



# Program the SAME Here and Over There – Intel® Data Parallel Programming Models and Intel® Many Integrated Core Architecture

Noah Clemons, Technical Consulting Engineer, Intel

Jim Jeffers, SW Product Application Engineer, Intel

CJ Lin, Software Engineer, Intel

## SFTS005

# Agenda

- **Overview**
  - Intel® Many Integrated Core Architecture (Intel® MIC Architecture) and Complementing Software Stack
  - Models of Computation
  - Intel® Xeon® Processor + Intel MIC Architecture: Writing Code for Today and Tomorrow
- **Intel® Parallel Composer**
  - Seamless Data Parallel: Turning “Small Knobs” to Run on Intel MIC Architecture
  - Array Notations
  - Carry Over Vectorization to Many Cores
- **Intel® Array Building Blocks**
  - C++ Library Front-End to Specify Arbitrary Data-Parallel Computation
  - Virtual Machine API to Create New Front-Ends
- **Programming for Intel MIC Architecture Without Using Offload**
- **Summary and Q&A**

The PDF for this Session presentation is available from our Technical Session Catalog at the end of the day at: [intel.com/go/idfsessions](http://intel.com/go/idfsessions)

URL is on top of Session Agenda Pages in Pocket Guide

# Agenda

- **Overview**
  - Intel® Many Integrated Core Architecture (Intel® MIC Architecture) and Complementing Software Stack
  - Models of Computation
  - Intel® Xeon® Processor + Intel MIC Architecture: Writing Code for Today and Tomorrow
- **Intel® Parallel Composer**
  - Seamless Data Parallel: Turning “Small Knobs” to Run on Intel MIC Architecture
  - Array Notations
  - Carry Over Vectorization to Many Cores
- **Intel® Array Building Blocks**
  - C++ Library Front-End to Specify Arbitrary Data-Parallel Computation
  - Virtual Machine API to Create New Front-Ends
- **Programming for Intel MIC Architecture Without Using Offload**
- **Summary and Q&A**

# Overview – A Step Forward in Performance with Excellent Programmability

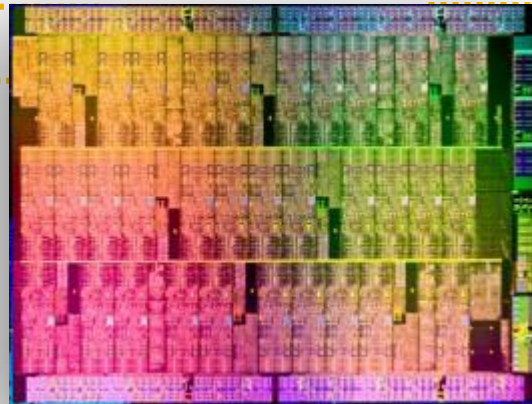
*First product to coincide with 22nm process codenamed Knights Corner*

## *Delivered Performance*

Launching on 22nm with >50 cores to provide outstanding performance for HPC users

## *Performance Density*

The compute density associated with specialty accelerators for parallel workloads



## *Programmability*

The many benefits of broad Intel® processor programming models, techniques, and familiar x86 developer tools

# Develop Using Parallel Models that Support Heterogeneous Computing

C/C++

Intel® Parallel Building Blocks

Intel® Cilk™ Plus

Intel® Threading Building Blocks

Intel® Array Building Blocks

Offload pragmas

OpenMP\*

OpenCL\*

Fortran

Co-array

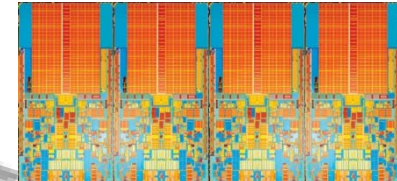
Offload directives

OpenMP

Intel® Math Kernel Library

Intel® Message Passing Interface Library

Multicore



Heterogeneous Computing



Many-core

The complementing software stack is common to both multicore and Many-core

# Invest in Common Tools and Programming Models

## Multicore

Intel® Xeon® processors are designed for intelligent performance and smart energy efficiency



Continuing to advance Intel Xeon processor family and instruction set (e.g., Intel® Advanced Vector Extensions, etc.)

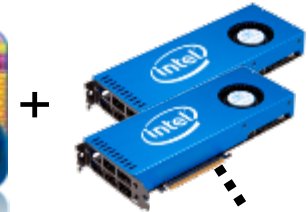
Your Application

```
REAL SUM(1)
CALL SYNC_ALL(WAIT=1)
DO WHILE (.TRUE.)
  IF (MOD(I,1000) == 0) THEN
    SUM = SUM + (SUM*(I))
  ENDIF
  CALL SYNC_ALL(WAIT=100)
ENDDO
```



## Many-core

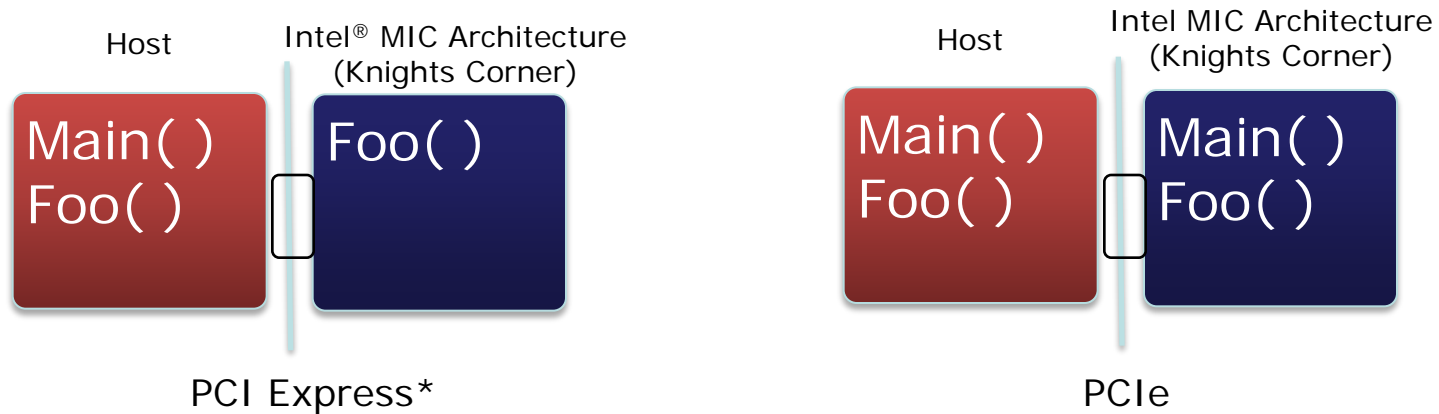
Intel® MIC Architecture - Co-processor ideal for highly parallel computing applications



