



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

The Hands-On Workshop (HOW) Series  
Session 5

*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.

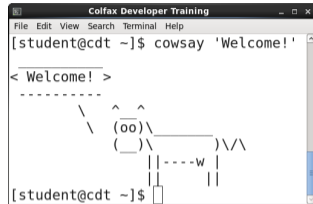
Welcome aboard. Post questions today!

## 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. PERFORMANCE OPTIMIZATION**

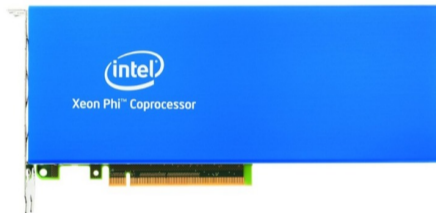
## Intel Xeon Processor



Current: Broadwell  
Upcoming: Skylake

Multi-Core Architecture

## Intel Xeon Phi Coprocessor, 1st generation



Knights Corner (KNC)

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

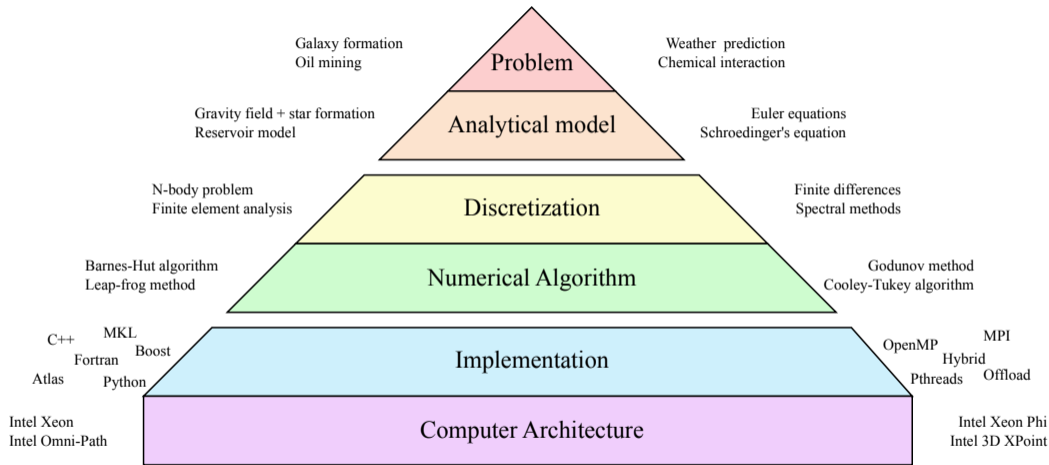


\* socket and coprocessor versions

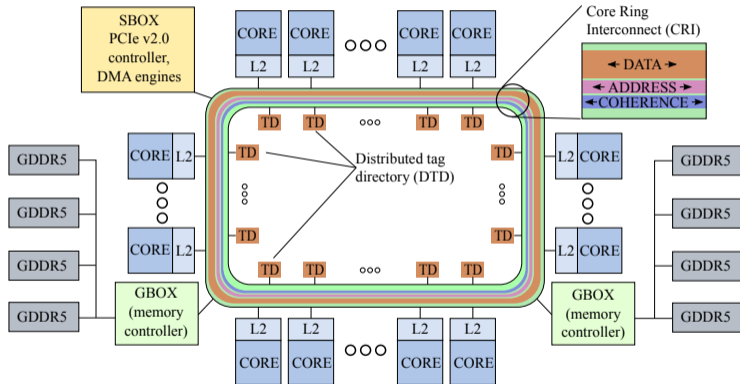
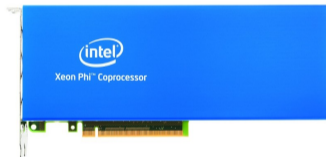
Knights Landing (KNL)

