



PROGRAMMING AND OPTIMIZATION FOR INTEL[®] ARCHITECTURE

Hands-On Workshop (HOW) Series "Deep Dive"

Session 4

Colfax International — colfaxresearch.com

May 2017

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.

- ▶ **Module I. Programming**
 - 01. Intel Architecture and Modern Code – May 15
 - 02. Xeon Phi, Coprocessors, Omni-Path – May 16
- ▶ **Module II. Expressing Parallelism**
 - 03. Automatic vectorization – May 17
 - 04. Multi-threading with OpenMP – May 18
 - 05. Distributed Computing, MPI – May 19
- ▶ **Module III. Optimization**
 - 06. Optimization Overview: N-body – May 22
 - 07. Scalar tuning, Vectorization – May 23
 - 08. Common Multi-threading Problems – May 24
 - 09. Multi-threading, Memory Aspect – May 25
 - 10. Access to Caches and Memory – May 26

May 2017						
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			

Course page:

colfaxresearch.com/how-17-05

- ▶ Slides
- ▶ Code
- ▶ Video
- ▶ Chat

More workshops:

colfaxresearch.com/training



GET YOUR QUESTIONS ANSWERED: CHAT



colfaxresearch.com/how-17-05

GET YOUR QUESTIONS ANSWERED: FORUMS



Forum

Colfax Cluster

Discussion of Colfax Cluster usage policies, troubleshooting.

Developer Training, HOW Series

Questions about any of the Colfax trainings? Usage of training servers, experience with specific exercises, inquiries on what's inside, suggestions for future trainings - post them here.

Performance Optimization and Parallelism

Discuss with Colfax Research and colleagues any topics related to computational science, parallel programming, performance optimization and code modernization.

colfaxresearch.com/discussion

- ▶ All registrants receive an invitation from `cluster@colfaxresearch.com`
- ▶ Queue-based access to Intel Xeon E5, Intel Xeon Phi (KNC and KNL)
- ▶ Can access the cluster the entire 2 weeks of the workshop





§2. EXPRESSING TASK PARALLELISM

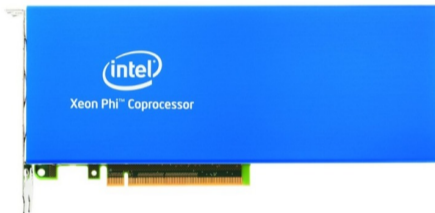
Intel Xeon Processor



Current: Broadwell
Upcoming: Skylake

Multi-Core Architecture

Intel Xeon Phi Coprocessor, 1st generation



Knights Corner (KNC)

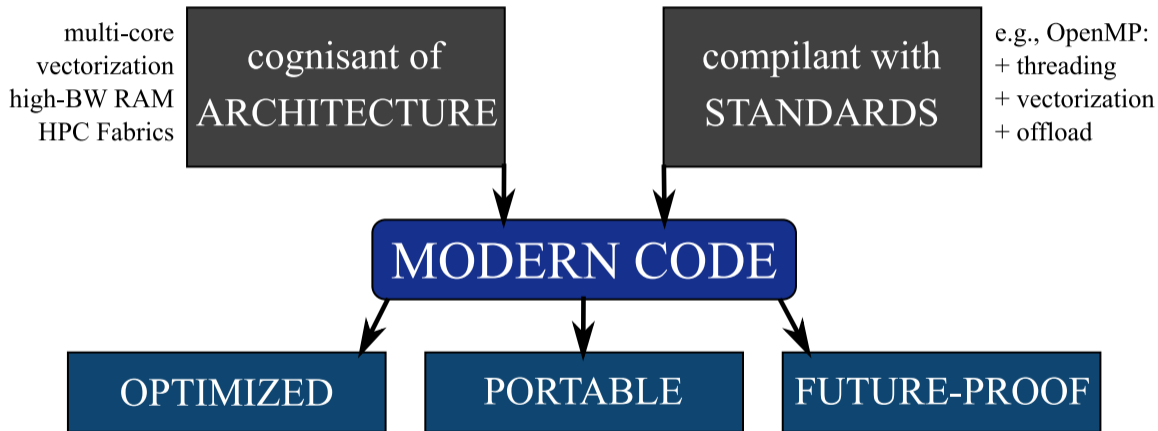
Intel Xeon Phi Processor, 2nd generation*



