MODERN CODE PRACTICES AND INTEL® ARCHITECTURE

Part 1 of 3

Colfax International — *colfaxresearch.com*

September 2016

© 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.

© Colfax International, 2013–2016

COURSE ROADMAP

- **Part 1**: Multi-threading strategies
  - Minimizing synchronization
  - Avoiding false sharing
  - Exposing parallelism
- **Part 2**: Vectorization tuning
  - Roles of compiler and developer
  - Tuning with directives
  - Data container optimization
  - Language extensions
- **Part 3**: Memory traffic control
  - Maximizing cache utilization
  - Optimizing memory bandwidth
  - Intel Xeon Phi processors: high-bandwidth memory

© Colfax International, 2013–2016

§2. INTEL ARCHITECTURE

COMPUTING PLATFORMS

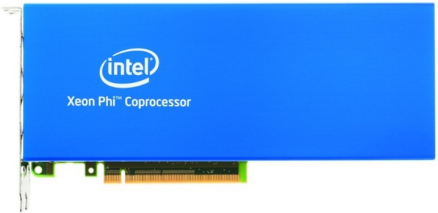
COMPUTING PLATFORMS

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

© Colfax International, 2013–2016

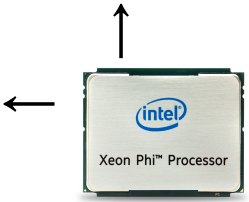
BOOTABLE INTEL XEON PHI PROCESSORS

- Bootable Host Processor
- RHEL/CentOS/SUSE/Win
- 64 cores × 4 HT, 1.3 GHz
- ≤ 384 GiB DDR4, > 90 GB/s
- 16 GiB HBM, > 400 GB/s
- PCIe bus for networking

dap.xeonphi.com

Servers:

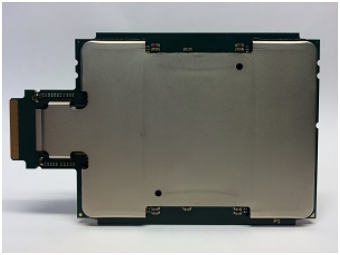
Workstations:

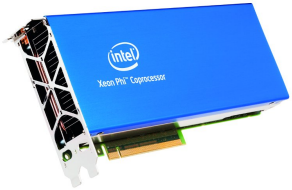


© Colfax International, 2013–2016

KNLF: KNL with Fabric

- Fabric integrated on CPU
  - Intel OmniPath Architecture
- Socket mount processor





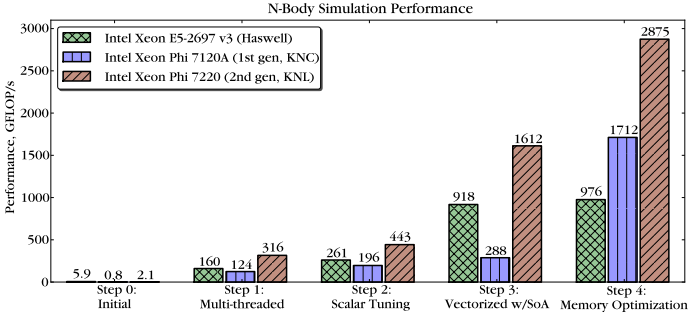
*KNC image

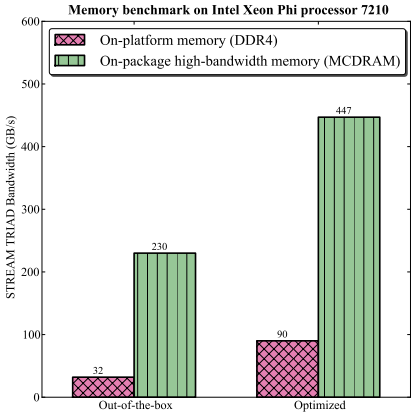
KNL Coprocessor

- PCIe add-in card
  - Requires host
- Multiple KNLs in a system

© Colfax International, 2013–2016

IT TAKES GOOD SOFTWARE TO UNLOCK THE PERFORMANCE!



