



PROGRAMMING AND OPTIMIZATION FOR INTEL[®] ARCHITECTURE

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

Session 8

Colfax International — colfaxresearch.com

May 2017

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.

- ▶ **Module I. Programming**
 - 01. Intel Architecture and Modern Code – May 15
 - 02. Xeon Phi, Coprocessors, Omni-Path – May 16
- ▶ **Module II. Expressing Parallelism**
 - 03. Automatic vectorization – May 17
 - 04. Multi-threading with OpenMP – May 18
 - 05. Distributed Computing, MPI – May 19
- ▶ **Module III. Optimization**
 - 06. Optimization Overview: N-body – May 22
 - 07. Scalar tuning, Vectorization – May 23
 - 08. Common Multi-threading Problems – May 24
 - 09. Multi-threading, Memory Aspect – May 25
 - 10. Access to Caches and Memory – May 26

| May 2017 | | | | | | |
|----------|----|----|----|----|----|----|
| 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 | | | |
| | | | | | | |

Course page:

colfaxresearch.com/how-17-05

- ▶ Slides
- ▶ Code
- ▶ Video
- ▶ Chat

More workshops:

colfaxresearch.com/training



GET YOUR QUESTIONS ANSWERED: CHAT



colfaxresearch.com/how-17-05

GET YOUR QUESTIONS ANSWERED: FORUMS



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/discussion

- ▶ 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. REFRESH



PERFORMANCE OPTIMIZATION

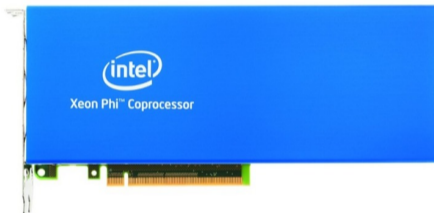
Intel Xeon Processor



Current: Broadwell
Upcoming: Skylake

Multi-Core Architecture

Intel Xeon Phi Coprocessor, 1st generation



Knights Corner (KNC)

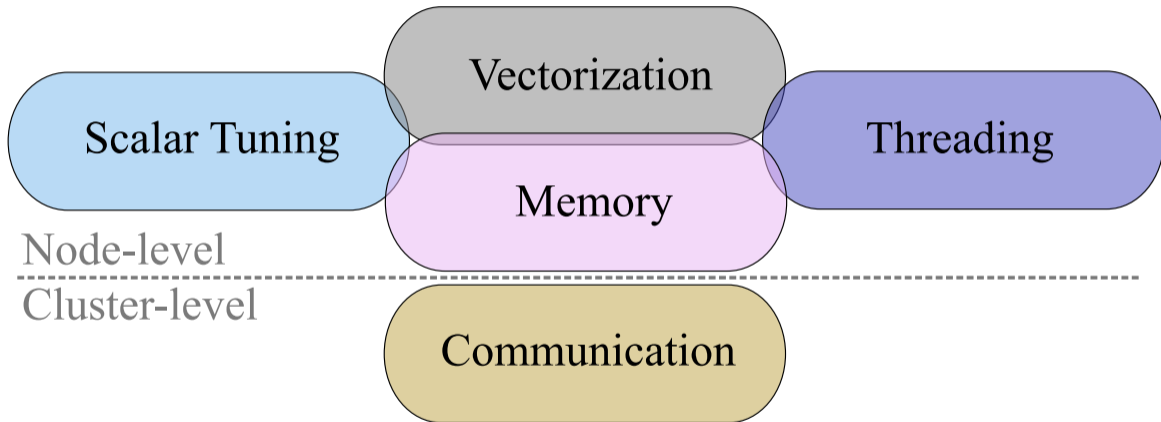
Intel Xeon Phi Processor, 2nd generation*



* socket and coprocessor versions

Knights Landing (KNL)

