Learning Objectives

• Recognize the differences of Intel® Xeon™ vs. Intel® Xeon Phi™

• Recognize Intel® Software Development Product support levels for Intel® Xeon Phi™

• Differentiate execution models

• Optimize for Intel® Xeon Phi™ Architecture

• Access additional resources for further information
Agenda

• Intel® Many Integrated Core (MIC) Architecture
  • Compiling/Optimizing

• Execution models
  • Offload
  • Native
  • Symmetric

• Tuning
Intel® Many Integrated Core (MIC) Architecture
Each Intel® Xeon Phi™ Coprocessor core is a fully functional multi-thread execution unit

>50 in-order cores
- Ring interconnect

64-bit addressing

Scalar unit based on Intel® Pentium® processor family
- Two pipelines
  - Dual issue with scalar instructions
- One-per-clock scalar pipeline throughput
  - 4 clock latency from issue to resolution

4 hardware threads per core
- Each thread issues instructions in turn
- Round-robin execution hides scalar unit latency
Each Intel® Xeon Phi™ Coprocessor core is a fully functional multi-thread execution unit

Optimized

- Single and Double Precision

Vector Processing Unit

- 512-bit SIMD Instructions (Not Intel® SSE, MMX™, or Intel® AVX)
- 32 512-bit wide vector registers
Data Types for Intel® MIC Architecture

- **16x floats**
- **8x doubles**
- **16x 32-bit integers**
- **8x 64-bit integers**
- **64x 8-bit bytes**
- **32x 16-bit shorts**
Parallel programming is the same on coprocessor and host

For Both Offload and Native Models

- Host Executable
- Coprocessor Executable
- PCIe
- Heterogeneous Compute

Offload Directives (Data Marshaling)
Offload Keywords (Shared Virtual Memory)

Tools
Intel® MKL
OpenMP
MPI
Intel® TBB
OpenCL
Intel® Cilk Plus
C++/Ftn

Tools
Intel® MKL
OpenMP
MPI
Intel® TBB
OpenCL
Intel® Cilk Plus
C++/Ftn

http://software.intel.com/mic-developer
Taming Intel® Xeon Phi™ with the Intel® Compilers

http://software.intel.com/mic-developer
Key to Peak Intel® Xeon Phi™ Performance

Parallelism

• Vectorization

• Scaling
SIMD Support

Vector Classes

• See mic/micvec.h and zmmintrin.h

• #include <immintrin.h>

Auto-Vectorization (align to 512 bits)

Explicit Vector Programming

• Intel® Cilk™ Plus extensions

• OpenMP 4.0 RC1 SIMD constructs
Auto-Vectorization

Similar on host and coprocessor

Enabled at `-O2`

Align data to 64 bytes

Many loops can be vectorized

- Masked vector instructions
- Gather/scatter instructions
- Fused multiply-add
How to Recognize Vectorization

Vectorization and Optimization Reports

-vec-report2 or –opt-report-phase hpo

Unmasked vector instructions

• No scalar instructions, uses masked vector inst.

Gather/Scatter instructions

Math library calls to libsvml
Vectorization Reports

By default, both host and target compilations may generate messages for the same loop, e.g.

```plaintext
icc -vec-report2 test_vec.c
```

*test_vec.c(10): (col. 1) remark: LOOP WAS VECTORIZED.*
*test_vec.c(10): (col. 1) remark: *MIC* LOOP WAS VECTORIZED.*

To get a vectorization report for the offload target compilation, but not for the host compilation:

```plaintext
icc -vec-report0 -offload-option,mic,compiler,"-vec-report2
test_vec.c"
```

*test_vec.c(10): (col. 1) remark: *MIC* LOOP WAS VECTORIZED.*
*test_vec.c(20): (col. 1) remark: *MIC* loop was not vectorized: existence of vector dependence.*
*test_vec.c(20): (col. 1) remark: *MIC* PARTIAL LOOP WAS VECTORIZED.*
Vectorization Requirements

See http://software.intel.com/en-us/articles/requirements-for-vectorizable-loops/ for full list

Innermost loop only (a few exceptions)

No function/subroutine calls (unless inline or vector)

No data-dependent exit conditions

Data should be contiguous
Vectorization Requirements (cont.)