Software development platforms ramping now

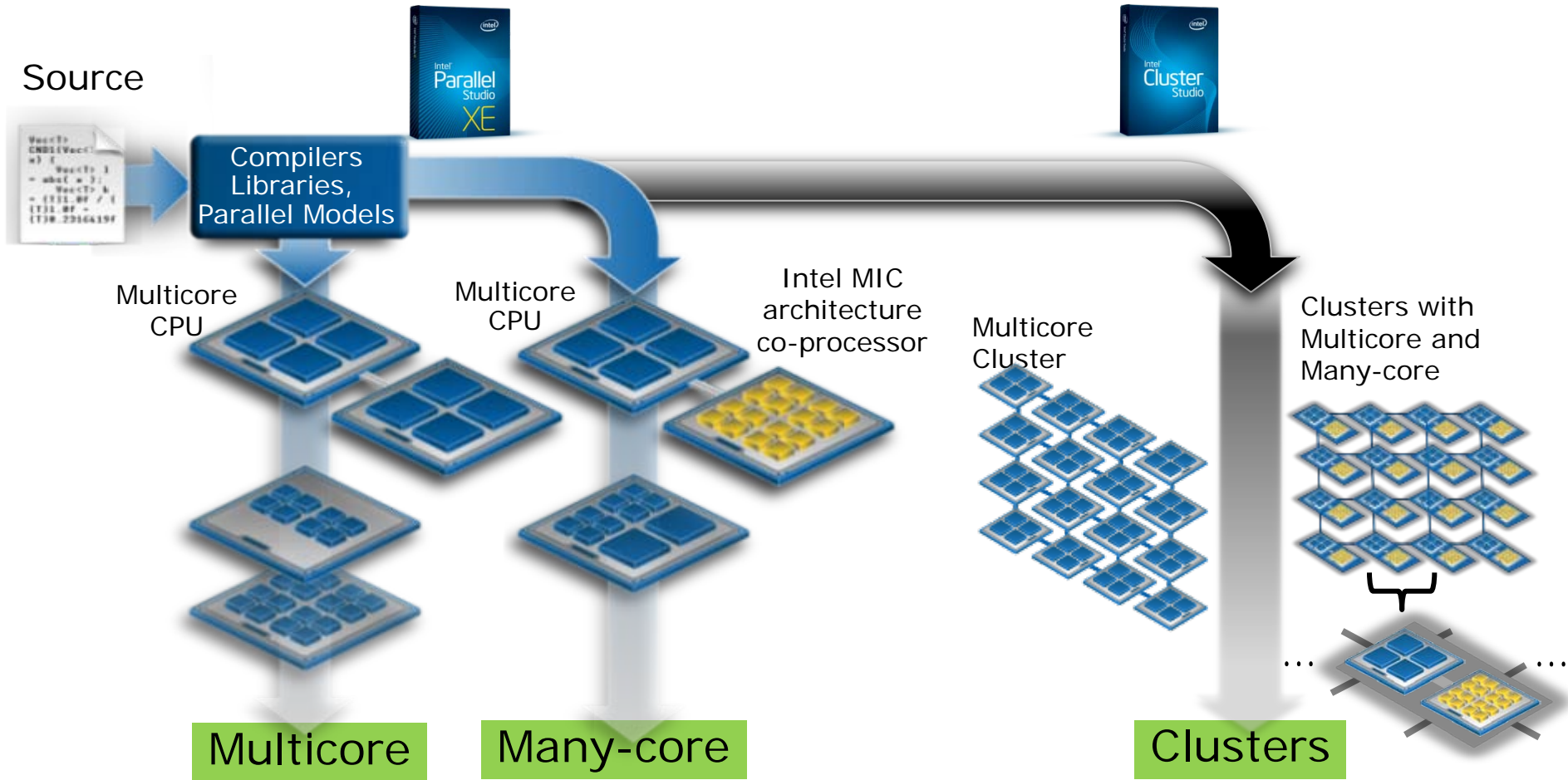
We extend Intel's Development Tools for Intel® Many Integrated Core Products

# Versatile Models of Computation for Intel® Many Integrated Core Products



Intel® MIC Architecture integrates full-featured processor cores and can either act as a coprocessor or host for applications.

