The Heterogeneous Offload Model for Intel® Many Integrated Core Architecture

**THEME:** Programming and Optimizing for Intel® MIC Architecture is based on the same, familiar tools and methods that you already use for multi-core systems

Developer Products Division
Executive Summary

• The programming model and compiler make it easy to develop or port code to run on a CPU + Intel® MIC system
• Full integration into both C/C++ and Fortran
• Enables use of our optimizing compilers on both CPU and Intel MIC
  • Vectorization
  • Parallel programming with TBB, Intel® Cilk™ Plus, OpenMP®, MPI
• Enables *co-operative* processing between CPU and Intel MIC device
  • Use both simultaneously for parallel processing
  • Use CPU for serial code, Intel MIC for data-parallel code

Using MIC is a simple extension of CPU programming
Agenda

• Programming Models
  • Native (non-offload) usage

• Language Extensions for Offload
  • (pragmas)
  • (keywords)
Programming Models

An Intel® MIC device(s) is accessed via the host system, but may be programmed as a coprocessor(s) or as an autonomous processor.

The appropriate model may depend on application and context.

**Heterogeneous (Offload)**
- Better serial processing
- More memory
- Better file access
- Makes fuller use of available resources

**Native**
- Simpler programming model
  - Easier or no code porting
- More constraints
- Maybe quicker route for initial testing of key kernels
Native model may be appropriate if Application:

- Contains very little serial processing
- Has a modest memory footprint
- Has a very complex code structure and/or does not have well-identified hot kernels than can be offloaded without substantial data transfer overhead
- Does not perform extensive I/O

Data parallelism, use of parallel algorithms and application scalability are criteria for targeting Intel® MIC Architecture, but not for distinguishing between native or offload modes.
NFS File Mounts

• Useful if application handles input/output of large data sets
  • There is no permanent file system on the card
• Super user permissions are required for configuration
• Use the typical method for mounting shares
  • Append /etc/exports on the host with
    – /mydir 192.168.1.100/24(rw,no_root_squash)
  • Append /etc/hosts.allow on the host with - ALL: 192.168.1.100
  • Restart the network file system on the host by running $ nfs start
  • Start exportfs on the host $ /usr/sbin/exportfs -a –v
  • Append /etc/fstab on the coprocessor with
    – 192.168.1.99:/mydir /mydir nfs rsize=8192,wsize=8192,nolock,intr 0 0
  • Create the /mydir directory on the coprocessor and run $ mount –a
  • Verify the mount is available on the coprocessor with $ df –h
• More information at https://mic-dev.intel.com “Building a Native App...”
Heterogeneous (Offload) Model

Parallel programming is the same on Intel® MIC and CPU
Heterogeneous Programming Model

- Programmer designates code sections to run on Intel® MIC target
  - No further programming / API usage is needed
  - Setup/teardown, data transfer, synchronization, are managed automatically by compiler and runtime
- Offload is optional
  - If Intel MIC device is missing, program may run entirely on CPU
    - If Intel MIC-specific code, may need alternate code path
    - There is an option to enforce failure if Intel MIC device is unavailable
Heterogeneous Memory Model
(Non-shared Memory)

• CPU and Intel® MIC device do not share a common memory

• Two techniques are used to maintain program semantics with/without offload
  • Emulate shared data by copying back and forth at point of offload
    – Good for large blocks of contiguous data
  • Maintain coherence in a range of virtual addresses on CPU and Intel MIC device, automatically in software
    – Good for complex data structures
Offload Compile/Run Overview

### CPU Program
```c
f()
{
    #pragma offload target(mic)
    a = b + g();
}
```
```c
__attribute__((target(mic))) g()
{
}
```
```c
h()
{
}
```

### Contents of MIC Program
```c
f_part1()
{
    a = b + g();
}
```
```c
__attribute__((target(mic))) g()
{
}
```
```c
__attribute__((target(mic))) g()
{
}
```

