



PROGRAMMING AND OPTIMIZATION FOR INTEL® ARCHITECTURE

Hands-On Workshop (HOW) Series "Deep Dive"
Session 6

Colfax International — colfaxresearch.com

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

- ▷ Module I. Programming Models
 - 01. Intel Architecture and Modern Code
 - 02. Xeon Phi, Coprocessors, Omni-Path
- ▷ Module II. Expressing Parallelism
 - 03. Automatic vectorization
 - 04. Multi-threading with OpenMP
 - 05. Distributed Computing, MPI
- ▷ Module III. Performance Optimization
 - 06. Optimization Overview: N-body
 - 07. Scalar tuning, Vectorization
 - 08. Common Multi-threading Problems
 - 09. Multi-threading, Memory Aspect
 - 10. Access to Caches and Memory

HOW SERIES ONLINE

Course page:
colfaxresearch.com/how-series

- ▷ Slides
- ▷ Code
- ▷ Video
- ▷ Chat

More workshops:
colfaxresearch.com/training



GET YOUR QUESTIONS ANSWERED: CHAT

leofernandesmo Hello from Recife/Brazil

gaesansi Hi, Naples, Italy

info2harish Harish f rom INDIA

hpcfan Hello, from Texas.

radekg1000 Hi, Poznan/Poland

zanton hello, Tokyo, JP

colfaxresearch.com/how-series

GET YOUR QUESTIONS ANSWERED: FORUMS

[READ](#)[WATCH](#)[LEARN](#)[FORUMS](#)[CONNECT](#)[JOIN](#)

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/forum

HANDS-ON EXERCISES AND REMOTE ACCESS

- ▶ 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. PERFORMANCE OPTIMIZATION

COMPUTING PLATFORMS

9

Intel Xeon
Processor



Current: Broadwell
Upcoming: Skylake

Intel Xeon Phi
Coprocessor, 1st generation



Knights Corner (KNC)

Intel Xeon Phi
Processor, 2nd generation*



* socket and coprocessor versions

Knights Landing (KNL)