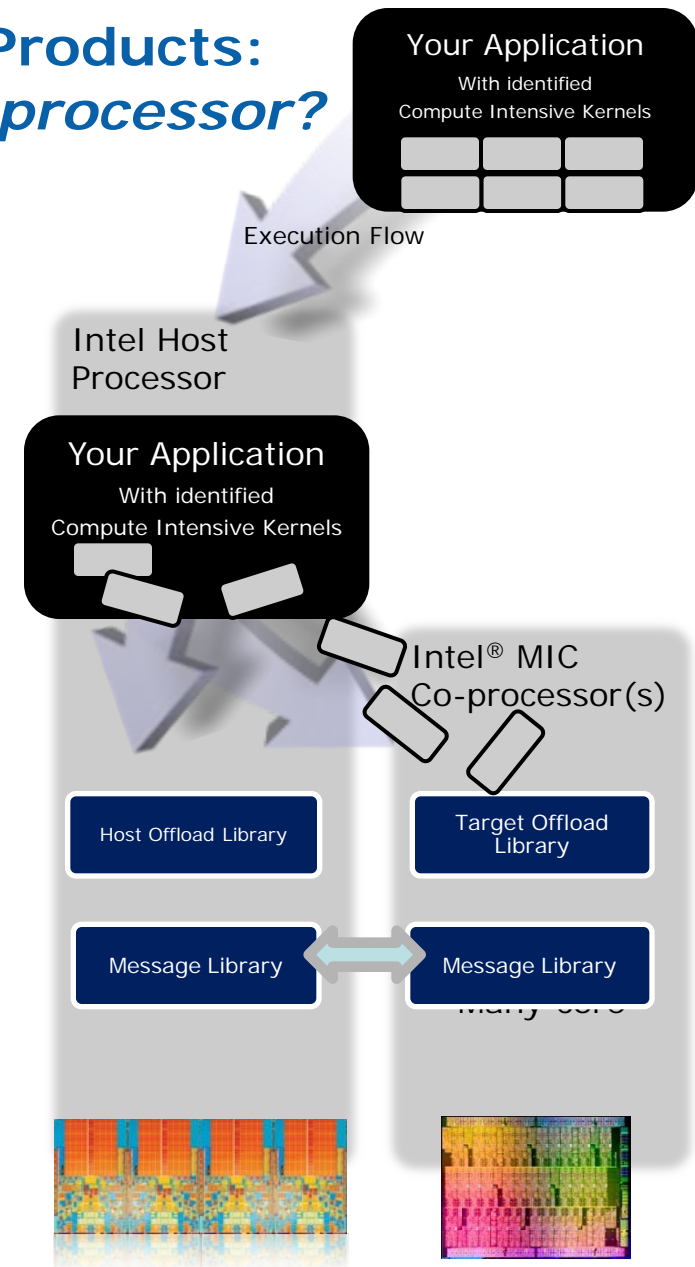
# Program for Multicore Today... Be Ready for Intel® Many Integrated Core Products



Eliminate Need for Dual Programming Architecture

# Intel® Xeon® Processors + Intel® MIC Products: What happens with and without the coprocessor?

Without: Intel® MIC Architecture Co-processor(s) are absent	With: Intel MIC Architecture Co-processor(s) are present
Application starts and executes on host	Application starts on host and executes portions on Intel MIC Co-processor(s)
	At runtime, if Intel® MIC Co-processor(s) are available, the target binary is loaded
At each offload, the construct runs on host cores/threads	At each offload, the construct runs on the Intel MIC Co-processor(s)
Normal program termination on host	At program termination, target binary is unloaded



