



# PROGRAMMING AND OPTIMIZATION FOR INTEL<sup>®</sup> ARCHITECTURE

The Hands-On Workshop (HOW) Series  
Session 4

*Colfax International* — [colfaxresearch.com](http://colfaxresearch.com)

September 2016

While best efforts have been used in preparing this training, Colfax International makes no representations or warranties of any kind and assumes no liabilities of any kind with respect to the accuracy or completeness of the contents and specifically disclaims any implied warranties of merchantability or fitness of use for a particular purpose. The publisher shall not be held liable or responsible to any person or entity with respect to any loss or incidental or consequential damages caused, or alleged to have been caused, directly or indirectly, by the information or programs contained herein. No warranty may be created or extended by sales representatives or written sales materials.

- ▶ **HOW to Program Intel Architecture**
  - 01. Parallelism, specialization, guided tour – Sep 26
  - 02. Programming Intel Xeon Phi (KNC, KNL) – Sep 27
- ▶ **HOW to Express Parallelism**
  - 03. Automatic vectorization – Sep 28
  - 04. Multi-threading with OpenMP – Sep 29
- ▶ **HOW to Get Performance**
  - 05. Comprehensive demo – Sep 30
  - 06. Scalar & vectorization tuning – Oct 3
  - 07. Multi-threading: common issues – Oct 4
  - 08. Multi-threading: memory aspect – Oct 5
  - 09. Memory traffic – Oct 6
- ▶ **HOW to Scale**
  - 10. Distributed Computing: MPI – Oct 7

September 2016						
S	M	T	W	H	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

October 2016						
S	M	T	W	H	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

■ — Webinar+remote access

Course page: [colfaxresearch.com/how-16-09](http://colfaxresearch.com/how-16-09)

- ▶ Slides (including this one), code downloads
- ▶ Video of recorded sessions
- ▶ Chat (during webinars or offline)



Additional resources:

- ▶ More workshops like this one: [colfaxresearch.com/training](http://colfaxresearch.com/training)
- ▶ Video courses: [colfaxresearch.com/video-courses](http://colfaxresearch.com/video-courses)

# GET YOUR QUESTIONS ANSWERED

## Chat (current):

[colfaxresearch.com/how-16-09](http://colfaxresearch.com/how-16-09)



## Forums (technical):

[colfaxresearch.com/discussion](http://colfaxresearch.com/discussion)

### COLFAX RESEARCH

CONTRIBUTING TO INNOVATIONS IN COMPUTING

[Log In/Register](#)

[/](#) [READ](#) [WATCH](#) [LEARN](#) [FORUMS](#) [CONNECT](#) [JOIN](#)

#### Join the Conversation

Welcome to Colfax Research forums, an online community for you to engage with HPC experts, software architects, developers, computational researchers, scientists, students and more—so you can acquire new knowledge, share ideas, and build new relationships.

Tap our experts and your peers to help meet the challenge of optimizing applications on modern hardware. This is the place to browse or post questions (and get answers) related to computational science, parallel programming and code modernization on Intel® Architecture.

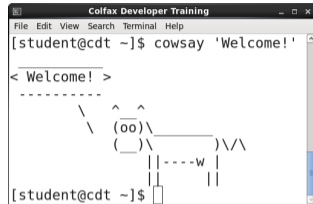
Welcome aboard. Post questions today!

## Email (organizational):

[training@colfax-intl.com](mailto:training@colfax-intl.com)

# HANDS-ON EXERCISES AND REMOTE ACCESS

- ▶ 96 people receive a remote access token
- ▶ Virtualized Intel Xeon CPU, real Intel Xeon Phi coprocessor (1st gen, KNC), SW tools
- ▶ Can access the system the entire 2 weeks of the workshop



```
Colfax Developer Training
File Edit View Search Terminal Help
[student@cdt ~]$ cowsay 'Welcome!'
< Welcome! >
-----
      \   ^__^
         (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||

[student@cdt ~]$
```

- ▶ Not among the 96? Stay tuned: follow along with instructor, use own system, or wait for a seat
- ▶ Use it or lose it: if you do not log in for a while, remote access token goes to next student on the list



