Intel® Xeon Phi™ Coprocessor

Offload Compilation
Legal Disclaimer

- INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked “reserved” or “undefined”. Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to:  
http://www.intel.com/design/literature.htm

- Sandy Bridge and other code names featured are used internally within Intel to identify products that are in development and not yet publicly announced for release. Customers, licensees and other third parties are not authorized by Intel to use code names in advertising, promotion or marketing of any product or services and any such use of Intel's internal code names is at the sole risk of the user

- 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. For more information go to http://www.intel.com/performance

- Intel, Core, Xeon, VTune, Phi and Cilk, Intel and Intel Sponsors of Tomorrow. and Intel Sponsors of Tomorrow. logo, and the Intel logo are trademarks of Intel Corporation in the United States and other countries.

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

- Copyright ©2012 Intel Corporation.

- Hyper-Threading Technology: Requires an Intel® HT Technology enabled system, check with your PC manufacturer. Performance will vary depending on the specific hardware and software used. Not available on all Intel® Core™ processors. For more information including details on which processors support HT Technology, visit http://www.intel.com/info/hyperthreading

- Intel® 64 architecture: Requires a system with a 64-bit enabled processor, chipset, BIOS and software. Performance will vary depending on the specific hardware and software you use. Consult your PC manufacturer for more information. For more information, visit http://www.intel.com/info/em64t

- Intel® Turbo Boost Technology: Requires a system with Intel® Turbo Boost Technology capability. Consult your PC manufacturer. Performance varies depending on hardware, software and system configuration. For more information, visit http://www.intel.com/technology/turboboost

Performance Notice

Module Goals

• You can list two pragmas or keywords that cause code to run on the Intel® Xeon Phi™ Coprocessor
• You can explain how data movement is controlled between the host and card
Module Outline

• High-Level overview of the Heterogeneous Compiler
  – Overview
  – Offload using Explicit Copies
  – Offload using Implicit Copies
  – Comparison of Techniques
  – Heterogeneous Compiler command-line options
  – Intel® Xeon Phi™ Coprocessor Native Compiler
Offload Compilation

• High-Level overview of the Heterogeneous Compiler
  – Overview
  – Offload using Explicit Copies
  – Offload using Implicit Copies
  – Comparison of Techniques
  – Heterogeneous Compiler command-line options
  – Intel® Xeon Phi™ Native Compiler
Heterogeneous Compiler – Programming Model Overview

• Add pragmas and new keywords to working host code to make sections of code run on the Intel® Xeon Phi™ Coprocessor
  – Similar to adding parallelism to serial code using OpenMP* pragmas or Intel® Cilk™ Plus keywords
  – Again, the Intel® Xeon Phi™ Coprocessor is best suited for highly-parallel vectorized code

• The Intel® Compiler generates code for both target architectures at once
  – The resulting binary runs whether or not a coprocessor is present
    o Unless you use _Cilk_offload_to or #pragma offload target(mic:cardnumber)
  – The compiler adds code to transfer data automatically to the coprocessor and to start your code running (with no extra coding on your part)
  – Hence the term “Heterogeneous Compiler” or “Offload Compiler”

• You can make further optimizations to your code that ensure full use of both the host and coprocessor
Heterogeneous Compiler – Data Transfer Overview

• The host CPU and the coprocessor do not share physical or virtual memory in hardware

• Two offload data transfer models are available:
  1. Explicit Copy
     o Programmer designates variables that need to be copied between host and card in the offload pragma/directive
     o Syntax: Pragma/directive-based
     o C/C++ Example: #pragma offload target(mic) in(data:length(size))
     o Fortran Example: !dir$ offload target(mic) in(a1:length(size))
  2. Implicit Copy
     o Programmer marks variables that need to be shared between host and card
     o The same variable can then be used in both host and coprocessor code
     o Runtime automatically maintains coherence at the beginning and end of offload statements
     o Syntax: keyword extensions based
     o Example: _Cilk_shared double foo; _Cilk_offload func(y);
Heterogeneous Compiler – Overview Summary

• Programmer designates code sections to offload
  – No further programming/API usage is needed
  – The compiler and the runtime automatically manage setup/teardown, data transfer, and synchronization
• Code marked for offload is not guaranteed to run on the coprocessor
  – If the coprocessor is unavailable, the offload section runs entirely on host
    o Unless you use _Cilk_offload_to or “#pragma offload target(mic:cardnumber)”, in which case the coprocessor must be present for the code to run
    o Future feature: The runtime may use coprocessor load to decide whether or not to execute code marked for “offload” on the coprocessor