### Execution
- At first offload, if Intel® MIC device is installed and available, MIC program is loaded.
- At each offload, if Intel MIC device is present, statement is run on device, else statement runs on CPU.
- At program termination, Intel MIC program is unloaded.
Language Extensions for Offload (C/C++ pragmas)

- Offload **pragma** for data marshalling
  - `#pragma offload` in C/C++
- Offloads the following OpenMP block or Intel® Cilk™ Plus construct or function call or compound statement
- All functions that will execute on the card must be declared as targeted for offload (so compiled for host and MIC device)
  - In both the callee (required) and the caller (recommended)
    - `__attribute__((target (MIC)))` or `__declspec(target(mic))`
    - For many such declarations, use
      `#pragma offload_attribute (push,target(mic)) | (pop)`
- Offloaded data must be scalars, arrays, bit-wise copyable structs
  - Excludes all but simplest C++ classes
  - All data types can be used within the target code
  - Data copy is explicit
Language Extensions for Offload (Fortran syntax)

- Offload **Directives** for data marshalling
  - `!dir$ offload`
  - `!dir$ omp offload` (historical; redundant)
    Offloads the following OpenMP block or subroutine/function call
    
    ```fortran
    RESULT = FUNC(A,B) ! but not RESULT = SCALE * FUNC(A,B)
    ```
  - `!dir$ offload begin ...`
    `!dir$ end offload` to offload other block of code

- All procedures that will execute on the card must be declared as targeted for offload (so compiled for host and MIC device)
  - In both the callee (required) and the caller (recommended)
  - `!DIR$ ATTRIBUTES TARGET:MIC :: procedure name`

- Offloaded data must be scalars, arrays or bit-wise copyable derived types (no embedded pointers or allocatable arrays)
  - Excludes most Fortran 2003 object-oriented constructs
  - All data types can be used within the target code
  - Data copy is explicit
## Offload Directives (contd.)

Variables restricted to scalars, bwc structs, arrays and pointers to scalars/structs/arrays

<table>
<thead>
<tr>
<th>Clauses / Modifiers</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target specification</td>
<td><code>target ( name [:num] )</code></td>
<td>Where to run construct</td>
</tr>
<tr>
<td>Inputs</td>
<td><code>in (var-list modifiers_opt)</code></td>
<td>Copy CPU to target</td>
</tr>
<tr>
<td>Outputs</td>
<td><code>out (var-list modifiers_opt)</code></td>
<td>Copy target to CPU</td>
</tr>
<tr>
<td>Inputs &amp; outputs</td>
<td><code>inout (var-list modifiers_opt)</code></td>
<td>Copy both ways</td>
</tr>
<tr>
<td>Non-copied data</td>
<td><code>nocopy (var-list modifiers_opt)</code></td>
<td>Data is local to target</td>
</tr>
</tbody>
</table>

### Modifiers

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specify pointer length</td>
<td><code>length (element-count-expr)</code></td>
<td>Copy that many pointer elements</td>
</tr>
<tr>
<td>Control pointer memory allocation</td>
<td><code>alloc_if ( condition )</code></td>
<td>Allocate/free new block of memory for pointer if condition is TRUE</td>
</tr>
<tr>
<td></td>
<td><code>free_if ( condition )</code></td>
<td></td>
</tr>
<tr>
<td>Alignment for pointer memory allocation</td>
<td><code>align ( expression )</code></td>
<td>Specify minimum data alignment</td>
</tr>
</tbody>
</table>
Offload Directive Examples: OpenMP, Intel® Cilk™ Plus

**C/C++ OpenMP**

```c
#pragma offload target (mic)
#pragma omp parallel for reduction(+:pi)
for (i=0; i<count; i++) {
    float t = (float)((i+0.5)/count);
    pi += 4.0/(1.0+t*t);
}
pi /= count;
```