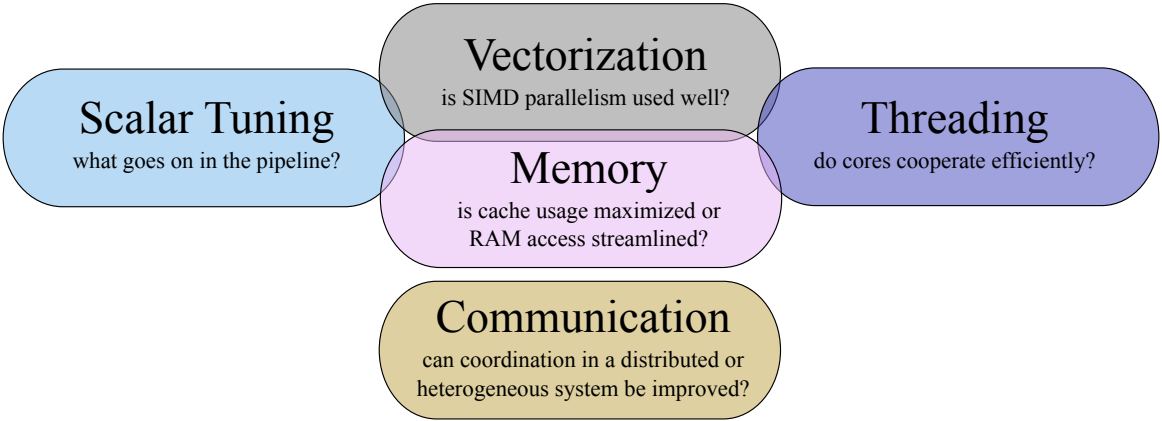


Details on N-body simulation in Chapter 23 of this book

© Colfax International, 2013–2016

PERFORMANCE OPTIMIZATION

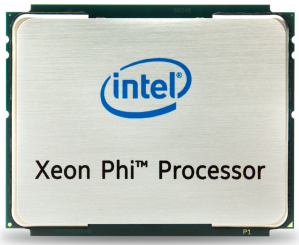
OPTIMIZATION AREAS

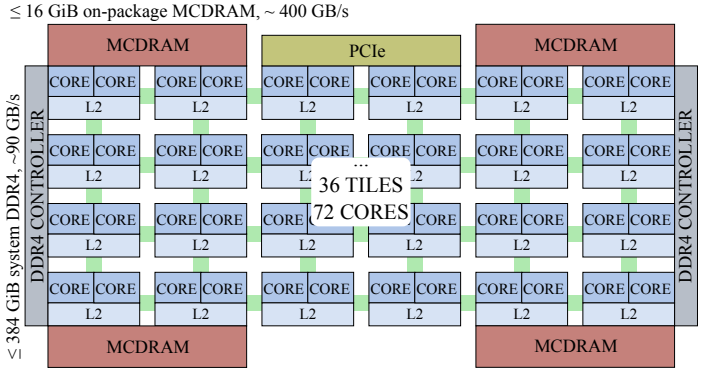


© Colfax International, 2013–2016

KNL DIE ORGANIZATION

- Mesh interconnect relaxes data locality requirement [somewhat]
- All-to-all, quadrant or sub-numa domain communication in mesh





© Colfax International, 2013–2016

§3. MULTI-THREADING: WHAT CAN GO WRONG?

THREADS AND OPENMP

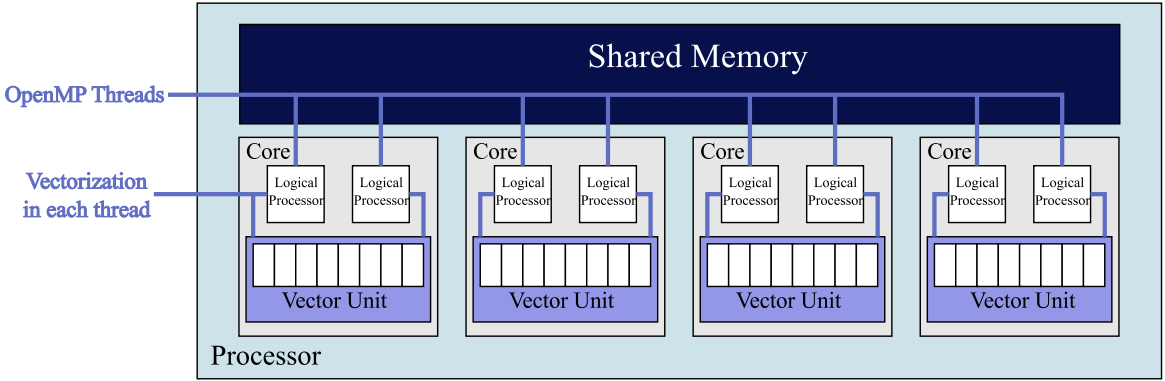
"HELLO WORLD" OPENMP PROGRAM

```
#include <omp.h>
#include <cstdio>

int main(){
  // This code is executed by only 1 thread
  const int nt=omp_get_max_threads();
  printf("OpenMP with %d threads\n", nt);

#pragma omp parallel
  {
    // This code is executed in parallel
    // by multiple threads
    printf("Hello World from thread %d\n",
                        omp_get_thread_num());
  }
}
```