See http://software.intel.com/en-us/articles/requirements-for-vectorizable-loops/ for full list

Pointers can be difficult

Use Directives/Pragmas to help
# Useful Loop Optimization Pragmas/Directives

<table>
<thead>
<tr>
<th>Pragma/Directive</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>IVDEP</td>
<td>Ignore vector dependency</td>
</tr>
<tr>
<td>LOOP COUNT</td>
<td>Advise typical iteration count(s)</td>
</tr>
<tr>
<td>UNROLL</td>
<td>Suggest loop unroll factor</td>
</tr>
<tr>
<td>DISTRIBUTE POINT</td>
<td>Advise where to split loop</td>
</tr>
<tr>
<td>VECTOR ...</td>
<td>Vectorization hints</td>
</tr>
<tr>
<td>Aligned</td>
<td>Assume data is aligned</td>
</tr>
<tr>
<td>Always</td>
<td>Override cost model</td>
</tr>
<tr>
<td>Nontemporal</td>
<td>Advise use of streaming stores</td>
</tr>
<tr>
<td>NOVECTOR</td>
<td>Do not vectorize</td>
</tr>
<tr>
<td>NOFUSION</td>
<td>Do not fuse loops</td>
</tr>
<tr>
<td>INLINE/FORCEINLINE</td>
<td>Invite/require function inlining</td>
</tr>
<tr>
<td>SIMD ASSERT</td>
<td>“Vectorize or die”</td>
</tr>
</tbody>
</table>
Data Alignment

Needed for vectorization

Should align on 64 byte boundaries for Intel® MIC Architecture (32 for AVX, 16 for SSE)

Different between C/C++ and Fortran

But the pragmas/directives are similar
Data Alignment in C/C++

Heap allocation aligned to n-byte boundary

```c
void* _mm_malloc(int size, int n);
int posix_memaligned(void **p, size_t n, size_t size);
```

Alignment for variable declarations

```c
__attribute__((aligned(n))) var_name;  //or
__declspec(align(n)) var_name;
```
Data Alignment in Fortran

Align array on n-byte boundary (n is power of 2)
• works for dynamic, automatic, and static arrays
• not in common blocks

```
!dir$ attributes align:n :: array
```

For a 2D array, choose column length to be a multiple of n, so consecutive columns have the same alignment (pad if necessary)

```
-align array64byte
```

compiler tries to align all array types
Alignment Pragmas/Directives

Ask compiler to vectorize, overriding cost model and assuming all array data access in loop is aligned for target processor. (Could cause fault if data not aligned.)

```
#pragma vector aligned
!dir$ vector aligned
```

Compiler may assume array is aligned to n-byte boundary

```
__assume_aligned(array,n);
!dir$ assume_aligned array:n [,array2:n2,...]
```
Explicit Vector Programming: Intel® Cilk™ Plus Array notation

void addit(double* a, double* b, int m, int n, int x)
{
    for (int i = m; i < m+n; i++) {
        a[i] = b[i] + a[i-x];
    }
}

loop was not vectorized: existence of vector dependence.

