Agenda

Quick overview of the Intel® Parallel Studio 2018 Beta

Intel® Advisor overview

Intel® Advisor AVX-512 profiling

Intel® Advisor Roofline automation

Intel® Advisor new features

Summary/call to action
What’s new in the “2018” release

1. Compiler diagnostics + Performance Data + SIMD efficiency information
   - Modules exclusions, MKL

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

3. “Precise” Trip Counts & FLOPs. Roofline analysis.
   - Call Counts, MKL, Instruction count, Hier. Roofline
   - Roofline is now a product feature!

4. Loop-Carried Dependency Analysis
   - Overhead decreased

5. Memory Access Patterns Analysis
   - Cache simulation (feature preview)
Advisor Survey: **Focus + Characterize.**

**Focus and order vectorized loops**

- **Efficiency** – my performance thermometer
- **Recommendations** – get tip on how to improve performance
  - (also apply to scalar loops)
Data Dependencies – Tough Problem #1
Is it safe to force the compiler to vectorize?

DO I = 1, N
  A(I) = A(I-1) * B(I)
ENDDO

or

void scale(int *a, int *b)
{
  for (int i = 0; i < 1000; i++)
    b[i] = z * a[i];
}

---

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

- Enable vectorization
  - Potential performance gain: Information not available until Beta Update release
  - Confidence this recommendation applies to your code: Information not available until Beta Update release

The Correctness analysis shows there is no real dependency in the loop for the given workload. Tell the compiler it is safe to vectorize using the `restrict` keyword or a `directive`.

<table>
<thead>
<tr>
<th>ICL/ICC/ICPC Directive</th>
<th>IFORT Directive</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>#pragma simd or #pragma omp simd</td>
<td>IDIRS SIMD or ISOMP SIMD</td>
<td>Ignores all dependencies in the loop</td>
</tr>
<tr>
<td>#pragma ivdep</td>
<td>IDIRS IVDEP</td>
<td>Ignores only vector dependencies (which is safest)</td>
</tr>
</tbody>
</table>

Read More:
- [User and Reference Guide for the Intel C++ Compiler 15.0](#) > Compiler Reference > Pragmas > Intel-specific Pragma Reference >
  - ivdep
  - omp simd
Advisor Memory Access Pattern (MAP): know your access pattern