- OpenMP = “Open Multi-Processing” = computing-oriented framework for shared-memory programming
- Threads – streams of instructions that share memory address space
- Distribute threads across CPU cores for parallel speedup

© Colfax International, 2013–2016

CO-EXISTENCE WITH VECTORS



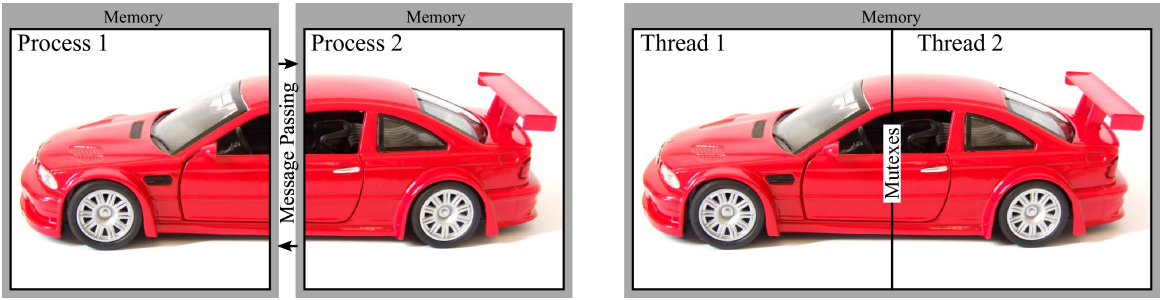
Utilize cores: run multiple threads/processes (MIMD)

Utilize vectors: each thread (process) issues vector instructions (SIMD)

© Colfax International, 2013–2016

THREADS VERSUS PROCESSES

Option 1: Partitioning data set between threads/processes

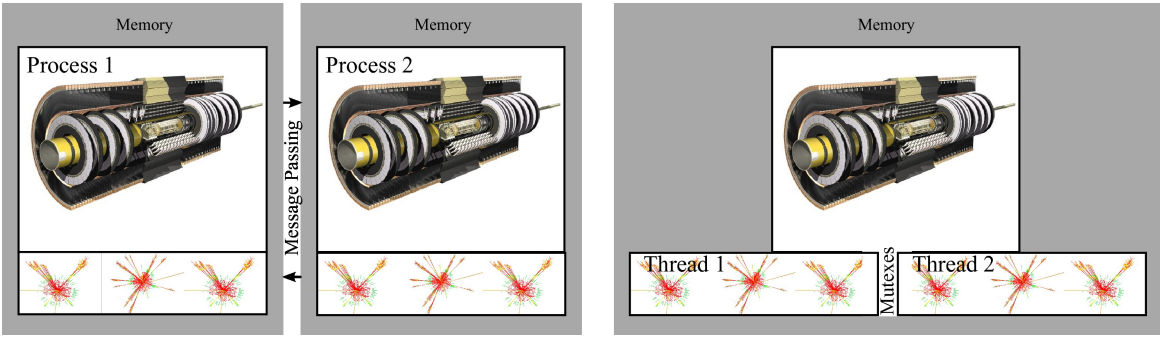


Examples: computational fluid dynamics (CFD), image processing.

© Colfax International, 2013–2016

THREADS VERSUS PROCESSES

Option 2: Sharing data set between threads/processes



Examples: particle transport simulation, machine learning (inference).

