From Scalar & Serial to Vector & Multi-Threaded

The Hands-On Tutorial (HOT) Series

Andrey Vladimirov, PhD — Colfax International
colfaxresearch.com

 © Colfax International, 2013–2016

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.

About the Series

Hands-On Tutorial (HOT) series: webinars on efficient programming for the Intel architecture. Select topics of parallel programming and performance optimization with detailed practical demonstrations.

colfaxresearch.com/hot-16-03

© Colfax International, 2013–2016

Learn More

Comprehensive Hands-On Workshop (HOW) series begins April 18, 2016. Free remote access to training servers (space is limited!):

colfaxresearch.com/how-16-04

© Colfax International, 2013–2016

§2. Intel Architecture

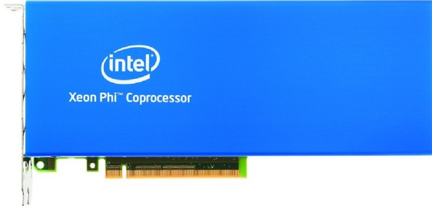
 © Colfax International, 2013–2016

Computing Platforms

Intel Xeon Processor

Current: Haswell
Upcoming: Broadwell

Intel Xeon Phi Corocessor, 1st generation



Current: Knights Corner (KNC)

Intel Xeon Phi Processor, 2nd generation*

* socket and coprocessor versions

Upcoming: Knights Landing (KNL)

Multi-Core Architecture

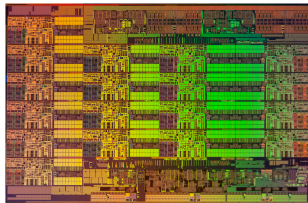
Intel Many Integrated Core (MIC) Architecture

© Colfax International, 2013–2016

Intel Xeon CPU: Purpose and Specifications

General-purpose platform for demanding computing applications.

- Up to ~ 1 TFLOP/s in DP
- Up to ~ 2 TFLOP/s in SP
- Up to 768 GiB DDR4 RAM
- ~ 126 GB/s bandwidth
- Hardware-rich: forgiving of sub-optimal code

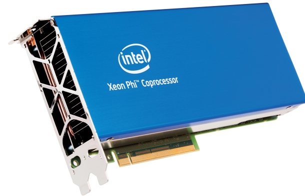


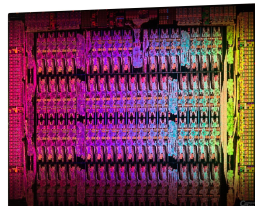
 © Colfax International, 2013–2016

Intel Xeon Phi Processors (1st Gen)

Specialized platform for demanding computing applications.

- PCIe end-point device
- $\sim$ 1.2 TFLOP/s in DP
- $\sim$ 2.4 TFLOP/s in SP
- Up to 16 GiB GDDR5 RAM
- $\sim$ 176 GB/s bandwidth
- Heterogeneous clustering
- Runs special Linux distribution



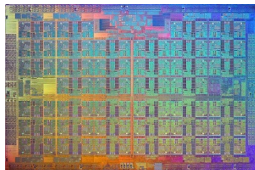


 © Colfax International, 2013–2016

Intel Xeon Phi Processors (2nd Gen)

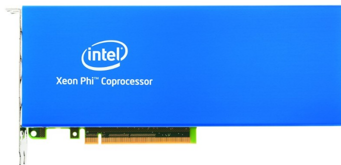
Specialized platform for demanding computing applications.

- Socket version or coprocessor
- 3+ TFLOP/s in DP
- 6+ TFLOP/s in SP
- Up to 16 GiB MCDRAM
- ~ 400 GB/s MCDRAM bandwidth
- Up to 384 GiB DDR4 RAM
- ~ 90 GB/s DDR4 bandwidth
- Supports common OS
- Public disclosures



© Colfax International, 2013–2016

“Standard Candle” Testbench



vs.