* socket and coprocessor versions

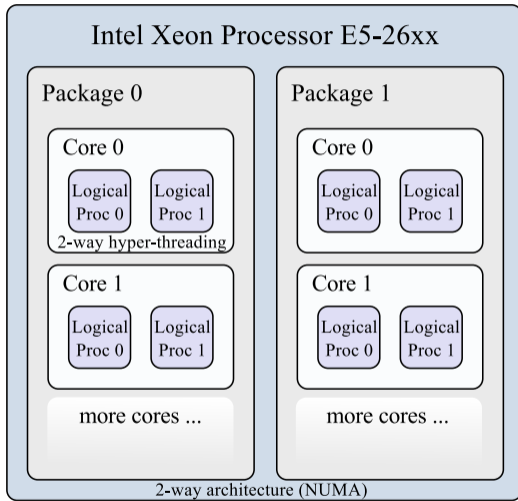
Knights Landing (KNL)

Intel Many Integrated Core (MIC) Architecture





HANDLING MULTIPLE CORES



Hierarchy:

Packages ->

Cores ->

Logical processors

OS Proc = numerical ID
of logical processor

Jargon:

"socket" = **package**

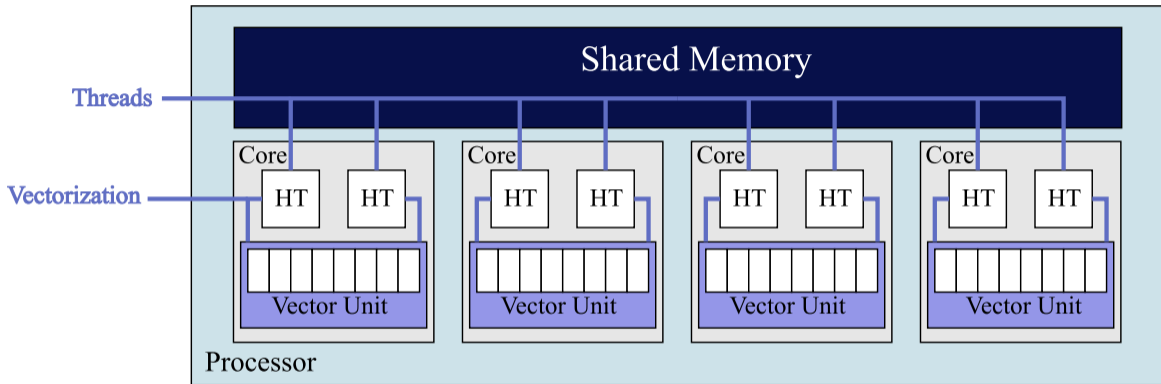
"logical core" =

"hyper-thread" =

"hardware thread" =

logical processor

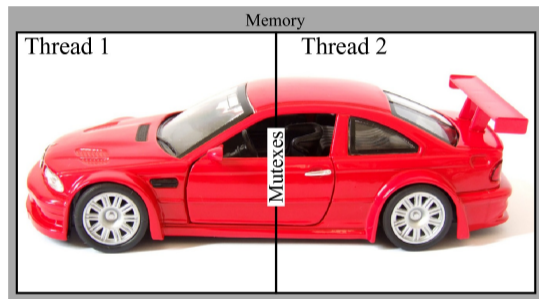
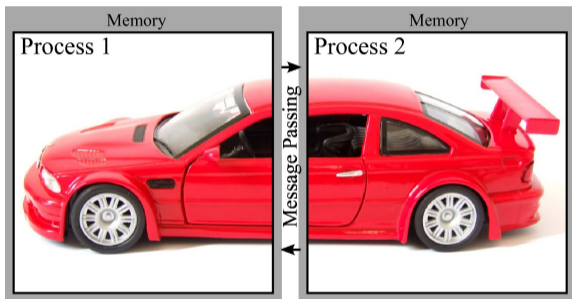
CO-EXISTENCE WITH VECTORS