## **§2. EXPRESSING TASK PARALLELISM**



## **HANDLING MULTIPLE CORES**

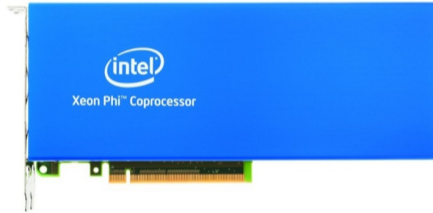
## Intel Xeon Processor



Current: Broadwell  
Upcoming: Skylake

Multi-Core Architecture

## Intel Xeon Phi Coprocessor, 1st generation



Knights Corner (KNC)

## Intel Xeon Phi Processor, 2nd generation\*

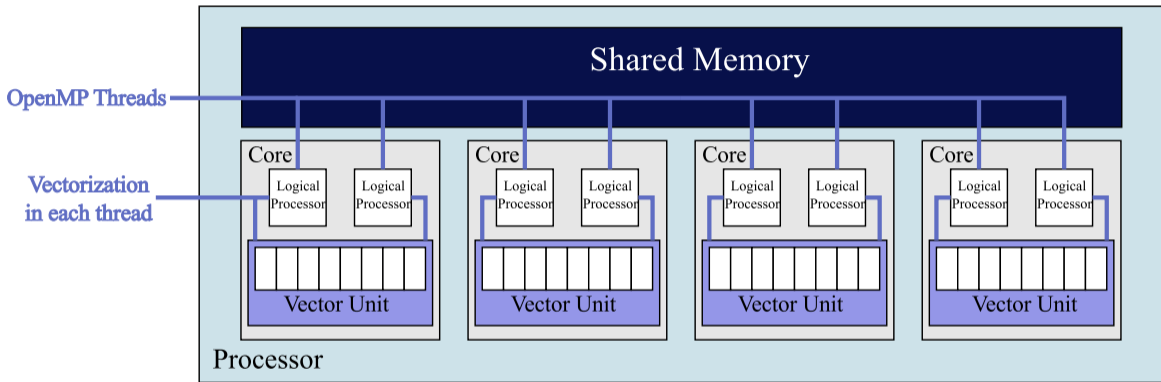


\* socket and coprocessor versions

Knights Landing (KNL)

Intel Many Integrated Core (MIC) Architecture

# CO-EXISTENCE WITH VECTORS



**Utilize cores:** run multiple threads/processes (MIMD)

**Utilize vectors:** each thread (process) issues vector instructions (SIMD)

# SIMULTANEOUS THREADING AND VECTORIZATION

This approach often works:

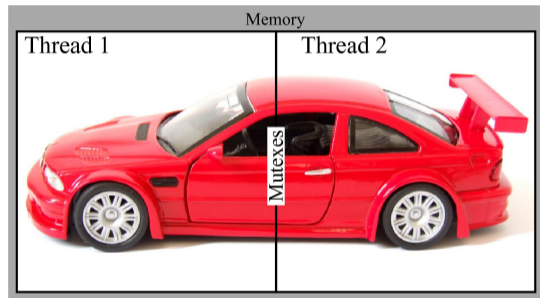
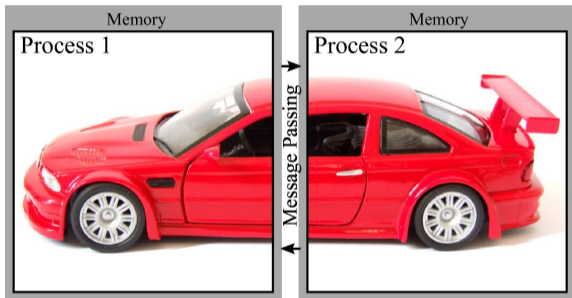
```
1 #pragma omp parallel for
2 for (int i = 0; i < n; i++) // Thread parallelism in outer loop
3 #pragma simd
4   for (int j = 0; j < m; j++) // Vectorization in inner loop
5     DoSomeWork(A[i][j]);
```

