Command line:**
```
advixe-cl -report roofline
-data-type=float
-memory-level=L2
-memory-operation-type=load
-project-dir /path/to/project/dir
```

Possible data types: float, int, mixed
Memory levels: L1, L2, L3, DRAM
Memory operation types: load, store, all

- Export Roofline from command line does not need GUI sub-system on clusters
- Useful for rooflines quick exchange
In a few words

Using Intel® Advisor, you can

- Collect the data for the Hierarchical and Integrated Roofline
- Analyze the roofline picture
- Focus on data you are interested in
- Compare roofline for different runs
- Share roofline results
- and more
COLLECTORS
# Collections vs Analysis

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Collections</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vectorization (basic)</td>
<td>Survey + Trip Counts</td>
</tr>
<tr>
<td>Vectorization (advanced)</td>
<td>As above + MAP + Dependencies</td>
</tr>
<tr>
<td>Roofline (CARM)</td>
<td>Survey + Trip Counts with FLOP</td>
</tr>
<tr>
<td>Roofline (Integrated)</td>
<td>Survey + Trip Counts with FLOP and Cache Simulator</td>
</tr>
<tr>
<td>Threading</td>
<td>Survey + Suitability + Dependencies</td>
</tr>
<tr>
<td>Custom Analysis (Python API)</td>
<td>Depends</td>
</tr>
</tbody>
</table>

*Mix and match as you wish*

*More data come with a cost*
HANDS-ON EXERCISE
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 CODE EXAMPLE

Consider solving differential equation with finite-difference method on 3-dimensional grid

Example: calculating Laplace operator of some field

```c
float * in = (float *) malloc(DIM*DIM*DIM * sizeof(float));
float * out = (float *) malloc(DIM*DIM*DIM * sizeof(float));

uint64_t iStride = 1;
uint64_t jStride = DIM;
uint64_t kStride = DIM * DIM;

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;
        }
    }
}
```

We encourage you to try the following steps on your own code, using the slides as a guide.
Activity 0: Building STENCIL
Build & Run

Setup environment:

$ source /soft/compilers/intel-2019/compilers_and_libraries/linux/bin/compilervars.sh intel64

Copy and unpack stencil sources:

$ cp /projects/ATPESC19_Instructors/advisor/advisor_lab.tar.gz ~ && cd && tar xzf advisor_lab.tar.gz

Go to working directory

$ cd ~/advisor_lab/src && git checkout ver0

Build application

$ make

Run application

$ ./stencil
Activity 0. Screenshot

```
[day1@clx-2 src]$ source /opt/intel/compilers_and_libraries/linux/bin/compilervars.sh intel64
[day1@clx-2 src]$ cd ~/advisor_lab/src && git checkout ver0
HEAD is now at 92efb0f... Initial commit
[day1@clx-2 src]$ make
icc -O3 -fopenmp -no-ipo -fno-inline-functions -g main.c bench_stencil.c -o stencil
[day1@clx-2 src]$ ./stencil
Naive: Dim= 512, nIterations= 10, Time= 4.102s, Useful GB/s= 5.297
[day1@clx-2 src]$
```
Activity 1: Doing Survey
Launch Advisor

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

Setup environment:

$ source /soft/compilers/intel-2019/advisor_2019/advixe_vars.sh

$ export ADVIXE_EXPERIMENTAL=int Roofline,roofline_guidance

Launch Advisor GUI:

$ advixe-gui &
Create Advisor Project

Welcome to Intel Advisor 2019
Vectorization Optimization and Thread Prototyping

1. Click to create a new project

2. Type name of the project

3. Click to finish creation

Create a Project

Project name: stencil
Location: /home/day1/advisor_lab/profile

Create Project  Cancel
Set UP Project

Set the application to launch: 
~/advisor_lab/src/stencil

Press OK button
Start Survey Analysis

Press “Collect” button in “1. Survey Target” section
Activity 1. Screenshot
Create a snapshot
Activity 2: Dealing with data type conversions
**LOOK AT THE RECOMMENDATIONS**

