



# 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

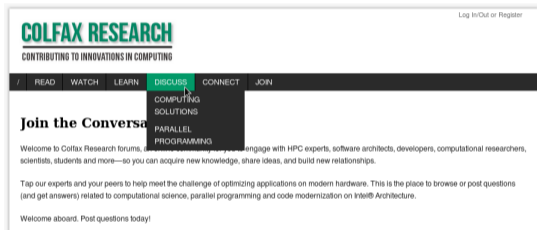
## Chat (current):

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



## Forums (technical):

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

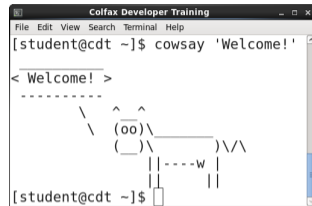


## Email (organizational):

[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. 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\*



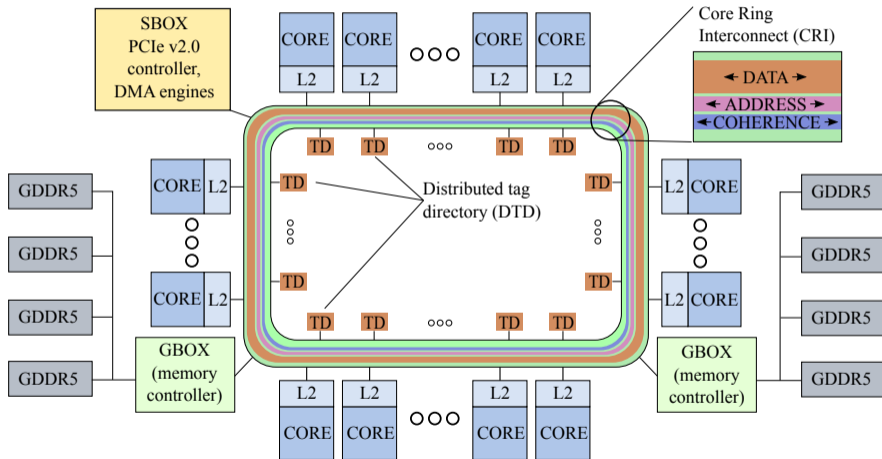
\* socket and coprocessor versions

Upcoming: Knights Landing (KNL)

Intel Many Integrated Core (MIC) Architecture

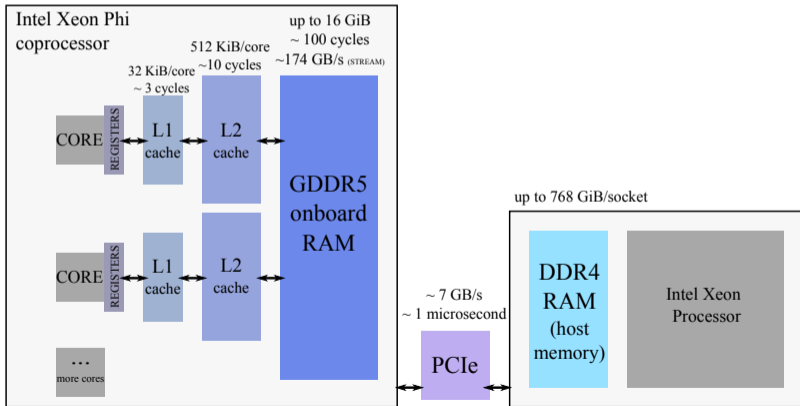
# KNC Die Organization

In a ring bus with distributed cache, data access locality is key.



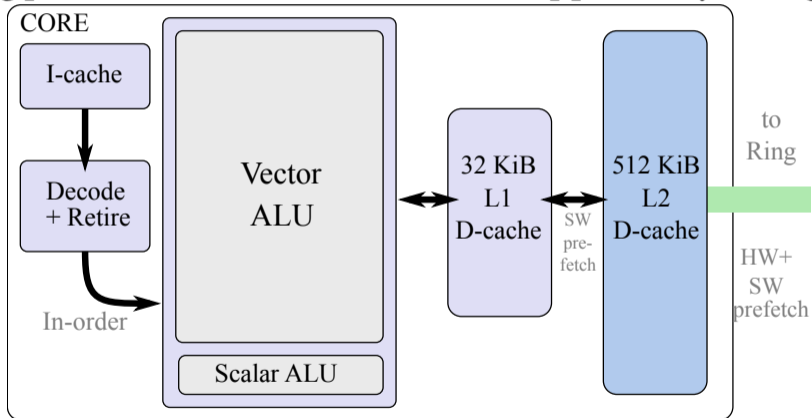
# KNC Memory Organization

- Direct access to  $\leq 16$  GiB of cached GDDR5 memory on board
- No access to system DDR4, connected to host via PCIe



# KNC Cores

Computing power is in vector units. Scalar support only for legacy usage.