That works as well:

```
1 #pragma omp parallel for simd
2 for (int i = 0; i < n; i++) // If the problem is all data-parallel
3   DoSomeWork(A[i]);
```

# THREADS VERSUS PROCESSES

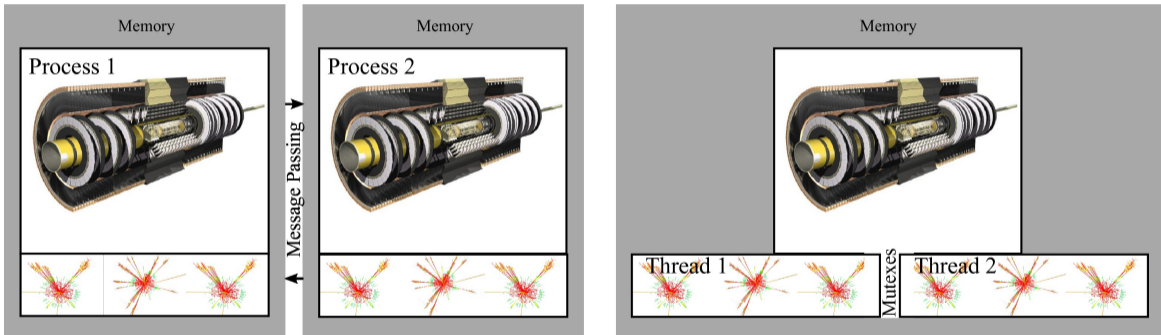
Option 1: Partitioning data set between threads/processes



Examples: computational fluid dynamics (CFD), image processing.

# THREADS VERSUS PROCESSES

## Option 2: Sharing data set between threads/processes



Examples: particle transport simulation, machine learning (inference).

# SCALABILITY EXPECTATIONS (CPU)

$T$  = number of threads

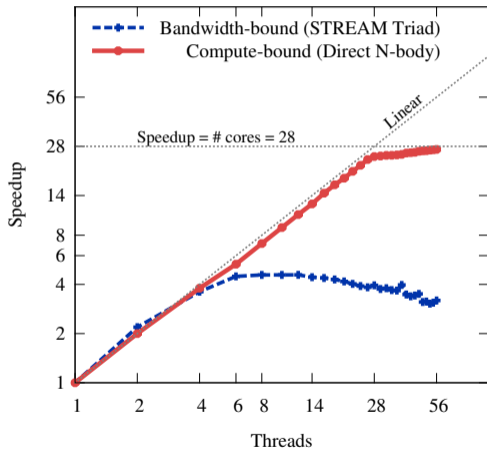
$$\text{Speedup}(T) = \frac{\text{Performance}(T)}{\text{Performance}(1)}$$

$$\text{Efficiency}(T) = \frac{\text{Speedup}(T)}{T}$$

Linear scaling (ideal case, 100% parallel efficiency):