Multi-Core Architecture

Intel Many Integrated Core (MIC) Architecture

ONE CODE FOR ALL PLATFORMS

multi-core
vectorization
high-BW RAM
HPC Fabrics

cognisant of
ARCHITECTURE

compliant with
STANDARDS

e.g., OpenMP:
+ threading
+ vectorization
+ offload

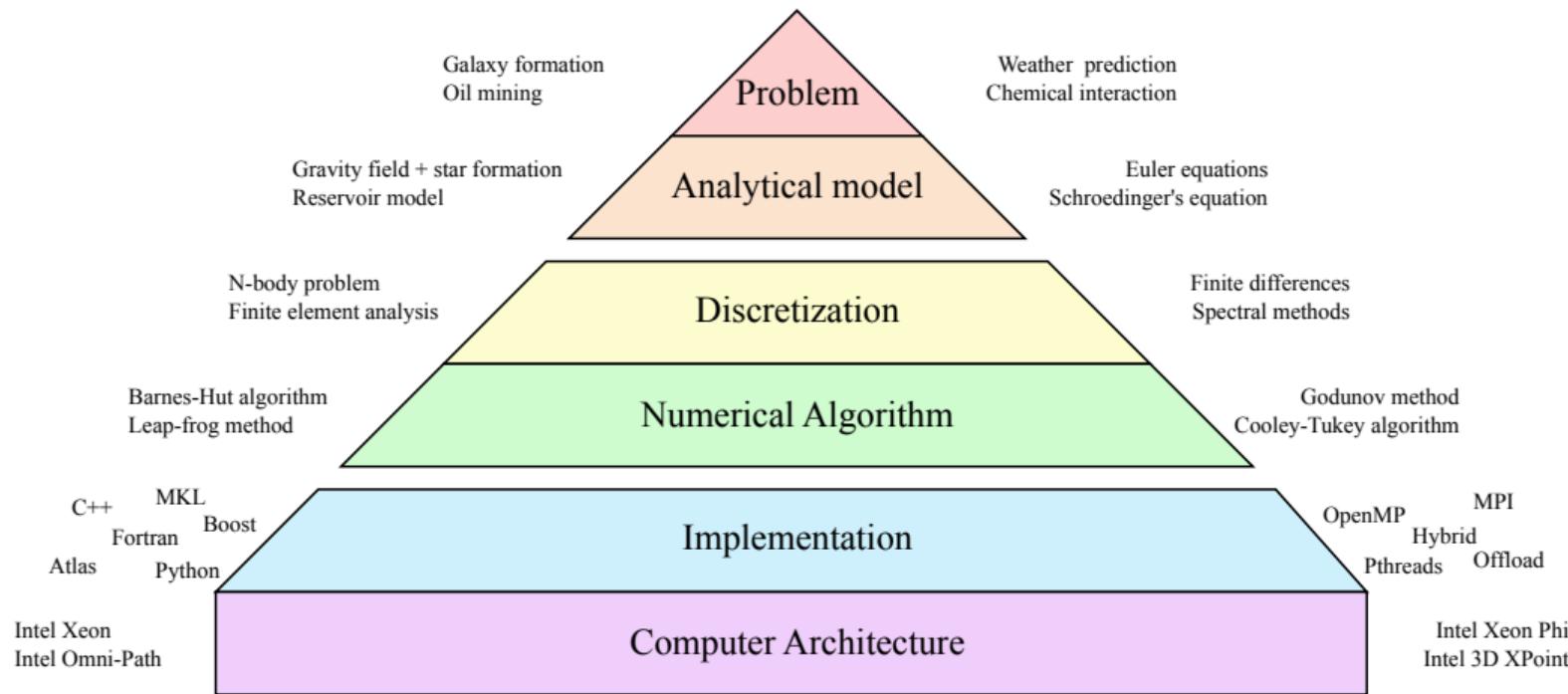
MODERN CODE

OPTIMIZED

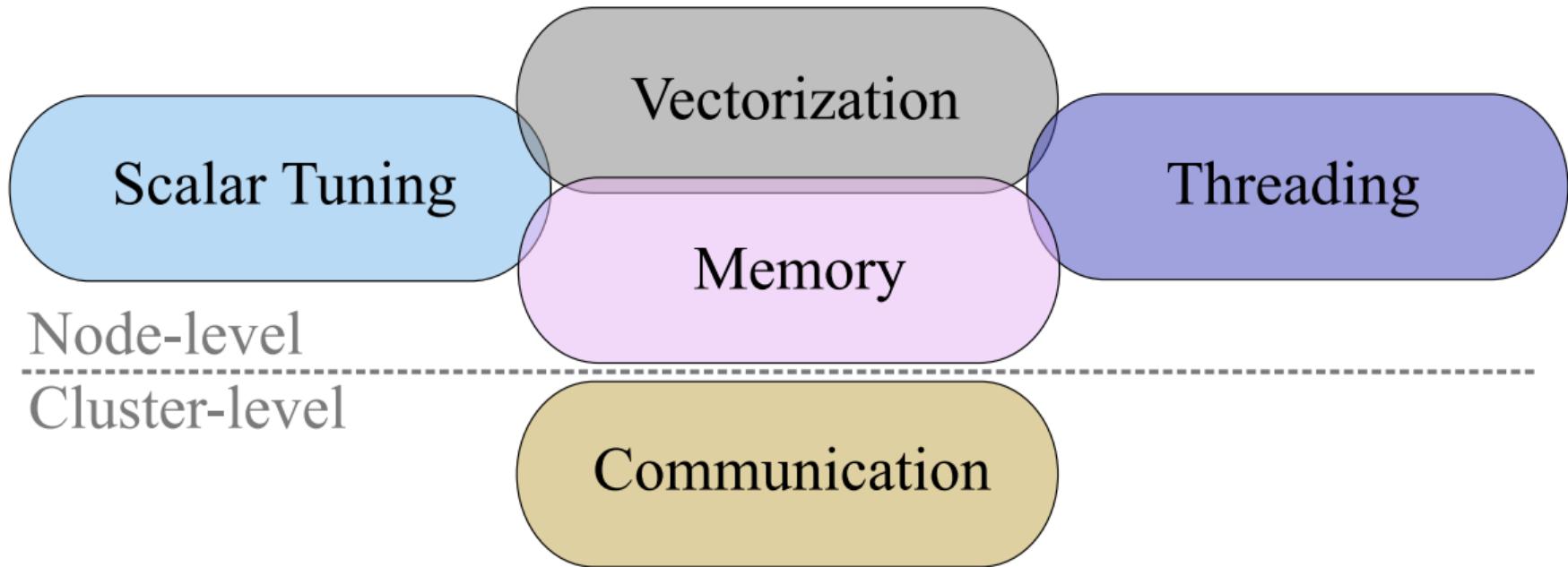
PORTABLE

FUTURE-PROOF