Utilize cores: run multiple threads/processes (MIMD)

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

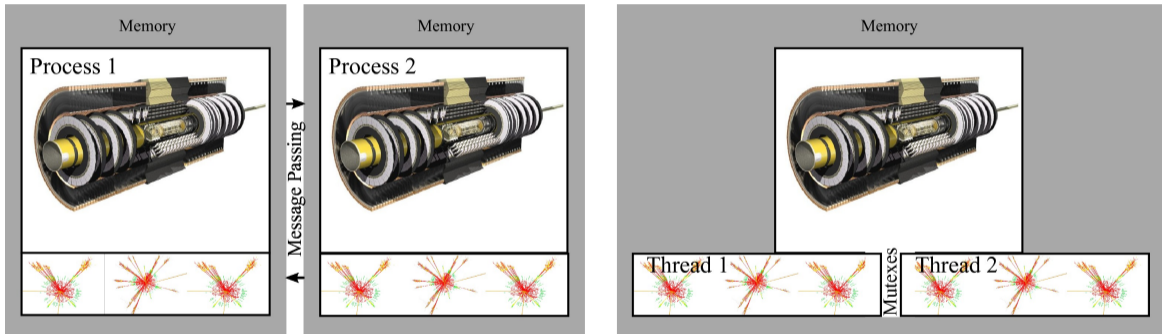
Option 1: Partitioning data set between threads/processes



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).

THREADING FRAMEWORKS

Framework	Functionality
C++11 Threads	Asynchronous functions; only C++
POSIX Threads	Fork/join; C/C++/Fortran; Linux
Cilk Plus	Async tasks, loops, reducers, load balance; C/C++
TBB	Trees of tasks, complex patterns; only C++
OpenMP	Tasks, loops, reduction, load balancing, affinity, nesting, C/C++/Fortran (+SIMD, offload)



OPENMP BASICS

"HELLO WORLD" OPENMP PROGRAM

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

- ▶ 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):

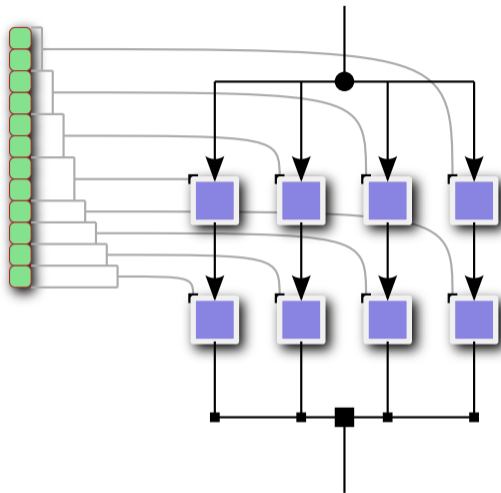
```
1 int A, B; // Variables declared at the beginning of a function
2 #pragma omp parallel private(A) shared(B)
3 {
4     // Each thread has its own copy of A, but B is shared
5 }
```

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

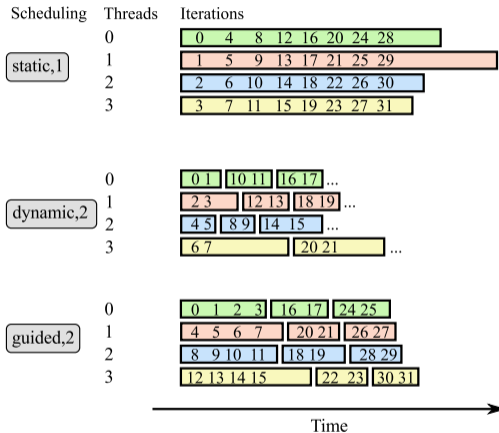
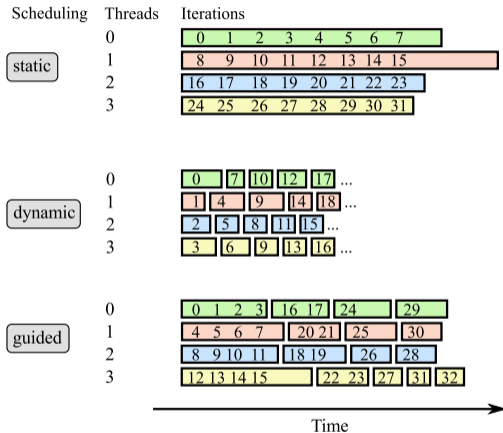
The OpenMP library will distribute the iterations of the loop following the `#pragma omp parallel for` across threads.

```
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