Intel Many Integrated Core (MIC) Architecture



"HELLO WORLD" OPENMP PROGRAM

```
1  #include <omp.h>
2  #include <stdio.h>
3
4  int main(){
5      // This code is executed by 1 thread
6      const int nt=omp_get_max_threads();
7      printf("OpenMP with %d threads\n", nt);
8
9      #pragma omp parallel
10     { // This code is executed in parallel
11       // by multiple threads
12       printf("Hello World from thread %d\n",
13             omp_get_thread_num());
14     }
15 }
```

- ▶ 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



§3. MULTI-THREADING: COMMON ISSUES



TOO MUCH SYNCHRONIZATION

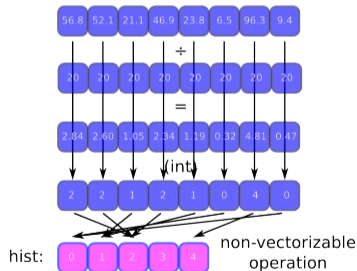
EXAMPLE: BINNING PROBLEM

```

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 grpWidth) {
11    for (int i = 0; i < n; i++) {
12        const int j = int(age[i]/grpWidth);
13        hist[j]++;
14    }
15 }

```

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



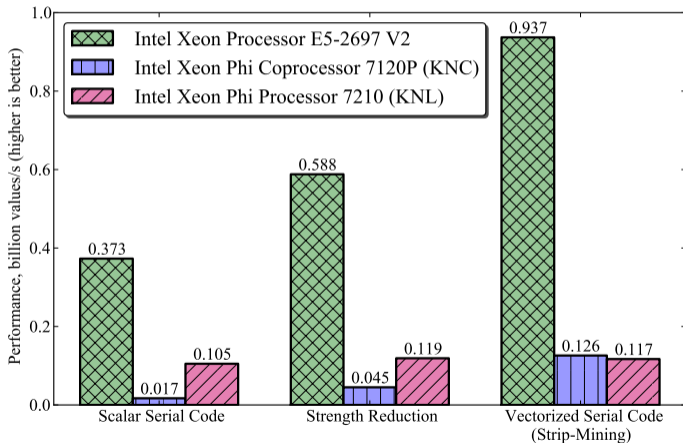
THE SAME CALCULATION, STRIP-MINED, VECTORIZED

```
1  const float recGrpWidth = 1.0f/grpWidth; // precompute the reciprocal
2
3  for (int ii = 0; ii < n; ii += 16) { // strip-mining
4
5      int index[16]; // a block of indices
6      for (int i = ii; i < ii + 16; i++) // vectorizable
7          index[i-ii] = (int) ( age[i] * recGrpWidth ); // unit-stride access
8
9      for (int c = 0; c < 16; c++) // not vectorizable
10         hist[index[c]]++; // indirect access
11 }
```

STRIP-MINING FOR VECTORIZATION

Vectorization improves performance.

More work is needed to take advantage of multiple cores.



HISTOGRAM CALCULATION EXAMPLE: ADDING THREAD PARALLELISM

Incorrect solution: unprotected data races

```
1 #pragma omp parallel for schedule(guided)
2 for (int ii = 0; ii < n; ii += vecLen) {
3     int index[vecLen] __attribute__((aligned(64)));
4     #pragma vector aligned
5     for (int i = ii; i < ii + vecLen; i++)
6         index[i-ii] = (int) ( age[i] * invGroupWidth );
7     for (int c = 0; c < vecLen; c++)
8         // Multiple threads will write into a single shared container
9         // These data races lead to incorrect results!
10        hist[index[c]]++;
11 }
```

HISTOGRAM CALCULATION EXAMPLE: ADDING THREAD PARALLELISM

Correct, but inefficient solution:

```
1  #pragma omp parallel for schedule(guided)
2  for (int ii = 0; ii < n; ii += vecLen) {
3      int index[vecLen] __attribute__((aligned(64)));
4      #pragma vector aligned
5          for (int i = ii; i < ii + vecLen; i++)
6              index[i-ii] = (int) ( age[i] * invGroupWidth );
7          for (int c = 0; c < vecLen; c++)
8              // Protect the ++ operation with the atomic mutex (inefficient!)
9          #pragma omp critical
10             { hist[index[c]]++; }
11 }
```