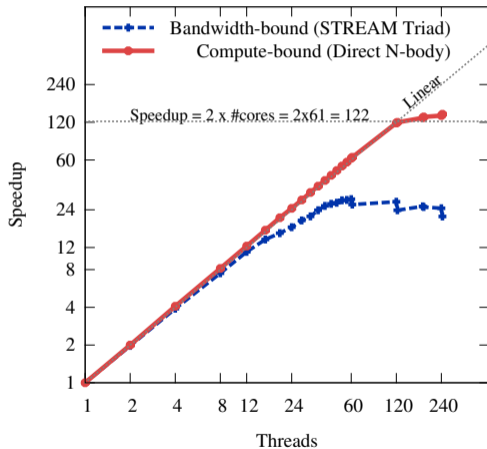
$$\text{Speedup}(T) = T$$

Performance on the CPU architecture

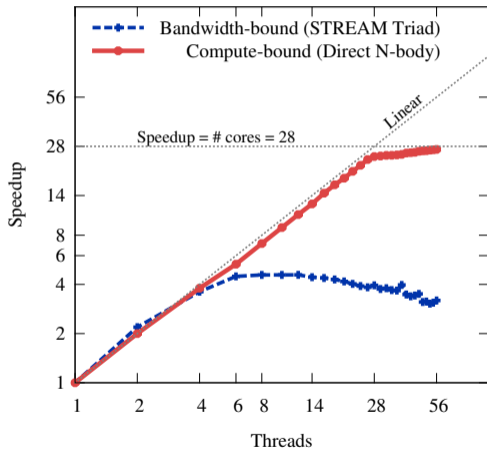


# SCALABILITY EXPECTATIONS: MIC VERSUS CPU

## Performance on the MIC architecture



## Performance on the CPU architecture



# THREADING FRAMEWORKS

Framework	Implementation	Complexity	Functionality
POSIX Threads	Various	Simple	Manually control everything
Cilk Plus	Intel, Public	Very simple	Automatic loops and tasks, no user control
TBB	Intel, Public	Complex	Automatic trees of tasks, au- tomatic scheduler
OpenMP	Various	Simple to Complex	HPC-specific functional- ity, automatic and manual control possible



# OPENMP BASICS

# "HELLO WORLD" OPENMP PROGRAM

```
1  #include <omp.h>
2  #include <stdio.h>
3
4  int main(){
5      // This code is executed by only 1 thread
6      const int nt=omp_get_max_threads();
7      printf("OpenMP with %d threads\n", nt);
8
9      #pragma omp parallel
10     {
11         // This code is executed in parallel
12         // by multiple threads
13         printf("Hello World from thread %d\n",
14                omp_get_thread_num());
15     }
16 }
```

- ▶ OpenMP = “Open Multi-Processing” = computing-oriented framework for shared-memory programming
- ▶ Threads – streams of instructions that share memory address space
- ▶ Distribute threads across CPU cores for parallel speedup

# COMPILING THE "HELLO WORLD" OPENMP PROGRAM

```
vega@lyra% icpc -qopenmp hello_omp.cc
vega@lyra% export OMP_NUM_THREADS=5
vega@lyra% ./a.out
OpenMP with 5 threads
Hello World from thread 0
Hello World from thread 3
Hello World from thread 1
Hello World from thread 2
Hello World from thread 4
```

OMP\_NUM\_THREADS controls number of OpenMP threads (default: logical CPU count)

# CONTROL OF VARIABLE SHARING

Method 1: using clauses in pragma omp parallel (C, C++, Fortran):

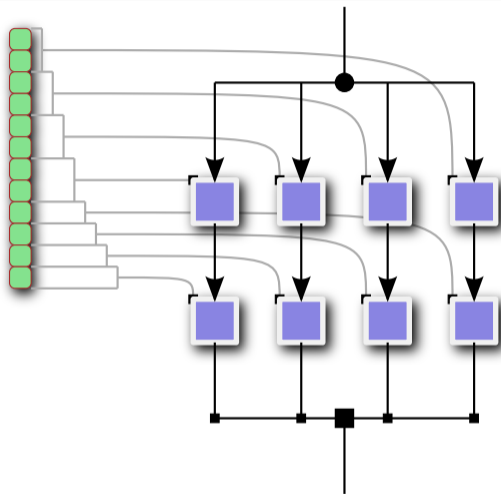
```
1 int A, B; // Variables declared at the beginning of a function
2 #pragma omp parallel private(A) shared(B)
3 {
4     // Each thread has its own copy of A, but B is shared
5 }
```

Method 2: using scoping (only C and C++):

```
1 int B; // Variable declared outside of parallel scope - shared by default
2 #pragma omp parallel
3 {
4     int A; // Variable declared inside the parallel scope - always private
5     // Each thread has its own copy of A, but B is shared
6 }
```

# LOOP-CENTRIC PARALLELISM: FOR-LOOPS IN OPENMP

- ▶ Simultaneously launch multiple threads
- ▶ Scheduler assigns loop iterations to threads
- ▶ Each thread processes one iteration at a time



Parallelizing a for-loop.