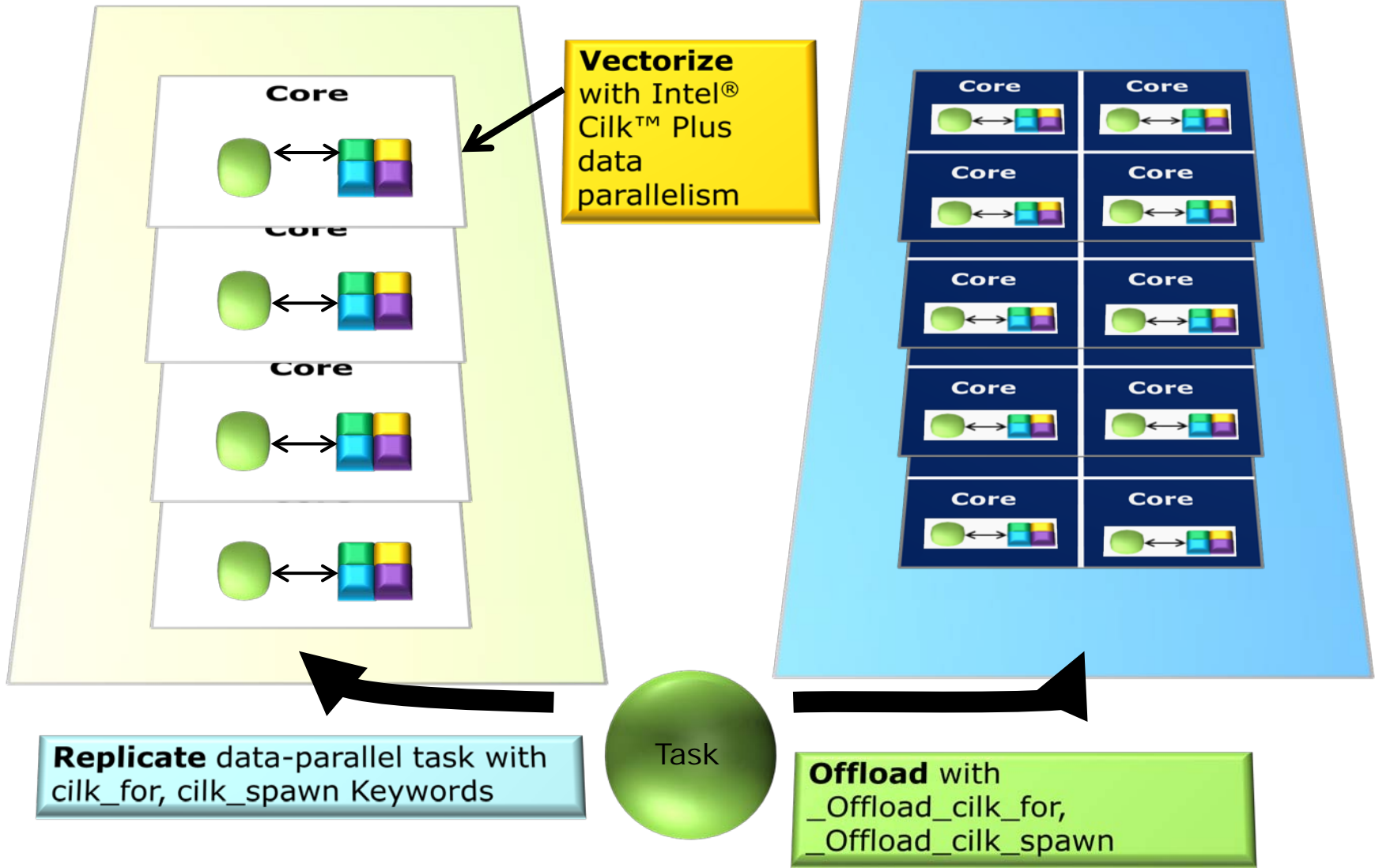
# Agenda

- **Overview**
  - Intel® Many Integrated Core Architecture (Intel® MIC Architecture) and Complementing Software Stack
  - Models of Computation
  - Intel® Xeon® Processor + Intel MIC Architecture: Writing Code for Today and Tomorrow
- **Intel® Parallel Composer**
  - Seamless Data Parallel: Turning “Small Knobs” to Run on Intel MIC Architecture
  - Array Notations
  - Carry Over Vectorization to Many Cores
- **Intel® Array Building Blocks**
  - C++ Library Front-End to Specify Arbitrary Data-Parallel Computation
  - Virtual Machine API to Create New Front-Ends
- **Programming for Intel MIC Architecture Without Using Offload**
- **Summary and Q&A**

# CPU

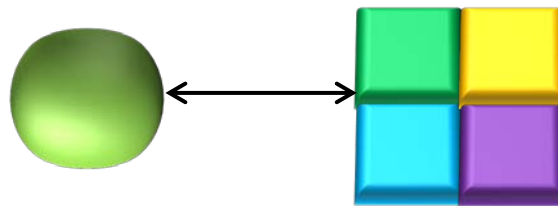
# Parallelizing With Intel® Cilk™ Plus

# Intel® MIC Architecture



# From Intel® Xeon® to Intel® MIC Products: Seamless Data Parallelism

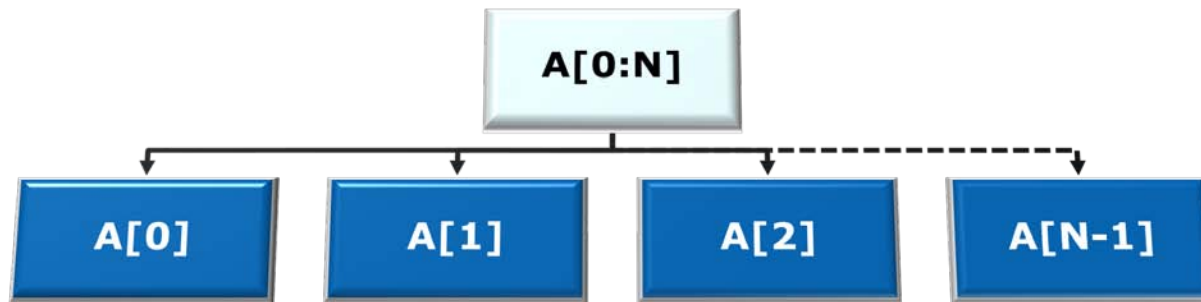
- Write architecture portable vector code with Intel® Cilk™ Plus Extensions
- No knowledge of Intel® Streaming SIMD Extensions, Advanced Vector Extensions or Intel® Many Integrated Core Instructions needed
- Same code can be recompiled to target different architectures



# Intel® Cilk™ Plus Array Notation: Simple and Powerful Vectorization

- Use a ":" in array subscripts to operate on *multiple elements*

```
A[:]           // all of array A
A[lower_bound : length]
A[lower_bound : length : stride]
```



## Examples

```
float A[64];    // Declare C/C++ array A as 64 floats
A[:]           // All elements of array A
A[2:6]         // Elements 2 to 7 of A
A[0:3:2]       // Elements 0,2,4 of A
```

# Operations on Array Sections

- C/C++ operators

```
c[:][:] = a[3:2][3:2] + b[5:2][5:2]; // 2x2 array addition
// sizes and "ranks" in assignments must be the same
```

- Function calls

```
b[:] = foo(a[:]); // Call foo() on each element of a[]
// Need builtin, inlining or elemental function for
// vectorization
```

- Reductions combine array elements to get a single result

```
sum = __sec_reduce_add(a[:]); // Add all elements of a
// many other reductions: all_zero, all_non_zero, max, min,
// etc.
```

- If-then-else and conditional operators allow masked operations

```
if (mask[:]) {
    a[:] = b[:];
}
```

# Elemental Functions

- A different style of writing vector code
- The compiler automatically vectorizes an entire function
- Single data elements are turned into vectors

```
__declspec(vector) float cxy(float a, float x, float y) {  
    return a*x + y; // a, x, y are turned into vectors  
}  
  
float a[64],x[64],y[64],r[64];  
r[:] = cxy(a[:],x[:],y[:]); // Can use a for-loop instead
```

- On Intel® AVX, the above code will generate a `cxy()` that processes 8 elements of the arrays at a time
- The caller will send data to `cxy()` in groups of 8
- Can specify scalar parameters, linear-increasing, etc.

# Data Parallelism with Intel® Cilk™ Plus is Easier!

```
void foo(float *dest, short *src, long len, float a) {  
    dest[0:len] = ((float) src[0:len]) * a;  
}
```

*or*

```
__declspec(vector, scalar(a))  
float foo(float dest, short src, long len, float a) {  
    dest = src * a;  
}
```

```
#include <pmmintrin.h>
```

**VS.**

```
void foo(float* dest, short* src, long len, float a) {  
    __m128 xmmMul = _mm_set1_ps(a);  
  
    for(long i = 0; i < len; i+=4) {  
        __m128i xmmSrc1i = _mm_loadl_epi64((__m128i*) &src[i]);  
        xmmSrc1i = _mm_cvtepi16_epi32(xmmSrc1i);  
        __m128 xmmSrc1f = _mm_cvtepi32_ps(xmmSrc1i);  
        xmmSrc1f = _mm_mul_ps(xmmSrc1f, xmmMul);  
        _mm_store_ps(&dest[i], xmmSrc1f);  
    }  
}
```

# Task Parallelism with Intel® Cilk™ Plus

```
// Example 1
cilk_for (int y = 0; y < height; y++) {
    render_one_line(y);
}
```

```
// Example 2
buffer = cilk_spawn load_image("1.jpg");
buffer2 = load_image("2.jpg");
```

- Create *parallel* for-loops or function calls
- Runtime maps tasks to system threads

# Offloading to Intel® MIC Products:

*Simple, easy, and works with your C/C++ code*

- Use `_Offload_cilk_for`, `_Offload_cilk_spawn`
- Your data-parallel code is optimized for Intel® MIC products and offloaded to the engine
- Compiler handles data transfer between host and “offload space”
  - We also offer keywords for optional user control
- Use our language features to write parallel code, and the tools put it where it needs to go!