• Array notation asks the compiler to vectorize
• asserts this is safe (for example, x<0)
• Improves readability

void addit(double* a, double * b, int m, int n, int x)
{
    // I know x<0
    a[m:n] = b[m:n] + a[m-x:n];
}

LOOP WAS VECTORIZED.
Explicit Vector Programming: Pragma example

Using `#pragma simd` (C/C++) or `!dir$ SIMD` (Fortran) or `#pragma omp simd` (OpenMP* 4.0 RC1)

```c
void addit(double* a, double* b, int m, int n, int x)
{
    for (int i = m; i < m+n; i++)
    {
        a[i] = b[i] + a[i-x];
    }
}
```

loop was not vectorized: existence of vector dependence.

```c
void addit(double* a, double * b, int m, int n, int x)
{
    #pragma simd // I know x<0
    for (int i = m; i < m+n; i++)
    {
        a[i] = b[i] + a[i-x];
    }
}
```

SIMD LOOP WAS VECTORIZED.

• Use when you KNOW that a given loop is safe to vectorize
SIMD Summary

The importance of SIMD parallelism is increasing

• Moore’s law leads to wider vectors as well as more cores
• Don’t leave performance “on the table”
• Be ready to help the compiler to vectorize, if necessary
  – With compiler directives and hints
  – With explicit vector programming
  – Use Intel® VTune™ Amplifier XE to find the best places (hotspots) to focus your efforts
• No need to re-optimize vectorizable code for new processors
  – Typically a simple recompilation
Adding the Intel® Math Kernel Library to the Mix
Using Intel® Math Kernel Library on Intel® Xeon Phi™ Coprocessors

High level parallelism is critical in maximizing performance.

- BLAS (Level 3) and LAPACK with large problem size get the most benefit.
- Scaling beyond 100’s threads, vectorized, good data locality

Minimize data transfer overhead when offload.

- Offset data transfer overhead with enough computation.
- Exploit data persistence: CAO to help!

You can always run on the host if offloading does not offer better performance.
Usage Models on Intel® Xeon Phi™ Coprocessors

• Automatic Offload
  - No code changes required
  - Automatically uses both host and target
  - Transparent data transfer and execution management

• Compiler Assisted Offload
  - Explicit controls of data transfer and remote execution using compiler offload pragmas/directives
  - Can be used together with Automatic Offload

• Native Execution
  - Uses the coprocessors as independent nodes
  - Input data and binaries are copied to targets in advance
Automatic Offload (AO)

- Offloading is automatic and transparent.
- Can take advantage of multiple coprocessors.
- By default, Intel® MKL decides:
  - When to offload
  - Work division between host and targets
- Users enjoy host and target parallelism automatically.
- Users can still specify work division between host and target. (for BLAS only)
Compiler Assisted Offload (CAO)

- Offloading is explicitly controlled by compiler pragmas or directives.
- All Intel® MKL functions can be offloaded in CAO.
  - In comparison, only a subset of Intel® MKL is subject to AO.
- Can leverage the full potential of compiler’s offloading facility.
- More flexibility in data transfer and remote execution management.
  - A big advantage is data persistence: Reusing transferred data for multiple operations.
Native Execution

• Use the coprocessor as an independent compute node.
  - Programs can be built to run only on the coprocessor by using the --mic build option.

• Intel® MKL function calls inside an offloaded code region executes natively.
  - Better performance if input data is already available on the coprocessor, and output is not immediately needed on the host side.
Clustering, with the Intel® MPI Library
Using the Intel® MPI Library with Intel® Xeon Phi™

Three execution models supported

• Offload
• Native
• Symmetric

Mostly similar to standard MPI programming
Job Schedulers

Job schedulers do not currently support Intel® Xeon Phi™

Best approach depends on chosen execution model

Usually involves additional scripts to manipulate hostfile and/or environment variables
The Execution Models - Offload
Offload Programming

Copies a portion of data and instructions to coprocessor

Executes on coprocessor

Copies data back to host

Can execute highly parallelized/vectorized code sections on coprocessor, serial sections on host
How to Offload?

