Programming and Optimization for Intel® Architecture

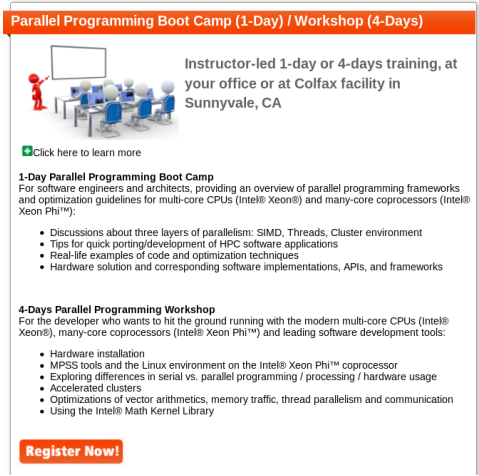
The Hands-On Workshop (HOW) Series

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.



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

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

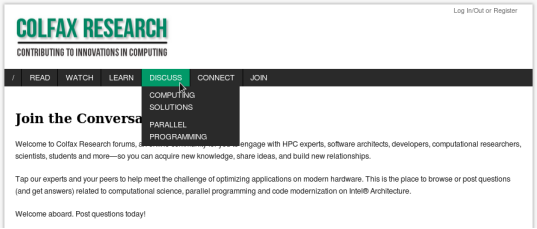
Additional resources:

- More workshops like this one: colfaxresearch.com/training
- Video courses: colfaxresearch.com/video-courses

Get Your Questions Answered

Chat (current):
colfaxresearch.com/how-16-05

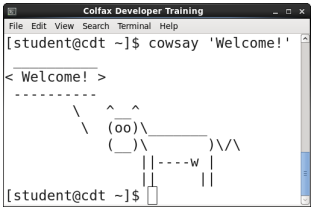
Forums (technical):
colfaxresearch.com/discussion



Email (organizational):
`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

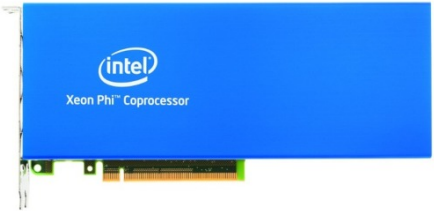
§2. Performance Optimization

Computing Platforms

Intel Xeon Processor

Current: Broadwell
Upcoming: Skylake

Intel Xeon Phi Coprocessor, 1st generation



Current: Knights Corner (KNC)

Intel Xeon Phi Processor, 2nd generation*

* socket and coprocessor versions

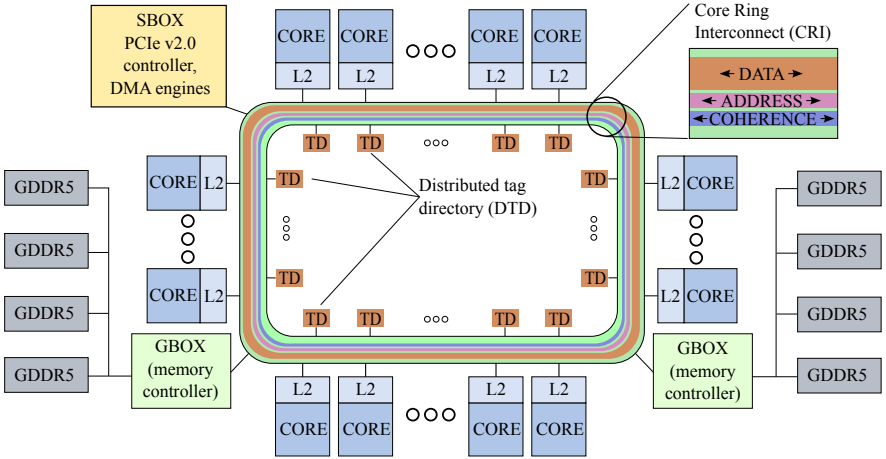
Upcoming: Knights Landing (KNL)

