



# Programming and Optimization for Intel<sup>®</sup> Architecture

The Hands-On Workshop (HOW) Series

Colfax International — @colfaxintl

May 2016 , Rev. 02c

# About This Document

This document represents the materials of a Web-based training “Programming and Optimization with Intel Architecture” developed and run by Colfax International.

© Colfax International, 2013–2016

Parallel Programming Boot Camp (1-Day) / Workshop (4-Days)



Instructor-led 1-day or 4-days training, at your office or at Colfax facility in Sunnyvale, CA

[Click here to learn more](#)

**1-Day Parallel Programming Boot Camp**  
 For software engineers and architects, providing an overview of parallel programming frameworks and optimization guidelines for multi-core CPUs (Intel® Xeon®) and many-core coprocessors (Intel® Xeon Phi™):

- Discussions about three layers of parallelism: SIMD, Threads, Cluster environment
- Tips for quick porting/development of HPC software applications
- Real-life examples of code and optimization techniques
- Hardware solution and corresponding software implementations, APIs, and frameworks

**4-Days Parallel Programming Workshop**  
 For the developer who wants to hit the ground running with the modern multi-core CPUs (Intel® Xeon®), many-core coprocessors (Intel® Xeon Phi™) and leading software development tools:

- Hardware installation
- MPSS tools and the Linux environment on the Intel® Xeon Phi™ coprocessor
- Exploring differences in serial vs. parallel programming / processing / hardware usage
- Accelerated clusters
- Optimizations of vector arithmetics, memory traffic, thread parallelism and communication
- Using the Intel® Math Kernel Library

Register Now!

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

# Disclaimer

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.

# Course Roadmap

- 1 Why Intel Parallel Architectures?
  - ▶ Parallelism and specialization – May 23
  - ▶ Programming model continuity – May 23
- 2 Programming models for Xeon Phi coprocessors
  - ▶ Native programming – May 23
  - ▶ Offload programming – May 24
- 3 Expressing Parallelism
  - ▶ Introduction to vectorization – May 25
  - ▶ Crash-course on OpenMP – May 26
- 4 Optimization – intro on April May 27
  - ▶ Vectorization tuning – May 30
  - ▶ Multi-threading – May 31, June 1
  - ▶ Memory traffic – June 2
- 5 Distributed Computing: MPI – June 3

May 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				

June 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		

— Lecture+remote access

# HOW Online

Course page: [colfaxresearch.com/how-16-05](http://colfaxresearch.com/how-16-05)

- 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

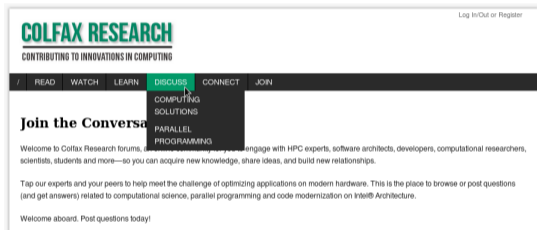
## Chat (current):

[colfaxresearch.com/how-16-05](http://colfaxresearch.com/how-16-05)



## Forums (technical):

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

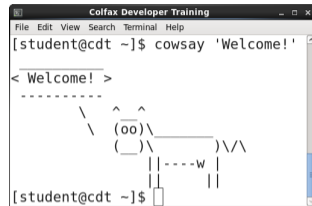


## Email (organizational):

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

# Hands-On Exercises and Remote Access

- 96 people receive a remote access token
- Can access the system the entire 3 weeks of the workshop
- 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