# 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)

## §3. N-body Simulation

# N-body Simulation on CPU and Coprocessor



## N-body simulation on...

Two  
Intel® Xeon®  
CPUs



One  
Intel® Xeon Phi™  
coprocessor



Two  
Intel® Xeon Phi™  
coprocessors



Paper: <http://xeonphi.com/papers/nbody-basic>

Demo: [click here](#)

# Physics

## Gravitational N-body dynamics:

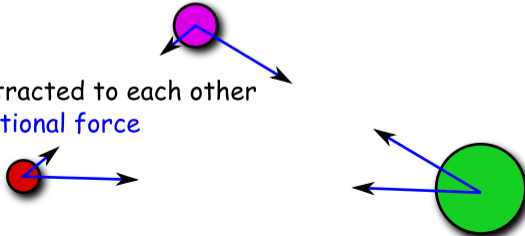
Newton's law of universal gravitation:

$$M_i \vec{R}_i''(t) = G \sum_j \frac{M_i M_j}{|\vec{R}_i - \vec{R}_j|^3} (\vec{R}_j - \vec{R}_i)$$

where:

$$|\vec{R}_i - \vec{R}_j| = \sqrt{(R_{i,x} - R_{j,x})^2 + (R_{i,y} - R_{j,y})^2 + (R_{i,z} - R_{j,z})^2}$$

particles are attracted to each other  
with **the gravitational force**



# Application

- 1 Astrophysics:
  - ▶ planetary systems
  - ▶ galaxies
  - ▶ cosmological structures
- 2 Electrostatic systems:
  - ▶ molecules
  - ▶ crystals

This work: “toy model” with all-to-all  $O(n^2)$  algorithm. Practical N-body simulations may use tree algorithms with  $O(n \log n)$  complexity.



Source: [APOD](#), credit: Debra Meloy Elmegreen (Vassar College) et al., & the Hubble Heritage Team (AURA/ STScI/ NASA)

# All-to-All Approach ( $O(n^2)$ Complexity Scaling)

Each particle is stored as a structure:

```
1 struct ParticleType {  
2     float x, y, z;  
3     float vx, vy, vz;  
4 };
```

main() allocates an array of ParticleType:

```
1 ParticleType* particle = new ParticleType[nParticles];
```

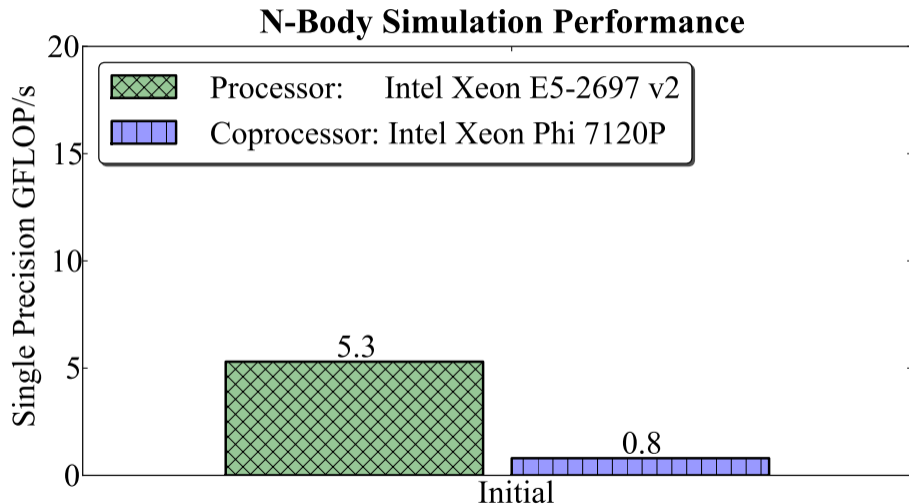
Particle propagation step is timed:

```
1 const double tStart = omp_get_wtime(); // Start timing  
2 MoveParticles(nParticles, particle, dt);  
3 const double tEnd = omp_get_wtime(); // End timing
```