# LOOP-CENTRIC PARALLELISM: FOR-LOOPS IN OPENMP

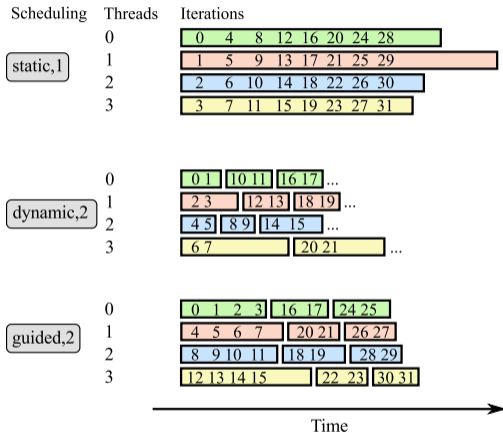
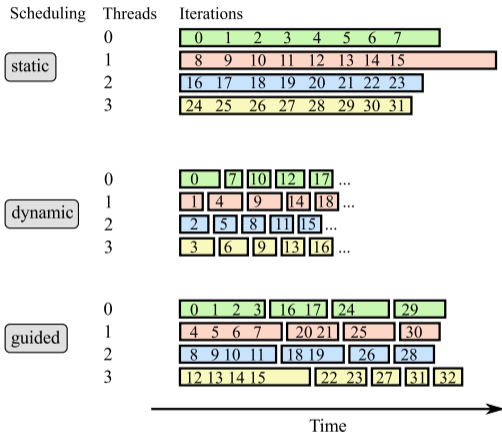
The OpenMP library will distribute the iterations of the loop following the `#pragma omp parallel for` across threads.

```
1  #pragma omp parallel for  
2  for (int i = 0; i < n; i++) {  
3      printf("Iteration %d is processed by thread %d\n",  
4          i, omp_get_thread_num());  
5      // ... iterations will be distributed across available threads...  
6  }
```

# LOOP-CENTRIC PARALLELISM: FOR-LOOPS IN OPENMP

```
1  #pragma omp parallel
2  {
3      // Code placed here will be executed by all threads.
4
5      // Alternative way to specify private variables:
6      // declare them in the scope of pragma omp parallel
7      int private_number=0;
8
9  #pragma omp for
10     for (int i = 0; i < n; i++) {
11         // ... iterations will be distributed across available threads...
12     }
13     // ... code placed here will be executed by all threads
14 }
```

# LOOP SCHEDULING MODES IN OPENMP





# THREAD SYNCHRONIZATION

# RACE CONDITIONS AND UNPREDICTABLE PROGRAM BEHAVIOR

```

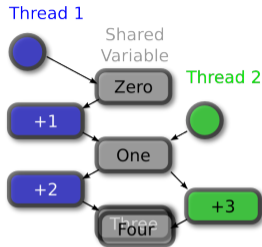
1  #include <omp.h>
2  #include <stdio>
3  int main() {
4      const int n = 1000;
5      int total = 0;
6      #pragma omp parallel for
7      for (int i = 0; i < n; i++) {
8          total = total + i; // Race condition
9      }
10     printf("total=%d (must be %d)\n", total,
11            ((n-1)*n)/2);
12 }

```

```

vega@lyra% icpc -o app omp-race.cc -qopenmp
vega@lyra% ./app
total=208112 (must be 499500)

```



**Race Condition!**

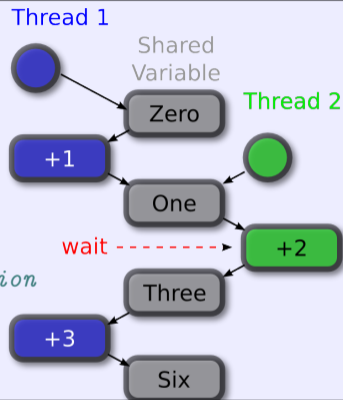
- Occurs when 2 or more threads access the same memory address, and at least one of these accesses is for writing

# PROTECTING RACE CONDITIONS WITH A CRITICAL SECTION

```

1 #include <omp.h>
2 #include <stdio.h>
3 int main() {
4     const int n = 1000;
5     int total = 0;
6     #pragma omp parallel for
7     for (int i = 0; i < n; i++) {
8     #pragma omp critical
9         { // Only one thread at a time can execute this section
10            total = total + i;
11        }
12    } }