**Fortran OpenMP**

```fortran
!dir$ offload target(mic)
!$omp parallel do
    do i=1,10
        A(i) = B(i) * C(i)
    enddo
```

**C/C++ Cilk**

```c
#pragma offload target(mic)
_Cilk_for (int i=0; i<count; i++)
{
    a[i] = b[i] * c + d;
}
```

```fortran
!dir$ offload target(mic)
!$omp parallel do
    do i=1,10
        A(i) = B(i) * C(i)
    enddo
```
Rules For Data Transfer

• Automatically detected and transferred as INOUT
  • Named arrays in lexical scope
  • Scalars in lexical scope

• User can override automatic transfer with explicit IN/OUT/INOUT clauses

• Not automatically transferred
  • Memory pointed to by pointers
    – This also needs a length parameter
  • Global variables used in functions called within the offloaded construct
When are alloc_if and free_if clauses needed?

• Needed for pointers or allocatable arrays
  • Default is to always allocate and free memory for pointers that are within the lexical scope of the offload, not otherwise
    – use free_if(0) if you want to memory and data to persist until next offload
    – Need alloc_if(1) for globals that are not lexically visible and are NOCOPY
    – Or use alloc_if(expression) to make dependent on runtime data

• Not needed for statically allocated data
  • These are statically allocated and persistent on the MIC device, even for arrays that are not lexically visible or have a NOCOPY clause.

• Syntax:
  • #pragma offload nocopy(myptr:length(n):alloc_if(expression))
  • !DIR$ OFFLOAD IN(FPTR:length(n):free_if(.false.))
    (or a logical expression in Fortran)
Data Transfer: non-shared memory

Buffer

input1
input2
output

CPU

Transfer to MIC

 MIC
input1
input2
output

PCIe
Memory Regions

A: COI buffer (4KB pages). All offloaded data (except B below) arrive here, then get re-copied to region C. Buffer size is dynamic (from 13.0 beta)

B: 2MB buffers/pages. C/C++ & Fortran pointer variables with size > **MIC_USE_2MB_BUFFERS** (off if unset) grows and shrinks dynamically. **Data not recopied**

C: dynamically allocated memory (malloc, ALLOCATE, ...) grows and shrinks dynamically with malloc, free,..

D: statically allocated data (.text, .data, .bss segments) size program-dependent but fixed.
Multi-card Support

```c
#pragma offload target(mic [ :<expr> ] ) ...

card # = <expr> % number_of_devices
```

- Code must run on card #, aborts if not available (counts from 0)
- If -1, runtime chooses card, aborts if not available
- If not present, runtime chooses card or runs on host if none available
- APIs: 
  ```c
  #include offload.h (C/C++);   USE MIC_LIB (Fortran)
  int _Offload_number_of_devices() (C/C++)
  result = OFFLOAD_NUMBER_OF_DEVICES() (Fortran)
  ```
  - Returns # of Intel® MIC devices installed, or 0 if none
  ```c
  int _Offload_get_device_number() (C/C++)
  result = OFFLOAD_GET_DEVICE_NUMBER() (Fortran)
  ```
  - Returns card number where executed, counting from 0, (or -1 for CPU)
  - Can use to share work explicitly by card number
Asynchronous offload

• New synchronization clauses  SIGNAL(&x) and  WAIT(&x)
  • Argument is a unique address (usually of the data being transferred)

• Data:
  • #pragma offload_transfer target(mic:n) IN(....) signal(&s1)
    – Standalone data offload
  • #pragma offload_wait target(mic:n) wait(&s1)
    – Standalone synchronization, host waits for transfer completion (blocking)

• Computation:
  • #pragma offload target(mic:n) wait(&s1) signal(&s2)
    – Offload computation when data transfer has completed
    – Computation on host then continues in parallel
  • #pragma offload_wait target(mic:n) wait(&s2)
    – Host waits for signal that offload computation completed

• There is also a non-blocking API to test signal value
Language Extensions for Offload (LEO)

• **Offload Keywords**
  
  • C/C++ keywords `_Cilk_shared`, `_Cilk_offload` and `_Cilk_offload_to()`
    
    – and eventually `_Cilk_borrow`.
    