HISTOGRAM CALCULATION EXAMPLE: ADDING THREAD PARALLELISM

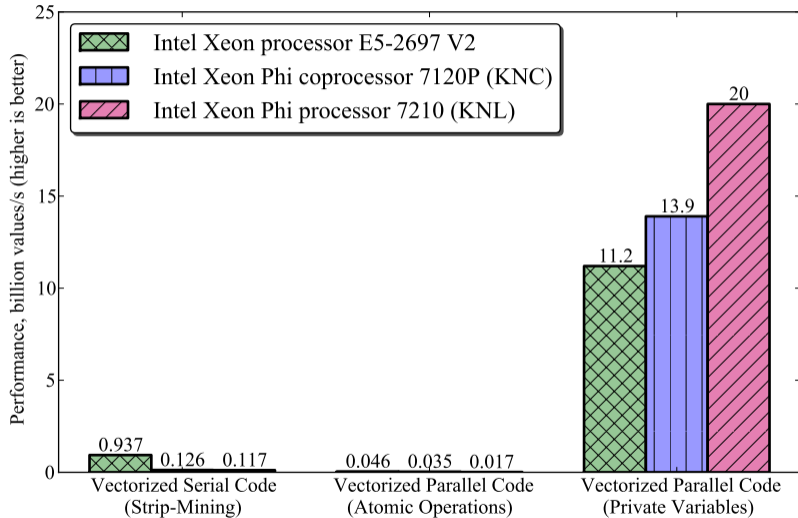
Correct, but inefficient solution:

```
1  #pragma omp parallel for schedule(guided)
2  for (int ii = 0; ii < n; ii += vecLen) {
3      int index[vecLen] __attribute__((aligned(64)));
4      #pragma vector aligned
5          for (int i = ii; i < ii + vecLen; i++)
6              index[i-ii] = (int) ( age[i] * invGroupWidth );
7          for (int c = 0; c < vecLen; c++)
8              // Protect the ++ operation with the atomic mutex (inefficient!)
9          #pragma omp atomic
10             hist[index[c]]++;
11 }
```

CORRECT AND EFFICIENT SOLUTION WITH REDUCTION

```
1  #pragma omp parallel
2  {
3      int hist_priv[m]; // Better idea: thread-private storage
4      hist_priv[:] = 0;
5      int index[vecLen] __attribute__((aligned(64)));
6      #pragma omp for schedule(guided)
7      for (int ii = 0; ii < n; ii += vecLen) {
8          #pragma vector aligned
9          for (int i = ii; i < ii + vecLen; i++)
10             index[i-ii] = (int) ( age[i] * invGroupWidth );
11         for (int c = 0; c < vecLen; c++)
12             hist_priv[index[c]]++;
13     }
14     for (int c = 0; c < m; c++) {
15         #pragma omp atomic
16         hist[c] += hist_priv[c];
17     } }
```