```



```

vega@lyra% icpc -o omp-critical omp-critical.cc -qopenmp
vega@lyra% ./omp-critical
total=499500 (must be 499500)

```

# AVOIDING RACES WITH ATOMIC OPERATIONS

This parallel fragment of code has predictable behavior, because the race condition was eliminated with *an atomic operation*:

```
1  #pragma omp parallel for
2  for (int i = 0; i < n; i++)
3  { // Lightweight synchronization
4  #pragma omp atomic
5     total += i;
6  }
```

# LIMITATIONS OF ATOMIC OPERATIONS

**Read** : operations in the form  $v = x$

**Write** : operations in the form  $x = v$

**Update** : operations in the form  $x++$ ,  $x--$ ,  $--x$ ,  $++x$ ,  $x \text{ binop} = \text{expr}$   
and  $x = x \text{ binop} \text{ expr}$

**Capture** : operations in the form  $v = x++$ ,  $v = x--$ ,  $v = -x$ ,  $v = ++x$ ,  
 $v = x \text{ binop} \text{ expr}$

- ▶ Here  $x$  and  $v$  are scalar variables
- ▶  $\text{binop}$  is one of  $+$ ,  $*$ ,  $-$ ,  $- /$ ,  $\&$ ,  $\wedge$ ,  $|$ ,  $\ll$ ,  $\gg$ .
- ▶ No “trickery” is allowed for atomic operations:
  - no operator overload,
  - no non-scalar types,
  - no complex expressions.



# PARALLEL REDUCTION

# REDUCTION CLAUSE IN PARALLEL REGION

```
1 #include <omp.h>
2 #include <stdio>
3
4 int main() {
5     const int n = 1000;
6     int total = 0;
7     #pragma omp parallel for reduction(+: total)
8     for (int i = 0; i < n; i++) {
9         total = total + i;
10    }
11    printf("total=%d (must be %d)\n", total, ((n-1)*n)/2);
12 }
```

```
vega@lyra% icpc -o omp-reduction omp-reduction.cc -qopenmp
vega@lyra% ./omp-reduction
total=499500 (must be 499500)
```

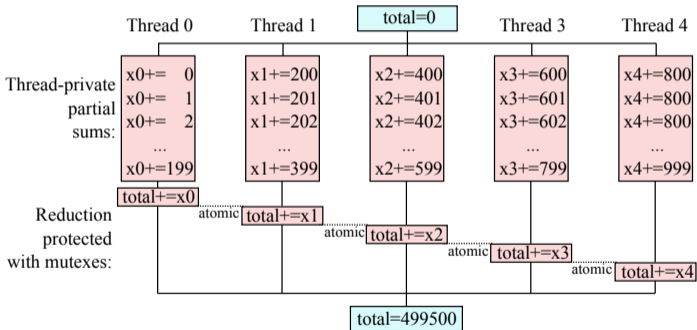
# AVOIDING RACES WITH THREAD-PRIVATE STORAGE

Correct and efficient code:

```

1  int total = 0;
2  #pragma omp parallel
3  {
4      int total_thr = 0;
5      #pragma omp for
6      for (int i=0; i<n; i++)
7          total_thr += i;
8
9      #pragma omp atomic
10     total += total_thr;
11
12 }

```

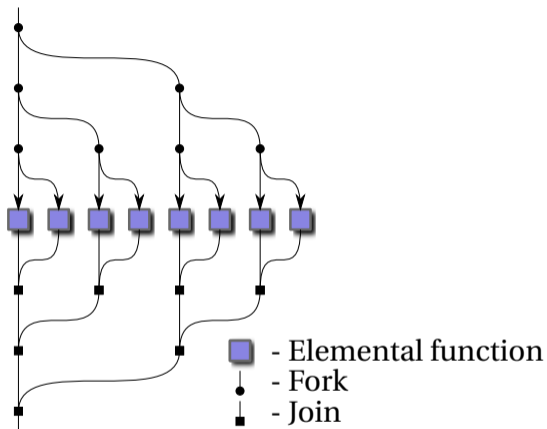




## TASKS IN OPENMP

# FORK-JOIN MODEL OF PARALLEL EXECUTION

- ▶ Each thread can spawn daughter threads
- ▶ Available threads pick up queued tasks
- ▶ Expresses algorithms that cannot be expressed in the loop model (e.g., parallel recursion)



Fork-join model of parallel execution.