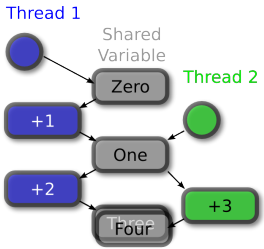
© Colfax International, 2013–2016

ISSUE 1: TOO MUCH SYNCHRONIZATION

Race conditions and unpredictable program behavior

```
#include <omp.h>
#include <cstdio>
int main() {
  const int n = 1000;
  int total = 0;
#pragma omp parallel for
  for (int i = 0; i < n; i++) {
    total = total + i; // Race condition
  }
  printf("total=%d (must be %d)\n", total,
                                   ((n-1)*n)/2);
}
```

```
vega@lyra% icpc -o app omp-race.cc -qopenmp
vega@lyra% ./app
total=208112 (must be 499500)
```



Race Condition!

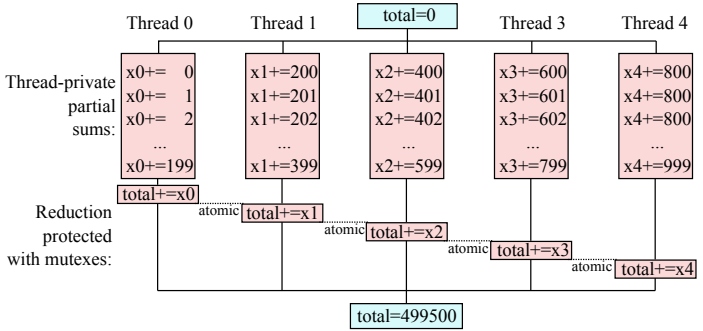
- Occurs when 2 or more threads access the same memory address, and at least one of these accesses is for writing

© Colfax International, 2013–2016

AVOIDING RACES WITH THREAD-PRIVATE STORAGE

Correct and efficient code:

```
1  int total = 0;
2  #pragma omp parallel
3  {
4    int total_thr = 0;
5  #pragma omp for
6    for (int i=0; i<n; i++)
7      total_thr += i;
8
9  #pragma omp atomic
10   total += total_thr;
11
12 }
```



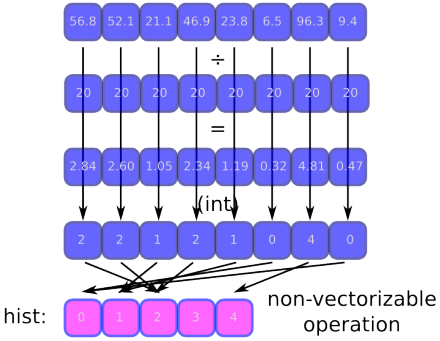
© Colfax International, 2013–2016

EXAMPLE: BINNING PROBLEM

Computing a histogram ($m << n$):

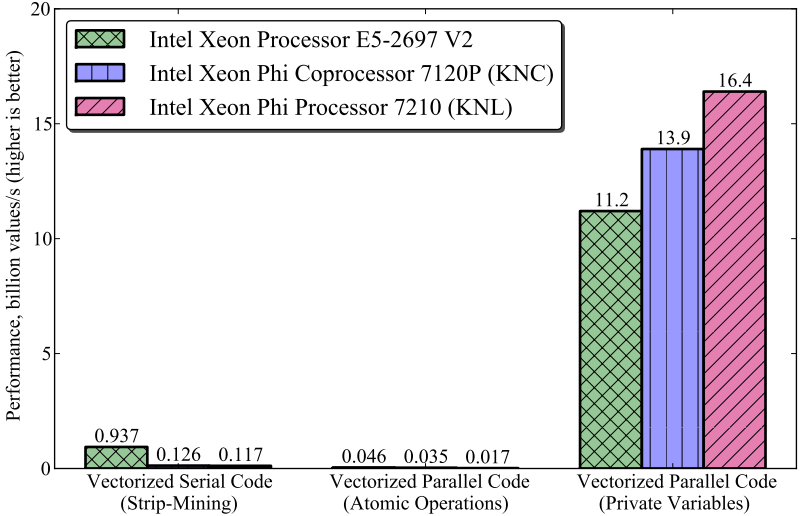
```
1  void Histogram(
2    // Ages, values from 0.0f to 100.0f:
3    const float* age,
4    // Size of array age, n=100000000:
5    const int n,
6    // Output: counts in groups:
7    int* const hist,
8    // Size of array hist, m=5:
9    const int m,
10   const float group_width) {
11     for (int i = 0; i < n; i++) {
12       const int j = int(age[i]/group_width);
13       hist[j]++;
14     }
15 }
```