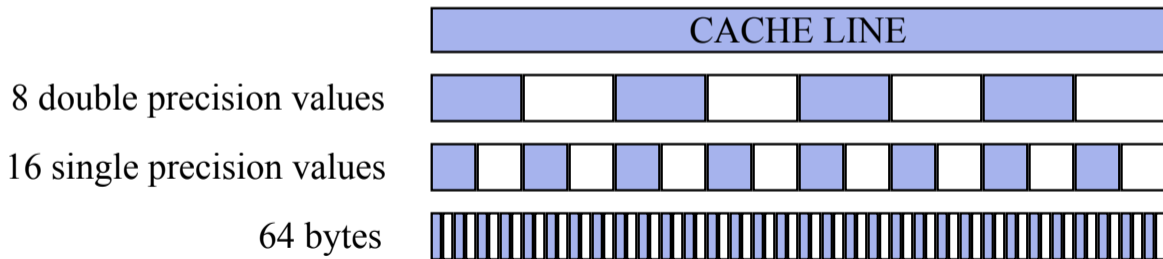
USING REDUCTION INSTEAD OF SYNCHRONIZATION



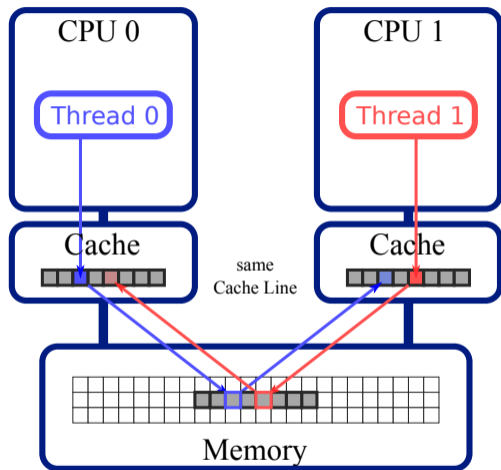


FALSE SHARING

- ▶ Minimal block of data transferred between memory and cache
- ▶ 64 bytes long in Intel Architecture
- ▶ Aligned on 64-byte boundaries in memory



FALSE SHARING. DATA PADDING AND PRIVATE VARIABLES



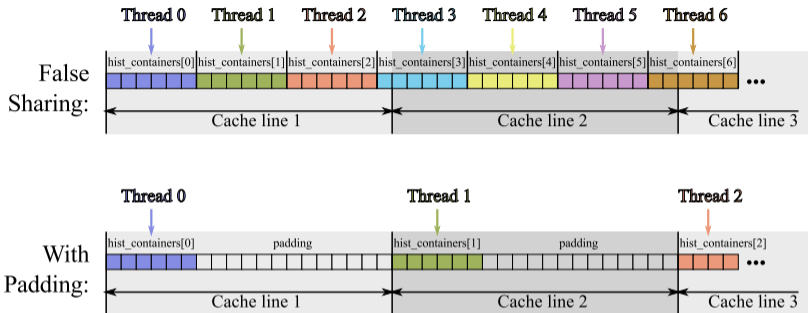
- ▶ Occurs when 2 or more threads access the same cache line, and at least one of the accesses is for writing
- ▶ Caused by *coherent caches*
- ▶ Cache line is 64-byte wide (in modern Intel architectures)

FALSE SHARING. DATA PADDING AND PRIVATE VARIABLES

```
1  const int m = 5;
2  int hist_thr[nThreads][m];
3  #pragma omp parallel for
4  for (int ii = 0; ii < n; ii += vecLen) {
5      // ...
6      // False sharing occurs here
7      for (int c = 0; c < vecLen; c++)
8          hist_thr[iThread][index[c]]++;
9  }
10 // Reducing results from all threads to the common histogram hist
11 for (int iThread = 0; iThread < nThreads; iThread++)
12     hist[0:m] += hist_thr[iThread][0:m];
```

- ▶ The value of $m=5$ is small
- ▶ Array elements `hist_thr[0][:]` are within $m*\text{sizeof}(\text{int})=20$ bytes of array elements `hist_thr[1][:]`

PADDING TO AVOID FALSE SHARING

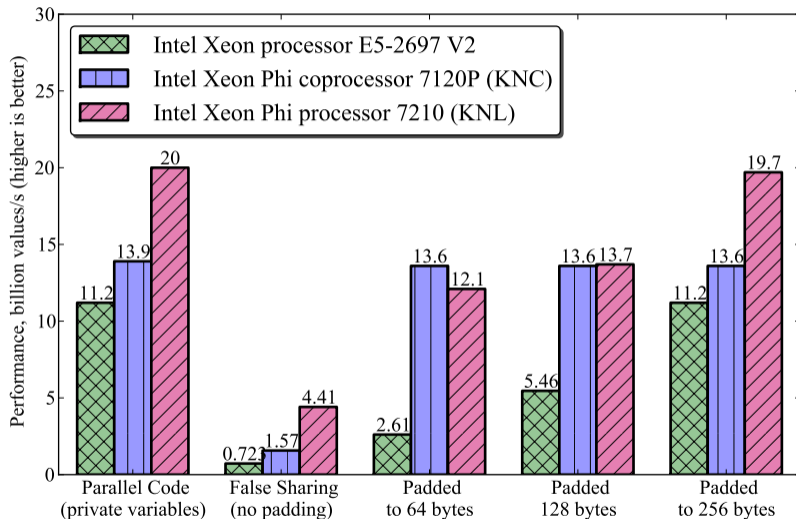


```

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