Multi-Core Architecture

Intel Many Integrated Core (MIC) Architecture

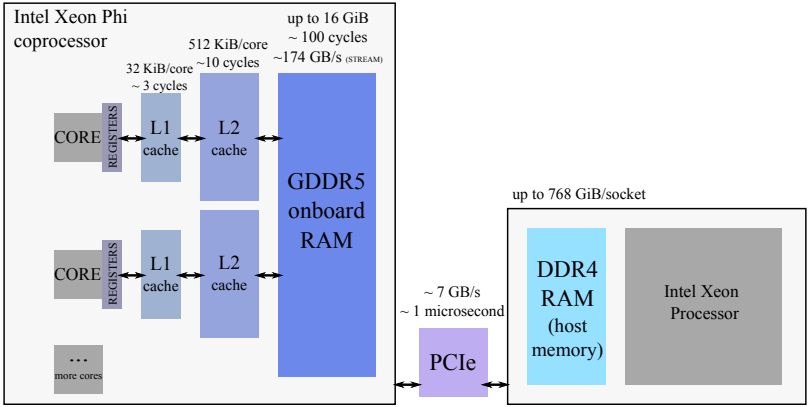
KNC Die Organization

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



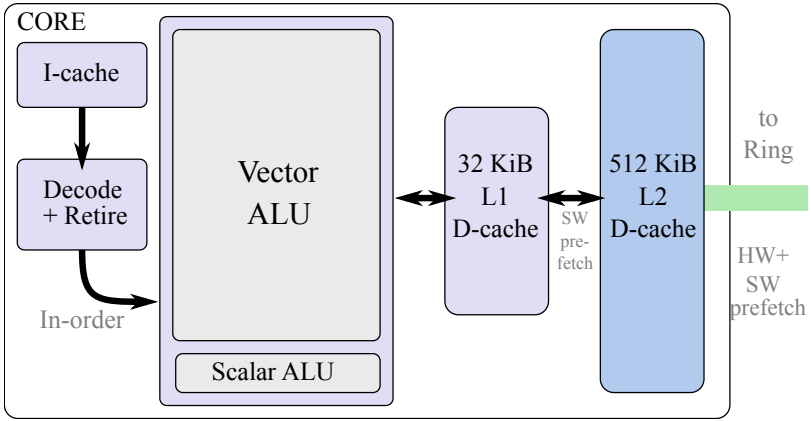
KNC Memory Organization

- Direct access to ≤ 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

COLFAX Customized Solutions

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

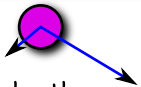
Gravitational N-body dynamics:

Newton's law of universal gravitation:

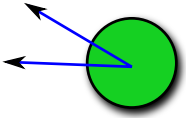
$$M_i \vec{R}_i''(t) = G \sum_j \frac{M_i M_j}{\left|\vec{R}_i - \vec{R}_j\right|^3} \left(\vec{R}_j - \vec{R}_i\right)$$

where:

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



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:

```
struct ParticleType {
  float x, y, z;
  float vx, vy, vz;
};
```

`main()` allocates an array of `ParticleType`:

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

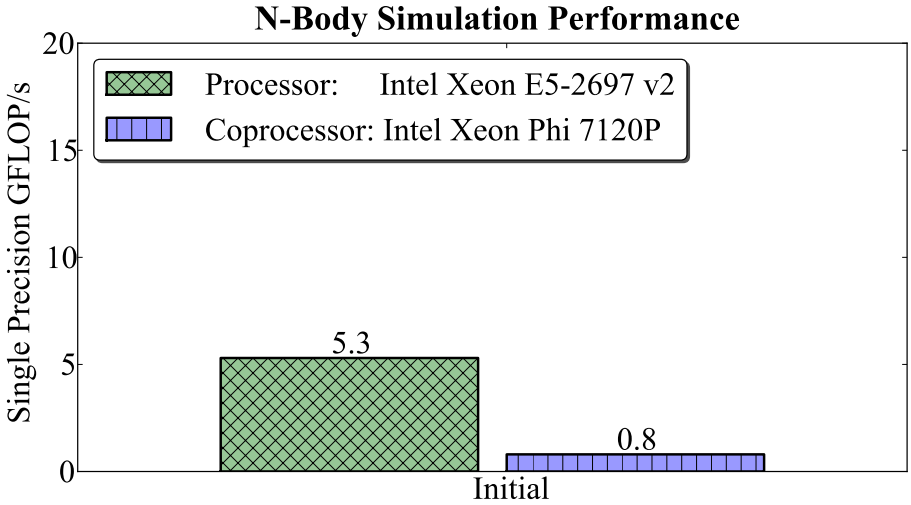
Particle propagation step is timed:

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


Particle Update Engine

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


Performance of Initial Implementation



Incorporating Thread Parallelism

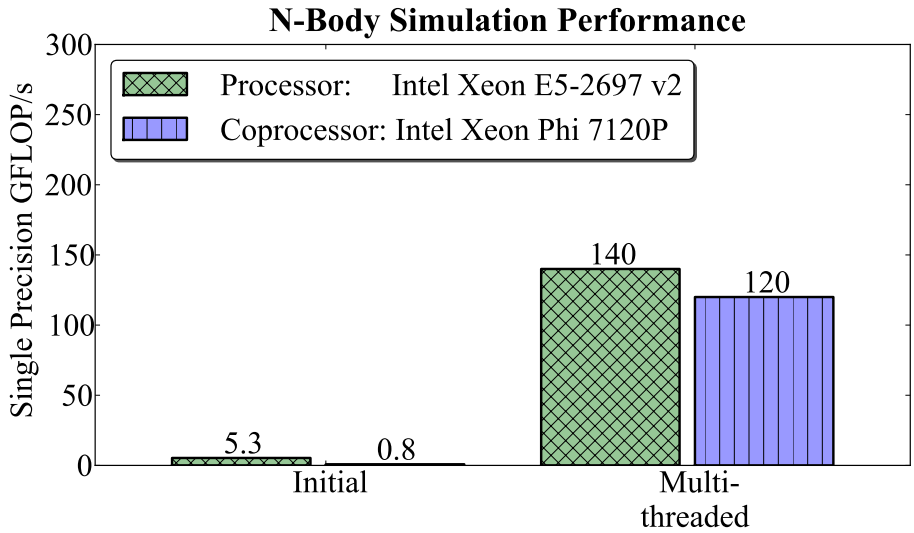
Before:

```
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

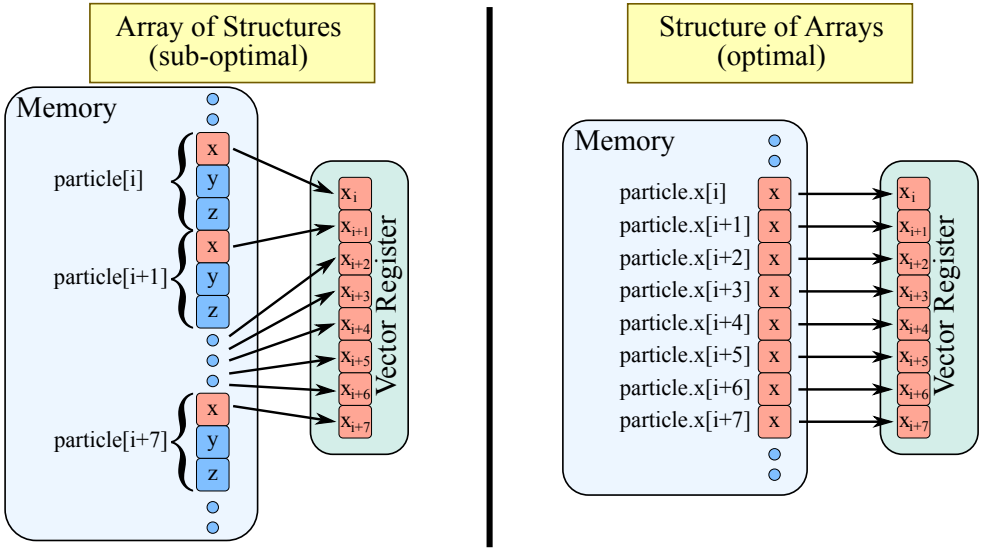
Before:

```
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;
```

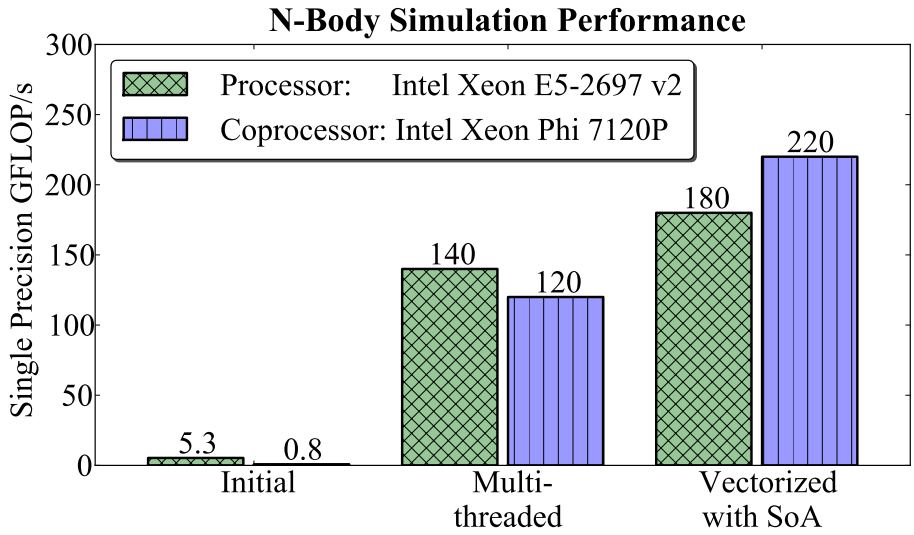
After:

```
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



Performance with Improved Vectorization



Improving Scalar Expressions

Before:

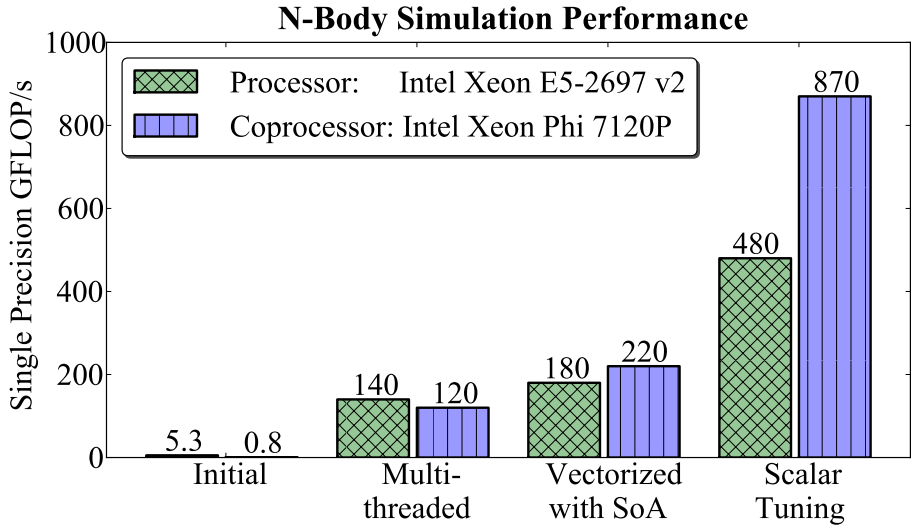
```
1       const float drSquared  = dx*dx + dy*dy + dz*dz + 1e-20;
2       const float drPower32  = pow(drSquared, 3.0/2.0);
3       // Calculate the net force
4       Fx += dx/drPower32;  Fy += dy/drPower32;  Fz += dz/drPower32;
```

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 → 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

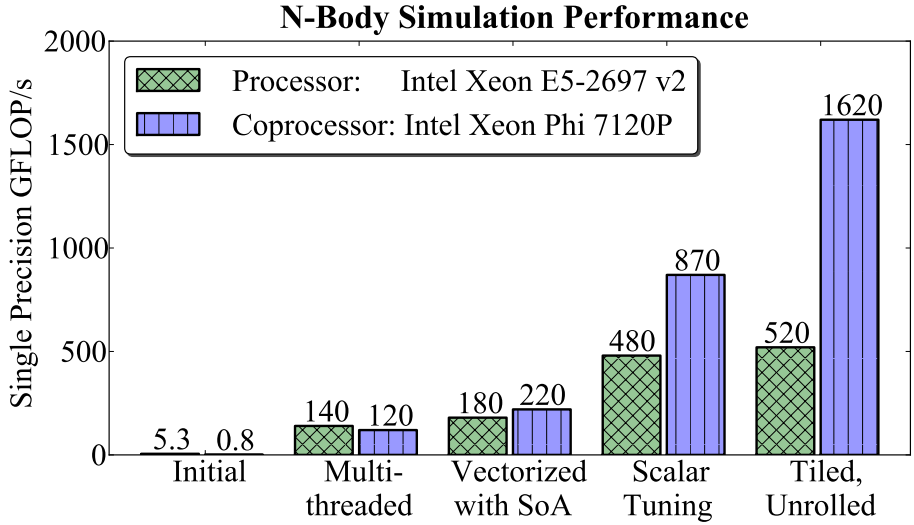
Before:

```
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`)

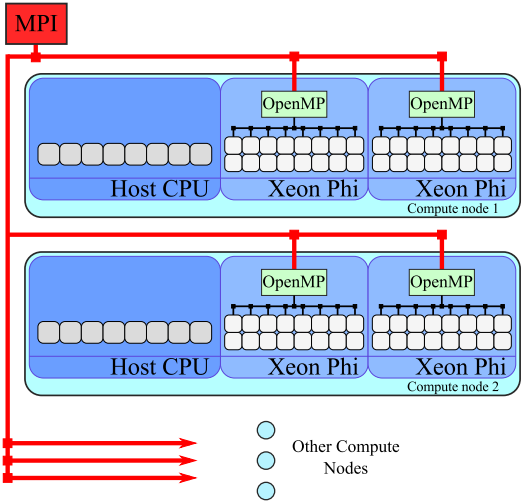
```
1   for (int ii = 0; ii < nParticles; ii += tileSize) { // Particle blocks
2     float Fx[tileSize], Fy[tileSize], Fz[tileSize]; // Force on particle block
3     Fx[:] = Fy[:] = Fz[:] = 0;
4 #pragma unroll(tileSize)
5     for (int j = 0; j < nParticles; j++) { // Particles that exert force
6       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