COMPUTING IN SCIENCE AND ENGINEERING

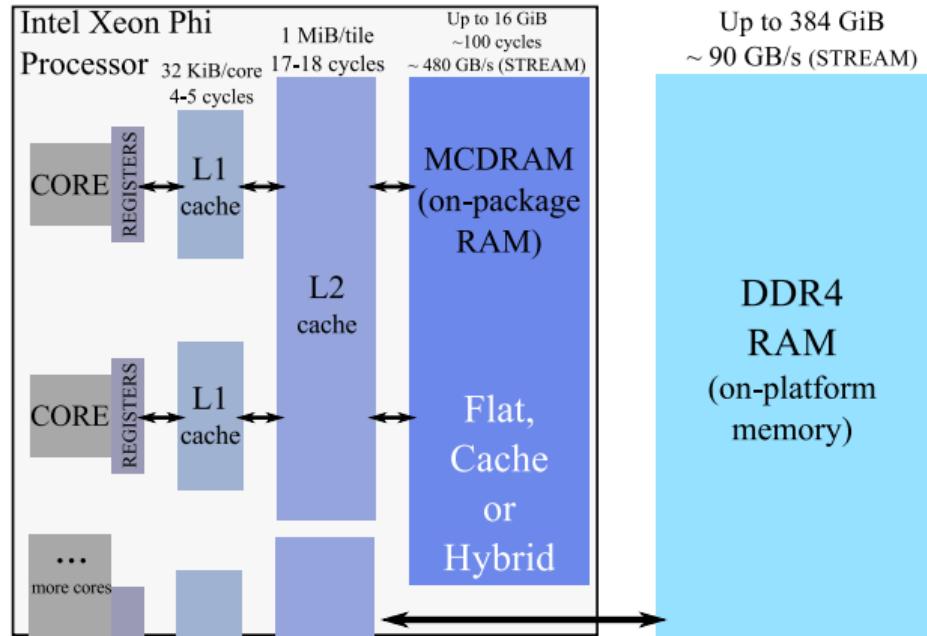


OPTIMIZATION AREAS



KNL MEMORY ORGANIZATION (BOOTABLE)

- ▶ On-package high-bandwidth memory (HBM) – MCDRAM
- ▶ Optimized for arithmetic performance and bandwidth (not latency)



§3. N-BODY SIMULATION



PHYSICS

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: <https://colfaxresearch.com/nbody-basic>

Demo: [click here](#)