- **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]
  ```
AVX-512 PROFILING WITH INTEL® ADVISOR
Intel® Advisor: AVX-512 specific performance insights

- Native AVX-512 profiling on KNL
- Precise FLOPs and Mask Utilization profiler
- AVX-512 Advices and “Traits”
- And more..
  - Performance Summary for AVX-512 codes
  - AVX-512 Gather/Scatter Profiler
- No access to AVX-512 Hardware yet?
  - Explore AVX-512 code with –axcode flags and new Advisor Survey capability!
Highlight “impactful” AVX-512 instructions

Survey Static Analysis - AVX-512 “Traits”

Presence of remarkable performance-impactful (negative or positive impact) instructions

<table>
<thead>
<tr>
<th>Vectorization Advisor Trait and/or Recommendation</th>
<th>Theoretical Performance Impact Comments</th>
<th>Corresponding AVX-512 Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compress / Expand Trait and Recommendation</td>
<td>&gt;&gt; 4x speedup</td>
<td>v(p) expand*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>v(p) compress*</td>
</tr>
<tr>
<td>Gather / Scatter Trait</td>
<td>Up to 10x slower than contiguous memory access &gt;2x faster than scalar</td>
<td>v(p) gather*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>v(p) scatter*</td>
</tr>
<tr>
<td>Conflict Detection</td>
<td></td>
<td>v(p) conflict*</td>
</tr>
<tr>
<td>Approximate Reciprocals/Reciprocal SQRT; AVX-512ER</td>
<td>&gt;10x faster than DIV/SQRT</td>
<td>vrcp*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>vrcsqrt*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>vdiv*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>vsqrt*</td>
</tr>
<tr>
<td>Exponent extraction Mantissa extraction Traits</td>
<td></td>
<td>vgetexp*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>vgetmant*</td>
</tr>
<tr>
<td>L1 (L2) Prefetch L1 (L2) Sparse prefetch Trait</td>
<td></td>
<td>prefetchw*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>vscatterpf*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>vgatherpf*</td>
</tr>
</tbody>
</table>
Gather/Scatter Analysis

Motivation

AVX-512 Gather/Scatter-based vectorization.

Much wider usage than before:

- Makes much more codes (profitably) vectorizable
- Gives good average performance, but often far from optimal.

Could be 2x faster than scalar mov

Could be 10x slower than vmovp*
Gather/Scatter Analysis
Advisor MAP detects gather “offset patterns”.

Vector loop (vector length = 8)

Horizontal stride
0x00 0x04 0x08 0x0C 0x10 0x14 0x18 0x1C

Vertical stride

Vectorized Loop (2 vector iterations, 16 original scalar iterations)

Horizontal stride
0x00 0x04 0x08 0x0C 0x10 0x14 0x18 0x1C

Vertical stride
0x20 0x24 0x28 0x2C 0x30 0x34 0x38 0x3C

Gather details
Pattern #1: "Invariant"
Instruction gathers values from the same memory throughout the loop
Horizontal stride: 8
Vertical stride: N/A
Mask is constant
Mask: [00000101]
Mask is filled to 25.0%.

<table>
<thead>
<tr>
<th>Pattern #</th>
<th>Pattern Name</th>
<th>Horizontal Stride Value</th>
<th>Vertical Stride Value</th>
<th>Example of Corresponding Fix(es)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Invariant</td>
<td>0</td>
<td>0</td>
<td>OpenMP uniform clause, simd pragmas/directive, retilting</td>
</tr>
<tr>
<td>2</td>
<td>Uniform (horizontal invariant)</td>
<td>0</td>
<td>Arbitrary</td>
<td>OpenMP uniform clause, simd pragmas/directive</td>
</tr>
<tr>
<td>3</td>
<td>Vertical Invariant</td>
<td>Constant</td>
<td>0</td>
<td>OpenMP private clause, simd pragmas/directive</td>
</tr>
<tr>
<td>4</td>
<td>Unit</td>
<td>1 or -1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Constant</td>
<td>Constant = X</td>
<td>Constant = X*VectorLength</td>
<td>Subject for AD &gt;= Sea transformation</td>
</tr>
</tbody>
</table>
Gather/scatter issue improvements

Compiler may generate gather/scatter instructions despite regular access pattern. In this case, performance can be improved by refactoring the code.

- Detecting regular patterns taking into account masking instructions
- Added new access pattern for gather profiling – Constant (Non-Unit Stride) with adjusted recommendation to transform AOS to SOA

**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 0x7f049a67f010</td>
<td>Constant (non-unit)</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.