**Optimization Notice**

Although static analysis presumes the loop may benefit from FMA instructions available with the AVX2 or higher ISA, no FMA instructions executed for this loop. To fix: Use the following compiler options:
- xCORE-AVX2 to compile for machines without AVX2 support
- xCOMMON-AVX312 to compile for machines with AVX2 support only
- xCOMMON-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.
Activity 2

Purpose: Dealing with data type conversions

Build a version with fixed conversions

$ git checkout ver1
$ make

Re-run Survey analysis

Create a snapshot

Compare with previous activity
Activity 2. Screenshots
Activity 3: Doing roofline analysis
Activity 3. 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
~ 4 minutes

Create a snapshot

For Integrated Roofline (NEW!)
Activity 3. Screenshot
Activity 3. Screenshot
Activity 4: Checking for dependencies
Activity 4. Collect data to GET Dependencies

Purpose: Find loop-carried dependencies

Select “loop in bench_stencil at bench_stencil.c:23”

Press “Collect” button in “2.2 Check Dependencies” section

~ 1 minute

Create a snapshot
Activity 4. Screenshot

Optimization Notice

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 5: Adding threading and enabling vectorization
Activity 5

Purpose: Add threading and enable vectorization

Build a version with threading and vectorization

$ git checkout ver3
$ make

Re-run Roofline analysis

Create a snapshot

Compare with previous activity
Activity 5. Screenshots

Vectorization Advisor

Vectorization Advisor is a vectorization analysis toolset that lets you identify loops that will benefit most from vector parallelism, discover performance vectorization and characterize your memory vs. vectorization bottlenecks with Advisor Roofline model automation.

Program metrics

- Elapsed Time: 2.74s
- Vector Instruction Set: SSE
- Number of CPU Threads: 4

Optimization Notice

Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
Activity 6: 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 6. Collect data to GET Memory Access Patterns

Purpose: Calculate strides

Select “loop in bench_stencil$omp$parallel_for@23 at bench_stencil.c:24”

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

~1 minute

Create a snapshot
Activity 6. Screenshot
Activity 7: Splitting task to tiles
Activity 7

Purpose: Improve memory access pattern

Build a version with tiling

$ git checkout ver4
$ make

Re-run Roofline analysis

Create a snapshot

Compare with previous activity

Vectorization Advisor

Vectorization Advisor is a vectorization analysis tool to vectorize and characterize your memory vs. vector.

Program metrics

- Elapsed Time: 1.32s
- Vector Instruction Set: SSE
- Number of CPU Threads: 4
Activity 6. Screenshot
Activity 8: Enabling AVX512
Activity 8

Purpose: Fix compilation options to use the highest available ISA

Build a version with new compilation flags

$ git checkout ver5

$ make clean && make

Re-run Survey analysis

Create a snapshot

Compare with previous activity

Review recommendations
Activity 8. Screenshots
Activity 8. Screenshots

Ineffective peeled/remainder loop(s) present

All or some source loop iterations are not executing in the loop body. Improve performance by moving source loop iterations from peeled/remainder loops to the loop body.

1. Disable dynamic alignment

The compiler automatically peeled iterations from the vector loop into a scalar loop to align the vector loop with a particular memory reference; however, this optimization may not be ideal. To possibly achieve better performance, disable automatic peel generation using the directive: 

```
#pragma vector nodynamic_align
```

Example (original code)

```
...  
#pragma vector nodynamic_align  
for (int i = 0; i < len; i++)  
...  
```
Activity 9: Disabling dynamic alignment
Activity 9

**Purpose: Exclude loop peel/reminder execution**

Build a version with new compilation flags

```bash
$ git checkout ver6
$ make
```

Re-run Roofline analysis

Create a snapshot

Compare with previous activity
Activity 9. Screenshots
Activity 10: Comparing roofline charts
ACTIVITY 10

Purpose: Graph roofline chart for optimized version, and compare with initial chart

Turn off “Show different memory level relationships” at Guidance tab

Compare with results for versions of source code “ver3” and “ver0”
Activity 10. Screenshot

Speedup: ~12x
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.

Copyright © 2016, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

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