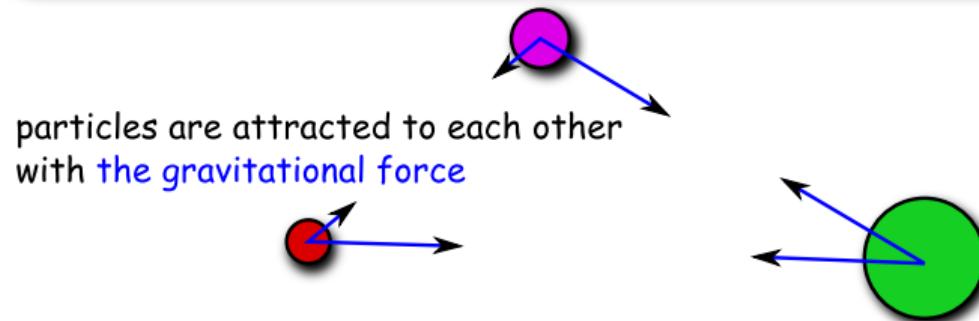
Gravitational N-body dynamics:

Newton's law of universal gravitation:

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

where:

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



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
```

OPTIMIZATION

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          ...
6      }
```

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             ...
7         }
```

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/sqrdf(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

VECTORIZING WITH UNIT-STRIDE MEMORY ACCESS

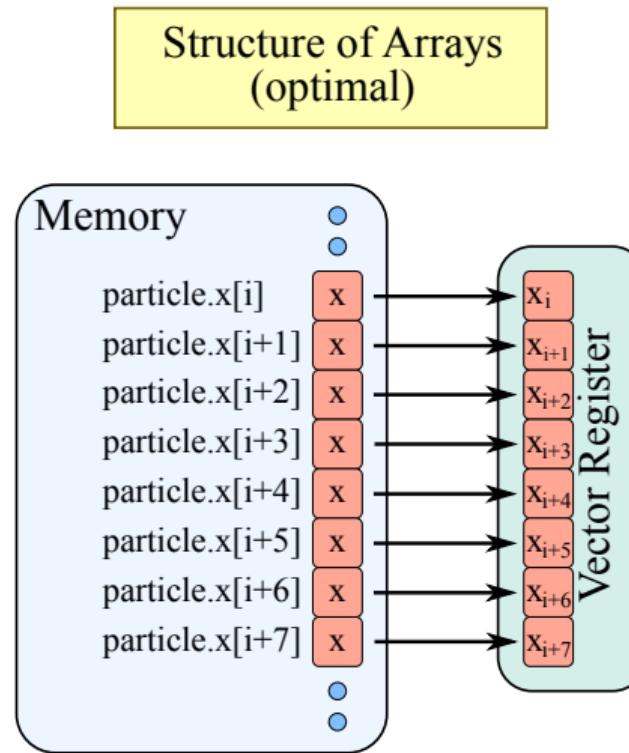
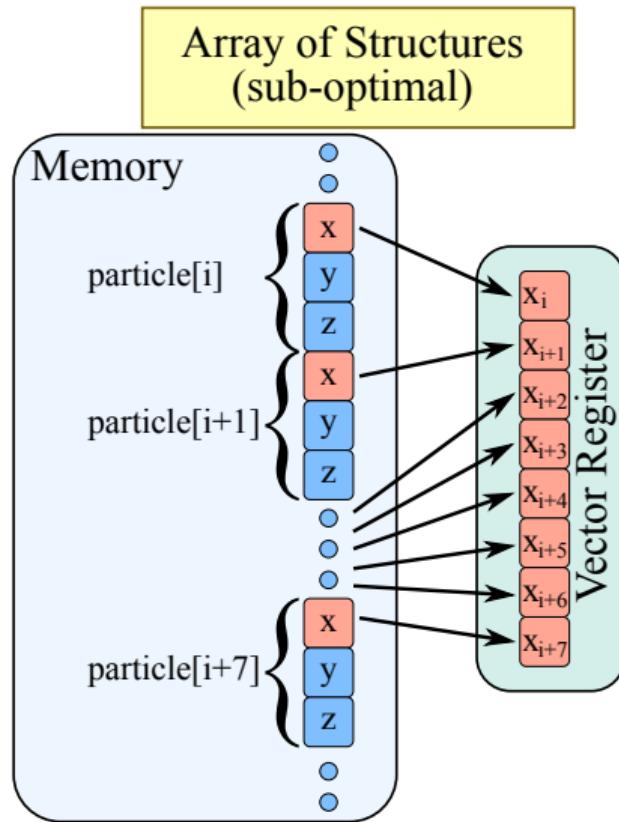
Before:

```
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];
```

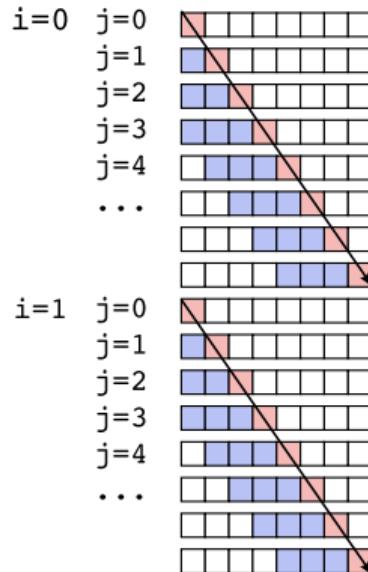
WHY AOS TO SOA CONVERSION HELPS: UNIT STRIDE



LOOP TILING: REGISTER BLOCKING

Original:

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



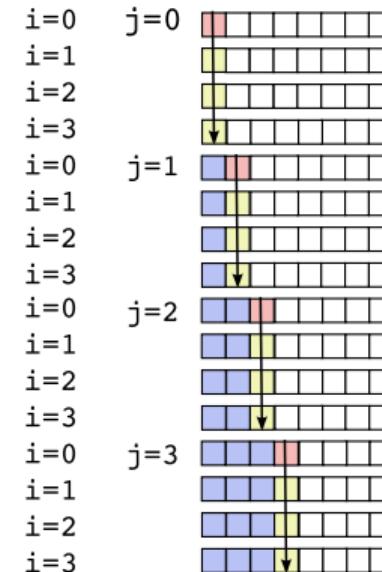
█ - cached, LRU eviction policy
█ - cache miss (read from memory, slow)
█ - cache hit (read from cache, fast)

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 (ii=0; ii<m; ii+=TILE)
    for (j=0; j<n; j++)
        for (i=ii; i<ii+TILE; i++)
            ...=...*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;