- Vector dependence in `hist[j]++`
- Strip-mine or use conflict detection



© Colfax International, 2013–2016

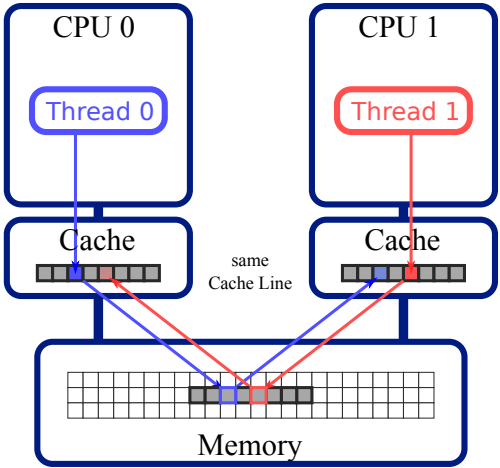
USING REDUCTION INSTEAD OF SYNCHRONIZATION



© Colfax International, 2013–2016

ISSUE 2: FALSE SHARING

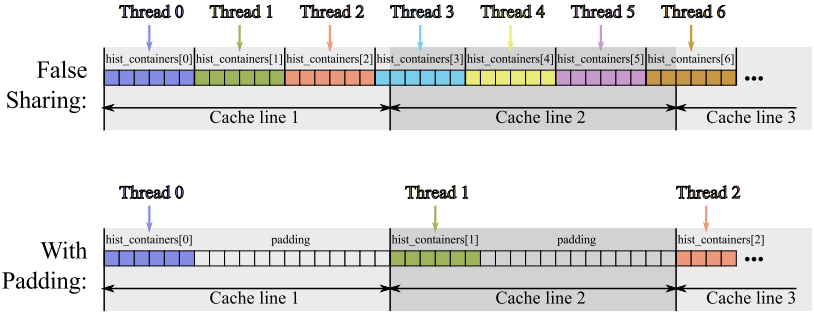
FALSE SHARING. DATA PADDING AND PRIVATE VARIABLES



- Occurs when 2 or more threads acess the same cache line, and at least one of the accesses is for writing
- Caused by *coherent caches*
- Cache line is 64-byte wide (in modern Intel architectures)

© Colfax International, 2013–2016

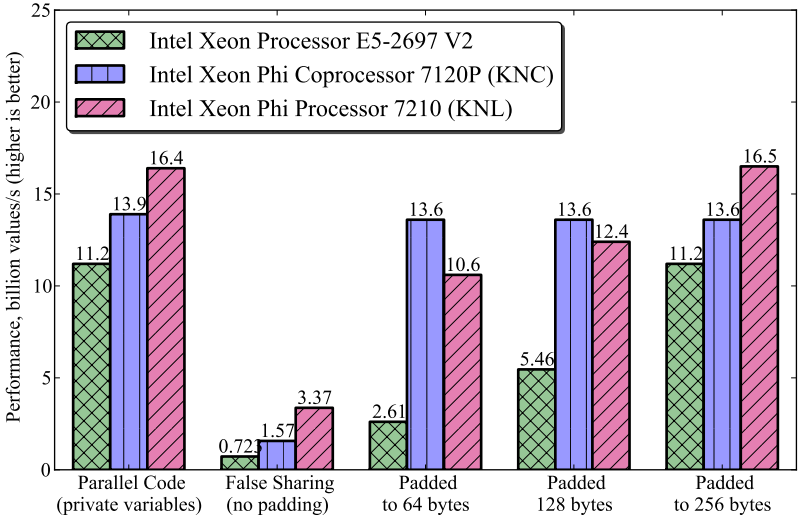
PADDING TO AVOID FALSE SHARING




```
1 // Padding to avoid sharing a cache line between threads
2 const int paddingBytes = 64;
3 const int paddingElements = paddingBytes / sizeof(int);
4 const int mPadded = m + (paddingElements-m%paddingElements);
5 int hist_containers[nThreads][mPadded]; // New container
```


