



Intel® Xeon Phi™ Coprocessor

New Developments

Klaus-Dieter Oertel
Software and Services Group
Intel Corporation

HIC for FAIR Physics Day, FIAS, 11-11-2014



Agenda

Knights Landing

OpenMP* 4.0

hStreams

Links

Knights Landing

Intel® Xeon Phi™ Coprocessor Generations



Pre 2013:

“Knights Ferry”

Engineering prototype




2013:

**Intel® Xeon Phi™
Coprocessor
x100 Product
Family**

“Knights Corner”

- 22 nm process
- 57-61 cores
- 6-16 GB memory
- 1 TeraFLOPs DP peak



2H 2015:

**Intel® Xeon Phi™
Coprocessor
x200 Product
Family**

“Knights Landing”

- 14 nm process
- 60+ cores
- On package, high-bandwidth memory
- Processor & coprocessor
- +3 TeraFLOPs DP peak

**In planning
Upcoming
Generation of the
Intel® MIC
Architecture**

“Future Knights”

Continued roadmap
commitment

Next Intel® Xeon Phi™ Product Family (Codenamed Knights Landing)



- ❖ Available in Intel cutting-edge 14 nanometer process
- ❖ Stand alone CPU or PCIe coprocessor – not bound by 'offloading' bottlenecks
- ❖ Integrated Memory - balances compute with bandwidth

Parallel is the path forward, Intel is your roadmap!

All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Knights Landing - Server Processor

Standalone bootable processor (running host OS) and a PCIe coprocessor (PCIe end-point device)

Platform memory capacity comparable to Intel® Xeon® processors

Reliability (“Intel server-class reliability”)

Power Efficiency (Over 25% better than discrete coprocessor)¹

Density (3+ KNL with fabric in 1U)²

All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

All projections are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect actual performance.

¹ Projected result based on internal Intel analysis using estimated performance and power consumption of a rack sized deployment of Intel® Xeon® processors and Knights Landing coprocessors as compared to a rack with KNL processors only

² Projected result based on internal Intel analysis comparing a discrete Knights Landing processor with integrated fabric to a discrete Intel fabric component card.

Knights Landing - Microarchitecture

Based on Intel's 14 nanometer manufacturing technology

Based on Intel® Atom™ core (based on Silvermont microarchitecture) with many HPC enhancements

Binary compatible with Intel® Xeon® Processors¹

Support for Intel® Advanced Vector Extensions 512 (Intel® AVX-512)

60+ cores in a 2D Mesh architecture

All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

All projections are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect actual performance.

¹ Binary compatible with Intel® Xeon® Processors v3 (Haswell) with the exception of Intel® TSX (Transactional Synchronization Extensions)

Knights Landing – Microarchitecture (ctd.)

4 Threads / Core

Deep Out-of-Order Buffers

Gather/scatter in hardware

Advanced Branch Prediction

High cache bandwidth

Cache-coherency

Knights Landing - Integration

Intel® Omni Path Architecture integration

Integrated high-performance on-package memory (MCDRAM)

- Developed in partnership with Micron Technology
- Up to 16GB at launch
- Over 5x STREAM vs. DDR4¹
- Over 5x Energy Efficiency vs. GDDR5²
- Over 3x Density vs. GDDR5²
- Flexible memory modes including cache and flat

All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

All projections are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect actual performance.

¹ Projected result based on internal Intel analysis of STREAM benchmark using a Knights Landing processor with 16GB of ultra high-bandwidth versus DDR4 memory with all channels populated.

² Projected result based on internal Intel analysis comparison of 16GB of ultra high-bandwidth memory to 16GB of GDDR5 memory used in the Intel® Xeon Phi™ coprocessor 7120P.