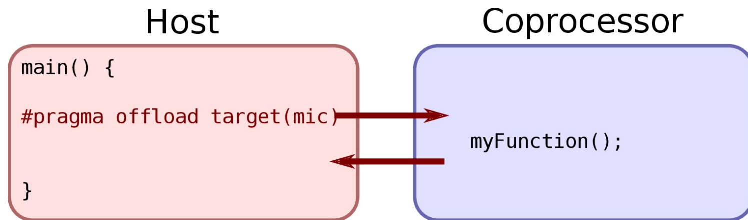
One Intel Xeon Phi 7120P coprocessor (2012)
TDP: 300 W, RCP: $4129

Two Intel Xeon E5-2697 v3 CPUs (2014)
TDP: 290 W, RCP: $5404

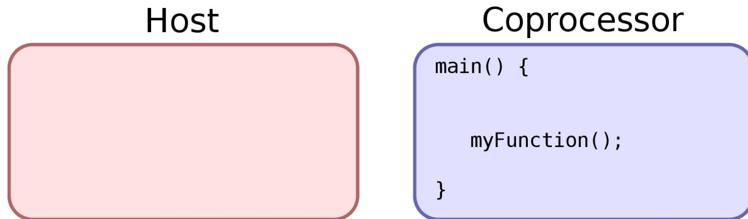
See also “Intel Xeon Product Family: Performance Brief”

Offload and Native models

- Offload model (explicit/virtual-shared memory/OpenMP 4.0):



- Native model (standalone application/MPI process):

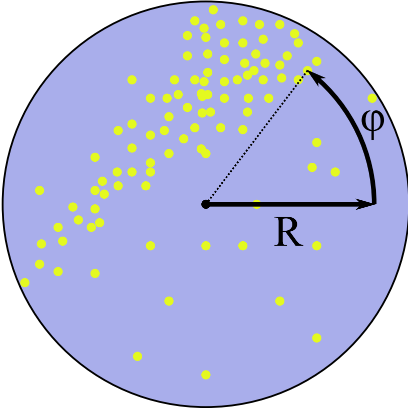


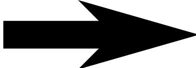
 © Colfax International, 2013–2016

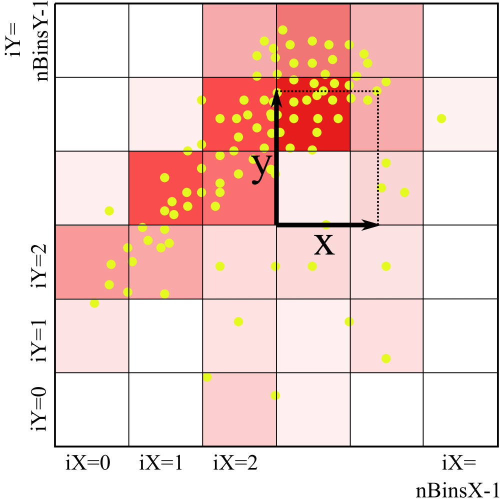
§3. Example Problem: Binning

© Colfax International, 2013–2016

Example Problem: Binning







 © Colfax International, 2013–2016

Initial Approach

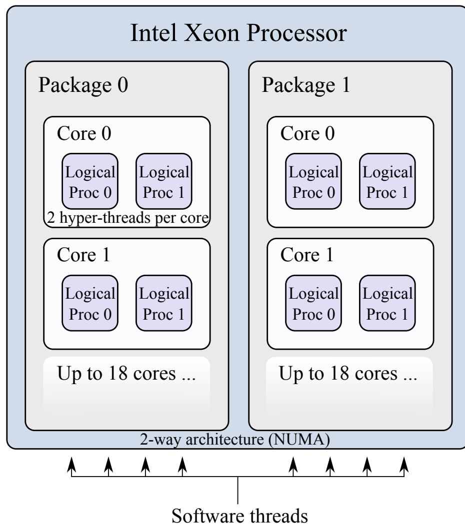
```
// Reference implementation: scalar, serial code without optimization
void BinParticlesReference(const InputDataType & inputData,
                           BinsType & outputBins) {
  // Loop through all particle coordinates
  for (int i = 0; i < inputData.numDataPoints; i++) {
    // Transforming from cylindrical to Cartesian coordinates:
    const FTYPE x = inputData.r[i]*COS(inputData.phi[i]);
    const FTYPE y = inputData.r[i]*SIN(inputData.phi[i]);

    // Calculating the bin numbers for these coordinates:
    const int iX = int((x - xMin)*binsPerUnitX);
    const int iY = int((y - yMin)*binsPerUnitY);

    // Incrementing the appropriate bin in the counter:
    outputBins[iX][iY]++;
  }
}
```