# Offloading to Intel® MIC Products:

*Simple, easy, and works with your C/C++ code*

- Use `_Offload_cilk_for`, `_Offload_cilk_spawn`
- Your data-parallel code is optimized for Intel® MIC products and offloaded to the engine
- Compiler handles data transfer between host and “offload space”
  - We also offer keywords for optional user control
- Use our language features to write parallel code, and the tools put it where it needs to go!

# Agenda

- **Overview**
  - Intel® Many Integrated Core Architecture (Intel® MIC Architecture) and Complementing Software Stack
  - Models of Computation
  - Intel® Xeon® Processor + Intel MIC Architecture: Writing Code for Today and Tomorrow
- **Intel® Parallel Composer**
  - Seamless Data Parallel: Turning “Small Knobs” to Run on Intel MIC Architecture
  - Array Notations
  - Carry Over Vectorization to Many Cores
- **Intel® Array Building Blocks**
  - C++ Library Front-End to Specify Arbitrary Data-Parallel Computation
  - Virtual Machine API to Create New Front-Ends
- **Programming for Intel MIC Architecture Without Using Offload**
- **Summary and Q&A**

# Intel® Array Building Blocks (Intel® ArBB)

*Ideal for data parallel programming*



- A generalized vector parallel programming solution
- Ideal for applications requiring data-intensive mathematical computations

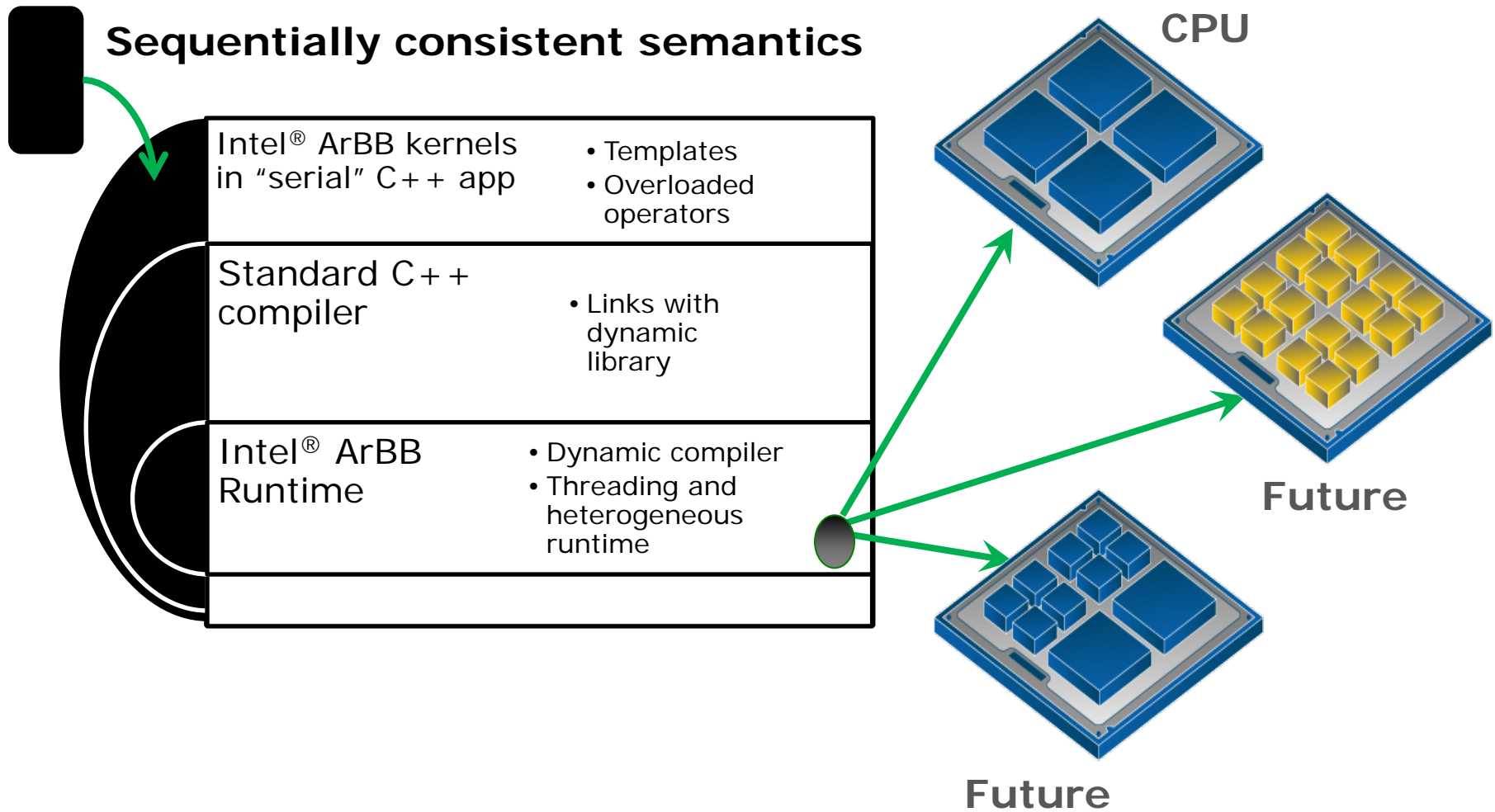
```
//Simple per element multiplication using
Intel® ArBB:

void dotprod(const arbb::dense<arbb::f32>& a,
             const arbb::dense<arbb::f32>& b,
             arbb::dense<arbb::f32>& c)
{
    c = a * b;
}
```