(#pragma omp task functionality)

# TASKS IN OPENMP: EXAMPLE

```
1 // Starting the first task:
2 #pragma omp parallel
3 { // Enter a parallel region
4 #pragma omp single
5   { // Start the first task
6     // from only one thread
7     RecursiveWorkload(args);
8   }
9 }
```

```
1 // Recursive task spawning:
2 void RecursiveWorkload(Arg* args) {
3   if (args->size > threshold) {
4     // Split work
5     Arg* args1=args->FirstHalf();
6     Arg* args2=args->SecondHalf();
7
8     // Parallel divide-and-conquer
9     #pragma omp task firstprivate(args1)
10    { RecursiveWorkload(args1); }
11    #pragma omp task firstprivate(args2)
12    { RecursiveWorkload(args2); }
13  } else {
14    // End of recursion
15    args->ProcessSmallestSubTask();
16  }
17 }
```



## **RECIPES FOR SUCCESS**

# RECIPE FOR SUCCESS: "TAKE A DEEP BREATH"

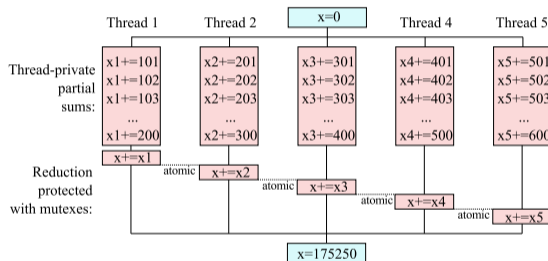
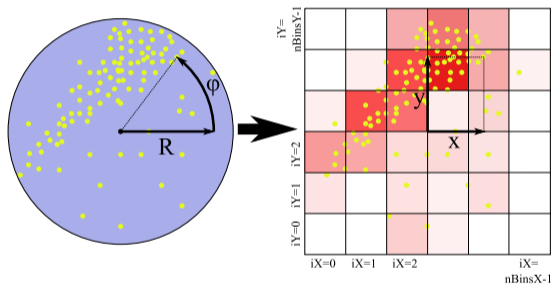
Where is parallelism in your application?



See also full interview with James Reinders at [Colfax Research](#)

# SUGGESTED ADDITIONAL READING

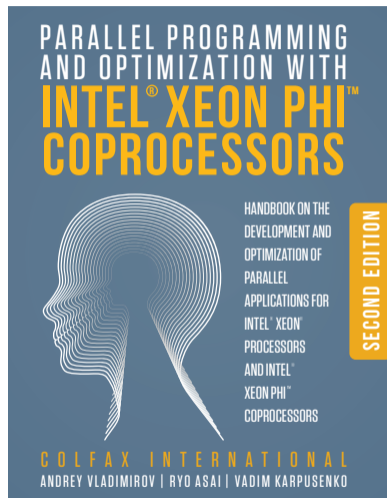
## Colfax Research tutorial on multi-threading in a binning code



<http://colfaxresearch.com/?p=6>

# ADDITIONAL MATERIALS ON OPENMP

1. OpenMP Specifications
2. Intel's OpenMP Video Course
3. LLNL tutorial: OpenMP
4. Book: “Parallel Programming and Optimization with Intel Xeon Phi Coprocessors” by Colfax.



# SUMMARY

Discussed today:

- ▶ Cores can run independent programs
- ▶ Use threads to scale across cores
- ▶ OpenMP – well-established parallel framework for HPC
- ▶ Data races lead to incorrect, unpredictable results
- ▶ Mutexes control data races at cost of performance
- ▶ Co-exist with have vectorization in each thread

Next session: introduction into optimization for Xeon and Xeon Phi.