© Colfax International, 2013–2016

§4. Multi-Threading: Happy Cores

 © Colfax International, 2013–2016

Cores and Threads

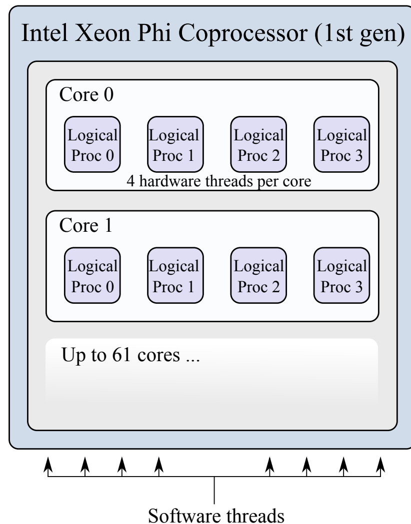


Hierarchy:
Packages ->
Cores ->
Logical Processors

Core presents itself as several logical processors due to one of:
a) hyper-threading (Xeon)
b) hardware threads (Xeon Phi)

Terminology:
OS Proc - numeric ID of logical processor
Software thread - stream of instructions in a process

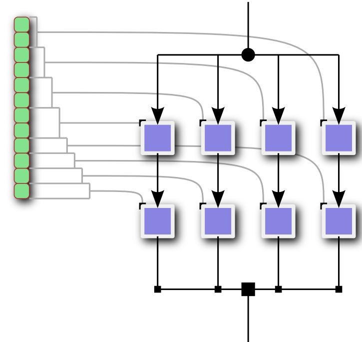
Jargon:
"physical core" = core
"logical core" = logical processor



 © Colfax International, 2013–2016

OpenMP Threads

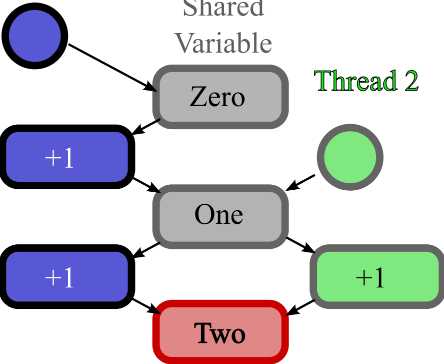
```
1  // Parallel, but INCORRECT implementation
2  void BinParticlesReference(
3     const InputDataType & inputData,
4                 BinsType & outputBins) {
5  // Distribute loop iterations across threads
6  #pragma omp parallel for
7    for (int i=0; i<inputData.numDataPoints; i++)
8      {
9        // ...
10       // Incrementing the appropriate
11       // bin in the counter:
12       outputBins[iX][iY]++;
13     }
14 }
```