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

[student@cdt ~]$
```

## §2. Refresh

# Performance Optimization

# Computing Platforms

## Intel Xeon Processor



Current: Broadwell  
Upcoming: Skylake

Multi-Core Architecture

## Intel Xeon Phi Coprocessor, 1st generation



Current: Knights Corner (KNC)

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

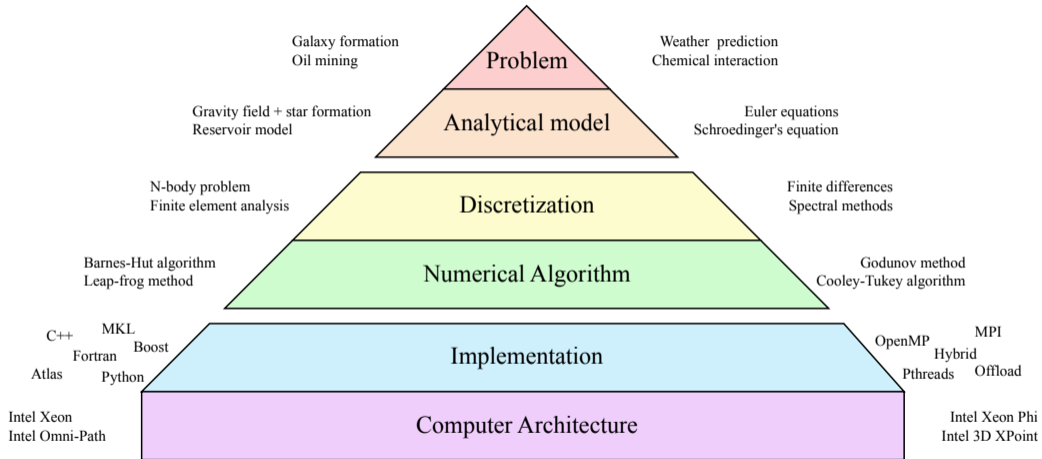


\* socket and coprocessor versions

Upcoming: Knights Landing (KNL)

Intel Many Integrated Core (MIC) Architecture

# Computing in Science and Engineering

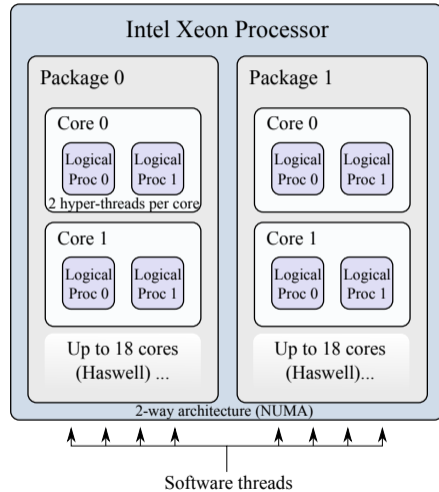
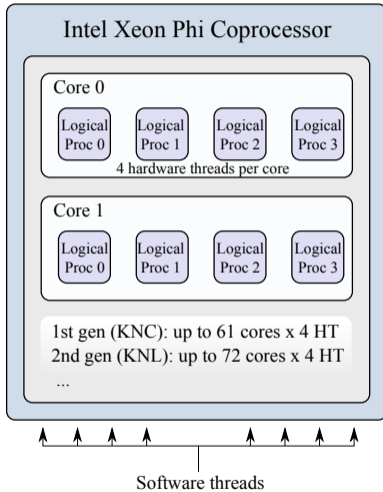


# Optimization Areas

- 1 **Scalar optimization** (compiler-friendly practices)
- 2 **Vectorization** (must use 16- or 8-wide vectors)
- 3 **Multi-threading** (must scale to 100+ threads)
- 4 **Memory access** (streaming access or tiling)
- 5 **Communication** (offload, MPI traffic control)

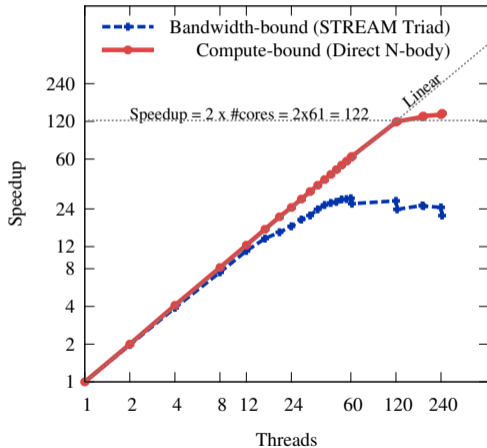
# Cores, Threads and OpenMP

# Processor Hierarchy

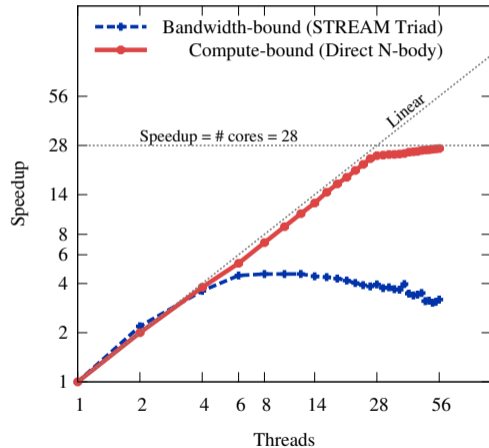


# Scalability Expectations: MIC versus CPU

## Performance on the MIC architecture



## Performance on the CPU architecture



# “Hello World” OpenMP Programs