Knights Landing Integrated On-Package Memory

Cache Model

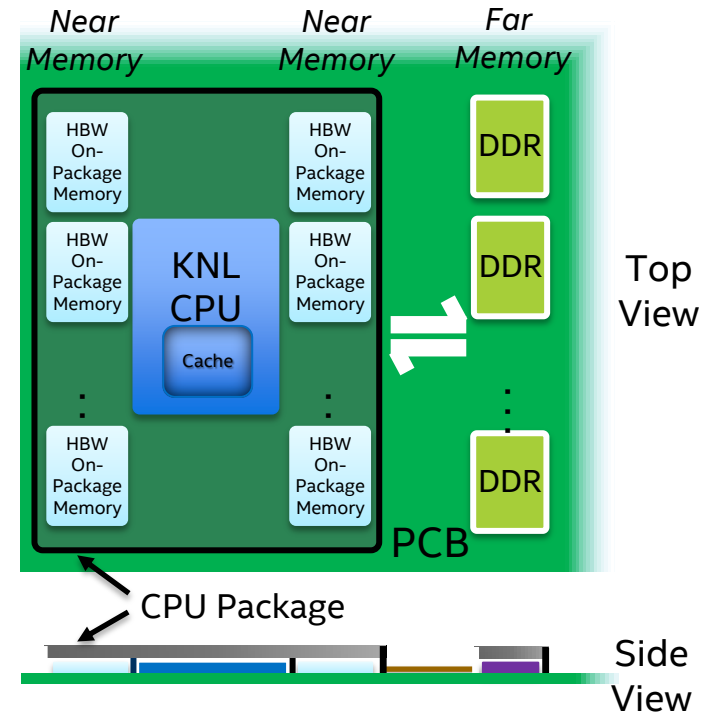
Let the hardware automatically manage the integrated on-package memory as an “L3” cache between KNL CPU and external DDR

Flat Model

Manually manage how your application uses the integrated on-package memory and external DDR for peak performance

Hybrid Model

Harness the benefits of both cache and flat models by segmenting the integrated on-package memory



Maximizes performance through higher memory bandwidth and flexibility¹

¹ As compared with Intel® Xeon Phi™ x100 Coprocessor Family

Diagram is for conceptual purposes only and only illustrates a CPU and memory – it is not to scale, and is not representative of actual component layout.

Knights Landing - Performance

Most of today's parallel optimizations carry forward to KNL

3x Single-Thread Performance compared to Knights Corner¹

3+ TeraFLOPS of double-precision peak theoretical performance per single socket node²

All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

All projections are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect actual performance.

¹ Projected peak theoretical single-thread performance relative to 1st Generation Intel® Xeon Phi™ Coprocessor 7120P

² Over 3 Teraflops of peak theoretical double-precision performance is preliminary and based on current expectations of cores, clock frequency and floating point operations per cycle.

Knights Landing - Availability

First commercial HPC systems in 2H'15

Already announced:

- Cori Supercomputer at NERSC (National Energy Research Scientific Computing Center at LBNL/DOE) became the first publically announced Knights Landing based system, with over 9,300 nodes slated to be deployed in mid-2016
- “Trinity” Supercomputer at NNSA (National Nuclear Security Administration) is a \$174 million deal awarded to Cray that will feature Haswell and Knights Landing, with acceptance phases in both late-2015 and 2016.

OpenMP* 4.0

OpenMP* 4.0 Specification

Released July 2013

- <http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf>
- A document of examples is expected to release soon

Changes from 3.1 to 4.0 (Appendix E.1):

- SIMD directives
- Device/Accelerator directives
- Taskgroup and dependant tasks
- Thread affinity
- Cancellation directives
- User-defined reductions
- Sequentially consistent atomics
- Fortran 2003 support

Intel® C/C++ and Fortran Compilers

Main SIMD and Offloading features are supported in 14.0 compilers

- *Places and thread affinity (14.0)*
- *Main features of SIMD directives (14.0)*
- *Main features of Device / Accelerator directives (14.0)*
- *Sequentially consistent atomics (14.0)*
- *Combined simd, parallel, target, teams, distribute constructs (15.0)*
- *Taskgroup and dependent tasks (15.0)*
- *Cancellation directives (15.0)*
- *Fortran 2003 support (15.0)*
- *User-defined reductions (TBD)*

SIMD Support - Motivation

Provides a portable high-level mechanism to specify SIMD parallelism (vectorization)

- Heavily based on Intel's SIMD directive

Two main new directives

- To SIMDize loops
- To create SIMD functions

SIMD Loop - Syntax

#pragma omp simd [*clauses*]

for-loop

!\$omp simd [*clauses*]

do-loops

[!\$omp end simd]

