INTEL ADVISOR: ROOFLINE AUTOMATION

Intel Software and Services, 2017
Zakhar Matveev, PhD, Product architect
5 Steps to Efficient Vectorization - Vector Advisor
(part of Intel® Advisor, Parallel Studio, Cluster Studio 2016)

### 1. Compiler diagnostics + Performance Data + SIMD efficiency information

<table>
<thead>
<tr>
<th>Function Call Statistics</th>
<th>Set Time</th>
<th>Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>vecsimd_total_time</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

**Issue:** Pivoted/remainder loop(s) present

- All or some source loops are not executing in the kernel loop. Improve performance by moving source loop iterations from pivoted/remainder loops to the kernel loop. Read more at [Vector Essentials](#).

**Recommendation: Align memory access**

Projected maximum performance gain: High Program confidence: Medium

### 3. “Accurate” Trip Counts + FLOPs: understand utilization, parallelism granularity & overheads

- **Trip Counts**
  - **Total Time**
  - **Max Infect Duration**
  - **Call Count**

### 4. Loop-Carried Dependency Analysis

#### Problems and Messages

<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
<th>Site Name</th>
<th>Sources</th>
<th>Modules</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Parallel site information</td>
<td>site2</td>
<td>dgttest2.cpp</td>
<td>dgttest2</td>
<td>Not a problem</td>
</tr>
<tr>
<td>P2</td>
<td>Read after write dependency</td>
<td>site2</td>
<td>dgttest2.cpp</td>
<td>dgttest2</td>
<td>New</td>
</tr>
<tr>
<td>P3</td>
<td>Read after write dependency</td>
<td>site2</td>
<td>dgttest2.cpp</td>
<td>dgttest2</td>
<td>New</td>
</tr>
<tr>
<td>P4</td>
<td>Write after write dependency</td>
<td>site2</td>
<td>dgttest2.cpp</td>
<td>dgttest2</td>
<td>New</td>
</tr>
<tr>
<td>P5</td>
<td>Write after write dependency</td>
<td>site2</td>
<td>dgttest2.cpp</td>
<td>dgttest2</td>
<td>New</td>
</tr>
<tr>
<td>P6</td>
<td>Write after read dependency</td>
<td>site2</td>
<td>dgttest2.cpp</td>
<td>dgttest2</td>
<td>New</td>
</tr>
<tr>
<td>P7</td>
<td>Write after read dependency</td>
<td>site2</td>
<td>dgttest2.cpp</td>
<td>dgttest2</td>
<td>New</td>
</tr>
</tbody>
</table>

### 5. Memory Access Patterns Analysis

#### Memory Access Patterns:

<table>
<thead>
<tr>
<th>ID</th>
<th>Store</th>
<th>Type</th>
<th>Source</th>
<th>Modules</th>
<th>Alignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>P22</td>
<td>0:1</td>
<td>Unit stride</td>
<td>runChwLoops.cpp:67</td>
<td>lcache.eue</td>
<td></td>
</tr>
</tbody>
</table>
Vectorization Analysis Workflow

1. Run Survey
2. Check Trip Counts
3. Check Dependencies
4. Check Memory Access Patterns

Start

Take Snapshot

Edit & Compile

Deeper-dive analysis
Use the same target binary within every cycle

Mark Loops for Deeper Analysis
Select loops in the Survey Report for Dependencies and/or Memory Access Patterns analysis.
2 loops are marked

1.1 Find Trip Counts
Collect

2.1 Check Dependencies
Collect

2.2 Check Memory Access Patterns
Collect

Batch mode
OFF

Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
Quickly **characterize** the efficiency of your code: Advisor Summary.

Use the summary view to quickly **characterize** your program.

**Time in Scalar vs. Vector** loops. **SIMD Efficiency.**

Focus on Hottest kernels.
**Advisor Survey:** Focus + Characterize.

One stop shop.