```
1  #include <omp.h>
2  #include <stdio.h>
3
4  int main(){
5      const int nt=omp_get_max_threads();
6      printf("OpenMP with %d threads\n", nt);
7
8      #pragma omp parallel
9      {
10         printf("Hello World from thread %d\n", omp_get_thread_num());
11     }
12 }
```

## §3. Optimization of Multi-Threading I

# Too Much Synchronization

# Example: Strip-Mining for Vectorization

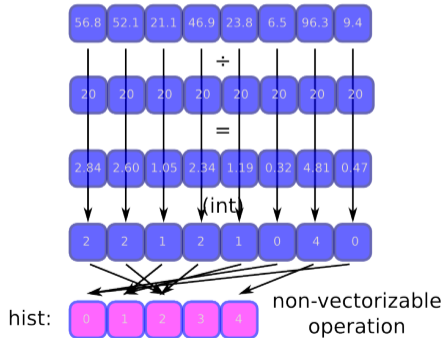
## Computing a histogram ( $m \ll n$ ):

```

1 void Histogram(
2     // Ages, values from 0.0f to 100.0f:
3     const float* age,
4     // Size of array age, n=100000000:
5     const int n,
6     // Output: counts in groups:
7     int* const hist,
8     // Size of array hist, m=5:
9     const int m,
10    // group_width=20.0f
11    const float group_width) {
12    for (int i = 0; i < n; i++) {
13        const int j = int(age[i]/group_width);
14        hist[j]++;
15    }
16 }

```

- Code cannot be auto-vectorized
- True vector dependence

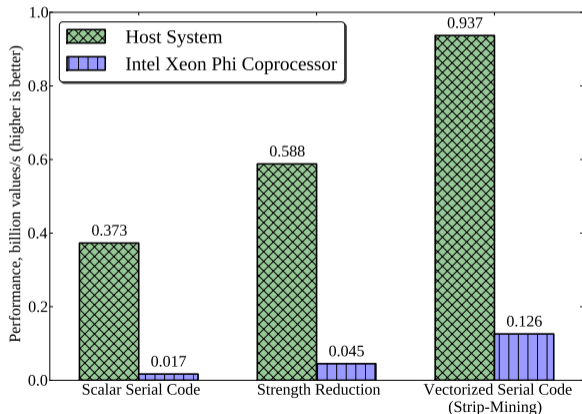


# The Same Calculation, Strip-Mined, Vectorized

```
1 void Histogram(const float* age, int* hist, const int n,  
2 const float group_width, const int m) {  
3     const int vecLen = 16; // Length of vectorized loop  
4     const float invGroupWidth = 1.0f/group_width; // Pre-compute the reciprocal  
5     // Strip-mining the loop in order to vectorize the inner short loop  
6     // Note: this algorithm assumes n%vecLen == 0.  
7     for (int ii = 0; ii < n; ii += vecLen) { //Temporary store vecLen indices  
8         int index[vecLen] __attribute__((aligned(64)));  
9         // Vectorize the multiplication and rounding  
10    #pragma vector aligned  
11        for (int i = ii; i < ii + vecLen; i++)  
12            index[i-ii] = (int) ( age[i] * invGroupWidth );  
13        // Scattered memory access, does not get vectorized  
14        for (int c = 0; c < vecLen; c++)  
15            hist[index[c]]++;  
16    }  
17 }
```

# Strip-Mining for Vectorization

Vectorization improves performance on both platforms. However, more work is needed to take advantage of the MIC architecture. See materials on multi-threading.



# Histogram Calculation Example: Adding Thread Parallelism

## Incorrect solution: unprotected data races