Loop has to be in “Canonical loop form”

- as do/for worksharing

SIMD Loop - Clauses

safelen (length)

- Maximum number of iterations that can run concurrently without breaking a dependence
 - in practice, maximum vector length
 - $A[i] = A[i\text{-shift}]$, safe if $\text{shift} \geq \text{VL}$

linear (list[:linear-step])

- The variable value is in relationship with the iteration number
 - $x_i = x_{\text{orig}} + i * \text{linear-step}$

aligned (list[:alignment])

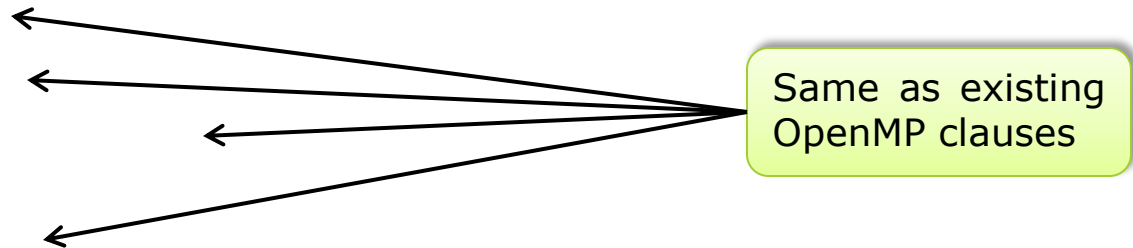
- Specifies that the list items have a given alignment
- Default is alignment for the architecture

private (list)

lastprivate (list)

reduction (operator:list)

collapse (n)



SIMD Loop - Example

```
double pi()  
{  
    double pi = 0.0;  
    double t;  
  
    #pragma omp simd private(t) reduction(+:pi)  
    for (i=0; i<count; i++) {  
        t = (double)((i+0.5)/count);  
        pi += 4.0/(1.0+t*t);  
    }  
    pi /= count  
    return pi;  
}
```

SIMD Functions - Syntax

#pragma omp declare simd [*clauses*]

[#pragma omp declare simd [*clauses*]]

function definition or declaration

!\$omp declare simd (*function-or-procedure-name*) [*clauses*]

Instructs the compiler to

- generate a SIMD-enabled version(s) of a given function
- that a SIMD-enabled version of the function is available to use from a SIMD loop

SIMD Functions - Clauses

simdlen(*length*)

- generate function to support a given vector length

uniform(*argument-list*)

- argument has a constant value between the iterations of a given loop

inbranch

- function always called from inside an if statement

notinbranch

- function never called from inside an if statement

linear(*argument-list[:linear-step]*)

aligned(*argument-list[:alignment]*)

reduction(*operator:list*)



Same as before

SIMD Functions - Example

```
#pragma omp declare simd notinbranch
```

```
float min(float a, float b) {  
    return a < b ? a : b;  
}
```

```
#pragma omp declare simd inbranch
```

```
float distsq(float x, float y) {  
    return (x - y) * (x - y);  
}
```

```
#pragma omp parallel for simd
```

```
    for (i=0; i<N; i++)  
        d[i] = min(distsq(a[i], b[i]), c[i]);
```

SIMD Combined Constructs

Distribute and vectorize the loop

Worksharing + SIMD

```
#pragma omp for simd [clauses]
```

```
!$omp do simd [clauses]
```

```
!$omp end do simd]
```

Parallel + worksharing + SIMD

```
#pragma omp parallel for simd [clause[[,] clause] ...]
```

```
!$omp parallel do simd [clause[[,] clause] ...]
```