• Setting up the compiler build environment
  CSH: source /opt/intel/bin/compilervars.csh intel64
  SH: source /opt/intel/bin/compilervars.sh intel64
Offload Compilation

- High-Level overview of the Heterogeneous Compiler
  - Overview
  - Offload using Explicit Copies
  - Offload using Implicit Copies
  - Comparison of techniques
  - Heterogeneous Compiler command-line options
  - Intel® Xeon Phi™ Native Compiler
# Heterogeneous Compiler – Offload using Explicit Copies

<table>
<thead>
<tr>
<th></th>
<th>C/C++ Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offload pragma</td>
<td>#pragma offload &lt;clauses&gt; &lt;statement&gt;</td>
<td>Allow next statement to execute on coprocessor or host CPU</td>
</tr>
<tr>
<td>Keyword for variable &amp;</td>
<td><strong>attribute</strong>((target(mic)))</td>
<td>Compile function for, or allocate variable on, both CPU and coprocessor</td>
</tr>
<tr>
<td>function definitions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Entire blocks of code</td>
<td>#pragma offload_attribute(push, target(mic))</td>
<td>Mark entire files or large blocks of code for generation on both host CPU and coprocessor</td>
</tr>
<tr>
<td></td>
<td>#pragma offload_attribute(pop)</td>
<td></td>
</tr>
</tbody>
</table>

## Fortran Syntax

<table>
<thead>
<tr>
<th></th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offload directive</td>
<td>Execute next OpenMP* parallel construct on coprocessor</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Execute next statement (function call) on coprocessor</td>
</tr>
<tr>
<td>Keyword for variable/</td>
<td>Compile function or variable for CPU and coprocessor</td>
</tr>
<tr>
<td>function definitions</td>
<td></td>
</tr>
</tbody>
</table>
Heterogeneous Compiler – Conceptual Transformation

Source Code

main()
{
 f();
}

f()
{
  #pragma offload
  a = b + g();
}

__attribute__((target(mic))) g()
{
}

Linux* Host Program

main()
{
 copy_code_to_coproc();
 f();
 unload_coproc();
}

f()
{
  if (coproc_available())
  {
    send_data_to_coproc();
    start f_part_coproc();
    receive_data_from_coproc();
  } else
  f_part_host();

f_part_host()
{
  a = b + g();
}

g() {
  ...
}

f_part_coproc()
{
  a = b + g_coproc();
}

g_coproc() {
  ...
}

This all happens automatically when you issue a single compile command.
Heterogeneous Compiler – Conceptual Transformation

Source Code

main()
{
 f();
}

f()
{
   #pragma offload
   a = b + g();
}

__attribute__((target(mic))) g()
{
}

Linux* Host Program

main()
{
   copy_code_to_coproc();
   f();
   unload_coproc();
}

f()
{
   if (coproc_available()){
      send_data_to_coproc();
      start f_part_coproc();
      receive_data_from_coproc();
   } else
      f_part_host();
}

f_part_host()
{
a = b + g();
}

f_part_coproc() {a = b + g_coproc();}

g_coproc() {

}
Heterogeneous Compiler – Conceptual Transformation

Source Code

```c
main()
{
    f();
}

f()
{
    #pragma offload
    a = b + g();
}

__attribute__((target(mic))) g()
{
}
```

Linux* Host Program

```c
main()
{
    copy_code_to_coproc();
    f();
    unload_coproc();
}

f()
{
    if (coproc_available()){
        send_data_to_coproc();
        start f_part_coproc();
        receive_data_from_coproc();
    } else
    f_part_host();
}

f_part_host()
{
    a = b + g();
}

g() { ... }
```

Intel® Xeon Phi™ Program

This all happens automatically when you issue a single compile command

```c
f_part_coproc() {a = b + g_coproc();}
g_coproc() { ... }
```
Heterogeneous Compiler – Conceptual Transformation

Source Code

```c
main() {
    f();
}

f() {
    #pragma offload
    a = b + g();
}

_attribute_ ((target(mic))) g() {
    
}
```

Linux* Host Program

```c
main() {
    copy_code_to_coproc();
    f();
    unload_coproc();
}

f() {
    if (coproc_available()){
        send_data_to_coproc();
        start f_part_coproc();
        receive_data_from_coproc();
    } else
    f_part_host();
}

f_part_host() {
    a = b + g();
}

g() {...}
```