```
1  #pragma omp parallel for schedule(guided)
2  for (int ii = 0; ii < n; ii += vecLen) {
3      int index[vecLen] __attribute__((aligned(64)));
4      #pragma vector aligned
5      for (int i = ii; i < ii + vecLen; i++)
6          index[i-ii] = (int) ( age[i] * invGroupWidth );
7      for (int c = 0; c < vecLen; c++)
8          // Multiple threads will write into a single shared container
9          // These data races lead to incorrect results!
10         hist[index[c]]++;
11 }
```

# Histogram Calculation Example: Adding Thread Parallelism

Correct, but inefficient solution:

```
1  #pragma omp parallel for schedule(guided)
2  for (int ii = 0; ii < n; ii += vecLen) {
3      int index[vecLen] __attribute__((aligned(64)));
4      #pragma vector aligned
5          for (int i = ii; i < ii + vecLen; i++)
6              index[i-ii] = (int) ( age[i] * invGroupWidth );
7          for (int c = 0; c < vecLen; c++)
8              // Protect the ++ operation with the atomic mutex (inefficient!)
9          #pragma omp critical
10             { hist[index[c]]++; }
11 }
```

# Histogram Calculation Example: Adding Thread Parallelism

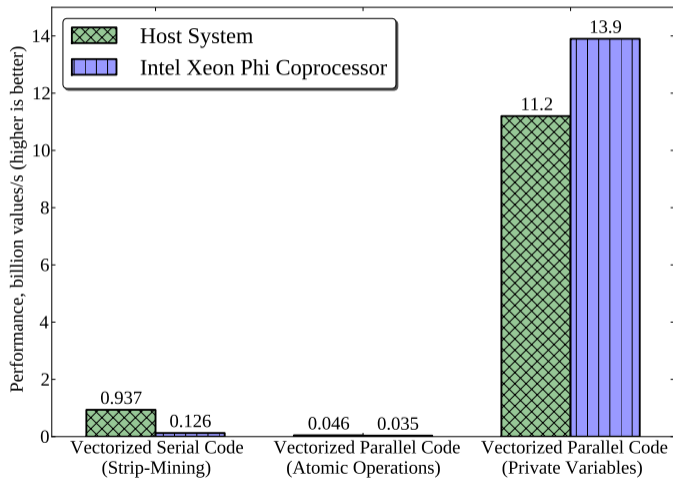
Correct, but inefficient solution:

```
1  #pragma omp parallel for schedule(guided)
2  for (int ii = 0; ii < n; ii += vecLen) {
3      int index[vecLen] __attribute__((aligned(64)));
4      #pragma vector aligned
5          for (int i = ii; i < ii + vecLen; i++)
6              index[i-ii] = (int) ( age[i] * invGroupWidth );
7          for (int c = 0; c < vecLen; c++)
8              // Protect the ++ operation with the atomic mutex (inefficient!)
9              #pragma omp atomic
10                 hist[index[c]]++;
11 }
```

# Correct and Efficient Solution with Reduction

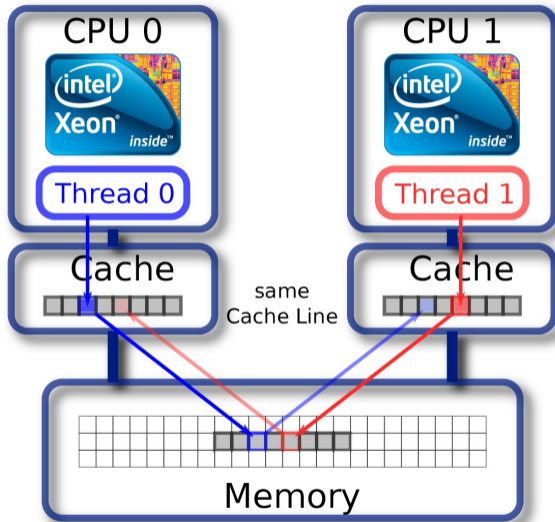
```
1  #pragma omp parallel
2  {
3      int hist_priv[m]; // Better idea: thread-private storage
4      hist_priv[:] = 0;
5      int index[vecLen] __attribute__((aligned(64)));
6      #pragma omp for schedule(guided)
7      for (int ii = 0; ii < n; ii += vecLen) {
8          #pragma vector aligned
9          for (int i = ii; i < ii + vecLen; i++)
10             index[i-ii] = (int) ( age[i] * invGroupWidth );
11         for (int c = 0; c < vecLen; c++)
12             hist_priv[index[c]]++;
13     }
14     for (int c = 0; c < m; c++) {
15         #pragma omp atomic
16         hist[c] += hist_priv[c];
17     } }
```

# Using Reduction instead of Synchronization



# False Sharing

# False Sharing. Data Padding and Private Variables



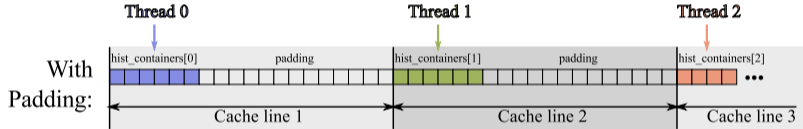
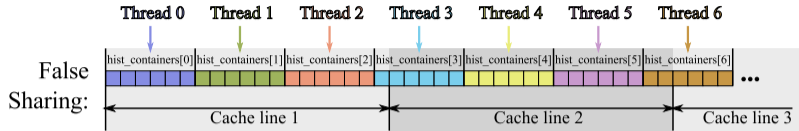
- *False sharing* is similar to *race condition*
- Threads accessing the same *cache line*
- Caused by *coherent caches*
- Cache line is 64-byte wide (in modern Intel architectures)

# False Sharing. Data Padding and Private Variables