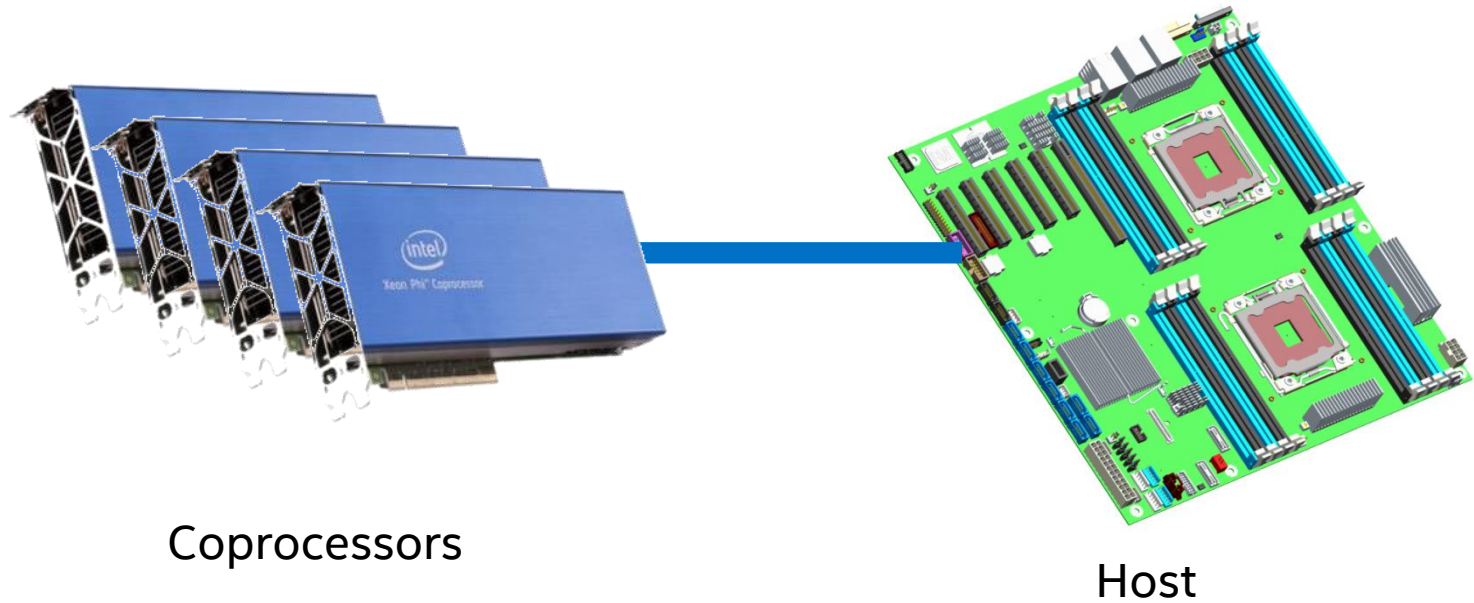
```
!$omp end parallel do simd
```

OpenMP* 4.0 - Device Model

OpenMP* 4.0 supports accelerators and coprocessors

Device model:

- One host
- Multiple accelerators/coprocessors of the same kind



declare target Construct

C/C++

#pragma omp declare target

[function-definition-or-functionvar-declaration]

#pragma omp end declare target

Fortran

!\$omp declare target *[(proc-name-list | list)]*

target Construct

#pragma omp target [*clause*[[,] *clause*],...]
structured-block

Clauses: **device**(*scalar-integer-expression*)
map(**alloc** | **to** | **from** | **tofrom**: *list*)
if(*scalar-expr*)

#pragma omp target data [*clause*[[,] *clause*],...]
structured-block

Clauses: **device**(*scalar-integer-expression*)
map(**alloc** | **to** | **from** | **tofrom**: *list*)
if(*scalar-expr*)

#pragma omp target update [*clause*[[,] *clause*],...]

Clauses: **to**(*list*)
from(*list*)
device(*integer-expression*)
if(*scalar-expression*)

Array Sections

OpenMP* 4.0 extends array subscript notation in C/C++ to allow array sections

- [*lower bound* : *length*]
 - specifies a section of length elements starting at lower bound
- if no lower bound is specified defaults to zero
- if no length is specified defaults to the remaining elements of that dimension for the array
 - only can be omitted if the size of the dimension is known

```
int a[10], *b;  
  
a[:] //legal  
b[:] // illegal
```