# Particle Update Engine

```
1 void MoveParticles(int nParticles, ParticleType* particle, float dt) {
2   for (int i = 0; i < nParticles; i++) { // Particles that experience force
3     float Fx = 0, Fy = 0, Fz = 0; // Gravity force on particle i
4     for (int j = 0; j < nParticles; j++) { // Particles that exert force
5       // Newton's law of universal gravity
6       const float dx = particle[j].x - particle[i].x;
7       const float dy = particle[j].y - particle[i].y;
8       const float dz = particle[j].z - particle[i].z;
9       const float drSquared = dx*dx + dy*dy + dz*dz + 1e-20;
10      const float drPower32 = pow(drSquared, 3.0/2.0);
11      // Calculate the net force
12      Fx += dx/drPower32; Fy += dy/drPower32; Fz += dz/drPower32;
13    }
14    // Accelerate particles in response to the gravitational force
15    particle[i].vx+=dt*Fx; particle[i].vy+=dt*Fy; particle[i].vz+=dt*Fz;
16  }
17  ...
```

# Performance of Initial Implementation



# Incorporating Thread Parallelism

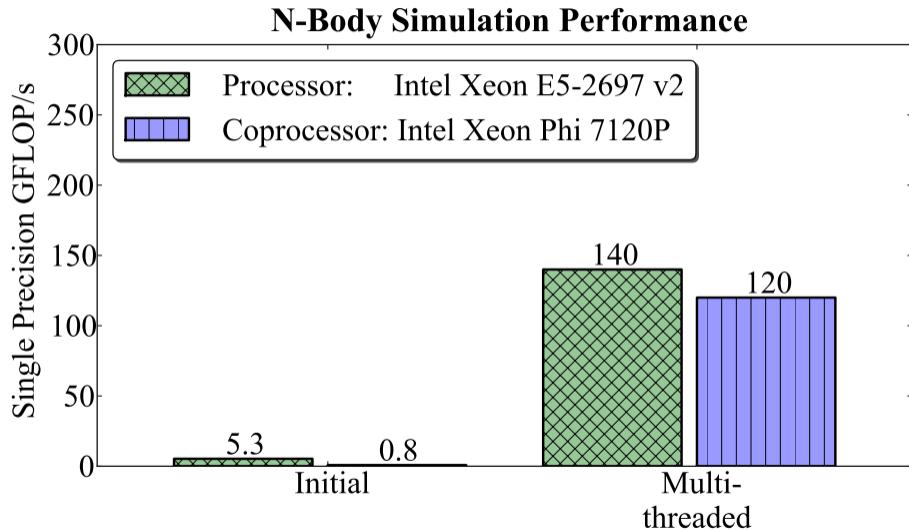
Before:

```
1  for (int i = 0; i < nParticles; i++) { // Particles that experience force
2      float Fx = 0, Fy = 0, Fz = 0; // Gravity force on particle i
3      for (int j = 0; j < nParticles; j++) { // Particles that exert force
4          // Newton's law of universal gravity
5          ...
```

After:

```
1  #pragma omp parallel for
2      for (int i = 0; i < nParticles; i++) { // Particles that experience force
3          float Fx = 0, Fy = 0, Fz = 0; // Gravity force on particle i
4          for (int j = 0; j < nParticles; j++) { // Particles that exert force
5              // Newton's law of universal gravity
6              ...
```

# Performance with Thread Parallelism



# Vectorizing with Unit-Stride Memory Access

Before:

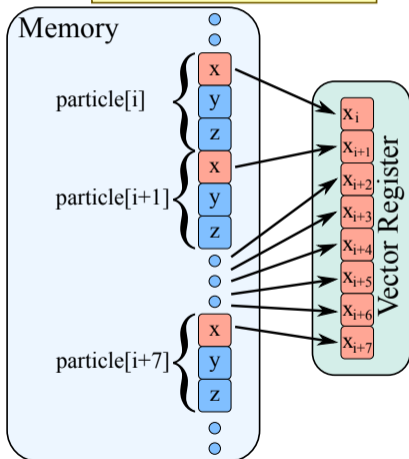
```
1 struct ParticleType {  
2     float x, y, z, vx, vy, vz;  
3 }; // ...  
4     const float dx = particle[j].x - particle[i].x;  
5     const float dy = particle[j].y - particle[i].y;  
6     const float dz = particle[j].z - particle[i].z;
```

After:

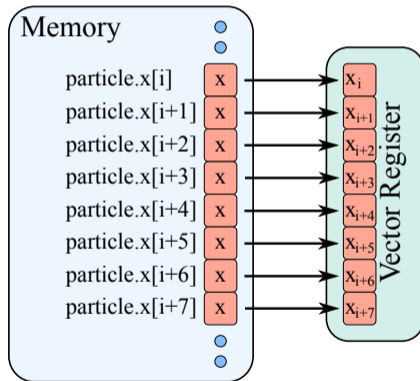
```
1 struct ParticleSet {  
2     float *x, *y, *z, *vx, *vy, *vz;  
3 }; // ...  
4     const float dx = particle.x[j] - particle.x[i];  
5     const float dy = particle.y[j] - particle.y[i];  
6     const float dz = particle.z[j] - particle.z[i];
```

# Why AoS to SoA Conversion Helps: Unit Stride

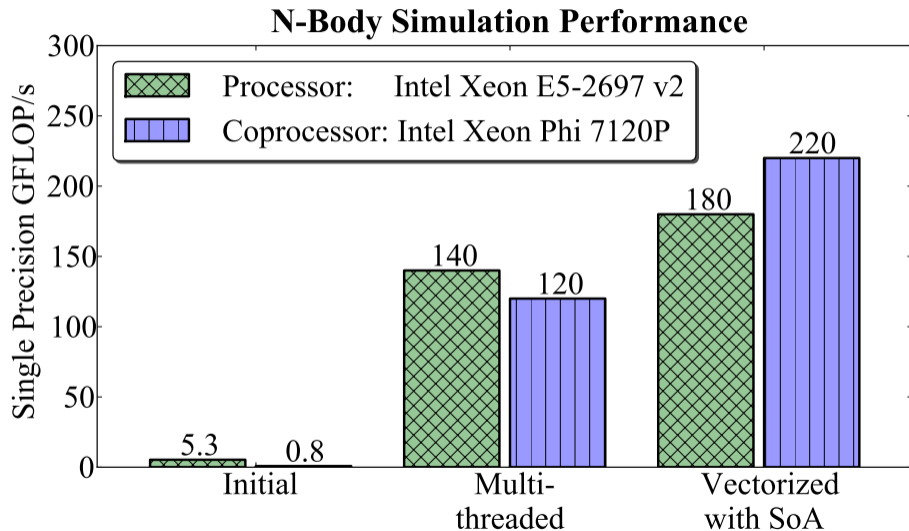
Array of Structures  
(sub-optimal)



Structure of Arrays  
(optimal)



# Performance with Improved Vectorization



# Improving Scalar Expressions

Before:

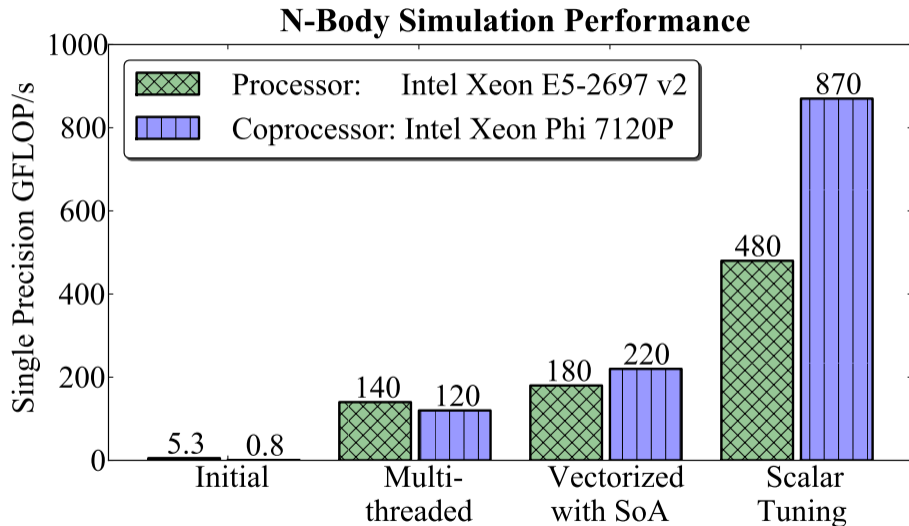
```
1  const float drSquared = dx*dx + dy*dy + dz*dz + 1e-20;  
2  const float drPower32 = pow(drSquared, 3.0/2.0);  
3  // Calculate the net force  
4  Fx += dx/drPower32;  Fy += dy/drPower32;  Fz += dz/drPower32;
```