**LEARN MORE**

# HOW SERIES: KNIGHTS LANDING

HOW SERIES "KNIGHTS LANDING":

PROGRAMMING AND OPTIMIZATION FOR  
INTEL XEON PHI X200 FAMILY

Free 2-hour video course



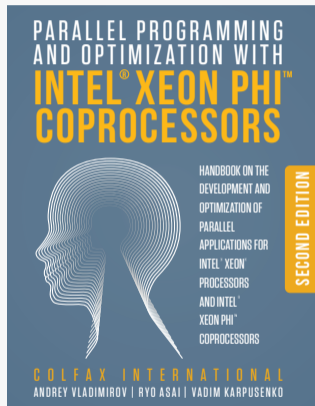
[colfaxresearch.com/how-knl/](http://colfaxresearch.com/how-knl/)

ISBN: 978-0-9885234-0-1 (508 pages, Electronic or Print)

Parallel Programming  
and Optimization with  
Intel® Xeon Phi™  
Coproprocessors

Handbook on the Development and  
Optimization of Parallel Applications  
for Intel® Xeon® Processors  
and Intel® Xeon Phi™ Coprocessors

© Colfax International, 2015



<http://xeonphi.com/book>

**COLFAX RESEARCH**
Log In/Out or Register

READ WATCH LEARN CONNECT JOIN

To search, type and hit enter

---

**Popular**

**The Hands-On Tutorials (HOT) webinars: details on efficient programming for Intel architecture**

**The Hands-On Workshop (HOW) Series**

**Introduction to Intel DAAL, Part 1: Polynomial Regression with Batch Mode Computation**

**Parallel Programming Book**

Introduction to parallel programming, deep discussion of optimization techniques, exercises. © 2015, Colfax International, 508 pages.

**Featured Video**

See Research material on vectorization in a training video

**Events**

**Presentations**

**Cardview**



**Introduction to Intel DAAL, Part 1: Polynomial Regression with Batch Mode Computation**

**Research and Educational Publications**

**Introduction to Intel DAAL, Part 1: Polynomial Regression with Batch Mode Computation**

**Software Developer's Introduction to the HGST Ultrastar Archive H700 SMR Drives**

**Optimization Techniques for the Intel MIC Architecture, Part 1 of 3: Multi-Threading and Parallel Reduction**


**Optimization Techniques for the Intel MIC Architecture, Part 3 of 3: False Sharing and Padding**

**Optimization Techniques for the Intel MIC Architecture, Part 2 of 3: Strip-Mining for Vectorization**

**Performance to Power and Performance to Cost Ratios with Intel Xeon Phi Coprocessors (and why ix Acceleration May Be Enough)**

## Consulting

Facebook
Twitter
LinkedIn
Google+
Share



**Colfax offers consulting services for enterprises, research help you:**


- Optimize your existing application to take advantage of parallelism, from vectors to cores to clusters and
- Future-proof your application for upcoming innovations
- Accelerate your application using coprocessor tech
- Investigate the potential system configurations that satisfy your cost, power, performance requirements.
- Take a clean slate to develop a novel approach to reduce your computing pro

**Episode 2.1 — Purpose of the MIC architecture**

**Parallel Computing in the Search for New Physics at LHC**

## Software Developer's Introduction to the HGST Ultrastar Archive H700 SMR Drives

Facebook
Twitter
LinkedIn
Google+
Share




**Configuration and Benchmarks of Peer-to-Peer Communication over Gigabit Ethernet and InfiniBand in a Cluster with Intel Xeon Phi Coprocessors**

**Interview with James Reinders: future of Intel MIC architecture, parallel programming, education**

**Parallel Computing in the Search for New Physics at LHC**

## Fluid Dynamics with Fortran on Intel Xeon Phi coprocessors

Facebook
Twitter
LinkedIn
Google+
Share



**Configuration and Benchmarks of Peer-to-Peer Communication over Gigabit Ethernet and InfiniBand in a Cluster with Intel Xeon Phi Coprocessors**

**Interview with James Reinders: future of Intel MIC architecture, parallel programming, education**

**Parallel Computing in the Search for New Physics at LHC**

http://colfaxresearch.com/