Intel Many Integrated Core (MIC) Architecture



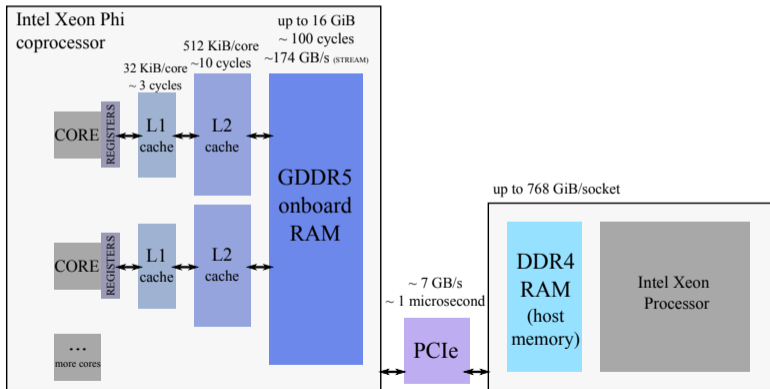
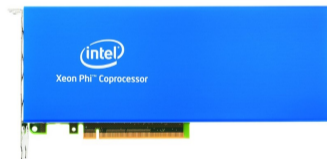


In a ring bus with distributed cache, data access locality is key.

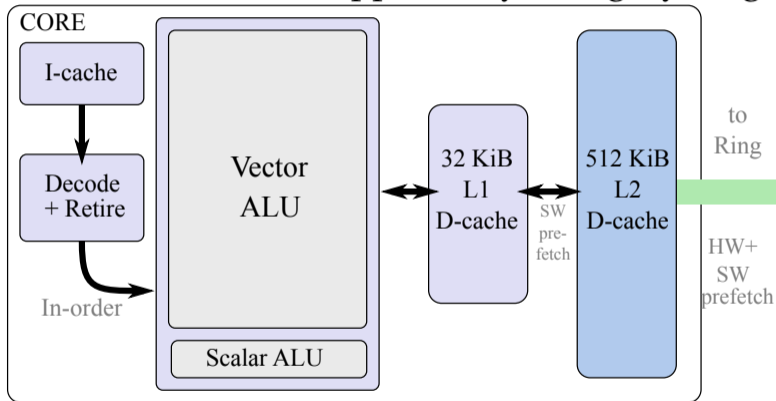
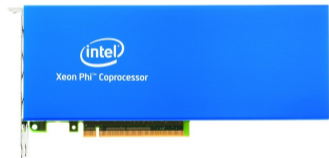