    All data types are supported
  
  • Implemented using “virtual shared memory” technology
    
    – Data copy is implicit
    
    – Further reduces the burden on the programmer
  
  • Enables pointer-based data, C structs, C++ classes and locks, enabling a wider class of apps to be run on Intel® MIC architecture
    
    – Pointers are no longer an issue when they point to shared data
  
  • Can combine keywords with pragmas in same application
MYO Virtual Shared Memory Model

- Virtual shared memory across processing units
  - Seamless sharing of complex pointer-containing data structures
  - Eliminates marshalling and data management
  - Supports different memory consistency models
  - Simple language extensions to C/C++
Language Extensions for Offload (keywords)
Virtual Shared-Memory Model

- Compiler allocates data exchanged between CPU and Intel® MIC device in MYO shared memory
  - Programmer must either declare data as \_Cilk\_shared
  - Or use MYO malloc/free routines, (see include/offload.h )
  - Functions that will run on target must be declared as \_Cilk\_shared (equivalent to _attributes((target(MIC))))

- Compiler supports Release consistency of shared memory between CPU and Intel MIC device
  - Acquire/release of memory done automatically around offloads
  - “Lazy” updates of MYO memory on MIC as needed, but in chunks of a page or more
  - A copy is kept
  - Only modified data are transferred back to CPU at end of offload
## Keyword Language Extensions

<table>
<thead>
<tr>
<th></th>
<th>Shared memory allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Shared malloc</strong></td>
<td><code>ptr = (_Shared int *) _Offload_shared_malloc(4);</code></td>
</tr>
<tr>
<td><strong>Shared aligned malloc</strong></td>
<td><code>ptr = (_Shared int *) _Offload_shared_aligned_malloc(4);</code></td>
</tr>
<tr>
<td><strong>Shared free</strong></td>
<td><code>_Offload_shared_free(ptr);</code></td>
</tr>
<tr>
<td><strong>Shared aligned free</strong></td>
<td><code>_Offload_shared_aligned_free(ptr);</code></td>
</tr>
</tbody>
</table>
| **Shared class malloc**      | `head = new(
   _Offload_shared_aligned_malloc (sizeof(CELO), 8)
) _Cilk_shared CELO;` |
# Keyword \_Cilk\_Shared for Data and Functions

<table>
<thead>
<tr>
<th>What</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function available on both sides</td>
<td><code>int _Cilk_shared f(int x) { return x+1 }</code></td>
<td>The function may execute on either side</td>
</tr>
<tr>
<td>Global</td>
<td><code>_Cilk_shared int x = 0;</code></td>
<td>Visible on both sides</td>
</tr>
<tr>
<td>File/Function static</td>
<td><code>static _Cilk_shared int x;</code></td>
<td>Visible on both sides, only to code within the file/function</td>
</tr>
<tr>
<td>Class</td>
<td><code>class _Cilk_shared x {...}</code></td>
<td>Static members are visible on both sides; methods and operators are available on both sides</td>
</tr>
<tr>
<td>Pointer to shared data</td>
<td><code>int _Cilk_shared *p;</code></td>
<td>P is allocated locally, can point to shared data</td>
</tr>
<tr>
<td>A shared pointer</td>
<td><code>int *_Cilk_shared p;</code></td>
<td>P is shared. (It should only point at shared data.)</td>
</tr>
<tr>
<td>Entire blocks of code</td>
<td>`#pragma offload_attribute (push</td>
<td>pop, _Cilk_shared)`</td>
</tr>
</tbody>
</table>
Limitations on data that may be _Cilk_shared

• The _Cilk_shared keyword may not be applied to:
  • Local variables within a function
  • Static variables (may not be unique)
  • A field of a structure (the whole structure has to be shared)
  • A pointer to non_shared data