```

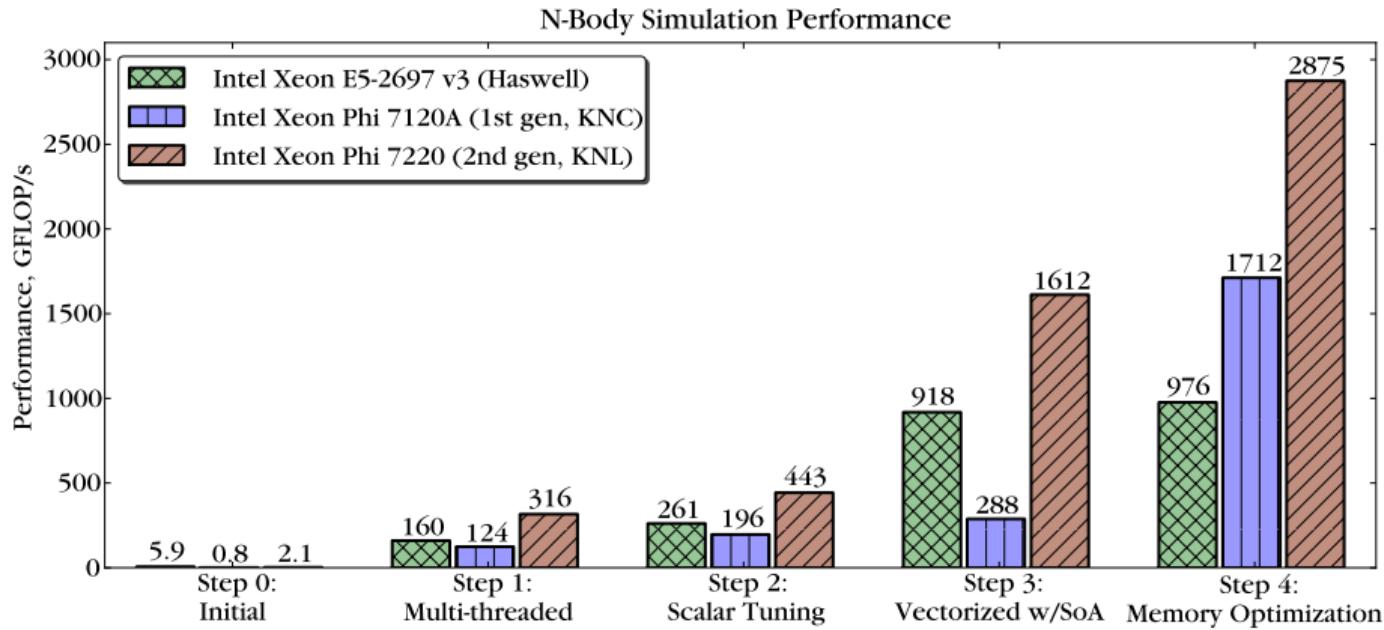
After: (tileSize = 16)

```

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

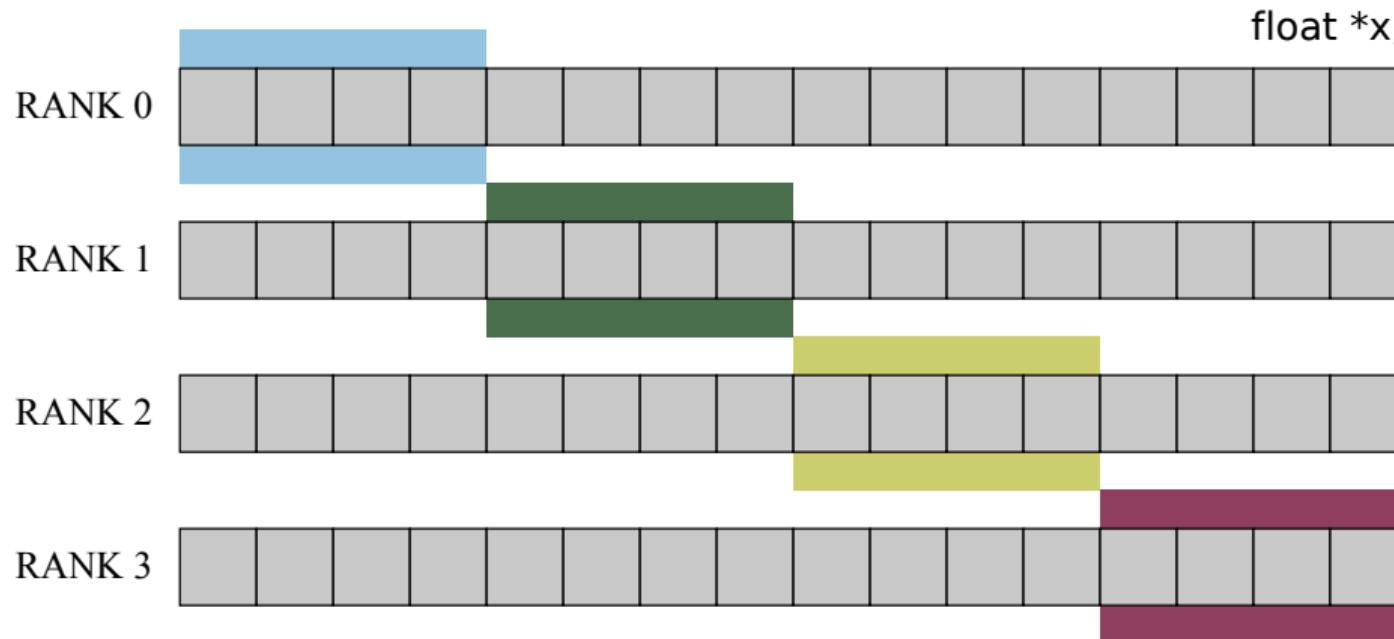


Contributed as Chapter 23 in “[Intel Xeon Phi Processor High Performance Programming, Knights Landing Edition](#)” (2016)

MPI IMPLEMENTATION

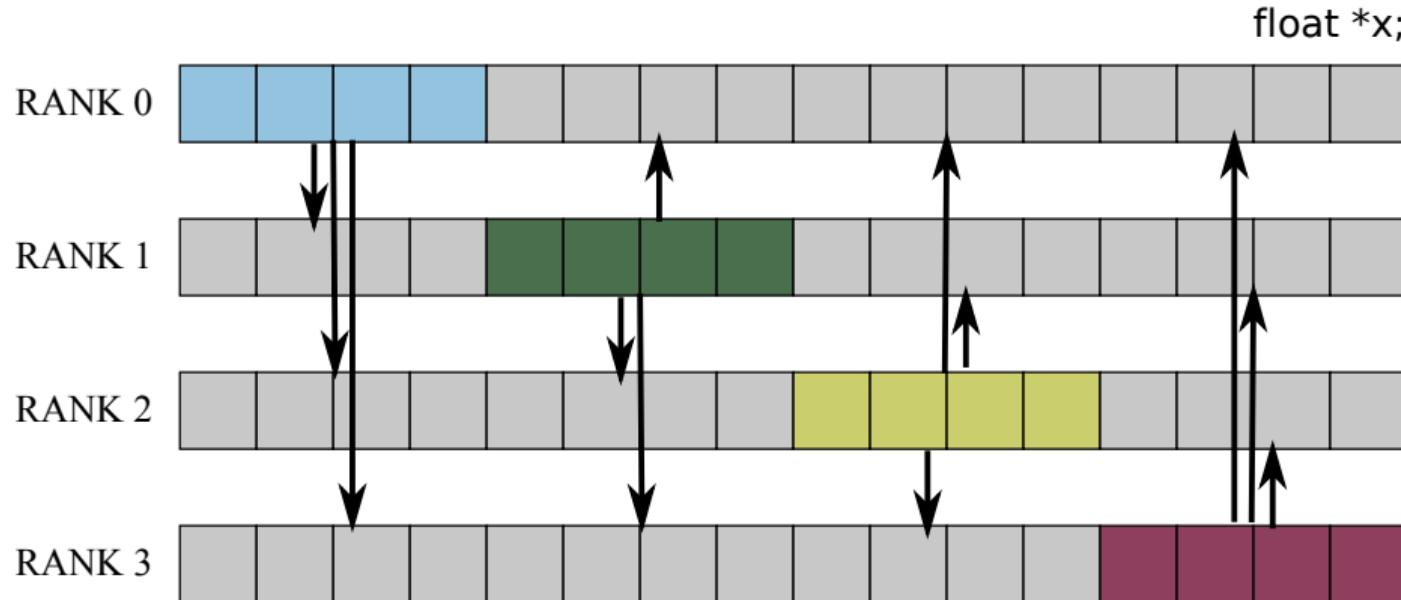
WORK DISTRIBUTION

- ▷ All processes start with same complete set of particle coordinates.
- ▷ Each processor moves only its share of particles



COMMUNICATION

- ▶ After time step, need to propagate modified particles to all peers
- ▶ Use the Allgather operation from MPI



COMMUNICATION

- ▷ All processors end up with the same data set again
- ▷ Only x, y, z need to be propagated. vx, vy, vz stay local



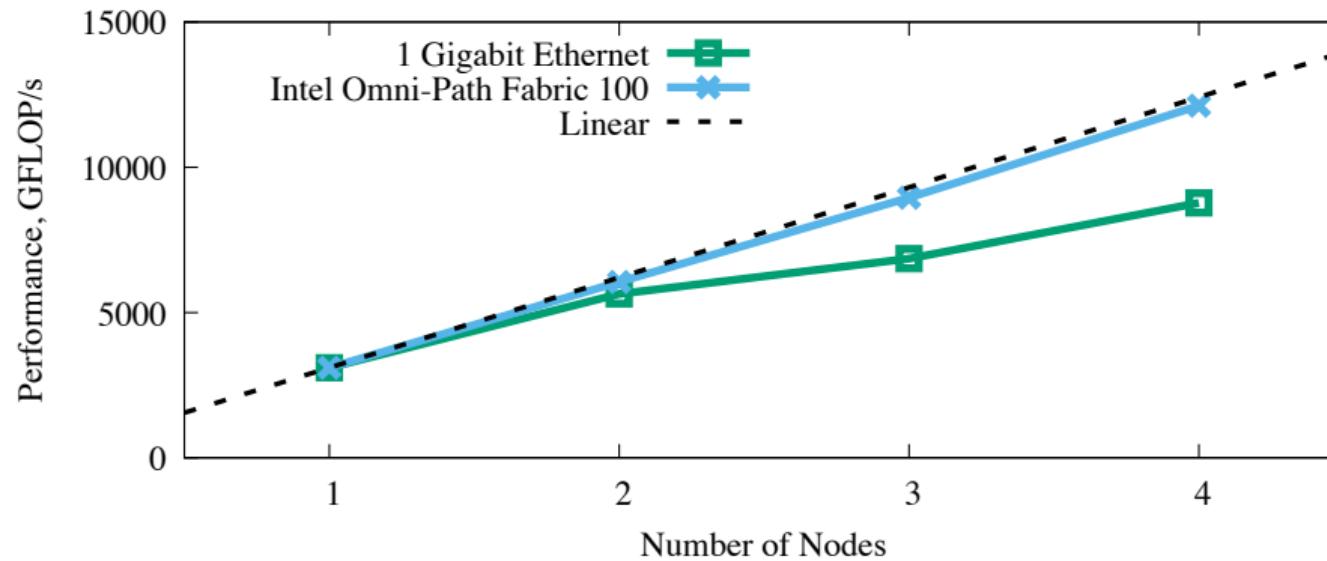