Pragmas

Explicit Offloading

Automatic Offloading (MKL)
Managing Offload

Target selection

• Offload target specified at compile-time or at run-time

Thread pinning

• Environment variables used to control thread pinning

• Can get complex, use –env option to help
Offload and Schedulers

If running one rank per host, no changes

If running multiple ranks per host, pass/set appropriate environment variables to control offload targets/pinning

Remember rank placement!
The Execution Models - Native
Native Programming

Cross-compile to Intel® MIC Architecture

Make binary and any dynamic libraries available on the coprocessor

Treat coprocessor as a unique system

No need to modify code to run on coprocessor

Good if code is already threaded
Considerations

Memory limitations

Recommended to run at most one rank per physical core, let threading utilize the rest

Avoid cores reserved for operating system
• 0, n-2, n-1, n
How to Use Native Model

Build for Intel® Xeon Phi™

```bash
mpiicc -mmic test.c -o hello.mic
```

Ensure binary is accessible on coprocessor

```bash
scp hello.mic node0-mic0:/home/user/test/hello
```

Launch application from host or coprocessor

```bash
$export I_MPI_MIC=enable
$cat mpi_hosts
node0-mic0
node1-mic0
$mpirun -n 2 -ppn 1 -hostfile mpi_hosts ./hello
Hello from rank 0 of 2 running on node0-mic0
Hello from rank 1 of 2 running on node1-mic0
```
Job Schedulers and Native Execution

Job schedulers will likely generate host file for hosts, not coprocessors

Need to modify host file for coprocessors

If not using NFS, might need to copy binaries and/or data files as part of job
The Execution Models - Symmetric
Symmetric Model

Ranks on both host and coprocessor

Heterogeneous runs

Best if serial code runs on host and parallel code runs on coprocessor

Can be run with no code modification
Symmetric Programming Model

Compile for both host and coprocessor

```
mpiicc test.c -o hello
mpiicc -mmic test.c -o hello.mic
```

Ensure binary is accessible on execution host

```
scp hello.mic node0-mic0:/home/user/test/hello
```

Launch the application on both

```
export I_MPI_MIC=enable
mpirun -hosts node0,node0-mic0 -n 2 -ppn 1 ./hello
```
Additional support for NFS shares

_I_MPI_MIC_POSTFIX_ – Adds the text to the end of the executable if execution host is a coprocessor

_I_MPI_MIC_PREFIX_ – Add the text to the end of the executable if execution host is a coprocessor

These only work if using \_hostfile or -machinefile
Example of NFS support

Begin in a shared folder with same path on host and coprocessor

```bash
$mpiicc test.c -o hello
$mpiicc -mmic test.c -o hello.mic
$export I_MPI_MIC=1
$export I_MPI_MIC_POSTFIX=.mic
$cat hosts
node0
node0-mic0
$mpirun -n 2 -ppn 1 -hostfile hosts ./hello
Hello from rank 0 of 2 running on node0
Hello from rank 1 of 2 running on node0-mic0
```
Argument Sets and Configuration Files

Use to run heterogeneous jobs

Global options applied to all sets

Local options only applied to current set

Pass different environment variables to each set

Run different executables in each set
Configuration Files

Store sets in a file, one per line

```bash
cat <configfile>
-genv I_MPI_DEBUG 2
-n 1 -host node000 ./hello.host
-n 1 -host node000-mic0 ./hello.mic
mpirun -configfile <configfile>
```
Argument sets

Separate with :

```
mpirun -n 1 -host node000 /home/user/test/printenv.host : -n 1 -host
node000-mic0 -env TESTVAR 25 /home/user/test/printenv.mic

Rank 0 of 1 running on node000, TESTVAR is not defined
Rank 1 of 1 running on node000-mic0, TESTVAR=25
```
Job Schedulers and Symmetric Execution

Similar to native execution, host file will likely only include hosts, no coprocessors.

Again, need to modify host file, but consider order of hosts.

Consider building a configuration file from the host file.
Tuning – Intel® Xeon Phi™ Style
Where to Begin?