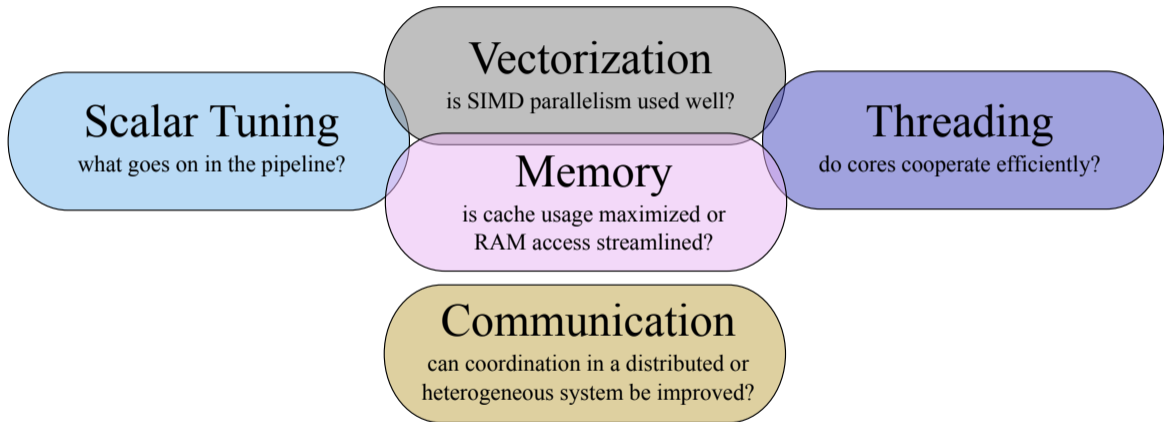
# KNC MEMORY ORGANIZATION

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



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







## **§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](#)

## Gravitational N-body dynamics:

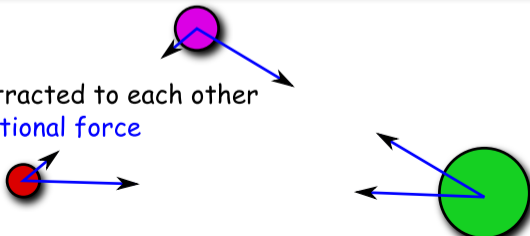
Newton's law of universal gravitation:

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

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  }
```

# 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  ...
```

# 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

# VECTORIZING WITH UNIT-STRIDE MEMORY ACCESS

Before:

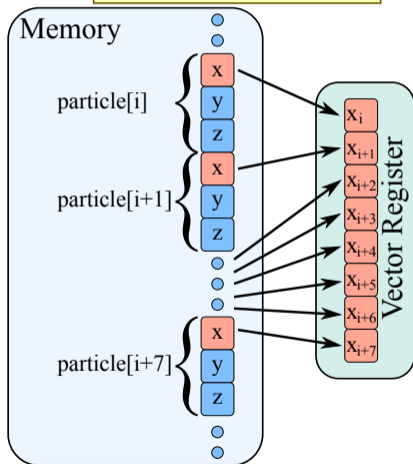
```
1 struct ParticleType {
2     float x, y, z, vx, vy, vz;
3 }; // ...
4     const float dx = particle[j].x - particle[i].x;
5     const float dy = particle[j].y - particle[i].y;
6     const float dz = particle[j].z - particle[i].z;
```

After:

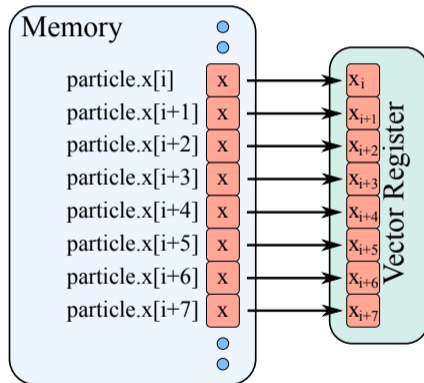
```
1 struct ParticleSet {
2     float *x, *y, *z, *vx, *vy, *vz;
3 }; // ...
4     const float dx = particle.x[j] - particle.x[i];
5     const float dy = particle.y[j] - particle.y[i];
6     const float dz = particle.z[j] - particle.z[i];
```

# WHY AOS TO SOA CONVERSION HELPS: UNIT STRIDE

Array of Structures  
(sub-optimal)



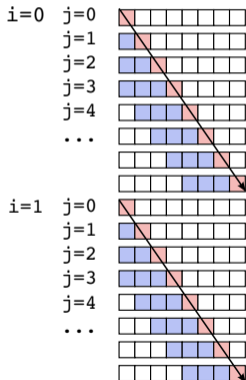
Structure of Arrays  
(optimal)



# LOOP TILING

## Original:

```
for (i=0; i<m; i++)
  for (j=0; j<n; j++)
    ...=...*b[j];
```



- - cached, LRU eviction policy
- - cache miss (read from memory, slow)
- - cache hit (read from cache, fast)

Cache size: 4

TILE=4

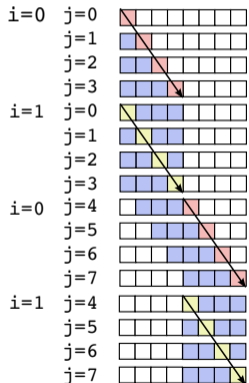
(must be tuned to cache size)