(Thus a shared pointer must always point to shared data;
A pointer to non-shared data may not be assigned to a pointer to shared data)

• In general, if a class is marked _Cilk_shared, then all operators and all function members are shared
  • Data members are shared only if the instance of the class is also shared
## Offloading using `_Cilk_offload`

<table>
<thead>
<tr>
<th>Feature</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offloading a function call</td>
<td><code>x = _Cilk_offload func(y);</code></td>
<td>Func executes on Intel® MIC device</td>
</tr>
<tr>
<td>Offloading asynchronously</td>
<td><code>x = _Cilk_spawn _Cilk_offload func(y);</code></td>
<td>Non-blocking offload</td>
</tr>
<tr>
<td>Callback from Intel MIC to CPU</td>
<td><code>x = _Cilk_borrow func(y);</code></td>
<td>The caller on Intel MIC device, the callee on the CPU</td>
</tr>
<tr>
<td></td>
<td>(planned, not yet available)</td>
<td></td>
</tr>
<tr>
<td>Offload a parallel for-loop</td>
<td><code>_Cilk_offload _Cilk_for</code></td>
<td>Loop executes in parallel on Intel MIC device. The loop is implicitly outlined as a function call.</td>
</tr>
<tr>
<td></td>
<td><code>(i = 0; i &lt; N; i++)</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>{</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>a[i] = b[i] + c[i];</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>}</code></td>
<td></td>
</tr>
<tr>
<td>Offload to a specific MIC device</td>
<td><code>x = _Cilk_offload_to(num) func(y);</code></td>
<td>Func executes on device number <code>num</code></td>
</tr>
</tbody>
</table>
Compiler Usage and Output Files (KNF)

\texttt{icc -c -offload-build a\.c} creates \texttt{a\.o aMIC\.o}
\texttt{aMIC\.o} is generated behind the scenes

\texttt{xiar … -qoffload-build a\.o b\.o} creates \texttt{lib\.a libMIC\.a}
\texttt{libMIC\.a} is generated behind the scenes

\texttt{xild … -qoffload-build lib\.a c\.o} creates \texttt{a\.out aMIC\.out}

\texttt{icc … -offload-build a\.c b\.c} creates \texttt{a\.out}
\texttt{aMIC\.out} is embedded within \texttt{a\.out} (formerly separate)

\texttt{ifort} is the same as \texttt{icc}

“-offload-build” is not be needed for the beta compiler for KNC, offload constructs are automatically detected and trigger an additional target compilation and link
(can prevent by \texttt{–no-offload})

\texttt{-offload-option,<tgt>,<tool>,”<option list>”} to set offload compiler, linker, etc. options
Compiler Usage  (KNF)

• Most compiler options for the host carry over to the offload
  • Including preprocessor macro definitions
  • Some parts of “-openmp” are on by default for offload builds, but not for native

• These defaults may be overridden, e.g.
  
  icc –vec-report0 –offload-build –offload-copts=“-vec-report2” a.c
  (generates vectorization report for the offload without generating the corresponding report for the host)

  ( icc -offload-option,mic,compiler,”-vec-report2” a.c   for beta compiler )
  -offload-ldopts=“...” to specify linker options, -offload-aropts=“...” for ar
  -opt-report-phase=offload   will report which variables are offloaded

• -offload-attribute-target=mic    flag all globals for offload
Compiler Usage and Output Files (KNC)

- Offload compiler is invoked automatically if the host compiler detects any offload language extensions (keywords or pragmas) in the source
  - This can be prevented by `--no-offload`