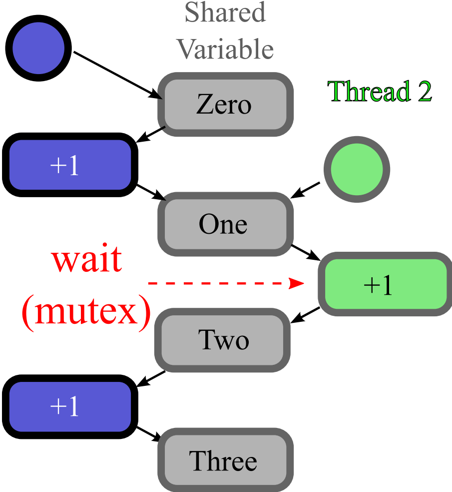


One software thread per logical processor or per core

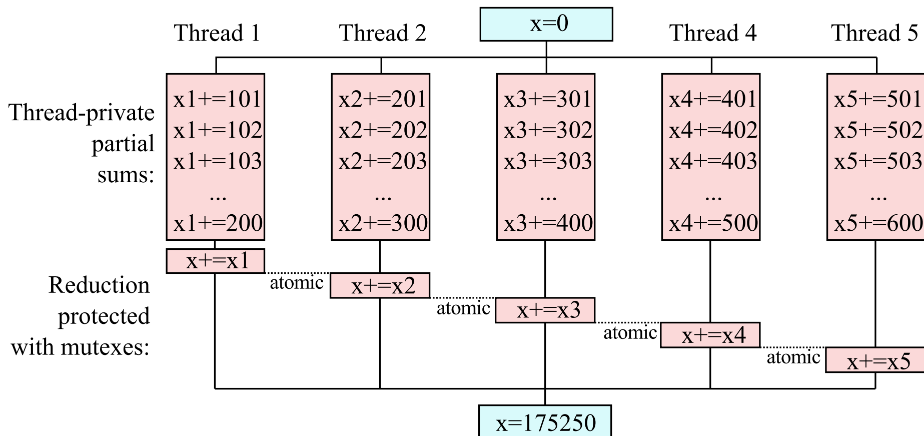
© Colfax International, 2013–2016

Data Races





Parallel Reduction



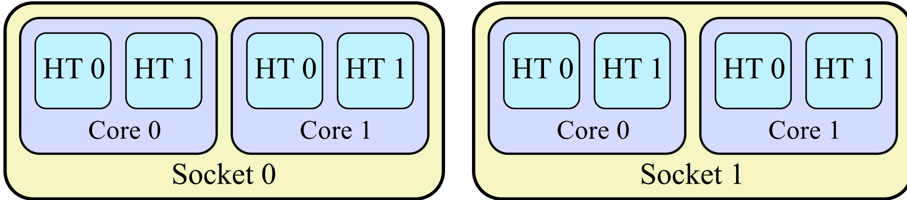
Key: using thread-private containers for partial sums; mutexes only after the loop

Multi-Threaded Version

```
1  void BinParticles_2(const InputDataType& inputData, BinsType& outputBins){
2  #pragma omp parallel
3    { BinsType threadPrivateBins; // Declare thread-private containers
4      for (int i = 0; i < nBinsX; i++)
5        for (int j = 0; j < nBinsY; j++)
6          threadPrivateBins[i][j] = 0;
7  #pragma omp for
8      for (int i = 0; i < inputData.numDataPoints; i++) {
9        // ...transforming from cylindrical to Cartesian coordinates
10       // ...calculating the bin numbers for these coordinates
11       threadPrivateBins[iX][iY]++; // Incrementing thread-private counter:
12     }
13     for(int i = 0; i < nBinsX; i++) // Reduction outside the parallel loop
14       for(int j = 0; j < nBinsY; j++)
15 #pragma omp atomic
16         outputBins[i][j] += threadPrivateBins[i][j];
17 }  }
```

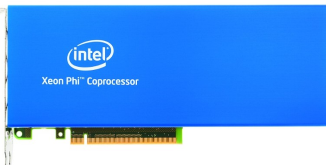

© Colfax International, 2013–2016

Thread Affinity



 © Colfax International, 2013–2016

Performance with Thread Parallelism



Performance, MP/s (higher is better)

3000
2500
2000
1500
1000
500
0

Intel Xeon Processor E5-2697 V3
Intel Xeon Phi Coprocessor 7120P

Baseline Code: 92, 5.4
Reduction with Atomics: 29, 45
Reduction with Private Variables: 890, 390
Thread Affinity: 2100, 760

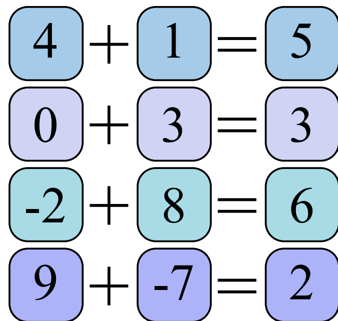
More work is needed to bring out coprocessor's performance!
Next step: vectorization.