© Colfax International, 2013–2016

PADDING TO AVOID FALSE SHARING

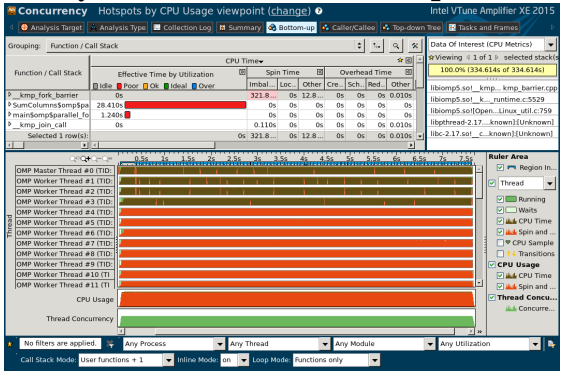


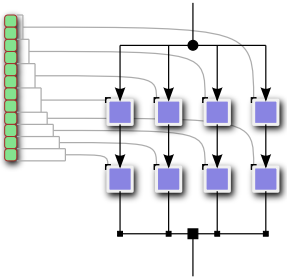
© Colfax International, 2013–2016

ISSUE 3: INSUFFICIENT PARALLELISM

INSUFFICIENT PARALLELISM

Analysis in Intel VTune Amplifier XE





- Occurs when there are not enough iterations or parallel work-items exposed to the parallel loop in OpenMP.

© Colfax International, 2013–2016

EXAMPLE: DEALING WITH INSUFFICIENT PARALLELISM

$$S_i = \sum_{j=0}^{n} M_{ij},\ i = 0 \ldots m. \tag{1}$$

- m=4 is small, smaller than the number of threads in the system
- n $\approx 10^8$ is large enough so that matrix does not fit into cache

```
1 void sum_unoptimized(const int m, const int n, long* M, long* s){
2 #pragma omp parallel for
3   for (int i=0; i<m; i++) { // m=4
4     long total=0;
5 #pragma vector aligned
6     for (int j=0; j<n; j++) // n=100000000
7       total+=M[i*n+j];
8   s[i]=total; }}
```

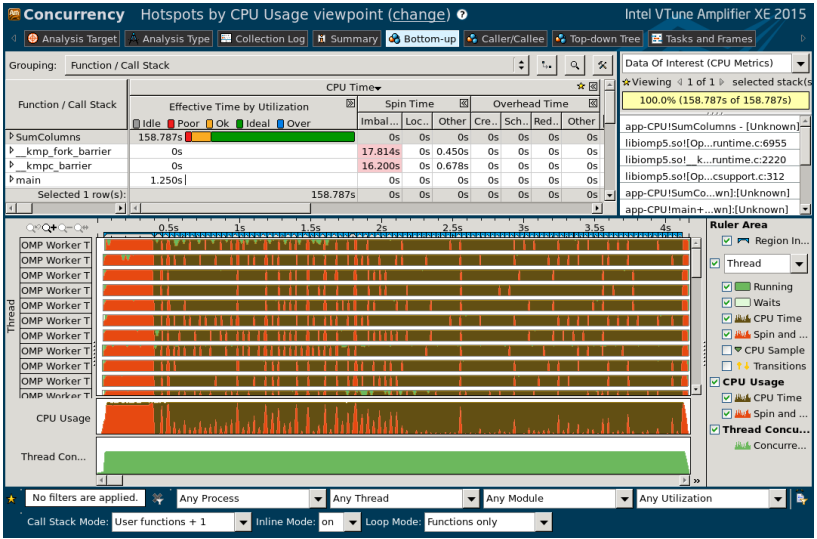

© Colfax International, 2013–2016

EXPOSING PARALLELISM: STRIP-MINING AND LOOP COLLAPSE

```
1  void sum_stripmine(const int m, const int n, long* M, long* s){
2    const int STRIP=1024;
3    assert(n%STRIP==0);
4    s[0:m]=0;
5  #pragma omp parallel
6    {
7      long total[m];   total[0:m]=0;
8  #pragma omp for collapse(2) schedule(guided)
9      for (int i=0; i<m; i++)
10       for (int jj=0; jj<n; jj+=STRIP)
11 #pragma vector aligned
12         for (int j=jj; j<jj+STRIP; j++)
13           total[i]+=M[i*n+j];
14     for (int i=0; i<m; i++)          // Reduction
15 #pragma omp atomic
16       s[i]+=total[i];
17 }   }
```