```

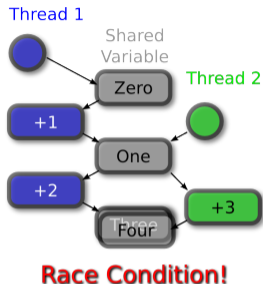
1  #include <omp.h>
2  #include <stdio.h>
3  int main() {
4      const int n = 1000;
5      int total = 0;
6      #pragma omp parallel for
7      for (int i = 0; i < n; i++) {
8          total = total + i; // Race condition
9      }
10     printf("total=%d (must be %d)\n", total,
11            ((n-1)*n)/2);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

```

```

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

```



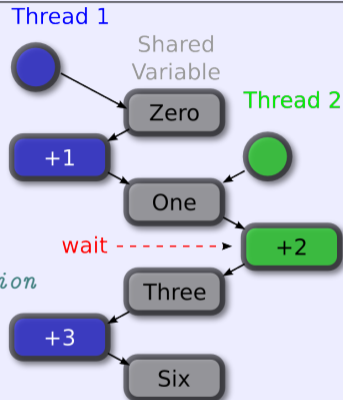
- Occurs when 2 or more threads access the same memory address, and at least one of these accesses is for writing

PROTECTING RACE CONDITIONS WITH A CRITICAL SECTION

```

1 #include <omp.h>
2 #include <stdio>
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
- ▶ 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)
```

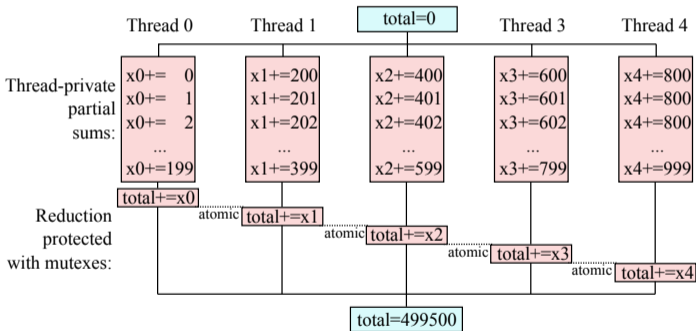
AVOIDING RACES WITH THREAD-PRIVATE STORAGE

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;
9      #pragma omp atomic
10     total += total_thr;
11 }
12

```





CO-EXISTENCE WITH VECTORS

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]);
```

SIMULTANEOUS THREADING AND VECTORIZATION

Sometimes the compiler may need a little help:

```
1  const int STRIP_SIZE = 128; // A multiple of vector length
2  const int nTrunc = n - n%STRIP_SIZE; // A multiple of vector length
3
4  #pragma omp parallel for
5  for (int ii = 0; ii < nTrunc; ii += STRIP_SIZE) // Thread parallelism in outer
6  #pragma simd
7      for (int i = ii; i < ii + STRIP_SIZE; i++) // Vectorization in inner loop
8          DoSomeWork(A[i]);
9
10 // Remainder loop:
11 for (int i = nTrunc; i < n; i++)
12     DoSomeWork(A[i]);
```



MORE TO LEARN ABOUT OPENMP

OPENMP CONCEPTS AND CONSTRUCTS

`#pragma omp parallel` – create threads

`#pragma omp for` – process loop with threads

`#pragma omp task/taskyield` – asynchronous tasks

`#pragma omp critical/atomic` – mutexes

`#pragma omp barrier/taskwait` – synchronization points

`#pragma omp sections/single` – blocks of code for individual threads

`#pragma omp flush` – enforce memory consistency

`#pragma omp ordered` – partial loop serialization

`OMP_*` – environment variables, `omp_*`() – functions

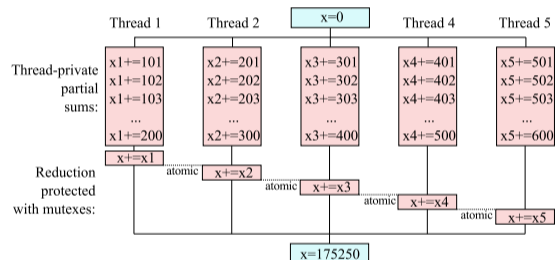
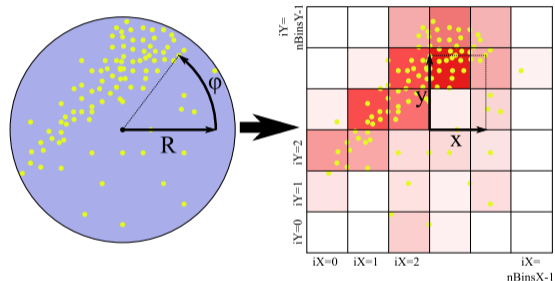
Click construct names for links to the [OpenMP reference from the LLNL](#)



ADDITIONAL READING

SUGGESTED ADDITIONAL READING

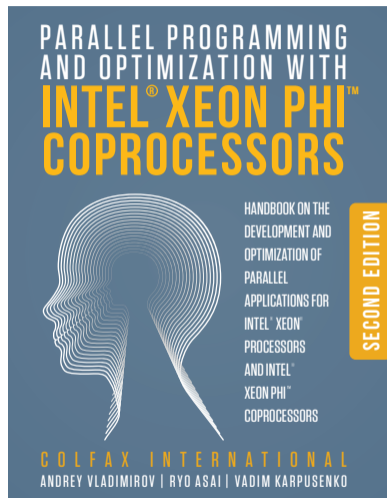
Colfax Research tutorial on multi-threading in a binning code



<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: distributed-memory computing with MPI.

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

Parallel Programming Book

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

© 2015, Colfax International, 508 pages.

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 H800 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 Performance to Cost Ratios with Intel Xeon Phi Coprocessors (and why ix Acceleration May Be Enough)

Featured Video


See Research material reconstruction to a streaming code





▶

[View Full Article](#)

Consulting



Share

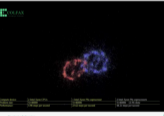



Colfax offers consulting services for enterprises, research help you to:

- Optimize your existing application to take advantage of parallelism, from vectors to cores to clusters and
- 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 architecture to realize your computing pro

48 Videos Courses | 101 VHS | Chapter 11 | Episode 1.1



Episode 2.1 — Purpose of the MIC architecture



▶

[View Full Article](#)



Software Developer's Introduction to the HGST Ultrastar Archive H800 SMR Drives

This paper will discuss the new HGST Ultrastar Archive H800 SMR drives, which are the world's first 7.2 TB drives. These drives are well suited for high volume archival applications. The paper will discuss the benefits of these drives and how they can be used in your applications.

[View Full Article](#)

Fluid Dynamics with Fortran on Intel Xeon Phi coprocessors

At the 2015 International Fortran Users Group Conference, a Fortran user presented a Fortran program that runs on Intel Xeon Phi coprocessors. The program simulates fluid dynamics and is a good example of how to use the coprocessors. The user will discuss the benefits of these coprocessors and how they can be used in your applications.

[View Full Article](#)

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



This paper discusses the configuration and benchmarks of a cluster with Intel Xeon Phi coprocessors. The cluster is configured with Gigabit Ethernet and InfiniBand. The paper will discuss the benefits of these networks and how they can be used in your applications.

[View Full Article](#)

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



A few weeks ago we conducted an interview with James Reinders, the Director of Intel's Parallel Computing Group. In this interview, we discussed the future of Intel MIC architecture, parallel programming, and education. James will discuss the benefits of these technologies and how they can be used in your applications.

[View Full Article](#)

http://colfaxresearch.com/