```

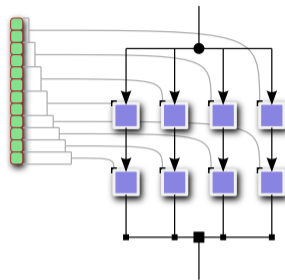
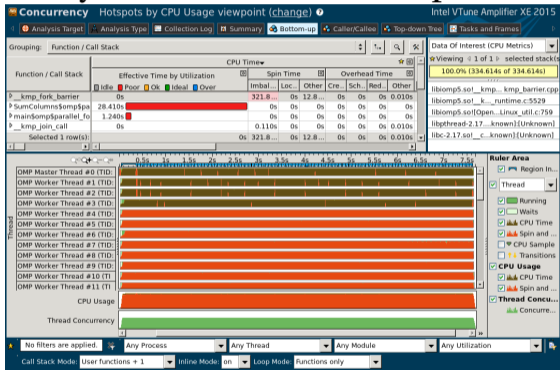
PADDING TO AVOID FALSE SHARING





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.

EXAMPLE: DEALING WITH INSUFFICIENT PARALLELISM

$$S_i = \sum_{j=0}^n M_{ij}, i = 0 \dots m. \quad (1)$$

- ▶ $m=4$ 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; }
```

DOES NOT WORK: PARALLELIZING INNER LOOP

Inner loop has more iterations, parallelize there?

```
1 void SumParallelInnerLoop(const int m, const int n, long* M, long* s){
2     for (int i = 0; i < m; i++) { // m=4
3         long total = 0;
4         #pragma omp parallel for reduction(+: total)
5         for (int j = 0; j < n; j++) { // n=100000000
6             total += M[i*n + j];
7         }
8         s[i] = total;
9     }
10 }
```

Does not work well: code must spawn and stop threads many times;
OpenMP does not see the entire parallel region.

LOOP COLLAPSE: PRINCIPLE

Idea: combine iterations spaces of the inner loop and the outer loop.

```
1 #pragma omp parallel for collapse(2)
2   for (int i = 0; i < m; i++)
3     for (int j = 0; j < n; j++) {
4         // ...
5         // ...
6     }
```