After:

```
1  const float drRecip    = 1.0f/sqrtf(dx*dx + dy*dy + dz*dz + 1e-20);  
2  const float drPowerN32 = drRecip*drRecip*drRecip;  
3  // Calculate the net force  
4  Fx += dx*drPowerN32;  Fy += dy*drPowerN32;  Fz += dz*drPowerN32;
```

- Strength reduction (division  $\rightarrow$  multiplication by reciprocal)
- Precision control (suffix `-f` on single-precision constants and functions)
- Reliance on hardware-supported reciprocal square root

# Performance after Scalar Tuning



# Improving Cache Traffic

Before:

```

1  for (int i = 0; i < nParticles; i++) { // Particles that experience force
2      float Fx = 0, Fy = 0, Fz = 0; // Gravity force on particle i
3      for (int j = 0; j < nParticles; j++) { // Particles that exert force
4          // ...
5          Fx += dx*drPowerN32; Fy += dy*drPowerN32; Fz += dz*drPowerN32;

```

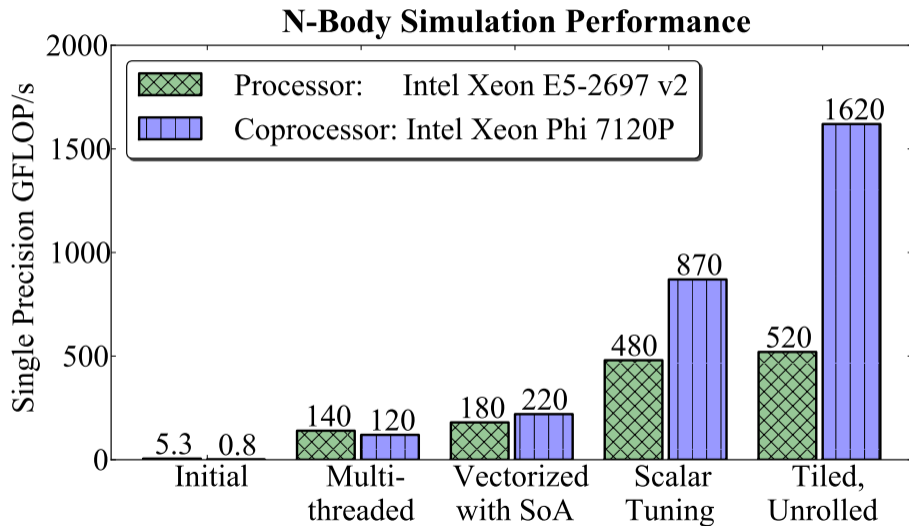
After: (tileSize = 16)

```

1  for (int ii = 0; ii < nParticles; ii += tileSize) { // Particle blocks
2      float Fx[tileSize], Fy[tileSize], Fz[tileSize]; // Force on particle block
3      Fx[:] = Fy[:] = Fz[:] = 0;
4      #pragma unroll(tileSize)
5      for (int j = 0; j < nParticles; j++) { // Particles that exert force
6          for (int i = ii; i < ii + tileSize; i++) { // Traverse the block
7              // ...
8          Fx[i-ii] += dx*drPowerN32; Fy[i-ii] += dy*drPowerN32; Fz[i-ii] += dz*drPowerN32;

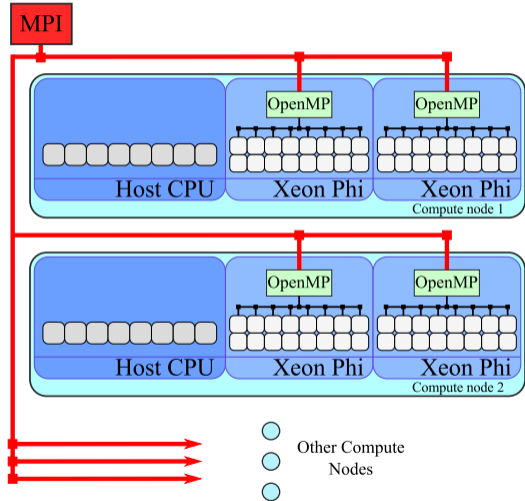
```

# Performance with Cache Optimization (Loop Tiling)



# Scaling Across a Cluster with Coprocessors with MPI

- We put MPI processes only on CPUs
- Subdivide data between coprocessors
- Concurrent offload from multiple host threads
- Synchronize data between nodes with MPI



# Core of MPI-only Implementation

Simple: all particles on each compute node; exchange updated particle coordinates.