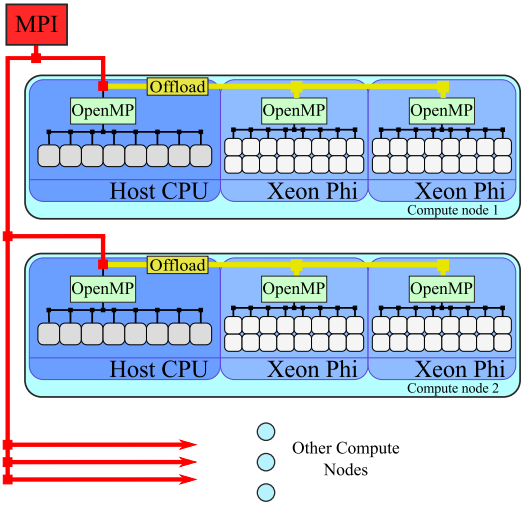
Core of MPI-only Implementation

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

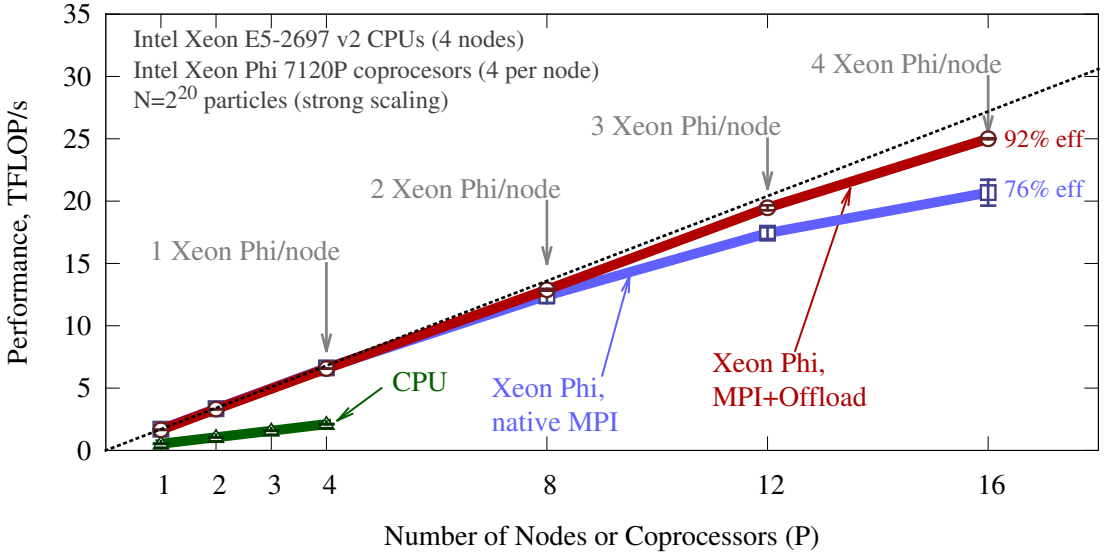
- We put MPI processes only on CPUs
- Subdivide data between coprocessors
- Concurrent offload from multiple host threads
- Synchronize data between nodes with MPI