---

**Where should I add vectorization and/or threading parallelism?**

<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>
<th>Why No Vectorization?</th>
<th>Vectorized Loops</th>
</tr>
</thead>
<tbody>
<tr>
<td>[loop in matvec at Multiply.c72]</td>
<td>✔️</td>
<td>✔️ Inefficient...</td>
<td>5.281s</td>
<td>5.281s</td>
<td>Vectorized (Bo...</td>
<td>AVX -55% 2.19x 4</td>
</tr>
<tr>
<td>[loop in matvec at Multiply.c66]</td>
<td>✔️</td>
<td></td>
<td>2.828s</td>
<td>2.828s</td>
<td>Vectorized (Bo...</td>
<td>AVX -51% 3.65x 4</td>
</tr>
<tr>
<td>[loop in matvec at Multiply.c49]</td>
<td>✔️</td>
<td></td>
<td>0.531s</td>
<td>5.812s</td>
<td>Scalar</td>
<td></td>
</tr>
<tr>
<td>[loop in matvec at Multiply.c49]</td>
<td>✔️</td>
<td></td>
<td>0.516s</td>
<td>3.344s</td>
<td>Scalar</td>
<td></td>
</tr>
<tr>
<td>[loop in matvec at Multiply.c85]</td>
<td>✔️</td>
<td></td>
<td>0.063s</td>
<td>0.063s</td>
<td>Scalar</td>
<td></td>
</tr>
<tr>
<td>[loop in main at River.c151]</td>
<td>✔️</td>
<td></td>
<td>0.016s</td>
<td>9.297s</td>
<td>Scalar</td>
<td></td>
</tr>
<tr>
<td>[loop in matvec at Multiply.c49]</td>
<td>✔️</td>
<td></td>
<td>0.000s</td>
<td>0.000s</td>
<td>Scalar</td>
<td></td>
</tr>
</tbody>
</table>

What prevents vectorization?

- Inefficient...
- Ineffective...
- Assumed dependence...
- Vector dependence...
- Logarithmic function...

---

**Optimization Notice**

Copyright © 2015, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.*
Advisor Memory Access Pattern (MAP): know your access pattern

Unit-Stride access

\[
\text{for } (i=0; i<N; i++) \quad A[i] = C[i] \times D[i]
\]

Constant stride access

\[
\text{for } (i=0; i<N; i++) \quad \text{point}[i].x = x[i]
\]

Variable stride access

\[
\text{for } (i=0; i<N; i++) \quad A[B[i]] = C[i] \times D[i]
\]

---

### Memory Access Patterns Report

<table>
<thead>
<tr>
<th>Site Location</th>
<th>Loop-Carried Dependencies</th>
<th>Strides Distribution</th>
<th>Access Pattern</th>
<th>Site Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>[loop in PropagationSwap at lbpSUB.cpp:1247]</td>
<td>No information available</td>
<td>33% / 50% / 12%</td>
<td>Mixed strides</td>
<td>loop_site_60</td>
</tr>
</tbody>
</table>

---

**Optimization Notice**

Copyright © 2015, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.*
Am I bound by VPU/CPU or by Memory?

ROOFLINE ANALYSIS

Better optimized – smaller potential (gap)

Big optimization gap
From “Old HPC principle” to modern performance model

“Old” HPC principles:

1. “Balance” principle (e.g. Kung 1986) – hw and software parameters altogether


More research catalyzed by memory wall/ gap growth and by GPGPU


Memory Wall

![Graph showing the performance of processors and memory from 1980 to 2010. The graph indicates a steady increase in performance over time.](Patterson, 2011)
From “Old HPC principle” to modern performance model

“Old” HPC principles:

1. “Balance” principle (e.g. Kung 1986) – hw and software parameters altogether


More research catalyzed by memory wall/ gap growth and by GPGPU:

- **2008, Berkeley**: generalized into Roofline Performance Model. Williams, Waterman, Patterson. "**Roofline: an insightful visual performance model for multicore**"

From “Old HPC principle” to modern performance model

“Old” HPC principles:

1. “Balance” principle (e.g. Kung 1986) – hw and software parameters altogether

More research catalyzed by memory wall

Density, Intensity, Machine balance

\[ \text{Arithmetic Intensity} = \frac{\text{Total Flops computed}}{\text{Total Bytes transferred}} \]

\[ \text{Arithmetic Operational Intensity} = \frac{\text{Total Flops computed}}{\text{Total Bytes transferred between DRAM (MCDRAM) and LLC}} \]

\[ \text{Arithmetic Intensity} = \frac{\text{Total Intops+Flops computed}}{\text{Total Bytes transferred between CPU and “memory”}} \]
Each Roof (slope) gives peak CPU/Memory throughput of your PLATFORM (benchmarked).

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

- Interactive mapping to source and performance profile
- Synergy between Vector Advisor and Roofline: FMA example
- Customizable chart
Roofline model: Am I bound by VPU/CPU or by Memory?