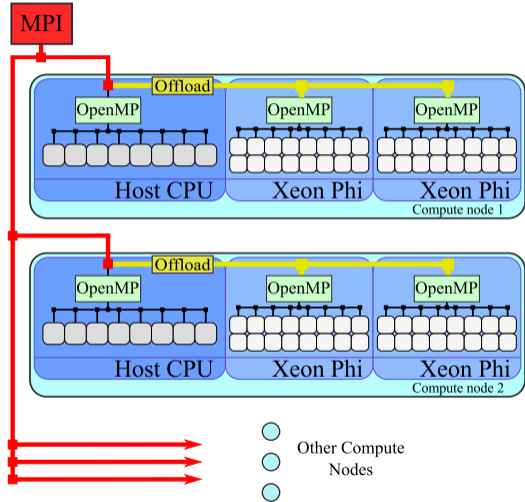
```

1 void MoveParticles(int nParticles, ParticleSet& particle, float dt,
2                   int mpiRank, int mpiWorldSize) {
3     const int myParticles = nParticles/mpiWorldSize;
4     const int startParticle = (mpiRank    )*myParticles;
5     const int endParticle   = (mpiRank + 1)*myParticles;
6     // Outer loop over only the subset of particles processed by present process
7     #pragma omp parallel for schedule(guided)
8     for (int ii = startParticle; ii < endParticle; ii += tileSize) {
9         for (int j = 0; j < nParticles; j++) // ...But inner loop over all particles
10            //...
11    }
12    // ... Propagate results of time step across the cluster
13    MPI_Allgather(MPI_IN_PLACE, 0, MPI_DATATYPE_NULL, particle.x,
14                 myParticles, MPI_FLOAT, MPI_COMM_WORLD);
15    // ...

```

# Scaling Across a Cluster with Coprocessors

- We put MPI processes only on CPUs
- Subdivide data between coprocessors
- Concurrent offload from multiple host threads
- Synchronize data between nodes with MPI



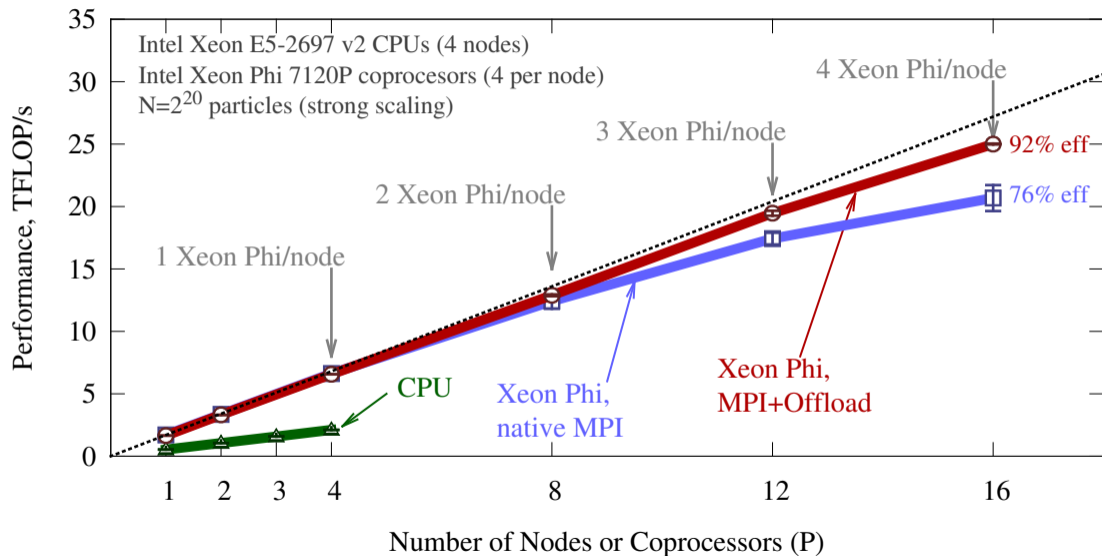
# MPI with offload implementation

```

1  const int nDevices = _Offload_number_of_devices();
2  const int particlesPerDevice=(nDevices==0 ? myParticles : myParticles/nDevices);
3  #pragma omp parallel num_threads(nDevices) if(nDevices>0)
4  {
5      const int iDevice = omp_get_thread_num();
6      const int startParticle = rankStartParticle + (iDevice )*particlesPerDevice;
7      #pragma offload target(mic:iDevice) if(nDevices>0)          \
8          in (x : length(nParticles)          alloc_if(alloc==1) free_if(0)) \
9          out(x [startParticle:particlesPerDevice] : alloc_if(0) free_if(alloc==-1)) \
10         in (vx: length(nParticles*alloc*alloc)          alloc_if(alloc==1) free_if(0)) \
11         //...
12         { // Loop over particles that experience force
13     #pragma omp parallel for schedule(guided)
14         for (int ii = startParticle; ii < endParticle; ii += tileSize) {
15             // ...