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          // ...
```

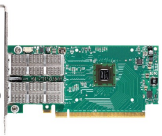

Results with MPI+Offload



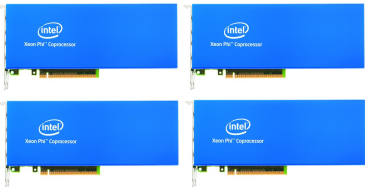
Comparative Benchmarks and System Configuration

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

Mellanox Connect-IB
InfiniBand HCA (FDR)



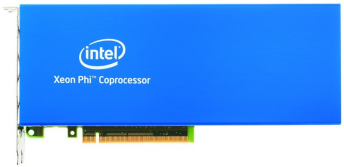
Intel Xeon Phi 7120P coprocessors
(4 per system)



Dual-socket Intel Xeon E5-2697 v2
processor

http://xeonphi.com/workstations

Coprocessor vs Processor Performance



vs.

One Intel Xeon Phi 7120P coprocessor

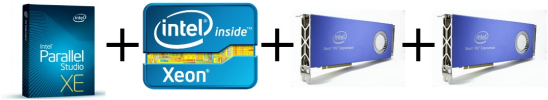
Two Intel Xeon E5-2697 v2 CPUs

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

See also "Intel Xeon Product Family: Performance Brief"

Additional Case Studies

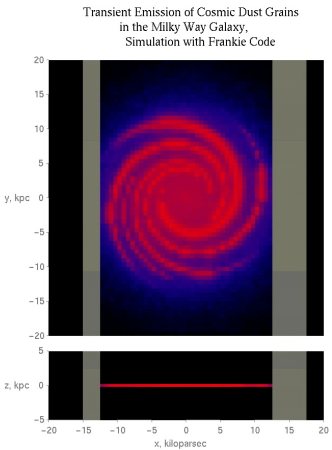
Astrophysical Code HEATCODE: an Offload Story



Porting to Intel® Xeon Phi™ coprocessors

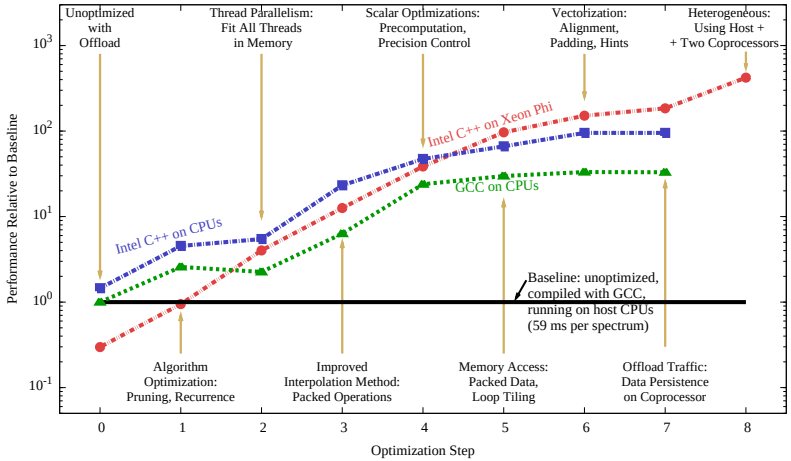
- We ported Frankie code using explicit offload model
- Same code & optimization methods for Xeon Phi™
- Simultaneous calculations on CPU and coprocessors with automatic load balancing was easy to implement