```
1  const int m = 5;
2  int hist_thr[nThreads][m];
3  #pragma omp parallel for
4  for (int ii = 0; ii < n; ii += vecLen) {
5      // ...
6      // False sharing occurs here
7      for (int c = 0; c < vecLen; c++)
8          hist_thr[iThread][index[c]]++;
9  }
10 // Reducing results from all threads to the common histogram hist
11 for (int iThread = 0; iThread < nThreads; iThread++)
12     hist[0:m] += hist_thr[iThread][0:m];
```

- The value of  $m=5$  is small
- Array elements `hist_thr[0][:]` are within  $m*\text{sizeof}(\text{int})=20$  bytes of array elements `hist_thr[1][:]`

# Padding to Avoid False Sharing

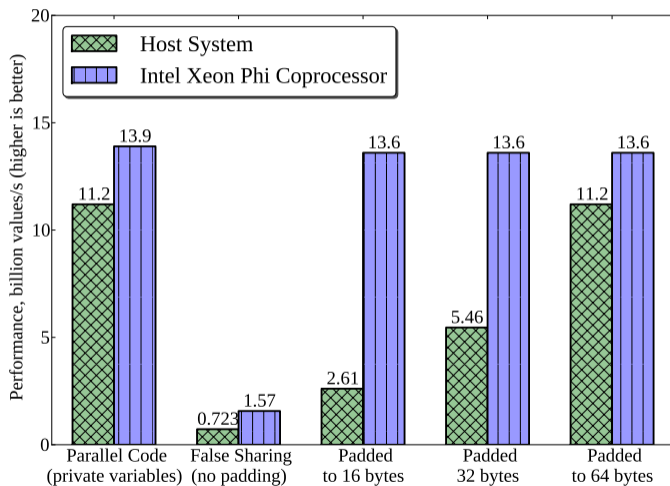


```

1 // Padding to avoid sharing a cache line between threads
2 const int paddingBytes = 64;
3 const int paddingElements = paddingBytes / sizeof(int);
4 const int mPadded = m + (paddingElements - m % paddingElements);
5 int hist_containers[nThreads][mPadded]; // New container

```

# Padding to Avoid False Sharing



# Insufficient Parallelism

## Example: Dealing with Insufficient Parallelism

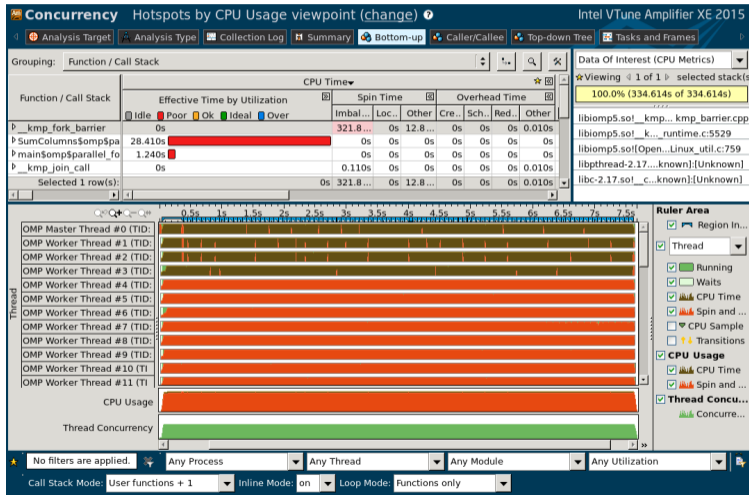
$$S_i = \sum_{j=0}^n M_{ij}, i = 0 \dots m. \quad (1)$$

- $m=4$  is small, smaller than the number of threads in the system
- $n \approx 10^8$  is large enough so that matrix does not fit into cache