```

# Results with MPI+Offload

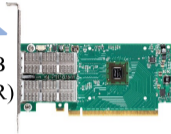


# Comparative Benchmarks and System Configuration

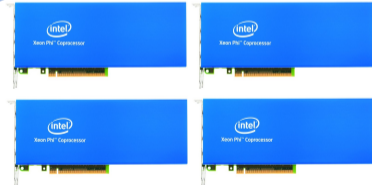
Colfax ProEdge SXP8600p  
rack-mountable workstations (cluster of 4)



Mellanox Connect-IB  
InfiniBand HCA (FDR)



Intel Xeon Phi 7120P coprocessors  
(4 per system)

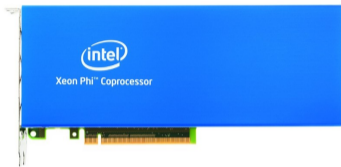


Dual-socket Intel Xeon E5-2697 v2  
processor



<http://xeonphi.com/workstations>

# Coprocessor vs Processor Performance



One Intel Xeon Phi 7120P  
coprocessor

*vs.*



Two Intel Xeon E5-2697 v2  
CPUs

- Why compare 1 coprocessor against 2 processors?  
Same thermal design power (TDP).

See also [“Intel Xeon Product Family: Performance Brief”](#)

# Additional Case Studies

# Astrophysical Code HEATCODE: an Offload Story

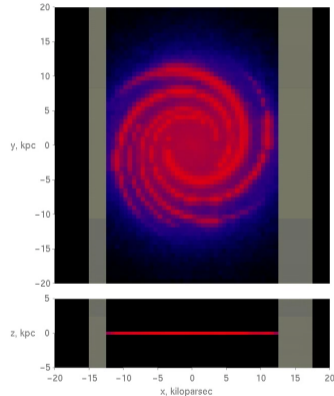


## Porting to Intel® Xeon Phi™ coprocessors

- We ported Frankie code using explicit offload model
- Same code & optimization methods for Xeon Phi™
- Simultaneous calculations on CPU and coprocessors with automatic load balancing was easy to implement
- With two Intel® Xeon Phi™ coprocessors, performance for high-res calculations is 3.2x better than with two Intel® Xeon® E5 processors alone.
- **RESULT:** estimated target project calculation time is now 2 weeks (down from 6+ years)

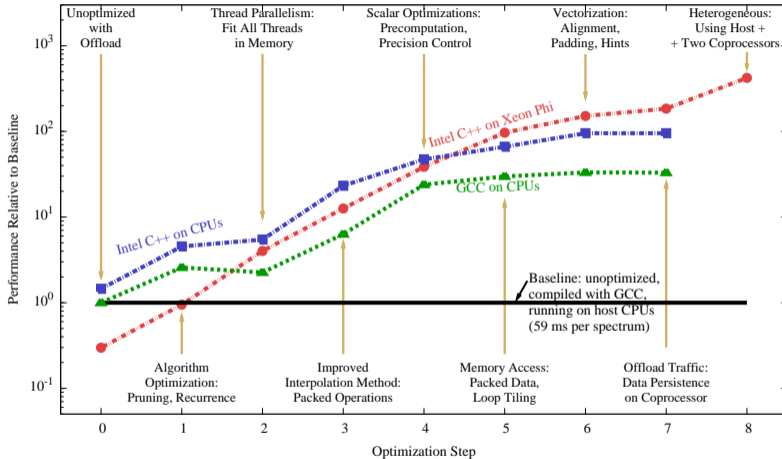
Goal achieved!