- With two Intel® Xeon Phi™ coprocessors, performance for high-res calculations is 3.2x better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

Goal achieved!



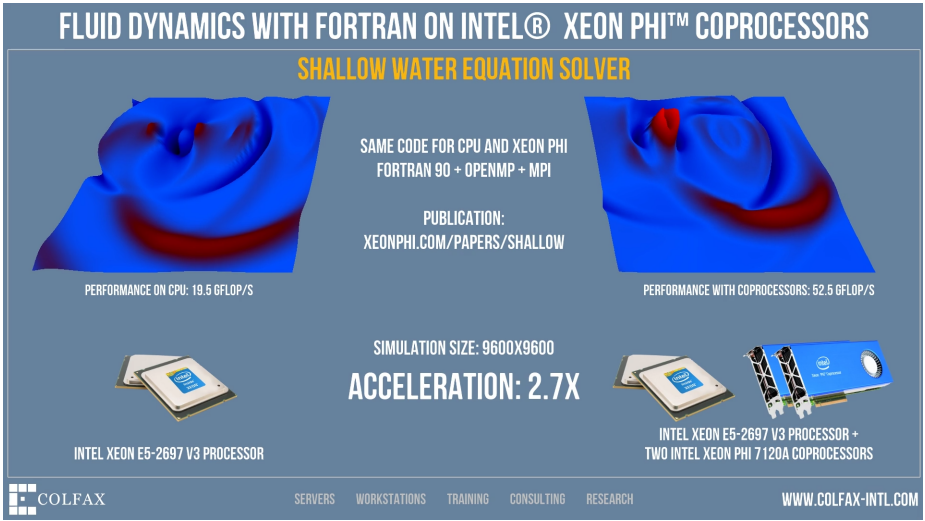
http://xeonphi.com/papers/heatcode

Astrophysical Code HEATCODE: an Offload Story



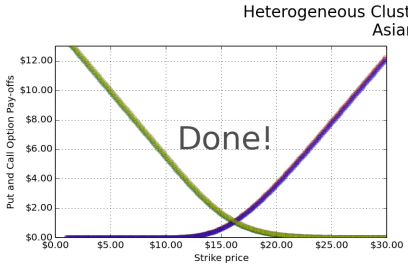
http://xeonphi.com/papers/heatcode

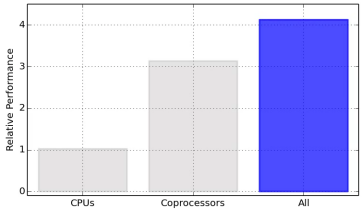
Computational Fluid Dynamics: Legacy Code

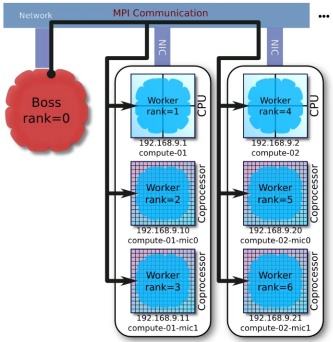


http://xeonphi.com/papers/shallow

Asian Option Pricing: Heterogeneous Clustering







COLFAX
Customized Solutions

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

colfaxresearch.com/how-tools-16-06

Developer's Guide to Knights Landing

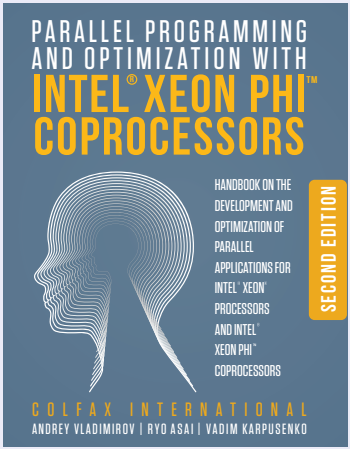
colfaxresearch.com/knl-webinar/

Textbook

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

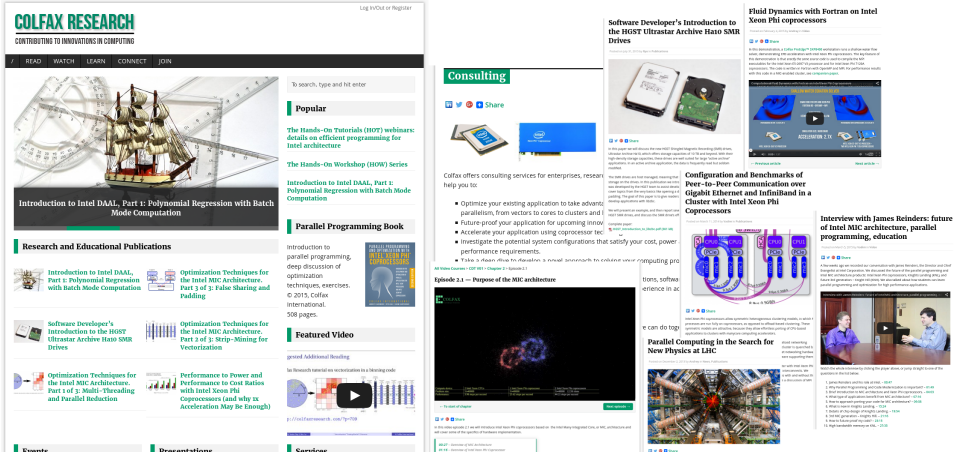
Parallel Programming
and Optimization with
Intel® Xeon Phi™
Coprocessors

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



http://xeonphi.com/book

Colfax Research



http://colfaxresearch.com/

(already registered? get $10 off the book)