Target Example

```
#pragma omp target data device(0) map(alloc:tmp[0:N]) map(to:input[:N]) map(from:result)
{
#pragma omp target device(0)
#pragma omp parallel for
    for (i=0; i<N; i++)
        tmp[i] = some_computation(input[i], i);

    update_input_array_on_the_host(input);

#pragma omp target update device(0) to(input[:N])

#pragma omp target device(0)
#pragma omp parallel for reduction(+:result)
    for (i=0; i<N; i++)
        result += final_computation(input[i], tmp[i], i)
}
```

host

target

host

target

host

Asynchronous Offload

OpenMP accelerator/coprocessor constructs rely on existing OpenMP features to implement asynchronous offloads.

```
#pragma omp parallel sections
{
  #pragma omp task
  {
    #pragma omp target map(in:input[:N]) map(out:result[:N])
    #pragma omp parallel for
      for (i=0; i<N; i++) {
        result[i] = some_computation(input[i], i);
      }
  }
  #pragma omp task
  {
    do_something_important_on_host();
  }
}
```



team Construct

```
#pragma omp team [clause[[, clause],...]  
structured-block
```

Clauses: **num_teams**(*integer-expression*)
num_threads(*integer-expression*)
default(**shared** | **none**)
private(*list*)
firstprivate(*list*)
shared(*list*)
reduction(*operator* : *list*)

If specified, a **teams** construct must be contained within a **target** construct. That **target** construct must contain no statements or directives outside of the **teams** construct.

distribute, **parallel**, **parallel loop**, **parallel sections**, and **parallel workshare** are the only OpenMP constructs that can be closely nested in the **teams** region.

distribute Constructs

#pragma omp distribute [*clause*[[, *clause*],...]
for-loops

Clauses: **private**(*list*)
firstprivate(*list*)
collapse(*n*)
dist_schedule(*kind*[, *chunk_size*])

A **distribute** construct must be closely nested in a **teams** region.

Team Example

```
#pragma omp target device(0)
#pragma omp teams num_teams(60) num_threads(4) // 60 physical cores, 4 h/w threads each
{
#pragma omp distribute // this loop is distributed across teams
  for (int i = 0; i < 2048; i++) {
#pragma omp parallel for // loop is executed in parallel by all threads (4) of the team
  for (int j = 0; j < 512; j++) {
#pragma omp simd // create SIMD vectors for the machine
    for (int k=0; k<32; k++) {
      foo(i,j,k);
    }
  }
}
}
```


Execution Environment

```
void omp_set_default_device(int device_num )
```

```
int omp_get_default_device(void)
```

```
int omp_get_num_devices(void);
```

```
int omp_get_num_teams(void)
```

```
int omp_get_team_num(void);
```

OMP_DEFAULT_DEVICE Variable

- The **OMP_DEFAULT_DEVICE** environment variable sets the device number to use in target constructs by setting the initial value of the *default-device-var* ICV.
- The value of this environment variable must be a non-negative integer value.

hStreams

hStreams Overview

Intel's newest concurrency offering is called hStreams – a library interface for concurrent streams in a heterogeneous context.

The Preview Release of hStreams is installed as part of the Intel® MPSS release.

hStreams code is portable to future implementations with heterogeneous clusters.

hStreams is designed to cover functionality that is similar to third-party streaming APIs, such that it can readily be enabled.

hStreams Functionality

hStreams primitives enable users:

- to enqueue computation and communication in streams,
- enabling concurrency among the host and one or more coprocessors,
- among tasks within a coprocessor,
- and between communication and computation.