Cache hit rate without tiling: 0%

Cache hit rate with tiling: 50%

## Tiled:

```
for (jj=0; jj<n; jj+=TILE)
  for (i=0; i<m; i++)
    for (j=jj; j<jj+TILE; j++)
      ...=...*b[j];
```





# 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;

```

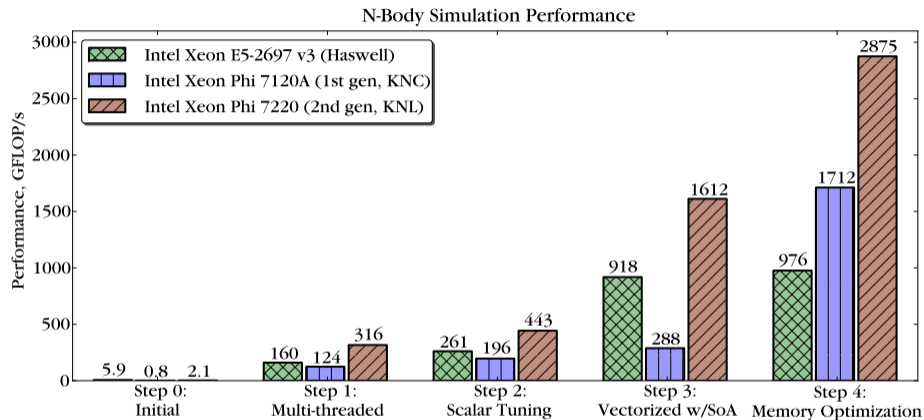
After: (tileSize = 16)

```

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;

```

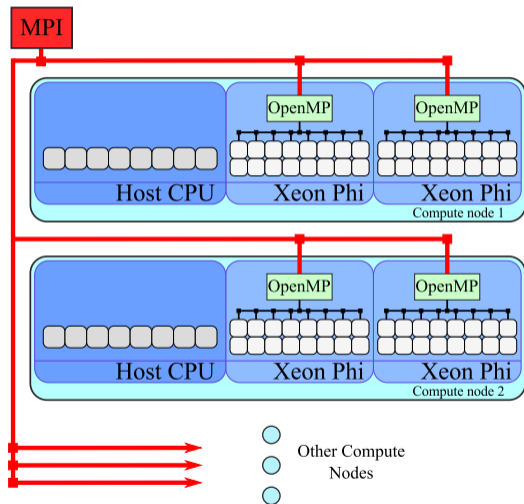
# IMPACT OF CODE OPTIMIZATION



Contributed as Chapter 23 in “[Intel Xeon Phi Processor High Performance Programming, Knights Landing Edition](#)” (2016)

# SCALING ACROSS A CLUSTER WITH COPROCESSORS WITH MPI

- ▶ MPI processes only on CPUs
- ▶ Divide 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.

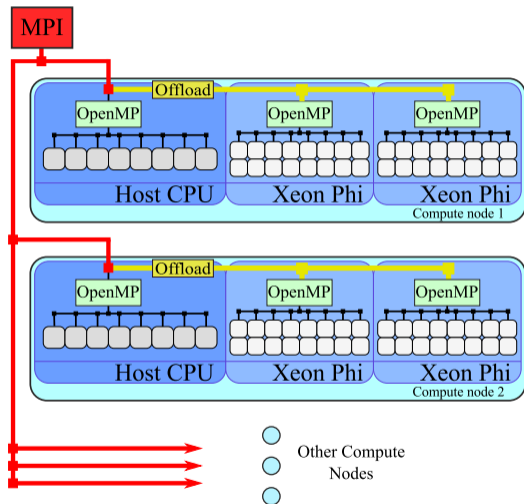
```

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

- ▶ MPI processes only on CPUs
- ▶ Divide data between coprocessors
- ▶ Concurrent offload from multiple host threads
- ▶ Synchronize data between nodes with MPI