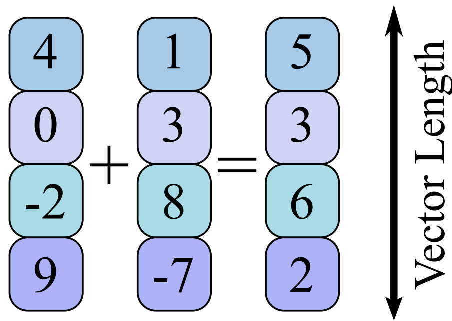
© Colfax International, 2013–2016

§5. SIMD Parallelism: Happy Vector Units

© Colfax International, 2013–2016

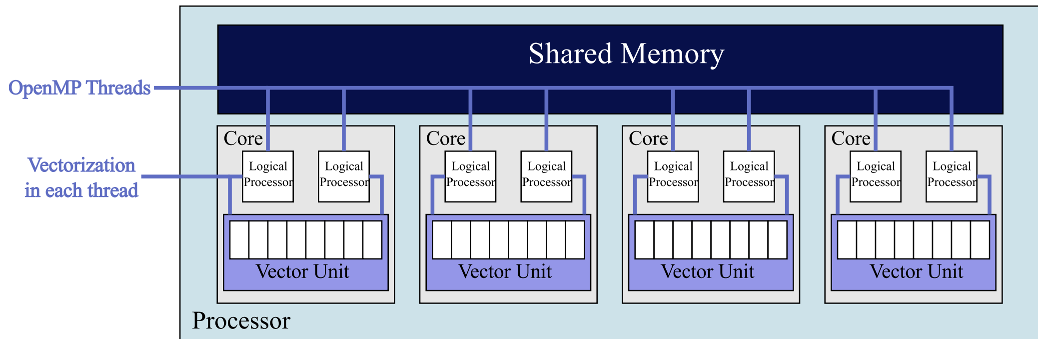
SIMD Instructions and Vectorization





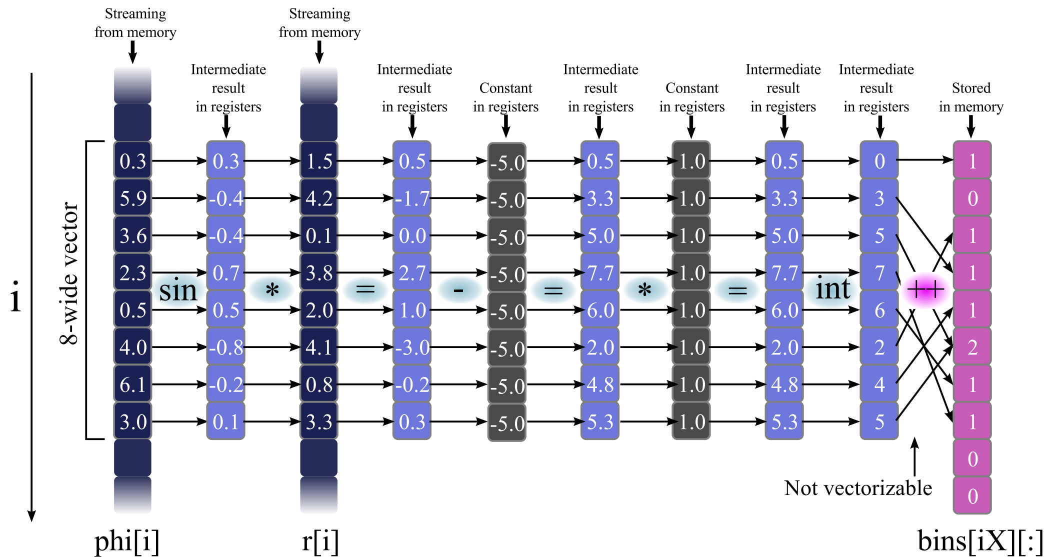
Intel archtecture implements SIMD processing with short vectors.

Vectors and Cores



Use threads to parallelize across cores, vector instructions for SIMD parallelism.