What makes loops A, B, C different?
Advisor Roofline: under the hood

Roofline application profile:

Axis Y: \( \text{FLOP/S} = \frac{\#\text{FLOP} \text{ (mask aware)}}{\#\text{Seconds}} \)

Axis X: \( \text{AI} = \frac{\#\text{FLOP}}{\#\text{Bytes}} \)

---

**Seconds**

- User-mode **sampling**
- **Root access not needed**

**#FLOP**

Binary **Instrumentation**

Does not rely on CPU counters

**Bytes**

Binary **Instrumentation**

Counts operands size (not cachelines)

---

**Performance**

- Flops/seconds

---

**Roofs**

Microbenchmarks

Actual peak for the current configuration
Getting Roofline in Advisor

<table>
<thead>
<tr>
<th>FLOP/S = #FLOP/Seconds</th>
<th>Seconds</th>
<th>#FLOP Count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>- Mask Utilization - #Bytes</td>
</tr>
</tbody>
</table>

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

**Step 2: FLOPS**
- Precise, instrumentation based
- Physically count Num-Instructions
- Output: #FLOP, #Bytes
Mask Utilization and FLOPS profiler

- Long-waiting in HPC: accurate HW independent FLOPs measurement tool
- Not just count FLOPs. Has following additions:
  - (AVX-512 only) Mask-aware. Masked-Memory/Unmasked-Compute pattern aware
  - Unique capability to correlate FLOPs with performance data (obtained without instrumentation). Gives FLOPs/s.
- Lightweight instrumentation, PIN-based, benefits from “threadchecker tools” and more generally Advisor framework integration.
Why Mask Utilization Important?

for(i = 0; i <= MAX; i++)
    c[i] = a[i] + b[i];
Why Mask Utilization Important?

```plaintext
for(i = 0; i <= MAX; i++)
    if (cond(i))
        c[i] = a[i] + b[i];
```

3 elements suppressed

SIMD Utilization = 5/8

62.5%
AVX-512 Mask Registers

8 Mask registers of size 64-bits
- k1-k7 can be used for predication
  - k0 can be used as a destination or source for mask manipulation operations

4 different mask granularities. For instance, at 512b:
- Packed Integer Byte use mask bits [63:0]
  - VPADDB zmm1 {k1}, zmm2, zmm3
- Packed Integer Word use mask bits [31:0]
  - VPADDW zmm1 {k1}, zmm2, zmm3
- Packed IEEE FP32 and Integer Dword use mask bits [15:0]
  - VADDPS zmm1 {k1}, zmm2, zmm3
- Packed IEEE FP64 and Integer Qword use mask bits [7:0]
  - VADDPD zmm1 {k1}, zmm2, zmm3

<table>
<thead>
<tr>
<th>element size</th>
<th>Vector Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte</td>
<td>128 256 512</td>
</tr>
<tr>
<td>Word</td>
<td>16 32 64</td>
</tr>
<tr>
<td>Dw ord/SP</td>
<td>4 8 16</td>
</tr>
<tr>
<td>Qw ord/DP</td>
<td>2 4 8</td>
</tr>
</tbody>
</table>
Survey+FLOPs Report on AVX-512: FLOP/s, Bytes and AI, Masks and Efficiency
### General efficiency (FLOPS) vs. VPU-centric efficiency (Vector Efficiency)

<table>
<thead>
<tr>
<th>Function Call Sites and Loops</th>
<th>Total Time</th>
<th>Type</th>
<th>Vectorized Loops</th>
<th>Gain</th>
<th>VL</th>
<th>Vector Issues</th>
<th>FLOPS And AVX-512 Mask Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Vectorized (Rem..)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>AVX512</td>
<td>44%</td>
<td></td>
<td>3.53x 8</td>
<td>0.847</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Vectorized (Rem..)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>AVX512</td>
<td>26%</td>
<td></td>
<td>3.55x 8</td>
<td>1.763</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Vectorized (Rem..)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>AVX512</td>
<td>44%</td>
<td></td>
<td>1.84x 8</td>
<td>0.750</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Vectorized (Rem..)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>AVX512</td>
<td>38%</td>
<td></td>
<td>3.05x 8</td>
<td>0.724</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Vectorized (Rem..)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>AVX512</td>
<td>24%</td>
<td></td>
<td>1.94x 8</td>
<td>1.529</td>
</tr>
</tbody>
</table>