Transient Emission of Cosmic Dust Grains  
in the Milky Way Galaxy,  
Simulation with Frankie Code



<http://xeonphi.com/papers/heatcode>

# Astrophysical Code HEATCODE: an Offload Story

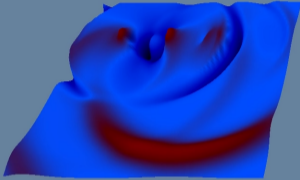


<http://xeonphi.com/papers/heatcode>

# Computational Fluid Dynamics: Legacy Code

## FLUID DYNAMICS WITH FORTRAN ON INTEL® XEON PHI™ COPROCESSORS

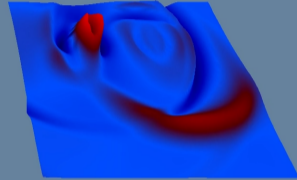
### SHALLOW WATER EQUATION SOLVER



PERFORMANCE ON CPU: 19.5 GFLOP/S

SAME CODE FOR CPU AND XEON PHI  
FORTRAN 90 + OPENMP + MPI

PUBLICATION:  
XEONPHI.COM/PAPERS/SHALLOW



PERFORMANCE WITH COPROCESSORS: 52.5 GFLOP/S



INTEL XEON E5-2697 V3 PROCESSOR

SIMULATION SIZE: 9600X9600

**ACCELERATION: 2.7X**



INTEL XEON E5-2697 V3 PROCESSOR +  
TWO INTEL XEON PHI 7120A COPROCESSORS



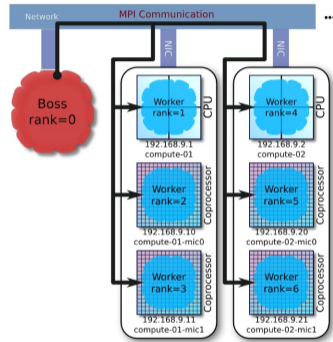
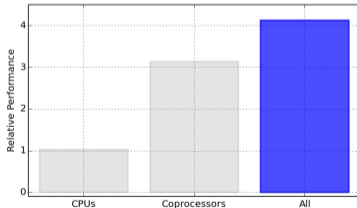
SERVERS WORKSTATIONS TRAINING CONSULTING RESEARCH

WWW.COLFAX-INTL.COM

<http://xeonphi.com/papers/shallow>

# Asian Option Pricing: Heterogeneous Clustering

Heterogeneous Clustering with Homogeneous Code:  
Asian Option Pricing



<http://xeonphi.com/papers/heterogeneous>

# Summary

Areas of optimization of applications for Intel Xeon and Intel Xeon Phi processors:

- 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)

Next session: optimization of vectorization.

[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/)

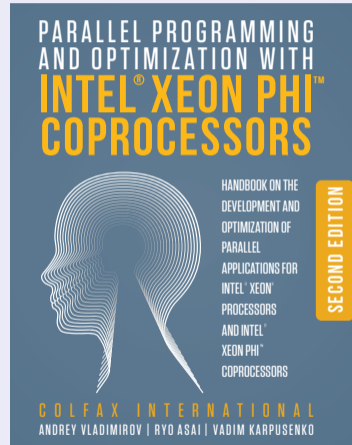
# Textbook

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

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 1 of 3: Strip-Mining for Vectorization**

**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 xx Acceleration May be Enough)**




**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 beyond
- 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 tour of a real-world example to explore your computing options, software experience in architecture, and more.

**Parallel Programming Book**

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

© 2015, Colfax International. 506 pages.

**Featured Video**

Additional Reading

See Research material on vectorization in a streaming code

<http://colfaxresearch.com/typ708>

## Consulting

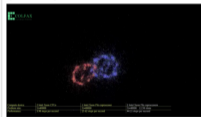


Colfax offers consulting services for enterprises, research help you to:

- Optimize your existing application to take advantage of parallelism, from vectors to cores to clusters and beyond
- 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 tour of a real-world example to explore your computing options, software experience in architecture, and more.

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

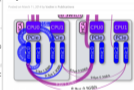
### Episode 2.1 — Purpose of the MIC architecture



In this video episode 2.1 we will introduce Intel's MIC architecture based on the Intel Many Integrated Core, or MIC, architecture and will discuss the role of the Intel Parallels environment.

**Download** Download the Intel Parallels environment

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



Intel networking... Download the Intel Parallels environment

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



Download the Intel Parallels environment

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



In this paper we will discuss the new HGST Ultrastar Archive HaaS SMR drives, which feature the ability to store up to 10 TB of data on a single drive. These drives are well suited for large-scale archival applications, such as data backup and recovery.

Download the Intel Parallels environment

## Fluid Dynamics with Fortran on Intel Xeon Phi coprocessors



Download the Intel Parallels environment

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

In this video you will see a conversation with James Reinders, the Director of Intel's Parallel Computing Group. We discuss the future of parallel programming and how the Intel MIC architecture is changing the way we think about computing and education.



Download the Intel Parallels environment

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