- Offload options set by `--offload-option,mic,<tool>,"<option list>"` where `<tool>` is compiler, as or ld.
  - e.g. `--offload-option,mic,compiler,"-vec-report2"

```
icc -c a.c creates a.o aMIC.o     (aMIC.o is generated behind the scenes)
xiar ... a.o b.o creates lib.a and libMIC.a
        (libMIC.a is generated behind the scenes)
xiild ... lib.a c.o creates a.out
icc ... a.c b.c creates a.out
```
- Intel® MIC executable is embedded inside a.out

ifort behaves the same as icc
Compiler Usage (KNC)

- Most compiler options for the host carry over to the offload
  - Including preprocessor macro definitions
  - `-openmp` is not enabled by default, but needed for OpenMP offloads
    - Some parts of the OpenMP runtime may be linked by default for offload builds, but not for native
- These defaults may be overridden, e.g.
  
  `icc -vec-report0 -offload-option,mic,compiler,"-vec-report2" a.c`
  (generates vectorization report for the offload without generating the corresponding report for the host)

- `-opt-report-phase=offload` will report which variables are offloaded

- `-offload-attribute-target=mic` flag all globals for offload
Preprocessor Macros

• __INTEL_OFFLOAD
  • Set automatically unless disabled by –no-offload (or –mmic)
  • Set for the host compilation as well as for target (Intel® MIC) compilation
  • Use to protect code on the host that is specific for offload
    e.g. omp_num_set_threads_target() family of APIs
    but must remember to set –no-offload for non-MIC builds

• __MIC__
  • NOT set for host compilation in an offload build
  • Set automatically for target (MIC) compilation in offload build
  • Also set automatically when building native MIC application
  • Use to protect code that is compiled & executed only on MIC device
    e.g. _mm512 intrinsics
## Intel® MIC-specific environment variables

<table>
<thead>
<tr>
<th>MIC Env Variable</th>
<th>Default Type</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIC_ENV_PREFIX</td>
<td>string</td>
<td>none</td>
<td>Environment variables (except those below) are stripped of this prefix and sent to MIC.</td>
</tr>
<tr>
<td>MIC_&lt;card #&gt;_ENV</td>
<td>string</td>
<td>none</td>
<td>List of environment variables to set on card #</td>
</tr>
<tr>
<td>MIC_LD_LIBRARY_PATH</td>
<td>string</td>
<td>set by compiler vars script</td>
<td>Search paths for shared libraries on MIC</td>
</tr>
<tr>
<td>MIC_USE_2MB_BUFFERS</td>
<td>integerB/K/M/G/T</td>
<td>none</td>
<td>Use 2M pages for (size &gt; MIC_USE_2M_BUFFERS)</td>
</tr>
<tr>
<td>MIC_STACKSIZE</td>
<td>integerB/K/M/G/T</td>
<td>12M</td>
<td>Main thread stack size limit for pthreads</td>
</tr>
<tr>
<td>MIC_PROXY_IO</td>
<td>integer</td>
<td>0</td>
<td>Proxy MIC I/O to host</td>
</tr>
<tr>
<td>MIC_PROXY_FS_ROOT</td>
<td>string</td>
<td>none</td>
<td>Root directory for I/O proxy for COI</td>
</tr>
<tr>
<td>H_TIME</td>
<td>integer</td>
<td>none</td>
<td>Control printing offload execution time</td>
</tr>
<tr>
<td>H_TRACE</td>
<td>integer</td>
<td>none</td>
<td>Control printing offload activity</td>
</tr>
<tr>
<td>H_TARGET_LOG</td>
<td>string</td>
<td>none</td>
<td>file name to output traces from MIC</td>
</tr>
<tr>
<td>H_HOST_LOG</td>
<td>string</td>
<td>none</td>
<td>file name to output traces from host</td>
</tr>
<tr>
<td>SEP_MONITOR</td>
<td>integer</td>
<td>0</td>
<td>Enable SEP at offload regions.</td>
</tr>
</tbody>
</table>
Resources