```
1 void sum_unoptimized(const int m, const int n, long* M, long* s){
2   #pragma omp parallel for
3     for (int i=0; i<m; i++) { // m=4
4       long sum=0;
5     #pragma vector aligned
6       for (int j=0; j<n; j++) // n=100000000
7         sum+=M[i*n+j];
8     s[i]=sum; }}
```

# Dealing with Insufficient Parallelism

## VTune Analysis: Row-Wise Reduction of a Short, Wide Matrix



# Does not Work: Parallelizing Inner Loop

Inner loop has more iterations, parallelize there?

```
1 void SumParallelInnerLoop(const int m, const int n, long* M, long* s){
2     for (int i = 0; i < m; i++) { // m=4
3         long sumOfColumns = 0;
4         #pragma omp parallel for reduction(+: sumOfColumns)
5             for (int j = 0; j < n; j++) { // n=100000000
6                 sumOfColumns += M[i*n + j];
7             }
8             s[i] = sumOfColumns;
9         }
10    }
```

Does not work well: code must spawn and stop threads many times;  
OpenMP does not see the entire parallel region.

# Loop Collapse: Principle

Idea: combine iterations spaces of the inner loop and the outer loop.

```
1 #pragma omp parallel for collapse(2)
2   for (int i = 0; i < m; i++)
3     for (int j = 0; j < n; j++) {
4         // ...
5         // ...
6     }
```

```
1 #pragma omp parallel for
2   for (int c = 0; c < m*n; c++) {
3       i = c / n;
4       j = c % n;
5       // ...
6   }
```

# Does not Work, but Correct Direction: Loop Collapse

Loop collapse applied to the wide short matrix example:

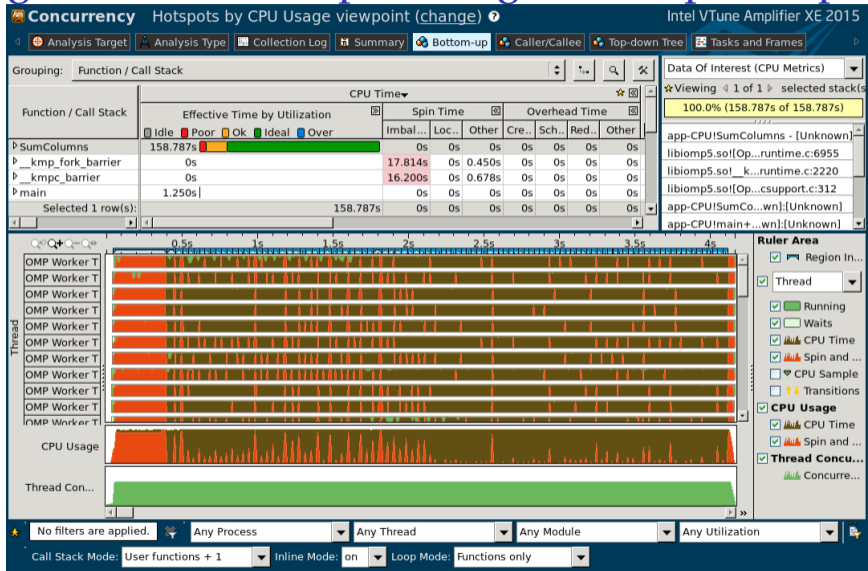
```
1 void SumCollapse(const int m, const int n, long* M, long* s){
2     s[:]=0;
3     #pragma omp parallel
4     { // Each thread will have a private container
5         long sumOfColumns[m]; sumOfColumns[:] = 0;
6         #pragma omp for collapse(2)
7         for (int i = 0; i < m; i++) // m=4
8             for (int j = 0; j < n; j++) // n=100000000
9                 sumOfColumns[i] += M[i*n + j];
10        for (int i = 0; i < m; i++)
11            #pragma omp atomic
12                s[i]=sumOfColumns[i];
13    } }
```

Does not work: automatic vectorization fails.

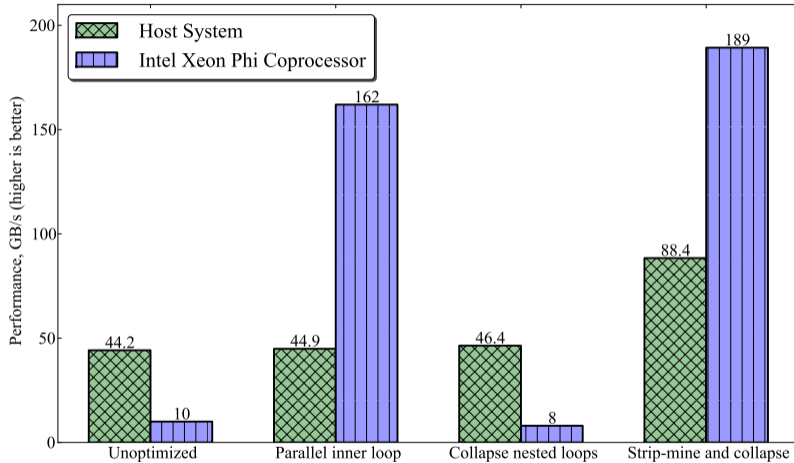
# Exposing Parallelism: Strip-Mining and Loop Collapse