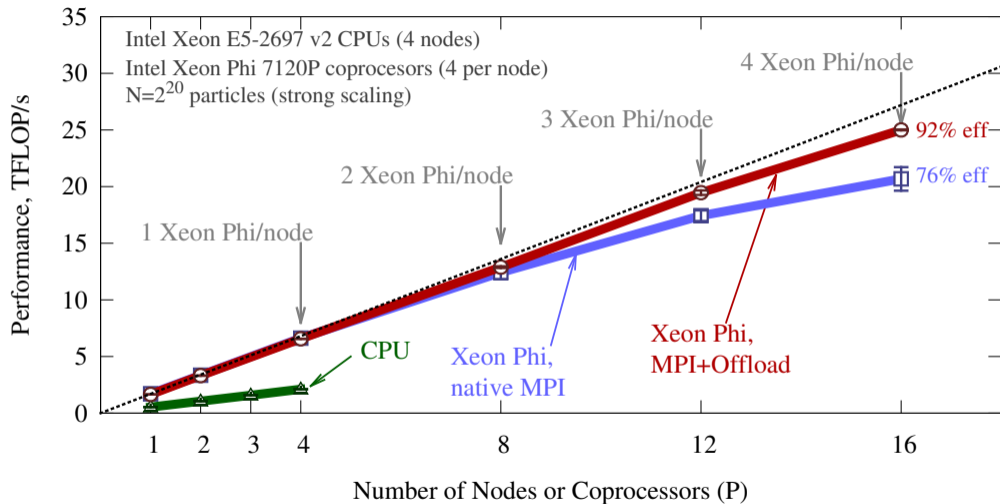
# MPI WITH OFFLOAD IMPLEMENTATION

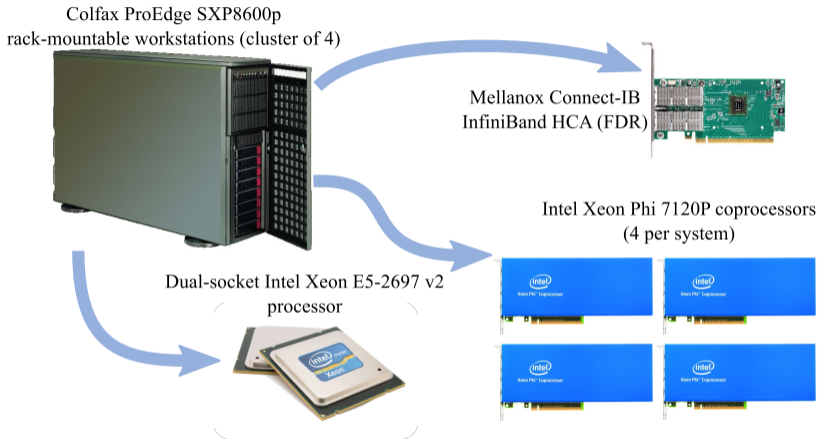
```

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

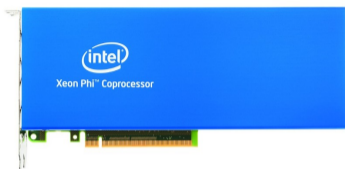




<http://xeonphi.com/workstations>



# 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”](#)

# 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 SERIES "KNIGHTS LANDING":

PROGRAMMING AND OPTIMIZATION FOR  
INTEL XEON PHI X200 FAMILY

Free 2-hour video course



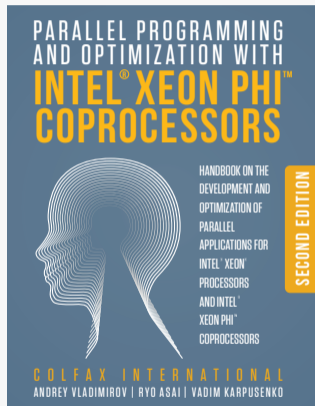
[colfaxresearch.com/how-knl/](http://colfaxresearch.com/how-knl/)

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

## Parallel Programming and Optimization with Intel® Xeon Phi™ Coprorocessors

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.

**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 on vectorization in a streaming mode



▶

Consulting

Like
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 approach to solve your computing problem

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



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 approach to solve your computing problem

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



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 approach to solve your computing problem

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



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 approach to solve your computing problem

http://colfaxresearch.com/

colfaxresearch.com/how-16-09
LEARN MORE
© Colfax International, 2013-