**Intel ArBB is applicable to multicore and many-core programming**

Learn more at <http://intel.com/go/arbb>

# How Does It Work?

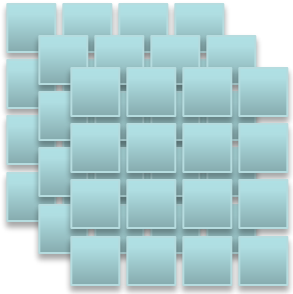


# Containers

## Regular Containers



*dense<T>*



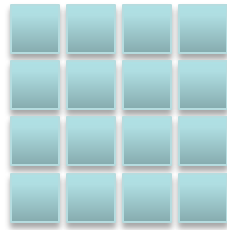
*dense<T,3>*



*array<...>*

```
struct user_type {..};
```

```
Class user_type { };
```



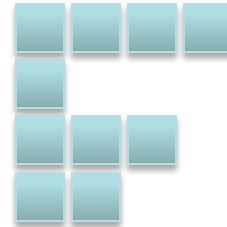
*dense<T, 2>*



*dense<array<...>>*

*dense<user\_type>*

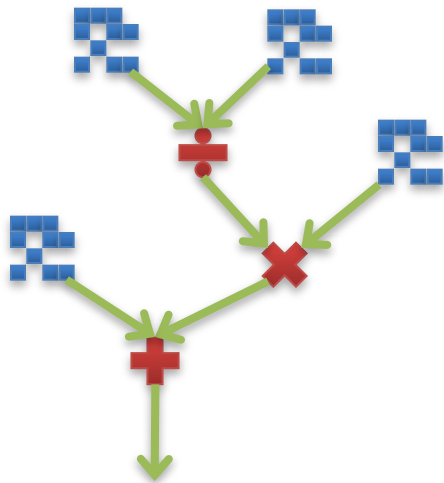
## Irregular Containers



*nested<T>*

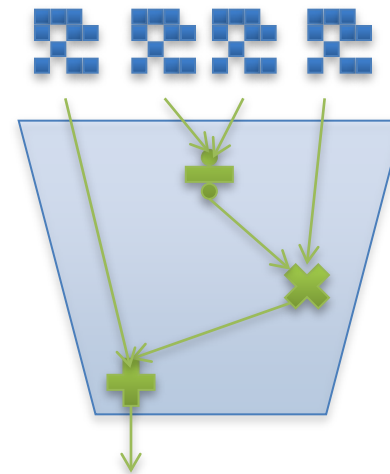
# Vector Processing *or* Scalar Processing

Vector Processing



```
dense<f32> A, B, C, D;  
A = A + B/C * D;
```

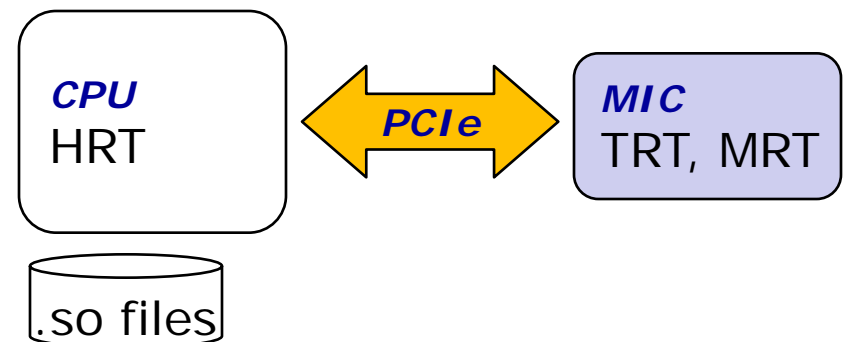
Scalar Processing



```
void kernel(f32& a, f32 b, f32 c, f32  
d) {  
    a = a + (b/c)*d;  
}  
...  
dense<f32> A, B, C, D;  
map(kernel)(A, B, C, D);
```

# Running on Intel® MIC Products?

- Intel® ArBB lib calls in app's main() invoke JIT, which generates target code and invokes the runtime
- Target Host or Intel® MIC product with environment variable
  - MIC: ARBB\_HETEROGENEOUS\_MODEL=OFFLOAD\_ONLY
  - CPU: ARBB\_HETEROGENEOUS\_MODEL=HOST\_ONLY
- JIT generates target code
- ARBB\_NUM\_CORES
  - Specifies the number of SW threads used as workers
  - Default setting is # cores
  - Can change from the API with arbb\_set\_num\_threads
- Heterogeneous runtime (HRT)
  - Moves the MIC .so files for TBB and ArBB at initialization
  - Moves the code and input data per ArBB call
  - Invokes the remote code
  - Moves the output data back
- Remote runtimes
  - Threading (TRT)
  - Memory (MRT)



# Agenda

- **Overview**
  - Intel® Many Integrated Core Architecture (Intel® MIC Architecture) and Complementing Software Stack
  - Models of Computation
  - Intel® Xeon® Processor + Intel MIC Architecture: Writing Code for Today and Tomorrow
- **Intel® Parallel Composer**
  - Seamless Data Parallel: Turning “Small Knobs” to Run on Intel MIC Architecture
  - Array Notations
  - Carry Over Vectorization to Many Cores
- **Intel® Array Building Blocks**
  - C++ Library Front-End to Specify Arbitrary Data-Parallel Computation
  - Virtual Machine API to Create New Front-Ends
- **Programming for Intel MIC Architecture Without Using Offload**
- **Summary and Q&A**

# Programming for Intel® Many Integrated Core Architecture without Offload

## *"Stand-alone" Intel MIC Computing*

- Intel® MIC Architecture software environment includes a highly functional, general purpose OS running on the co-processor with:
  - A familiar Unix\* flavored interactive shell
  - A file system that supports subdirectories, file reads, writes, etc.
  - standard i/o including printf
  - Virtual memory management
  - Process, thread management & scheduling
  - Interrupt and exception handling
  - Semaphores, mutexes, etc...
- What does this mean?
  - A large majority of existing code even with OS oriented calls like fork() can port with a simple recompile
  - Intel MIC Architecture natively supports parallel coding models like Intel® Cilk™ Plus, Intel® Threading Building Blocks, pThreads\*, OpenMP\*

foeey.c

```
main( )
{
    printf("running Foo()\n");
    Foo( );
}

Foo()
{
    printf("foeey\n");
}
```