```
1 #pragma omp parallel for
2   for (int c = 0; c < m*n; c++) {
3       i = c / n;
4       j = c % n;
5       // ...
6   }
```

DOES NOT WORK, BUT CORRECT DIRECTION: LOOP COLLAPSE

Loop collapse applied to the wide short matrix example:

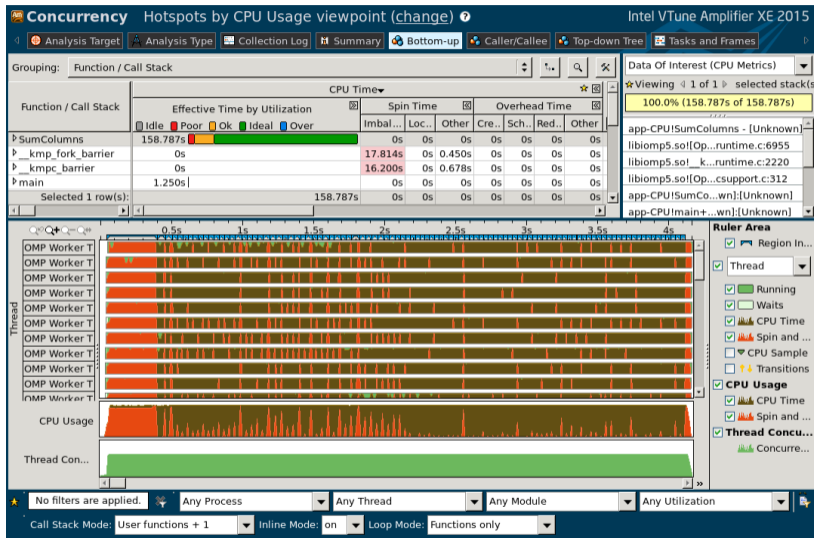
```
1 void SumCollapse(const int m, const int n, long* M, long* s){
2     s[:] = 0;
3     #pragma omp parallel
4     { // Each thread will have a private container
5         long total[m]; total[:] = 0;
6         #pragma omp for collapse(2)
7         for (int i = 0; i < m; i++) // m=4
8             for (int j = 0; j < n; j++) // n=100000000
9                 total[i] += M[i*n + j];
10        for (int i = 0; i < m; i++)
11        #pragma omp atomic
12            s[i] = total[i];
13    } }
```

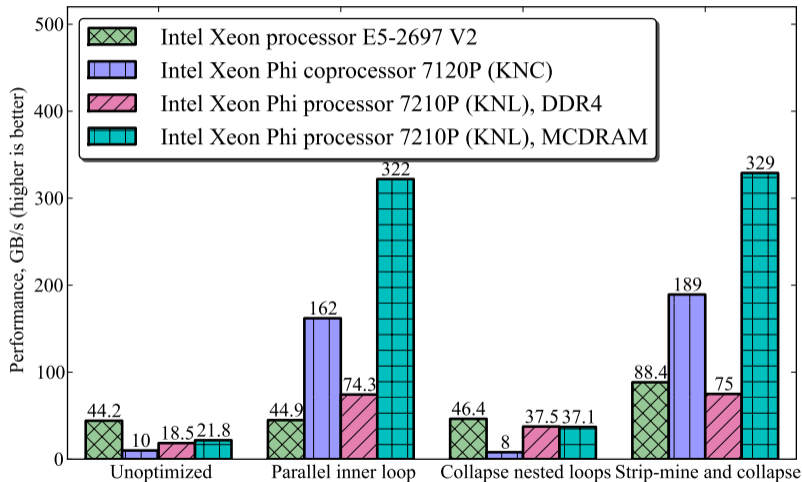
Does not work: automatic vectorization fails.

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    } }
```

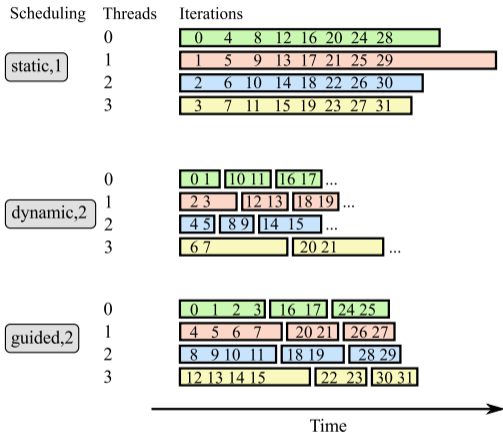
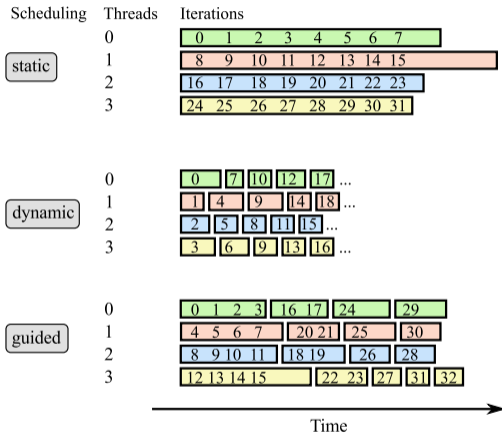
EXPOSING PARALLELISM: STRIP-MINING AND LOOP COLLAPSE