An array is the most common type of data structure containing a contiguous collection of data items that can be accessed by an ordinal index. You can organize this data as an array of structures (AoS) or as a structure of arrays (SoA). Detected constant stride might be the result of AoS implementation. While this organization is excellent for encapsulation, it can hinder effective vector processing. To fix:

Rewrite code to organize data using SoA instead of AoS.

However, the cost of rewriting code to organize data using SoA instead of AoS may outweigh the benefit. To fix: Use Intel SIMD Data Layout Templates (Intel SDLT), introduced in version 16.1 of the Intel compiler, to mitigate the cost. Intel SDLT is a C++11 template library that may reduce code rewrites to just a few lines.
AVX-512-specific performance trade-offs

Advisor AVX-512 Recommendations

Increasing Vector Register Size  ->

Increase fraction of time spent in Remainders

<table>
<thead>
<tr>
<th>Function Call Sites and Loops</th>
<th>Vector Issues</th>
<th>Self Time</th>
<th>Total Time</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>[loop in fCollisionBGKShanChenSom...](loop in fCollisionBGKShanChenSom...)</td>
<td>1 Ineffective peeled/remainder loop(s)</td>
<td>0,110s</td>
<td>0,110s</td>
<td>Vectorized (Remainder; [Body]) AVX512</td>
</tr>
<tr>
<td>[loop in fCollisionBGKShanChenSom...](loop in fCollisionBGKShanChenSom...)</td>
<td></td>
<td>0,110s</td>
<td>0,110s</td>
<td>Vectorized (Remainder) AVX512</td>
</tr>
<tr>
<td>[loop in fCollisionBGKShanChenSom...](loop in fCollisionBGKShanChenSom...)</td>
<td></td>
<td>n/a</td>
<td>n/a</td>
<td>Vectorized (Body) [Not Executed] AVX512</td>
</tr>
<tr>
<td>[loop in fGetFracSite at lbpGET.cpp;19...](loop in fGetFracSite at lbpGET.cpp;19...)</td>
<td>1 Ineffective peeled/remainder loop(s)</td>
<td>0,060s</td>
<td>0,060s</td>
<td>Vectorized (Peeled; Remainder; [Body]) AVX512</td>
</tr>
<tr>
<td>[loop in fGetFracSite at lbpGET.cpp;19...](loop in fGetFracSite at lbpGET.cpp;19...)</td>
<td></td>
<td>0,040s</td>
<td>0,040s</td>
<td>Vectorized (Peeled) AVX512</td>
</tr>
<tr>
<td>[loop in fGetFracSite at lbpGET.cpp;19...](loop in fGetFracSite at lbpGET.cpp;19...)</td>
<td></td>
<td>0,020s</td>
<td>0,020s</td>
<td>Vectorized (Body) [Not Executed] AVX512</td>
</tr>
<tr>
<td>[loop in fCalcInteraction_ShankChen a...](loop in fCalcInteraction_ShankChen a...)</td>
<td>1 Ineffective peeled/remainder loop(s)</td>
<td>0,060s</td>
<td>0,060s</td>
<td>Vectorized (Remainder; [Body]) AVX512</td>
</tr>
<tr>
<td>[loop in fCalcInteraction_ShankChen a...](loop in fCalcInteraction_ShankChen a...)</td>
<td></td>
<td>0,060s</td>
<td>0,060s</td>
<td>Vectorized (Remainder) AVX512</td>
</tr>
<tr>
<td>[loop in fCalcInteraction_ShankChen a...](loop in fCalcInteraction_ShankChen a...)</td>
<td></td>
<td>n/a</td>
<td>n/a</td>
<td>Vectorized (Body) [Not Executed] AVX512</td>
</tr>
<tr>
<td>[loop in fGetOneMassSite at lbpGET.c...](loop in fGetOneMassSite at lbpGET.c...)(not vect.)</td>
<td>1 Ineffective peeled/remainder loop(s)</td>
<td>0,050s</td>
<td>0,050s</td>
<td>Vectorized (Remainder; [Body]) AVX512</td>
</tr>
<tr>
<td>[loop in fGetOneMassSite at lbpGET.c...](loop in fGetOneMassSite at lbpGET.c...)(not vect.)</td>
<td></td>
<td>0,040s</td>
<td>0,040s</td>
<td>Vectorized (Remainder) AVX512</td>
</tr>
<tr>
<td>[loop in fGetOneMassSite at lbpGET.c...](loop in fGetOneMassSite at lbpGET.c...)(not vect.)</td>
<td></td>
<td>0,030s</td>
<td>0,030s</td>
<td>Vectorized (Remainder) AVX512</td>
</tr>
<tr>
<td>[loop in fGetOneMassSite at lbpGET.c...](loop in fGetOneMassSite at lbpGET.c...)(not vect.)</td>
<td></td>
<td>0,030s</td>
<td>0,030s</td>
<td>Vectorized (Remainder) AVX512</td>
</tr>
<tr>
<td>[loop in fGetOneMassSite at lbpGET.c...](loop in fGetOneMassSite at lbpGET.c...)(not vect.)</td>
<td></td>
<td>0,020s</td>
<td>0,020s</td>
<td>Vectorized (Remainder) AVX512</td>
</tr>
</tbody>
</table>

Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others."
Ineffective masked remainder for AVX512 codes

- Compiler generates vector masked remainder due to the number of iterations (trip count) not being divisible by vector length. In case of executing a few iterations, it is ineffective comparing to scalar versions of the loop.

- Using AVX512 mask profiler and trip-counts data to prove the issue.