Intel MIC Architecture  
(Knights Corner console)

```
mymic>ls
foeey
mymic>./foeey
running Foo()
foeey
mymic>
```

# Summary

## *What Makes a Great Model for Data Parallel?*

### **Composable**

- **Portable**
- **Performance**
- **Safety**

# There are Many Parallel Programming Models for C, C++ and Fortran

*If you write good code, it can be applied to both Intel® Xeon® and Intel® Many Integrated Core Architecture*

## Intel® Parallel Building Blocks

### Intel® Cilk™ Plus

Language extensions to simply parallelism

### Intel® Threading Building Blocks

Widely used C++ template library for parallelism

### Intel® Array Building Block

Powerful C++ Library for data parallelism

Mix and match to optimize your applications performance

## Domain-Specific Libraries

Intel® Integrated Performance Primitives

Intel® Math Kernel Library

## Established Standards

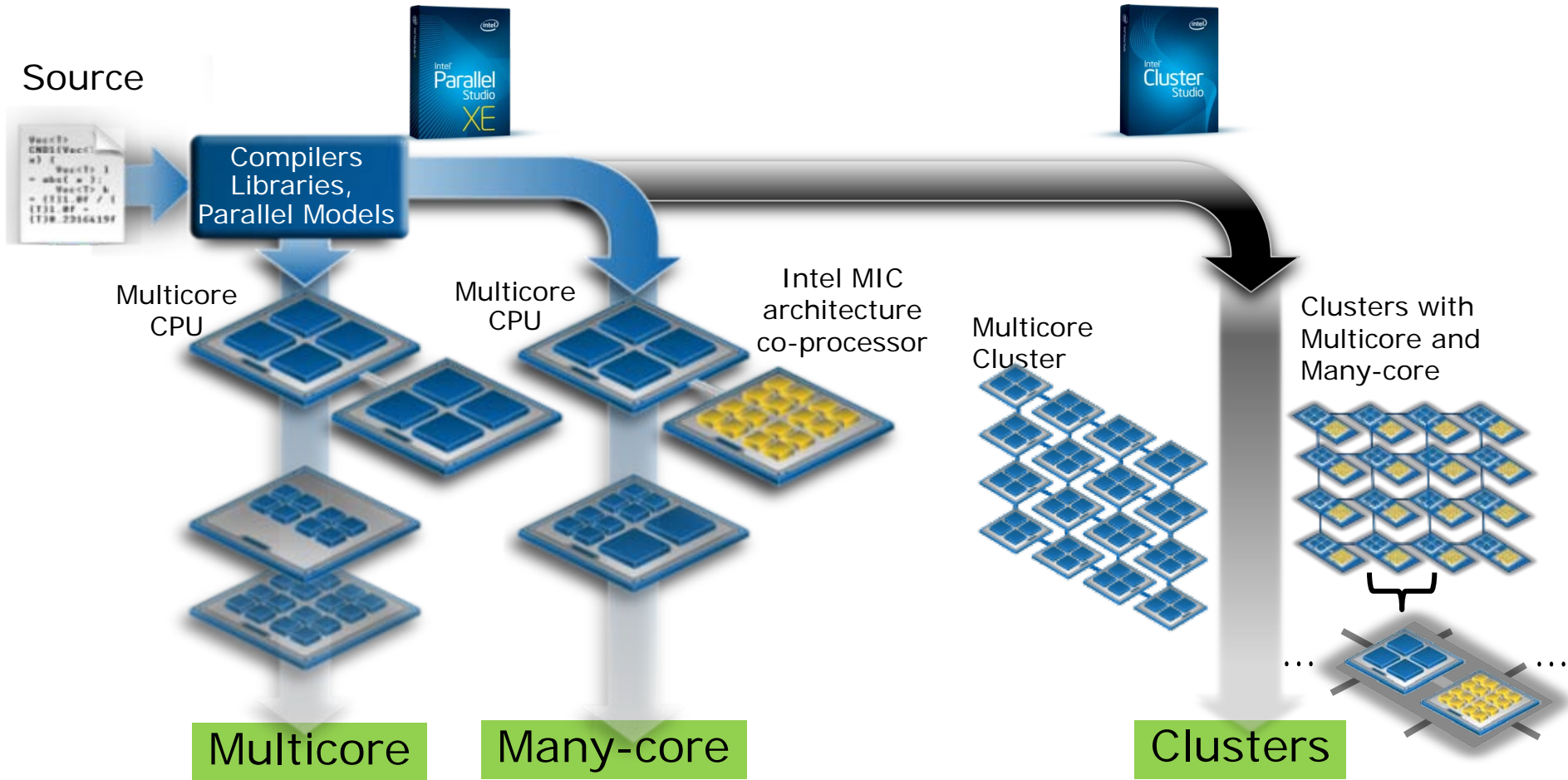
Intel® Message Passing Interface (MPI) Library

OpenMP\*

Fortran Coarray

OpenCL\*

# Program for Multicore Today... Be Ready for Intel® Many Integrated Core Products



Eliminate Need for Dual Programming Architecture

# Session, Lab, and Demo Schedule

## Tuesday—September 13

**Demo—Intel® IDF Technology Showcase**

Time: 5:00 p.m.—7:00 p.m.

Location: Software Community area of the Technology Showcase

**Session - A Three-Pronged Approach to Improving Software Stability Using Intel® Software Correctness Tools**

Time: 11:20 a.m.—12:10 p.m.

Location: Room 2011

Presenter: David Mackay

**Intel® Cluster Checker Tool – An Automated Approach and Extensible Tool for Analyzing High-Performance Computing Clusters**

Time: 1 p.m. –4:00 p.m.