CORE OF MPI-ONLY IMPLEMENTATION

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    // ...
```

PERFORMANCE WITH MPI

Intel Xeon Phi 7210 processors, $N = 2^{18}$ particles



Impact of communication decreases with increasing N .

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: scalar tuning, optimization of vectorization.

COLFAX RESEARCH

COLFAX RESEARCH
CONTRIBUTING TO INNOVATIONS IN COMPUTING

READ WATCH LEARN CONNECT JOIN

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

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 1 of 3: False Sharing and Padding

Software Developer's Introduction to the HOST Ultrastar Archive Hard 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 Acceleration May Be Enough)

Events

Demonstrations

Log In/Out or Register

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 I
- Future-proof your application for upcoming Intel® Xeon® Processors
- Accelerate your application using coprocessor technology
- Investigate the potential system configurations that satisfy your cost, power performance requirements.
- Take a closer look at the hardware & virtual environments for evaluation servers computing products.

Episode 2.3 --- Purpose of the MIC architecture

Parallel Programming Book

Introduction to parallel programming, deep discussion of optimization techniques, exercises. © 2015, Colfax International. 508 pages.

Featured Video

Parallel Computing in the Search for New Physics at LHC

Fluid Dynamics with Fortran on Intel Xeon Phi coprocessors

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

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

<https://colfaxresearch.com/>