```c
#pragma simd reduction(+:mean)
for(int j = 0; j < size; j++) {
    mean += data[order[j]] / N;
    data[order[j]] = 10.f / (j+1);
}
```

E.g. bad performance if ((size) % (loop_body_vl) == 1), in case of float number it results in 12.5% mask bits utilization only, in addition leads to gathers, scatters...
ROOFLINE PERFORMANCE MODEL
CASE STUDY
Roofline Analysis to Tune an MRI Image Reconstruction Benchmark

The 514.pomriq SPEC ACCEL Benchmark

An MRI image reconstruction kernel described in Stone et al. (2008). MRI image reconstruction is a conversion from sampled radio responses to magnetic field gradients. The sample coordinates are in the space of magnetic field gradients, or K-space.

The algorithm examines a large set of input, representing the intended MRI scanning trajectory and the points that will be sampled.

The input to 514.pomriq consists of one file containing the number of K-space values, the number of X-space values, and then the list of K-space coordinates, X-space coordinates, and Phi-field complex values for the K-space samples.
Hot loop is vectorized

Optimization Notice

Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Intel Advisor summary view

1 vectorized loop that we spend 98.8% of our time in

Need more information to see if we can get more performance
What is our performance?
Relative to peak system performance

Our hot loop is below the MCDRAM roof
Potential memory bottleneck
Get detailed Advice from intel® Advisor

Possible inefficient memory access. Gather stride.

Recommendations – need more information, confirm inefficient memory access
Irregular access patterns decreases performance!
Gather profiling

Run Memory Access Pattern Analysis (MAP)

2.2 Check Memory Access Patterns

- Collect
- Analyze

-- Nothing to analyze --

- 0%: percentage of memory instructions with unit stride or stride 0 accesses
  - Unit stride (stride 1) = Instruction accesses memory that consistently changes by one element from iteration to iteration
  - Uniform stride (stride 0) = Instruction accesses the same memory from iteration to iteration

- 50%: percentage of memory instructions with fixed or constant non-unit stride accesses
  - Constant stride (stride N) = Instruction accesses memory that consistently changes by N elements from iteration to iteration
  - Example: for the double floating point type, stride 4 means the memory address accessed by this instruction increased by 32 bytes, (4 * sizeof(double)) with each iteration

- 50%: percentage of memory instructions with irregular (variable or random) stride accesses
  - Irregular stride = Instruction accesses memory addresses that change by an unpredictable number of elements from iteration to iteration
  - Typically observed for indirect indexed array accesses, for example, a[index[i]]
  - Gather (irregular) accesses, detected for v(p)gather* instructions on AVX2 Instruction Set Architecture
Irregular access patterns
Bad for vectorization performance

Hint: use the Intel Advisor details!

Specific recommendation for your application

The compiler assumes indirect or irregular stride access to data used for vector operations. Improve memory access by altering the compiler to detected regular stride access patterns, such as:

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invariant</td>
<td>The instruction accesses values in the same memory throughout the loop.</td>
</tr>
<tr>
<td>Uniform (Horizontal Invariant)</td>
<td>The instruction accesses values in the same memory within the vector iteration.</td>
</tr>
<tr>
<td>Vertical Invariant</td>
<td>The instruction accesses the memory locations using the same offset across all vector iterations.</td>
</tr>
<tr>
<td>Unit</td>
<td>The instruction accesses values in contiguous memory throughout the loop, and the stride between vector iterations = vector length.</td>
</tr>
</tbody>
</table>

Recommendation: Refactor code with detected regular stride access patterns

The Memory Access Patterns Report shows the following regular stride accesses:

Names: block 0x7f0045867010 allocated at main.c:99
Remove gather instructions

step #1 – use newer version of the intel compiler can recognize the access pattern

Removed gathers

Gathers replacement is performed by the “Gather to Shuffle/Permuters” compiler transformation

Increased GFLOPS (from 266.42 to 342.67)
Remove gather instructions
step #1 – newer version of the intel compiler can recognize the access pattern

Now above MCDRAM roof
Greater GFLOPS
Remove gather instructions
step #2 - Use structure of arrays instead of array of structures T

struct kValues {
    float Kx;
    float Ky;
    float Kz;
    float PhiMag;
};

SDLT_PRIMITIVE(kValues, Kx, Ky, Kz, PhiMag)

sdlt::soa1d_container<kValues> inputKValues(numK);
auto kValues = inputKValues.access();
for (k = 0; k < numK; k++) {
    kValues[k].Kx() = kx[k];
    kValues[k].Ky() = ky[k];
    kValues[k].Kz() = kz[k];
    kValues[k].PhiMag() = phiMag[k];
}

auto kVals = inputKValues.const_access();
#pragma omp simd private(expArg, cosArg, sinArg) reduction(+:QrSum, QiSum)
for (indexK = 0; indexK < numK; indexK++) {
    expArg = PIx2 * (kVals[indexK].Kx() * x[indexX] + kVals[indexK].Ky() * y[indexX] + kVals[indexK].Kz() * z[indexX]);
    cosArg = cosf(expArg);
    sinArg = sinf(expArg);
    float phi = kVals[indexK].PhiMag();
    QrSum += phi * cosArg;
    QiSum += phi * sinArg;
}

This is a classic vectorization efficiency strategy
But it can yield poorly designed code

Intel® SIMD Data Layout Templates makes this transformation easy and painless!
Remove gather instructions

step #2 - Transform code using the Intel® SIMD Data Layout Templates

The loop is no longer red. This means it takes less time now

Has more GFLOPS, putting it close to the L2 roof

The total performance improvement is almost 3x for the kernel and 50% for the entire application.
ROOFLINE PERFORMANCE MODEL

AUTOMATION
Acknowledgments/References

Roofline model proposed by Williams, Waterman, Patterson:
http://www.eecs.berkeley.edu/~waterman/papers/roofline.pdf


At Intel:
Roman Belenov, Zakhar Matveev, Julia Fedorova
SSG product teams, Hugh Caffey,
in collaboration with Philippe Thierry
Roofline model: Am I bound by VPU/CPU or by Memory?

What makes loops A, B, C different?
Roofline in Intel® Advisor

Automatic and integrated – first class citizen in Intel® Advisor

Switch between the roofline and the grid

Source for the selected loop

Loop data hint
Find Effective Optimization Strategies

Intel Advisor: Cache-aware roofline analysis

Roofs Show Platform Limits
- Memory, cache & compute limits

Dots Are Loops
- Bigger, red dots take more time so optimization has a bigger impact
- Dots farther from a roof have more room for improvement

Higher Dot = Higher GFLOPs/sec
- Optimization moves dots up
- Algorithmic changes move dots horizontally

Which loops should we optimize?
- A and G have the biggest impact & biggest gap
- B has room to improve, but will have less impact
- E and H are perfectly optimized already

Roofline tutorial video
### Getting Roofline data in Intel® Advisor

<table>
<thead>
<tr>
<th>FLOP/S</th>
<th>Seconds</th>
<th>#FLOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>= #FLOP/Seconds</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Step 1: Survey
- Non intrusive. *Representative*
- Output: Seconds (+much more)

### Step 2: Trip counts+FLOPS
- Precise, instrumentation based
- Physically count Num-Instructions
- Output: #FLOP, #Bytes
Find Effective Optimization Strategies
Intel Advisor: Cache-aware roofline analysis

Roofline Performance Insights
- Highlights poor performing loops
- Shows performance “headroom” for each loop
  - Which can be improved
  - Which are worth improving
- Shows likely causes of bottlenecks
- Suggests next optimization steps
Is My Application Bound by a Memory Bandwidth or a Compute Peak?

Often it’s a combination of the two

- Applications in area 1 are purely memory bandwidth bound
- Applications in area 3 are purely compute bound
- In area 2 we need more information

![Diagram showing the relationship between Arithemetic Intensity (flops/byte) and Attainable Performance (Gflops/s). Areas are labeled with:

1. Memory Bound
2. Memory/Compute Bound
3. Compute Bound]

Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
Ask Yourself “Why am I Here?” and “Where am I going?”

Usually, it is more complicated...

You won't be on any ceiling. Or if you are, it is kind of coincidence.

BUT - asking the questions “why am I not on a higher ceiling?” and “what should I do to reach it?” is always productive.
Perform the right optimization for your region

Roofline: characterization regions

Optimization Notice

Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
WHAT’S NEW 2018
Intel® Advisor 2018 What’s New

Hierarchical Roofline (Experimental)

MKL Summary

Python API

Cache simulation (Experimental)
Hierarchical (top-down) Roofline: new in 2018 release