Start on the host

• Inspector XE currently only supports Intel® Xeon™

• Some VTune™ Amplifier XE reports are not supported on Intel® Xeon Phi™

Errors (races, deadlocks, etc.) will be the same on both architectures

Disable offloading in initial collections

Remember load balancing
Collect Xeon Phi™ Events

VTune™ Amplifier XE settings:

- Application: ssh
- Parameters: node0-mic0 <application and args>
- Working Directory: Usually won’t matter (ssh)
- Remember to set search directories

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

Application: ssh
Application parameters: mic0 /tmp/diffusionMIC tiled
Working directory: /home/michome/rredd/projects/MICtest/perfMIC

http://software.intel.com/mic-developer

Copyright© 2013, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.
Coproprocessor Collections Have Separate Analysis Types
Performance Analysis

Only “Lightweight Hotspots” for Intel® Xeon Phi™

• Uses Event-Based Sampling
• Uses the Performance Monitoring Unit
• No instrumentation (“Locks & Waits”, “Concurrency”, etc.)
• More to come!

Other analysis types need to be configured

• Create a copy of Lightweight Hotspots
• Add desired counters
• Add useful name and documentation
• Multi-event collections (over 2) can multiplex or use multiple runs

http://software.intel.com/mic-developer
Configuring a User-Defined Analysis

- Analysis name: My Shiny New KNC Analysis
- Description: This is the greatest analysis I ever created!

Available events:
- BANK_CONFLICTS
- BRANCHES
- BRANCHES_MISPREDICTED
- CODE_CACHE_MISS
- CODE_READ
- DATA_CACHE_LINES_WRITTEN_BACK
- DATA_PAGE_WALK
- DATA_READ
- DATA_READ_MISS
- DATA_READ_MISS_OR_WRITE_MISS
- DATA_READ_OR_WRITE
- DATA_WRITE
- DATA_WRITE_MISS
- EXFC_STAGE_CYCLES

Number of memory read accesses that miss the internal

List of MIC cards (e.g. 0.1.2.3): 0

Command line name: shiny-new-analysis

http://software.intel.com/mic-developer

Copyright © 2013, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.
Hardware Counters and Events

2 counters in core, most are thread specific

4 outside the core (uncore) that get no thread or core details

See PMU documentation for full list
Collecting Hardware Performance Data

Invoke from VTune™ Amplifier XE

If collecting more than 2 core events:
• Select multi-run for more precise results OR
• Default multiplexed collection, all in one run

Uncore events are limited to 4 at a time in a single run

Uncore event sampling needs a source of PMU interrupts (programming cores to CPU_CLK_UNHALTED)
Efficiency Metrics

CPI – Cycles Per Instruction
- Lower is better
- Minimum depends on number of hardware threads used
- Investigate if >4.0 per thread or >1.0 per core (or if increasing)

Compute to Data Access Ratio
- Two measures of computational density
- Investigate if
  - L1 Compute to Data Access Ratio < Vectorization Intensity
  - L2 Compute to Data Access Ratio < 100x L1 Compute to Data Access Ratio
Problem Areas

Vectorization Intensity
• Investigate if <8 for double precision, <16 for single

L1 Hit Rate
• Investigate if <95%

Estimated Latency Impact
• Investigate if >145
Problem Areas

TLB Usage
• Investigate if
  - L1 TLB Miss Ratio > 1%
  - L2 TLB Miss Ratio > 0.1%
  - L1 TLB Misses Per L2 TLB Miss > 100x

Memory Bandwidth
• Investigate if < 80 GB/s
Event collections on the coprocessor can generate volumes of data
dgemm: on 60+ cores

Tip: Use cpu-mask to reduce data set, while maintaining the same accuracy.
Resources

http://software.intel.com/mic-developer


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 © , Intel Corporation. All rights reserved. Intel, the Intel logo, Xeon, Xeon Phi, Core,
VTune, and Cilk 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
Backup Slides
Vector Elemental Function