Intel® Xeon Phi™ Program

```c
f_part_coproc() {
    a = b + g_coproc();
}

g_coproc() {...}
```

This all happens automatically when you issue a single compile command.
Heterogeneous Compiler – Offload using Explicit Copies – OpenMP* & Intel® Cilk™ Plus examples

**C/C++ OpenMP**

```c
#pragma omp parallel for
for (i=0; i<count; i++)
{
    a[i] = b[i] * c + d;
}
```

**Fortran OpenMP**

```fortran
!$omp parallel do
    do i=1, count
        A(i) = B(i) * c + d
    end do
!$omp end parallel
```

**C/C++ Intel® Cilk™ Plus**

Note: There is no direct equivalent _Cilk_for notation in the explicit offload model.*

*There is one in the implicit model.
Heterogeneous Compiler – Offload using Explicit Copies – Modifiers

Variables and pointers restricted to scalars, structs of scalars, and arrays of scalars

<table>
<thead>
<tr>
<th>Clauses / Modifiers</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target specification</td>
<td>target( name[:card_number] )</td>
<td>Where to run construct</td>
</tr>
<tr>
<td>Conditional offload</td>
<td>if (condition)</td>
<td>Boolean expression</td>
</tr>
<tr>
<td>Inputs</td>
<td>in(var-list modifiers&lt;opt&gt;)</td>
<td>Copy from host to coprocessor</td>
</tr>
<tr>
<td>Outputs</td>
<td>out(var-list modifiers&lt;opt&gt;)</td>
<td>Copy from coprocessor to host</td>
</tr>
<tr>
<td>Inputs &amp; outputs</td>
<td>inout(var-list modifiers&lt;opt&gt;)</td>
<td>Copy host to coprocessor and back when offload completes</td>
</tr>
<tr>
<td>Non-copied data</td>
<td>nocopy(var-list modifiers&lt;opt&gt;)</td>
<td>Data is local to target</td>
</tr>
</tbody>
</table>

**Modifiers**

<table>
<thead>
<tr>
<th></th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specify pointer length</td>
<td>length(element-count-expr)</td>
<td>Copy N elements of the pointer’s type</td>
</tr>
<tr>
<td>Control pointer memory allocation</td>
<td>alloc_if ( condition )</td>
<td>Allocate memory to hold data referenced by pointer if condition is TRUE</td>
</tr>
<tr>
<td>Control freeing of pointer memory</td>
<td>free_if ( condition )</td>
<td>Free memory used by pointer if condition is TRUE</td>
</tr>
<tr>
<td>Control target data alignment</td>
<td>align ( expression )</td>
<td>Specify minimum memory alignment on target</td>
</tr>
</tbody>
</table>
Heterogeneous Compiler – Offload using Explicit Copies – Modifier Example

```c
float reduction(float *data, int numberOf)
{
    float ret = 0.f;
    #pragma offload target(mic) in(data:length(numberOf))
    {
        #pragma omp parallel for reduction(+:ret)
        for (int i=0; i < numberOf; ++i)
            ret += data[i];
    }
    return ret;
}
```

Note: copies `numberOf*sizeof(float)` elements to the coprocessor, not `numberOf` bytes – the compiler knows `data`'s type
Heterogeneous Compiler – Offload using Explicit Copies - Rules & Limitations

• The Host⇔Coprocessor data types allowed in a simple offload:
  – Scalar variables of all types
    o Must be *globals or statics* if you wish to use them with *nocopy*, *alloc_if*, or *free_if* (i.e. if they are to persist on the coprocessor between offload calls)
  – Structs that are bit-wise copyable (no pointer data members)
  – Arrays of the above types
  – Pointers to the above types

• What is allowed *within* coprocessor code?
  – All data types can be used (incl. full C++ objects)
  – Any parallel programming technique (Pthreads*, Intel® TBB, OpenMP*, etc.)
  – Intel® Xeon Phi™ versions of Intel® IPP and Intel® MKL
Heterogeneous Compiler – Offload using Explicit Copies – Data Movement

- Default treatment of in/out variables in a `#pragma offload` statement
  - At the start of an offload:
    - Space is allocated on the coprocessor
    - in variables are transferred to the coprocessor
  - At the end of an offload:
    - out variables are transferred from the coprocessor
    - Space for both types (as well as inout) is deallocated on the coprocessor

```c
#pragma offload inout(pA:length(n)) {...}
```
**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 SSE3 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