```
1 void sum_stripmine(const int m, const int n, long* M, long* s){
2     const int STRIP=1024;
3     assert(n%STRIP==0);
4     s[0:m]=0;
5     #pragma omp parallel
6     {
7         long sum[m];    sum[0:m]=0;
8         #pragma omp for collapse(2) schedule(guided)
9         for (int i=0; i<m; i++)
10            for (int jj=0; jj<n; jj+=STRIP)
11                #pragma vector aligned
12                    for (int j=jj; j<jj+STRIP; j++)
13                        sum[i]+=M[i*n+j];
14        for (int i=0; i<m; i++)                // Reduction
15            #pragma omp atomic
16                s[i]+=sum[i];
17    } }
```

# Exposing Parallelism: Strip-Mining and Loop Collapse



# Dealing with Insufficient Parallelism



Techniques that did not work well are discussed in the book.

## §4. Review and What's Next

# Summary

This session:

- 1 Synchronization is necessary to resolve data races, but mutexes must be moved out of innermost loops
- 2 False sharing can be resolved with padding
- 3 Strip-mining can help to expose parallelism

Next session: optimization of thread affinity, NUMA locality, nested parallelism and loop scheduling.

[Learn More](#)

# HOW “Tools” Series

Hands-On Workshop (HOW “Tools” Series): webinars on efficient programming for the Intel architecture with the help of dedicated software development tools



**GOT THE TOOLS - NOW WHAT?**

Learn workflows and methodology with the “HOW” tools\* training and hands-on demos

\* Intel MKL | Intel Advisor | Intel VTune Amplifier

PARALLEL STUDIO XE

The graphic features a blue header with the text 'GOT THE TOOLS - NOW WHAT?'. Below this is a light-colored wood-grain background. On the left is a book cover for 'PARALLEL STUDIO XE' with the Intel logo. To the right of the book is the text 'Learn workflows and methodology with the “HOW” tools\* training and hands-on demos' and a list of tools: '\* Intel MKL | Intel Advisor | Intel VTune Amplifier'. At the bottom right, three silver wrenches are arranged vertically.

[colfaxresearch.com/how-tools-16-06](http://colfaxresearch.com/how-tools-16-06)

# Developer's Guide to Knights Landing



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

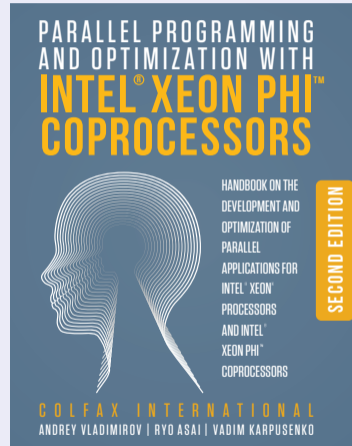
# Textbook

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

## COLFAX RESEARCH

CONTRIBUTING TO INNOVATIONS IN COMPUTING

[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

**Research and Educational Publications**

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

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

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

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

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

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

**Parallel Programming Book**

Introduction to parallel programming, deep discussion of optimization techniques, exercises.

© 2015, Colfax International. 506 pages.

**Featured Video**

Get Research material via webcast in a streaming video

<http://colfaxresearch.com/par1>

**Consulting**

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

- Optimize your existing application to take advantage of parallelism, from vectors to cores to clusters and GPUs
- Future-proof your application for upcoming innovations
- Accelerate your application using coprocessor technology
- Investigate the potential system configurations that satisfy your cost, power, performance requirements.