Vectorization Opportunity in Binning



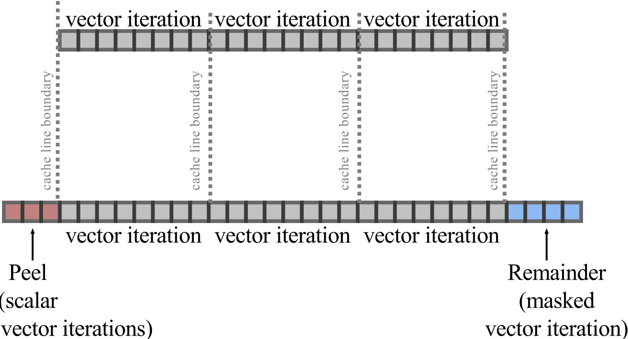
Strip-Mining to the Rescue

```
1  const int STRIP_WIDTH = 16;
2  for (int ii = 0; ii < inputData.numDataPoints; ii += STRIP_WIDTH) {
3    int iX[STRIP_WIDTH], iY[STRIP_WIDTH];
4    const FTYPE* r   = &(inputData.r[ii]);
5    const FTYPE* phi = &(inputData.phi[ii]);
6    for (int c = 0; c < STRIP_WIDTH; c++) { // Vector loop
7      const FTYPE x = r[c]*COS(phi[c]); // Transforming from cylindrical
8      const FTYPE y = r[c]*SIN(phi[c]); // to Cartesian coordinates
9      iX[c] = int((x - xMin)*binsPerUnitX); // Calculating the bin numbers
10     iY[c] = int((y - yMin)*binsPerUnitY); // for these coordinates
11   }
12   for (int c = 0; c < STRIP_WIDTH; c++) // Scalar loop
13     threadPrivateBins[iX[c]][iY[c]]++;
14 }
```

 © Colfax International, 2013–2016

Fine-Tuning Vectorization

```
for (i = 0; i < n; i++)  A[i] = ...
```



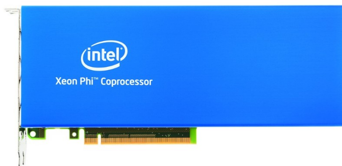
 © Colfax International, 2013–2016

Better Vectorization

```
// Allocating data on a 64-byte aligned memory heap address
rawData.r   = (FTYPE*) _mm_malloc(sizeof(FTYPE)*n, 64);
rawData.phi = (FTYPE*) _mm_malloc(sizeof(FTYPE)*n, 64);

// Later in the code:
for (int ii = 0; ii < inputData.numDataPoints; ii += STRIP_WIDTH) {
  int iX[STRIP_WIDTH] __attribute__((aligned(64))); // Aligned allocation
  int iY[STRIP_WIDTH] __attribute__((aligned(64))); // on the stack
  // ...

  // Compiler hint: we promise alignment, no need for peeling
#pragma vector aligned
  for (int c = 0; c < STRIP_WIDTH; c++) {
    // ...
  }
}
```


© Colfax International, 2013–2016

Performance with Vectorization



Performance, MP/s (higher is better)

	Intel Xeon Processor E5-2697 V3	Intel Xeon Phi Coprocessor 7120P
Baseline Code	92	5.4
Reduction with Atomics	29	45
Reduction with Private Variables	890	390
Thread Affinity	2100	760
Strip-Mining, Alignment	4500	7400