Location: Room 2000

Presenter: Jeremy Siadal

**Task Parallel Evolution and Revolution—Intel® Cilk™ Plus and Intel® Threading Building Blocks**

Time: 2:10 p.m.—3:00 p.m.

Location: Room 2011

Presenters: Noah Clemons and Victoria Gromova

**Program the SAME, Here and over There—Intel® Data Parallel Programming Models with Respect to Manycore**

Time: 3:20 p.m.—4:10 p.m.

Location: Room 2011

Presenters: Noah Clemons and Chang-Sun Lin, Jr.

## Wednesday—September 14

**Demo—Intel® IDF Technology Showcase**

Time: 11 a.m.—1 p.m. and 4 p.m.—7 p.m.

Location: Software Community area of the Technology Showcase

**Embedded System Tools for Development and Validation of Intel® Atom™ Processor-Based Devices**

Time: 11:20 a.m.—12:10 p.m.

Location: Room 2011

Presenter: Robert Mueller

**Intel® Cluster Checker Tool – An Automated Approach and Extensible Tool for Analyzing High-Performance Computing Clusters**

Time: 1 p.m.—4:00 p.m.

Location: Room 2000

Presenter: Jeremy Siadal

**Intel® Faces of Parallelism Lab: Parallel Models for Multi/Many Core**

Time: 1:05 p.m.—5:05 p.m.

Location: Room 2010

Presenters: Noah Clemons

## Thursday—September 15

**Demo—Intel® IDF Technology Showcase**

Time: 11 a.m.—2 p.m.

Location: Software Community area of the Technology Showcase

**Intel® Cluster Checker Tool – An Automated Approach and Extensible Tool for Analyzing High-Performance Computing Clusters**

Time: 1 p.m.—4:00 p.m.

Location: Room 2000

Presenter: Jeremy Siadal

**Performance Profiling Secrets: The new Intel® VTune™ Amplifier XE for Beginning and Experienced Tuners**

Time: 2:05 p.m.—2:55 p.m.

Location: Room 2011

Presenter: Shannon Cepeda

**Parallel Programming Methods from an Intel and Microsoft\* Perspective**

Time: 2:05 p.m.—2:55 p.m.

Location: Room 2009

Presenters: James Reinders (Intel) and Steve Teixeira (Microsoft)

# 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.
- UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.
- 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>.
- 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.
- Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. Go to: [http://www.intel.com/products/processor\\_number](http://www.intel.com/products/processor_number).
- Intel product plans in this presentation do not constitute Intel plan of record product roadmaps. Please contact your Intel representative to obtain Intel's current plan of record product roadmaps.
- Knights Corner 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.
- Intel, Xeon, Cilk, Sponsors of Tomorrow 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 ©2011 Intel Corporation.

# Risk Factors

The above statements and any others in this document that refer to plans and expectations for the second quarter, the year and the future are forward-looking statements that involve a number of risks and uncertainties. Words such as “anticipates,” “expects,” “intends,” “plans,” “believes,” “seeks,” “estimates,” “may,” “will,” “should,” and their variations identify forward-looking statements. Statements that refer to or are based on projections, uncertain events or assumptions also identify forward-looking statements. Many factors could affect Intel’s actual results, and variances from Intel’s current expectations regarding such factors could cause actual results to differ materially from those expressed in these forward-looking statements. Intel presently considers the following to be the important factors that could cause actual results to differ materially from the company’s expectations. Demand could be different from Intel’s expectations due to factors including changes in business and economic conditions, including supply constraints and other disruptions affecting customers; customer acceptance of Intel’s and competitors’ products; changes in customer order patterns including order cancellations; and changes in the level of inventory at customers. Potential disruptions in the high technology supply chain resulting from the recent disaster in Japan could cause customer demand to be different from Intel’s expectations. Intel operates in intensely competitive industries that are characterized by a high percentage of costs that are fixed or difficult to reduce in the short term and product demand that is highly variable and difficult to forecast. Revenue and the gross margin percentage are affected by the timing of Intel product introductions and the demand for and market acceptance of Intel’s products; actions taken by Intel’s competitors, including product offerings and introductions, marketing programs and pricing pressures and Intel’s response to such actions; and Intel’s ability to respond quickly to technological developments and to incorporate new features into its products. The gross margin percentage could vary significantly from expectations based on capacity utilization; variations in inventory valuation, including variations related to the timing of qualifying products for sale; changes in revenue levels; product mix and pricing; the timing and execution of the manufacturing ramp and associated costs; start-up costs; excess or obsolete inventory; changes in unit costs; defects or disruptions in the supply of materials or resources; product manufacturing quality/yields; and impairments of long-lived assets, including manufacturing, assembly/test and intangible assets. Expenses, particularly certain marketing and compensation expenses, as well as restructuring and asset impairment charges, vary depending on the level of demand for Intel’s products and the level of revenue and profits. The majority of Intel’s non-marketable equity investment portfolio balance is concentrated in companies in the flash memory market segment, and declines in this market segment or changes in management’s plans with respect to Intel’s investments in this market segment could result in significant impairment charges, impacting restructuring charges as well as gains/losses on equity investments and interest and other. Intel’s results could be affected by adverse economic, social, political and physical/infrastructure conditions in countries where Intel, its customers or its suppliers operate, including military conflict and other security risks, natural disasters, infrastructure disruptions, health concerns and fluctuations in currency exchange rates. Intel’s results could be affected by the timing of closing of acquisitions and divestitures. Intel’s results could be affected by adverse effects associated with product defects and errata (deviations from published specifications), and by litigation or regulatory matters involving intellectual property, stockholder, consumer, antitrust and other issues, such as the litigation and regulatory matters described in Intel’s SEC reports. An unfavorable ruling could include monetary damages or an injunction prohibiting us from manufacturing or selling one or more products, precluding particular business practices, impacting Intel’s ability to design its products, or requiring other remedies such as compulsory licensing of intellectual property. A detailed discussion of these and other factors that could affect Intel’s results is included in Intel’s SEC filings, including the report on Form 10-Q for the quarter ended April 2, 2011.

Rev. 5/9/11