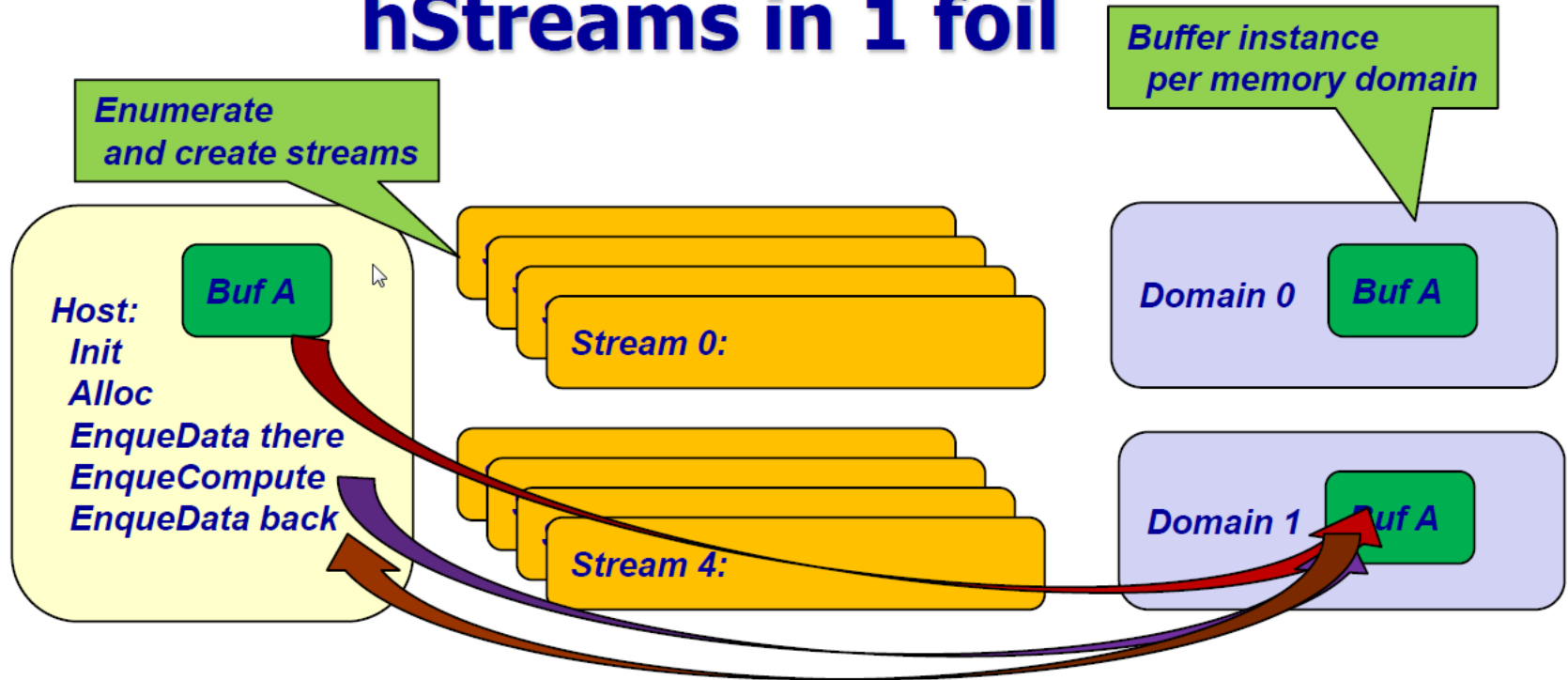
hStreams makes it easy to manage remote memory allocation and transfers between the host and coprocessor.

hStreams allows for explicit synchronization between the host and card and among streams when necessary.

Users can write and compile their own code for remote execution on an Intel® Xeon Phi™ coprocessor, or use stock gemm, memcpy and memset building blocks.

The user-provided code can be threaded and vectorized to make full use of a portion of the machine resources, e.g. using OpenMP 4.x.

hStreams in 1 foil



- Streams are FIFOs, mapped to arbitrary subsets of cards; subsets can exactly match to oversubscribe
- Everything is async by default

hStreams Documentation

Provided as part of the MPSS installation in
`/usr/share/doc/hStreams/`

- `hStreams_Overview.pdf`
- `hStreams_Usage.pdf` plus reference code
- `hStreams_Reference.pdf`

Links

Links

Developer portal: <https://software.intel.com/mic-developer>
→ Applications and Solutions Catalog

Intel® Developer Zone Forum:
<https://software.intel.com/en-us/forums/intel-many-integrated-core>

Intel® Xeon Phi™ Product Family: <http://intel.com/xeonphi>
→ Performance (benchmark results from different areas)

User support: <https://premier.intel.com>

Special Promotion: Intel® Xeon Phi™ Coprocessor 31S1P (≈\$200)
<https://software.intel.com/en-us/articles/special-promotion-intel-xeon-phi-coprocessor-31s1p>

Questions?



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 © 2014, 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