Compiler generates vector version of a scalar function that can be called from a vectorized loop:

```c
__attribute__((vector(uniform(y, xp, yp))))
float func(float x, float y, float xp, float yp) {
    float denom = (x-xp)*(x-xp) + (y-yp)*(y-yp);
    denom = 1./sqrtf(denom);
    return denom;
}
```

These clauses are required for correctness, just like for OpenMP.*

```c
#pragma simd  private(x)  reduction(+:sumx)
for (i=1; i<nx; i++)  {
    x = x0 + (float)i * h;
    sumx = sumx + func(x, y ,xp, yp);
enddo
```

SIMD LOOP WAS VECTORIZED.

y, xp and yp are constant, x can be a vector

SIMD LOOP WAS VECTORIZED.
Clauses for Vector Functions

__attributes__((vector))  (Intel)
#pragma omp declare simd  (OpenMP* 4.0 RC1)

Available clauses  (both OpenMP and Intel versions)
• LINEAR  (additional induction variables)
• UNIFORM  (arguments that are loop constants)
• PROCESSOR  (Intel)
• VECTORLENGTH  (Intel)
• MASK / NOMASK  (Intel)
• INBRANCH / NOTINBRANCH  (OpenMP 4.0 RC1)
• SIMDLEN  (OpenMP 4.0 RC1)
• ALIGNED  (OpenMP 4.0 RC1)
Efficiency Metric: CPI

- Cycles per Instruction
- Can be calculated per hardware thread or per core
- Is a ratio! So varies widely across applications and should be used carefully.

<table>
<thead>
<tr>
<th>Number of Hardware Threads / Core</th>
<th>Minimum (Best) Theoretical CPI per Core</th>
<th>Minimum (Best) Theoretical CPI per Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>0.5</td>
<td>1.0</td>
</tr>
<tr>
<td>3</td>
<td>0.5</td>
<td>1.5</td>
</tr>
<tr>
<td>4</td>
<td>0.5</td>
<td>2.0</td>
</tr>
</tbody>
</table>
**Efficiency Metric: CPI**

- Measures how latency affects your application’s execution
- Look at how optimizations applied to your application affect CPI
- Address high CPIs through any optimizations that aim to reduce latency

<table>
<thead>
<tr>
<th>Metric</th>
<th>Formula</th>
<th>Investigate if</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPI Per Thread</td>
<td>CPU_CLK_UNHALTED/INSTRUCTIONS_EXECUTED</td>
<td>&gt; 4.0, or increasing</td>
</tr>
<tr>
<td>CPI Per Core</td>
<td>(CPI per Thread) / Number of hardware threads used</td>
<td>&gt; 1.0, or increasing</td>
</tr>
</tbody>
</table>

Efficiency Metric: Compute to Data Access Ratio

• Measures an application’s computational density, and its suitability for Intel® MIC Architecture

<table>
<thead>
<tr>
<th>Metric</th>
<th>Formula</th>
<th>Investigate if</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 Compute to Data Access Ratio</td>
<td>VPU_ELEMENTS_ACTIVE / DATA_READ_OR_WRITE</td>
<td>&lt; Vectorization Intensity</td>
</tr>
<tr>
<td>L2 Compute to Data Access Ratio</td>
<td>VPU_ELEMENTS_ACTIVE / DATA_READ_MISS_OR_WRITE_MISS</td>
<td>&lt; 100x L1 Compute to Data Access Ratio</td>
</tr>
</tbody>
</table>

• Increase computational density through vectorization and reducing data access (see cache issues, also, DATA ALIGNMENT!)
Problem Area: VPU Usage

• Indicates whether an application is vectorized successfully and efficiently

<table>
<thead>
<tr>
<th>Metric</th>
<th>Formula</th>
<th>Investigate if</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vectorization Intensity</td>
<td>VPU_ELEMENTS_ACTIVE / VPU_INSTRUCTIONS_EXECUTED</td>
<td>&lt;8 (DP), &lt;16(SP)</td>
</tr>
</tbody>
</table>