• Intel® Composer XE 2013 for Linux* User and Reference Guide

• http://mic-dev.intel.com (may need to request access)
  • Getting Started Guide
  • Technical Papers, e.g.
    – Intel® MIC Optimization Guide
    – Intel® MIC Profiling Guide
    – Building a Native Application...
    – Using Intel® MPI...
    – Intel® MIC Debugger Use Cases, etc, etc
  • New User Forum at https://mic-dev.intel.com/forum/

• Intel Premier Support https://premier.intel.com
Summary

• Intel’s optimizing compilers and libraries take full advantage of CPU and Intel® MIC architecture
  => Performance

• Offload compiler makes parallel programming on Intel® MIC architecture as easy as programming the CPU
  => Productivity
<table>
<thead>
<tr>
<th>Optimization Notice</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
</tr>
</tbody>
</table>

Notice revision #20110804
Legal Disclaimer

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.

Performance tests and ratings are measured using specific computer systems and/or components
and reflect the approximate performance of Intel products as measured by those tests. Any
difference in system hardware or software design or configuration may affect actual performance.
Buyers should consult other sources of information to evaluate the performance of systems or
components they are considering purchasing. For more information on performance tests and on

BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Atom, Centrino Atom Inside, Centrino
Inside, Centrino logo, Cilk, Core Inside, FlashFile, i960, InstantIP, Intel, the Intel logo, Intel386,
Intel486, IntelDX2, IntelDX4, IntelSX2, Intel Atom, Intel Atom Inside, Intel Core, Intel Inside,
NetStructure, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel Viiv, Intel vPro, Intel
XScale, Itanium, Itanium Inside, MCS, MMX, Oplus, OverDrive, PDCcharm, Pentium, Pentium
Inside, skoool, Sound Mark, The Journey Inside, Viiv Inside, vPro Inside, VTune, Xeon, and Xeon
Inside are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2011. Intel Corporation.

http://intel.com/software/products
Thank you
#pragma offload_attribute (push,target(mic))
#include "tbb/concurrent_hash_map.h"
#include "tbb/task_scheduler_init.h"
...
static void CountOccurrences() {
    StringTable table;
    parallel_for( blocked_range<string*>( Data, Data+N, 1000 ), Tally(table) );
    int n = 0;
    for( StringTable::iterator i=table.begin(); i!=table.end(); ++i ) {
        ...
    }
}
static void CreateData() {
    ...
    for( int i=0; i<N; ++i ) {
        Data[i] = Adjective[rand()%n_adjective];
        ...
    }
}
#pragma offload_attribute (pop)

int main() {
    #pragma offload target(mic) in(Verbose, NThread)
    {
        task_scheduler_init init(NThread);
        CreateData();
        CountOccurrences();
    }
}
Suggestion: Simplify use of alloc_if and free_if

- For Readability define macros  (unless condition is variable)
  - `#define ALLOC alloc_if(1)`
  - `#define FREE free_if(1)`
  - `#define RETAIN free_if(0)`
  - `#define REUSE alloc_if(0)`

  ```
  #pragma offload target(mic) in( p:length(l) )
  ...
  ```
  - Allocate and do not free
    ```
    #pragma offload target(mic) in (p:length(l) ALLOC RETAIN)
    ...
    ```
  - Reuse memory allocated above and do not free
    ```
    #pragma offload target(mic) in (p:length(l) REUSE RETAIN)
    ...
    ```
  - Reuse memory allocated above and free
    ```
    #pragma offload target(mic) in (p:length(l) REUSE FREE)
    ```
Memory Regions - consequences

- All data “IN” data from a single offload pragma must fit into buffer A, unless a pointer designated for 2MB pages. All such “OUT” data must (separately) fit into buffer A.
- If a single data offload is too large for buffer A, it may be split into two offloads.
- Offloaded data in buffer A must be recopied to a separate allocation in C or D, freeing up the buffer A for reuse.
- Offloaded data allocated in buffer B (2MB pages) remain there; just a pointer is passed - they do not need to be copied. (So this is more efficient use of memory for large objects).
Sample Program – Multicard

```c
no_device = __Offload_number_of_devices()
no_device = min(2, no_device);
Cilk_for(j=0; j<no_device; j++) {
  #pragma offload target(mic:j)
  {
    my_size = problem_size/no_device;
    my_start = __Offload_get_device_number() * my_size;
    for(j=my_start; j<(my_start+my_size); j++) {
      output[j] = input1[j] + input2[j];
    }
  }
}
```