```
export ADVIXE_EXPERIMENTAL=roofline_ex
```
Hierarchical Roofline (based on stacks w/ FLOPS)

> source advixe-vars.sh

> export ADVIXE_EXPERIMENTAL=roofline_ex

> advixe-cl --collect survey --project-dir ./your_project -- <your-executable-with-parameters>

> advixe-cl --collect tripcounts -flops-and-masks -callstack-flops --project-dir ./your_project -- <your-executable-with-parameters>

> export ADVIXE_EXPERIMENTAL=roofline_ex

> advixe-gui ./your_project
Roofline in Action: neural networks profiling

Inefficient implementation for pooling and convolutional layers
Get insights on how effectively you are using MKL

**MKL summary**

- **Elapsed time:** 62.49s
- **Vector Instruction Set:** AVX2, AVX, SSE2, SSE
- **Number of CPU Threads:** 72
- **Total GFLOP Count:** 549.89
- **Total GFLOPS:** 8.80
- **Total Arithmetic Intensity:** 0.10

### Program metrics

- **Elapsed Time:** 62.49s
- **Vector Instruction Set:** AVX2, AVX, SSE2, SSE
- **Number of CPU Threads:** 72
- **Total GFLOP Count:** 549.89
- **Total GFLOPS:** 8.80
- **Total Arithmetic Intensity:** 0.10

### Loop metrics

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Total</th>
<th>User</th>
<th>MKL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total CPU time</td>
<td>3604.58s</td>
<td>3593.13s</td>
<td>99.7%</td>
</tr>
<tr>
<td>Time in 10 vectorized loops</td>
<td>162.29s</td>
<td>156.18s</td>
<td>11.45</td>
</tr>
<tr>
<td>Time in scalar code</td>
<td>3442.29s</td>
<td>3436.95s</td>
<td>6.11s</td>
</tr>
<tr>
<td>Total GFLOP Count</td>
<td>549.89</td>
<td>412.32</td>
<td>75.0%</td>
</tr>
<tr>
<td>Total GFLOPS</td>
<td>8.80</td>
<td>6.66</td>
<td>245.6</td>
</tr>
</tbody>
</table>

See how effectively you are using MKL!

See GFLOPS for user and MKL code

Also see how much time you are spending in MKL relative to total program time.
See MKL code on the Roofline chart
Drill down into MKL loop details
Verify vectorization and memory access patterns
Access the Intel® Advisor database using Python

Python api

• You can now access the Intel® Advisor database using our new Python API
• We have provided several reference examples on how to use this new functionality.

> source advixe-vars.sh

> advixe-cl --collect survey --project-dir ./your_project -- <your-executable-with-parameters>

> advixe-cl --collect tripcounts -flops-and-masks -callstack-flops --project-dir ./your_project -- <your-executable-with-parameters>

> python /opt/intel/advisor_2018/pythonapi/joined.py ./your_project
> & report.txt
>

Optimization Notice

Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
Flexible way to report on useful program metrics

Over 500 metric elements can be displayed. (See column.txt)

We also provide a way to generate customizable html reports