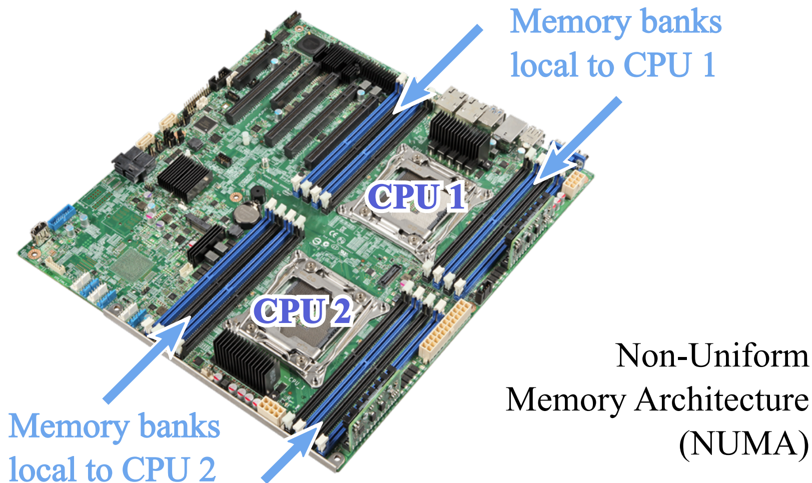
We can get more out of the CPU

Next step: memory traffic tuning.

§6. Memory Tuning: Happy Controllers

 © Colfax International, 2013–2016

NUMA Architecture in Intel Xeon Processors

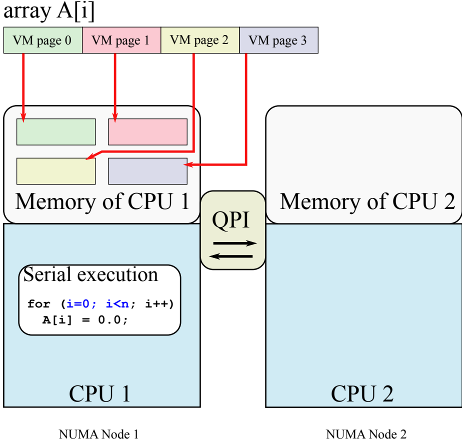


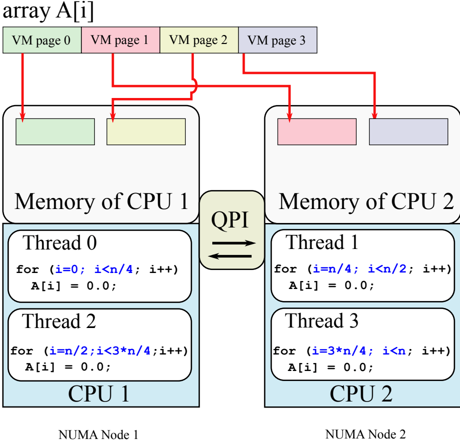
Non-Uniform
Memory Architecture
(NUMA)

Intel Xeon Phi processors of 2nd generation (KNL) have NUMA support as well.

© Colfax International, 2013–2016

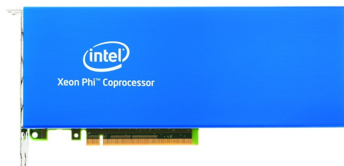
First-Touch Allocation Policy





© Colfax International, 2013–2016

Performance with Parallel First Touch



Performance, MP/s (higher is better)

	Intel Xeon Processor E5-2697 V3	Intel Xeon Phi Coprocessor 7120P
Baseline Code	92	5.4
Reduction with Atomics	29	45
Reduction with Private Variables	890	390
Thread Affinity	2100	760
Strip-Mining, Alignment	4500	7400
First Touch Optimization	7800	7400

Code is ready to scale to future Intel architectures.

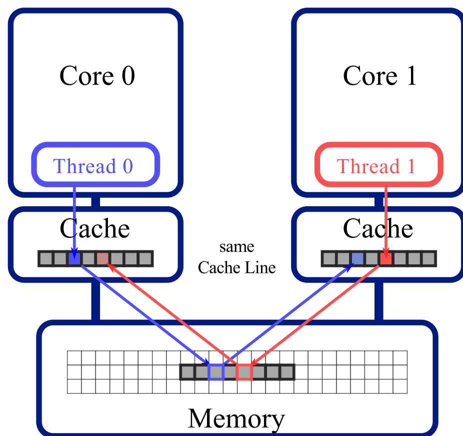
Is the Xeon Phi performance sufficient? See colfaxresearch.com/?p=11

© Colfax International, 2013–2016

§7. Extra Credit: False Sharing

 © Colfax International, 2013–2016

False Sharing



False sharing occurs if one thread when multiple threads access the same cache line, and one of the accesses is a write operation.