**High Vector Efficiency, Low FLOPS**

**Low Vector Efficiency, High FLOPS**
Interpreting Roofline Data: advanced ROI analysis.

**Final Limits**
(assuming perfect optimization)
Long-term ROI, optimization strategy

**Current Limits**
(what are my current bottlenecks)
Next step, optimization tactics

**Finally compute-bound**
Invest more into effective CPU/VPU (SIMD) optimization

**Finally memory-bound**
Invest more into effective cache utilization

Check your Advisor Survey and MAP results
Batch Mode Workflow Saves Time
Intel® Advisor - Vectorization Advisor

Turn On Batch Mode
Run several analyses in batch as a single run

Select analyses to run
Contains pre-selected criteria for advanced analyses

Click Collect all
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 © 2015, 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
## Configurations for Binomial Options SP

**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

Performance measured in Intel Labs by Intel employees

### Platform Hardware and Software Configuration

<table>
<thead>
<tr>
<th>Platform</th>
<th>Unscaled Core Frequency</th>
<th>Cores/Socket</th>
<th>Num Sockets</th>
<th>L1 Data Cache</th>
<th>L1 I Cache</th>
<th>L2 Cache</th>
<th>L3 Cache</th>
<th>Memory</th>
<th>Memory Frequency</th>
<th>Memory Access</th>
<th>H/W Prefetchers Enabled</th>
<th>HT Enabled</th>
<th>Turbo Enabled</th>
<th>C States Enabled</th>
<th>O/S Name</th>
<th>Operating System</th>
<th>Compiler Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® Xeon™ 5472 Processor</td>
<td>3.0 GHZ</td>
<td>4</td>
<td>2</td>
<td>32K</td>
<td>32K</td>
<td>12 MB</td>
<td>None</td>
<td>32 GB</td>
<td>800 MHZ</td>
<td>UMA</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Disable d</td>
<td>Fedora d</td>
<td>3.11.10-301.fc20</td>
<td>icc version 14.0.1</td>
</tr>
<tr>
<td>Intel® Xeon™ X5570 Processor</td>
<td>2.93 GHZ</td>
<td>4</td>
<td>2</td>
<td>32K</td>
<td>32K</td>
<td>256K</td>
<td>8 MB</td>
<td>48 GB</td>
<td>1333 MHZ</td>
<td>NUMA</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Disable d</td>
<td>Fedora d</td>
<td>3.11.10-301.fc20</td>
<td>icc version 14.0.1</td>
</tr>
<tr>
<td>Intel® Xeon™ X5680 Processor</td>
<td>3.33 GHZ</td>
<td>6</td>
<td>2</td>
<td>32K</td>
<td>32K</td>
<td>256K</td>
<td>12 MB</td>
<td>48 MB</td>
<td>1333 MHZ</td>
<td>NUMA</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Disable d</td>
<td>Fedora d</td>
<td>3.11.10-301.fc20</td>
<td>icc version 14.0.1</td>
</tr>
<tr>
<td>Intel® Xeon™ E5 2690 Processor</td>
<td>2.9 GHZ</td>
<td>8</td>
<td>2</td>
<td>32K</td>
<td>32K</td>
<td>256K</td>
<td>20 MB</td>
<td>64 GB</td>
<td>1600 MHZ</td>
<td>NUMA</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Disable d</td>
<td>Fedora d</td>
<td>3.11.10-301.fc20</td>
<td>icc version 14.0.1</td>
</tr>
<tr>
<td>Intel® Xeon™ E5 2697v2 Processor</td>
<td>2.7 GHZ</td>
<td>12</td>
<td>2</td>
<td>32K</td>
<td>32K</td>
<td>256K</td>
<td>30 MB</td>
<td>64 GB</td>
<td>1867 MHZ</td>
<td>NUMA</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Disable d</td>
<td>Fedora d</td>
<td>3.11.10-301.fc20</td>
<td>icc version 14.0.1</td>
</tr>
<tr>
<td>Codename Haswell</td>
<td>2.2 GHZ</td>
<td>14</td>
<td>2</td>
<td>32K</td>
<td>32K</td>
<td>256K</td>
<td>35 MB</td>
<td>64 GB</td>
<td>2133 MHZ</td>
<td>NUMA</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Disable d</td>
<td>Fedora d</td>
<td>3.13.5-202.fc20</td>
<td>icc version 14.0.1</td>
</tr>
</tbody>
</table>