- Take a Hands-On Workshop: a virtual workshop to explore your computing problems, software experience in architecture, software development, and hardware design.

colfaxresearch.com/consulting

## Consulting

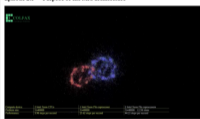


Colfax offers consulting services for enterprises, research help you to:

- Optimize your existing application to take advantage of parallelism, from vectors to cores to clusters and GPUs
- Future-proof your application for upcoming innovations
- Accelerate your application using coprocessor technology
- Investigate the potential system configurations that satisfy your cost, power, performance requirements.
- Take a Hands-On Workshop: a virtual workshop to explore your computing problems, software experience in architecture, software development, and hardware design.

All Video Courses - COP 001 - Chapter 2 - Episode 2.1

### Episode 2.1 — Purpose of the MIC architecture



In this video episode 2.1 we will introduce Intel Micro Architecture based on the Intel Many Integrated Core, or MIC, architecture and will discuss the role of the Intel Fabric Engine.

**par1** - Overview of the Architecture  
[Download](#) | [View](#) | [Share](#)

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

Published by: 2015-08-04 10:00 AM

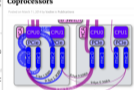


In this paper we will discuss the new HGST Ultrastar Archive HaaS SMR drive, a different feature set which offers storage capacities of 10 TB and beyond. These drives have high density and large capacities. These drives are well suited for large "data center" applications, such as data archive applications. The drive has a high level of performance.

We have done our best to manage, making the storage in the drive. In this publication we will discuss the potential of the HGST Ultrastar Archive HaaS SMR drive for enterprise data storage. It is a good fit for the enterprise data storage applications with their high capacity and performance.

**colfaxresearch.com/par1**  
[Download](#) | [View](#) | [Share](#)

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



In this paper we will discuss the configuration and benchmarks of peer-to-peer communication over Gigabit Ethernet and InfiniBand in a cluster with Intel Xeon Phi coprocessors.

**colfaxresearch.com/par1**  
[Download](#) | [View](#) | [Share](#)

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



**colfaxresearch.com/par1**  
[Download](#) | [View](#) | [Share](#)

### Fluid Dynamics with Fortran on Intel Xeon Phi coprocessors

Published by: 2015-08-04 10:00 AM

In this paper we will discuss the Fortran program for fluid dynamics on Intel Xeon Phi coprocessors. The program is written in Fortran and is designed to run on Intel Xeon Phi coprocessors. The program is designed to run on Intel Xeon Phi coprocessors.



**colfaxresearch.com/par1**  
[Download](#) | [View](#) | [Share](#)

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

Published by: 2015-08-04 10:00 AM

In this video we will discuss an interview with James Reinders, the Director and Chief Architect of Intel Coprocessors. We discuss the future of the parallel programming and how the performance can be improved by using coprocessors, high speed data and data generation, programming models, and also other related topics. We discuss parallel programming and software for coprocessors.



**colfaxresearch.com/par1**  
[Download](#) | [View](#) | [Share](#)

<http://colfaxresearch.com/>  
 (already registered? **get \$10 off the book**)