```python
python ./to_html.py ./adv
```
Generate a Roofline chart using the Python api

- python /opt/intel/advisor_2018/pythonapi/roofline.py ./your_project
Experiment with how you using utilizing cache
Experimental cache simulation feature

> source advixe-vars.sh

> export ADVIXE_EXPERIMENTAL=cachesim

> advixe-cl --collect survey --project-dir ./your_project -- <your-executable-with-parameters>

Select your loops of interest using the mark-up-list (you can generate this using the Advisor GUI)

> advixe-cl --collect map -mark-up-list=4 --project-dir ./your_project -- <your-executable-with-parameters>

> python cache.py ./your_project
Cache simulation setting in Project properties
Cache simulation
Model how effectively you are utilizing cache

Site: loop_site_9
Location: loop_site_9
File  Line: 69

Cache model settings:
Associativity  = 8
Sets  = 4096

Cache model results:
Writes  = 46
Reads  = 92
Read misses  = 50

Evicted cache lines utilization:
Average utilization = 6.25%

Bytes used | Evicted lines
4 | 38
Call to Action

Modernize your Code

- To get the most out of your hardware, you need to modernize your code with vectorization and threading.

- Taking a methodical approach such as the one outlined in this presentation, and taking advantage of the powerful tools in Intel® Parallel Studio XE, can make the modernization task dramatically easier.
  - The Professional and Cluster Edition both include Advisor

- Join the 2018 beta of Intel Parallel Studio XE to get the latest version

- Send e-mail to vector_advisor@intel.com to get the latest information on some exciting new capabilities that are currently under development.
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