- Divides up the work as you would for OpenMP threads using # threads, thread #
Example of Offload Using Keywords

typedef class _Cilk_shared _LinkedListNode{
    int data;
    class _LinkedListNode* next;
} LinkedListNode;

typedef  LinkedListNode LinkedList;

_Cilk_shared LinkedList listhead;
_Cilk_shared void calculateOnMIC();

void initialize() {
    listhead.data = 5;
    listhead.next->data = 7;
    listhead.next->next->data = 9;
}

_Cilk_shared void calculateOnMIC()
{
    listhead.data += 100;
    listhead.next->data += 100;
    listhead.next->next->data += 100;
}

main()
{
    initialize();

    _Cilk_offload calculateOnMIC();
    _Cilk_offload calculateOnMIC();
}
Differences between beta compiler for KNC and alpha compiler for KNF

- Very few usage changes for KNC compared to KNF
  - But the instruction set (& hence intrinsics) has differences
  - Can’t run KNF executables on KNC; must recompile
  - Separate compilers –beta compiler does not support KNF
    - It’s not just a different command line switch
  - The MPSS stacks are also different

- There is no –offload-build switch for KNC
  - -offload-copts replaced by –offload-option,mic,compiler,”....”
Example – share work between card(s) and host

```fortran
$dir$ attributes offload:mic :: threed_int
call omp_set_nested(.true.)
!$OMP PARALLEL SECTIONS
!$OMP SECTION
!DIR$ OFFLOAD TARGET(MIC) IN(points) OUT(ppotential:length(np/2))
!$OMP PARALLEL DO PRIVATE (xp, yp, zp)
do ip = 1,np/2
   xp = points(1,ip); yp = points(2,ip); zp = points(3,ip)
   ppotential(ip) = threed_int(x0,xn,y0,yn,z0,zn,nx,ny,nz,xp,yp,zp)
enddo
!$OMP SECTION
!$OMP PARALLEL DO PRIVATE (xp, yp, zp)
do ip = np/2+1,np
   xp = points(1,ip); yp = points(2,ip); zp = points(3,ip)
   ppotential(ip) = threed_int(x0,xn,y0,yn,z0,zn,nx,ny,nz,xp,yp,zp)
enddo
!$OMP END SECTIONS
!$OMP END PARALLEL
```

Add an extra section for each additional Intel® MIC co-processor.

ppotential is a pointer
## Usage Models for Intel® MKL on Intel® MIC

<table>
<thead>
<tr>
<th>Offload mechanism</th>
<th>Link line modifications</th>
<th>Hetero. computations</th>
<th>Interface modifications</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Native calls</strong></td>
<td>Direct execution on MIC side / explicit data management via user platform APIs</td>
<td>Using ‘-mkl’ or explicit linking with Intel MIC native library</td>
<td>Must be defined by the user</td>
<td>Not required</td>
</tr>
<tr>
<td><strong>Compiler-assisted Offload</strong></td>
<td>Explicit data management via offload compiler RTL</td>
<td>Not required*</td>
<td>Automatically uses both host and MIC device(s)</td>
<td>None; 2 new functions to specify how work is divided</td>
</tr>
<tr>
<td><strong>Automatic Offload</strong></td>
<td>Automatic for the user (uses MKL own private API)</td>
<td>Not required*</td>
<td>Automatically uses both host and MIC device(s)</td>
<td>None; 2 new functions to specify how work is divided</td>
</tr>
</tbody>
</table>