 © Colfax International, 2013–2016

Global Containers for Thread-Private Data

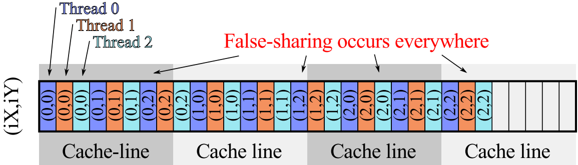
```
// Using a global container in
// threads-first layout
int nThrds=omp_get_max_threads();

// Instead of storing scalars in
// each bin, store an array with
// values for each thread =
int glBins[nBinsX][nBinsY][nThrds];

#pragma omp parallel
{
  int iThd = omp_get_thread_num();
  // ... later, memory access:
  glBins[iX[c]][iY[c]][iThrd]++;
}
```

```
// Using a global container in
// threads-last layout
int nThrds=omp_get_max_threads();

// Instead of storing scalars in
// each bin, store an array with
// values for each thread
int glBins[nThrds][nBinsX][nBinsY];

#pragma omp parallel
{
  int iThd = omp_get_thread_num();
  // ... later, memory access:
  glBins[iThrd][iX[c]][iY[c]]++;
}
```

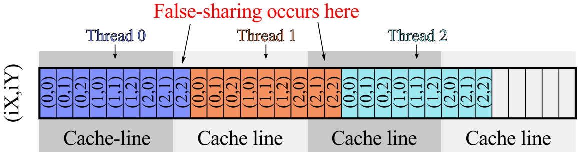
 © Colfax International, 2013–2016

False Sharing and Data Layout

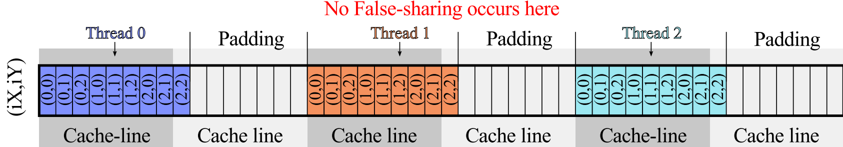
Case #1:
global container
threads-first layout



Case #2:
global container
threads-last layout



Case #3:
global container
threads-last layout
with padding



Bins

(0,0)	(1,0)	(2,0)
(0,1)	(1,1)	(2,1)
(0,2)	(1,2)	(2,2)

 © Colfax International, 2013–2016

Performance with False Sharing and Padding

Performance, MP/s (higher is better)

	Intel Xeon Processor E5-2697 V3	Intel Xeon Phi Coprocessor 7120P
Threads-first Layout	220	430
Threads-last Layout	5100	5500
Threads-last with Padding	8600	5900
Reference (Part 2)	7800	7400

False sharing can be eliminated with padding.
Xeon needs more padding than Xeon Phi.

© Colfax International, 2013–2016

§8. Resources

© Colfax International, 2013–2016

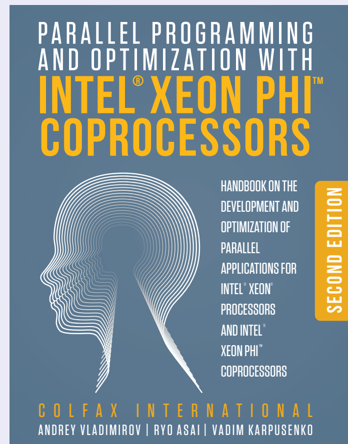
Supplementary Materials: Textbook

ISBN: 978-0-9885234-0-1 (2nd edition, 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

© Colfax International, 2015



http://xeonphi.com/book

Colfax Research

http://colfaxresearch.com/

Learn More

Comprehensive Hands-On Workshop (HOW) series begins April 18, 2016. Free remote access to training servers (space is limited!):

colfaxresearch.com/how-16-04

 © Colfax International, 2013–2016

Slides, Code, Video

You can download slides, code and watch the video recording of this webinar here (requires registration for a free Colfax Research account):

colfaxresearch.com/hot-16-03

Next webinar on March 23, 2016: "Finding the Low-Hanging Fruit for Optimization":

Register

© Colfax International, 2013–2016