• Tuning Suggestions:
  - Use the Compiler vectorization report!
  - For data dependencies preventing vectorization, try using Intel® Cilk™ Plus #pragma SIMD (if safe!)
  - Align data and tell the Compiler!
  - Re-structure code if possible: Array notations, AOS->SOA
Problem Area: L1 Cache Usage

- Significantly affects data access latency and therefore application performance

<table>
<thead>
<tr>
<th>Metric</th>
<th>Formula</th>
<th>Investigate if</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 Misses</td>
<td>DATA_READ_MISS_OR_WRITE_MISS + L1_DATA_HIT_INFLIGHT_PF1</td>
<td></td>
</tr>
<tr>
<td>L1 Hit Rate</td>
<td>(DATA_READ_OR_WRITE - L1 Misses) / DATA_READ_OR_WRITE</td>
<td>&lt; 95%</td>
</tr>
</tbody>
</table>

- Tuning Suggestions:
  - Software prefetching
  - Tile/block data access for cache size
  - Use streaming stores
  - If using 4K access stride, may be experiencing conflict misses
  - Examine Compiler prefetching (Compiler-generated L1 prefetches should not miss)
Problem Area: Data Access Latency

- Significantly affects application performance

<table>
<thead>
<tr>
<th>Metric</th>
<th>Formula</th>
<th>Investigate if</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estimated Latency Impact</td>
<td>(CPU_CLK_UNHALTED - EXEC_STAGE_CYCLES - DATA_READ_OR_WRITE) / DATA_READ_OR_WRITE_MISS</td>
<td>&gt;145</td>
</tr>
</tbody>
</table>

- Tuning Suggestions:
  - Software prefetching
  - Tile/block data access for cache size
  - Use streaming stores
  - Check cache locality – turn off prefetching and use CACHE_FILL events - reduce sharing if needed/possible
  - If using 64K access stride, may be experiencing conflict misses

http://software.intel.com/mic-developer
Problem Area: TLB Usage

- Also affects data access latency and therefore application performance

<table>
<thead>
<tr>
<th>Metric</th>
<th>Formula</th>
<th>Investigate if</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 TLB miss ratio</td>
<td>DATA_PAGE_WALK/DATA_READ_OR_WRITE</td>
<td>&gt; 1%</td>
</tr>
<tr>
<td>L2 TLB miss ratio</td>
<td>LONG_DATA_PAGE_WALK / DATA_READ_OR_WRITE</td>
<td>&gt; .1%</td>
</tr>
<tr>
<td>L1 TLB misses per L2 TLB miss</td>
<td>DATA_PAGE_WALK / LONG_DATA_PAGE_WALK</td>
<td>&gt; 100x</td>
</tr>
</tbody>
</table>

- Tuning Suggestions:
  - Improve cache usage & data access latency
  - If L1 TLB miss/L2 TLB miss is high, try using large pages
  - For loops with multiple streams, try splitting into multiple loops
  - If data access stride is a large power of 2, consider padding between arrays by one 4 KB page

http://software.intel.com/mic-developer
Problem Area: Memory Bandwidth

- Can increase data latency in the system or become a performance bottleneck

<table>
<thead>
<tr>
<th>Metric</th>
<th>Formula</th>
<th>Investigate if</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Bandwidth</td>
<td>(UNC_F_CH0_NORMAL_READ + UNC_F_CH0_NORMAL_WRITE + UNC_F_CH1_NORMAL_READ + UNC_F_CH1_NORMAL_WRITE) x 64/time</td>
<td>&lt; 80GB/sec (practical peak 140GB/sec) (with 8 memory controllers)</td>
</tr>
</tbody>
</table>

- Tuning Suggestions:
  - Improve locality in caches
  - Use streaming stores
  - Improve software prefetching