© Colfax International, 2013–2016

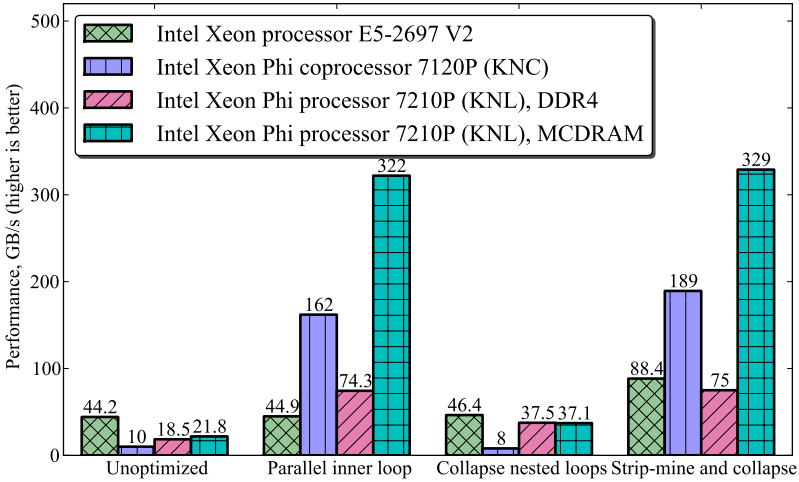
EXPOSING PARALLELISM: STRIP-MINING AND LOOP COLLAPSE



ISSUE 3: INSUFFICIENT PARALLELISM

© Colfax International, 2013–2016

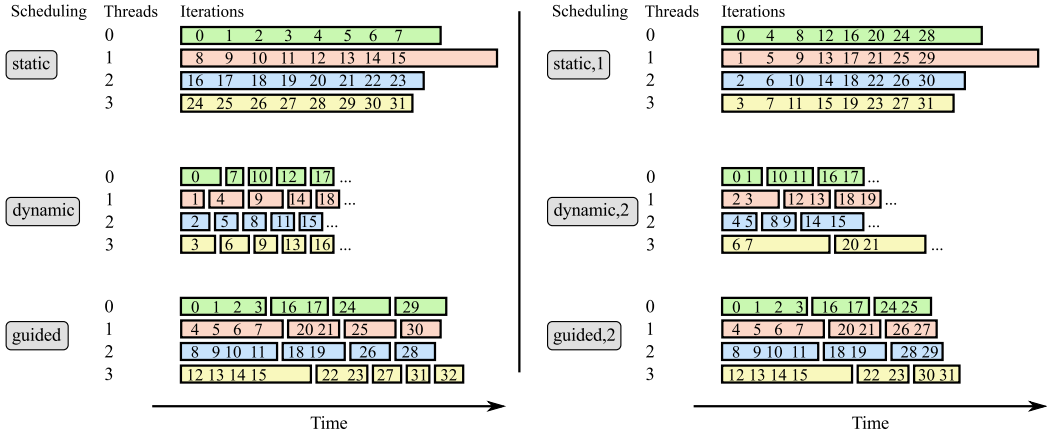
DEALING WITH INSUFFICIENT PARALLELISM



© Colfax International, 2013–2016

OTHER OPENMP CONTROLS

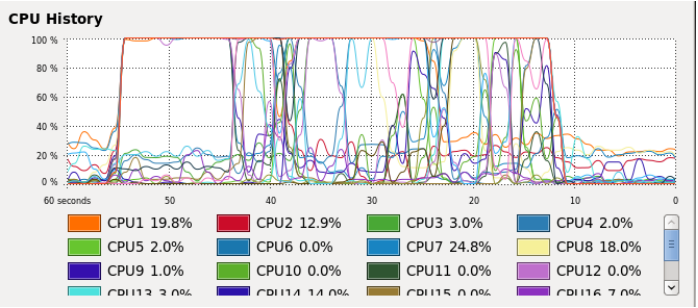
LOOP SCHEDULING MODES IN OPENMP



© Colfax International, 2013–2016

WHAT IS THREAD AFFINITY

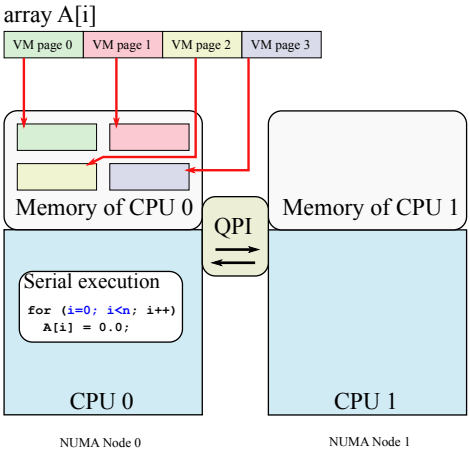
- OpenMP threads may migrate between cores
- Forbid migration — improve locality — increase performance
- Affinity patterns “scatter” and “compact” may improve cache sharing, relieve thread contention



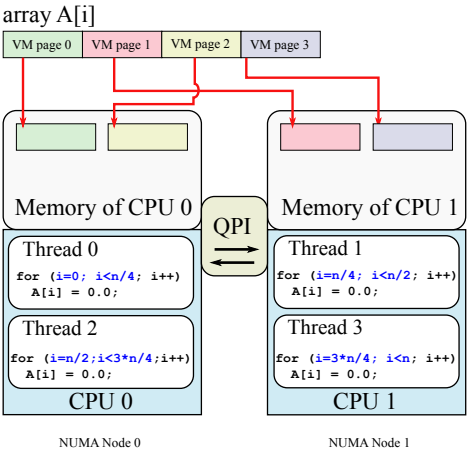
© Colfax International, 2013–2016

FIRST-TOUCH ALLOCATION POLICY

Poor First-Touch Allocation

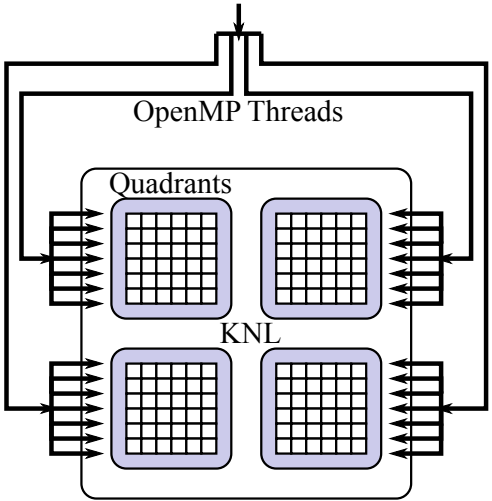


Good First-Touch Allocation



© Colfax International, 2013–2016

NESTED PARALLELISM WITH OPENMP




```
1 #pragma omp parallel
2 {
3 #pragma omp parallel
4   {
5     // ...
6   }
7 }
```

- Tune granularity of parallelism
- Improve resource sharing in NUMA systems

© Colfax International, 2013–2016

LEARN MORE

HOW SERIES

Interested? Sign-up at:

colfaxresearch.com/how-series

© Colfax International, 2013–2016

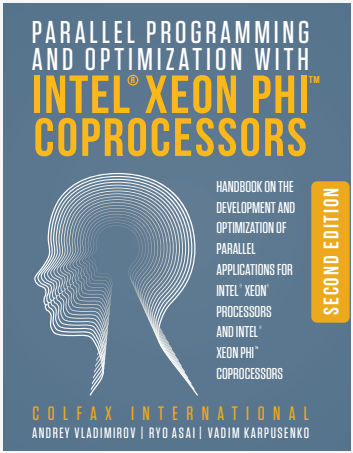
TEXTBOOK

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

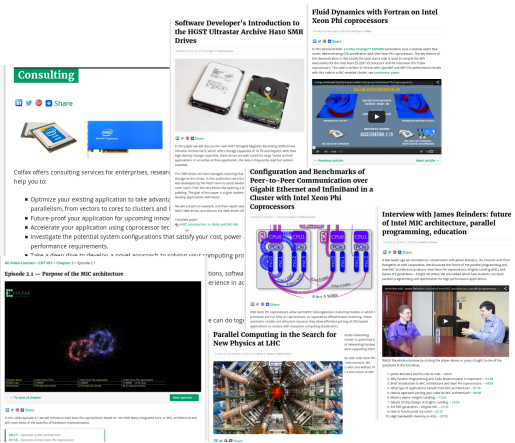
© Colfax International, 2015



http://xeonphi.com/book

© Colfax International, 2013–2016

COLFAX RESEARCH



http://colfaxresearch.com/

© Colfax International, 2013–2016

DEVELOPER ACCESS PROGRAM (DAP)

Can't wait to get your hands on Knights Landing?

Find out more at dap.xeonphi.com
or contact us at dap@colfax-intl.com

© Colfax International, 2013–2016