LOAD IMBALANCE

LOOP SCHEDULING MODES IN OPENMP



CONTROL OF SCHEDULING MODES

To set scheduling for a particular loop in code (example):

```
1 #pragma omp parallel for schedule(dynamic,4)
2 // ...
```

To set scheduling for the entire application at run time (example):

```
1 #pragma omp parallel for schedule(runtime)
2 // ...
```

```
vega@lyra% export OMP_SCHEDULE=dynamic,4
vega@lyra% ./run-my-app
```

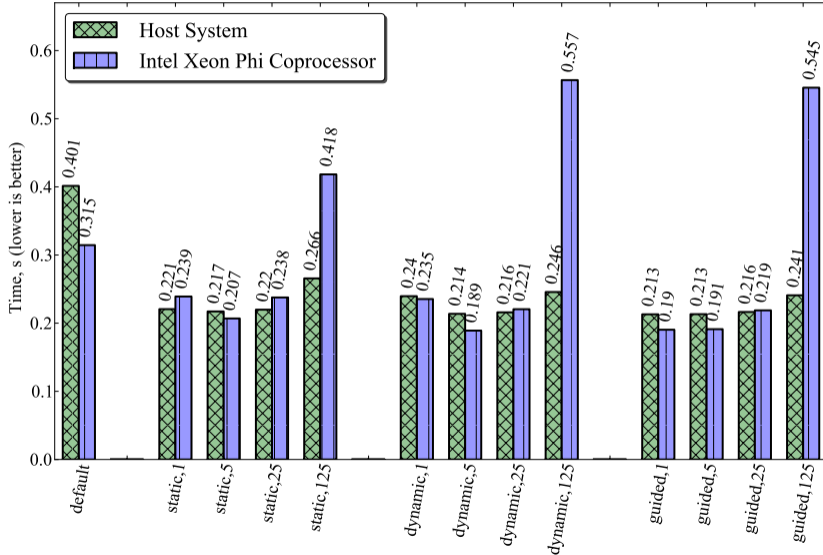
ITERATIVE JACOBI SOLVER

```

1 int IterativeSolver(int n, double* M, double* b, double* x, double minAccuracy){
2     double accuracy; int iters=0; double bTrial[n] __attribute__((aligned(64)));
3     x[0:n] = 0.0; // Initial guess
4     do { iters++; // The Jacobi method - iterate until convergence
5         for (int i = 0; i < n; i++) {
6             double c = 0.0;
7             #pragma vector aligned
8                 for (int j = 0; j < n; j++) c += M[i*n+j]*x[j]; // Iterate
9                 x[i] = x[i] + (b[i] - c)/M[i*n+i]; }
10            bTrial[:] = 0.0; // Verification
11            for (int i = 0; i < n; i++)
12                #pragma vector aligned
13                    for (int j = 0; j < n; j++) bTrial[i] += M[i*n+j]*x[j];
14            accuracy = RelativeNormOfDifference(n, b, bTrial); // Check convergence
15        } while (accuracy > minAccuracy); // Must achieve the requested accuracy
16    return iters; }

```


PERFORMANCE OF ITERATIVE JACOBI SOLVER





§4. REVIEW AND WHAT'S NEXT

SUMMARY

This session:

1. Synchronization is necessary to resolve data races
2. Mutexes must be moved out of innermost loops
3. False sharing can be resolved with padding
4. Loop collapse can help to expose parallelism
5. Strip-mining to make vectorization co-exist with threading
6. Trade-off between load balance and low scheduling overhead

Next session: optimization of thread affinity, NUMA locality, nested parallelism and loop scheduling.

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 of 3: 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 H700 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 reconstruction to a streaming code

▶

Consulting

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 H700 SMR Drives

Colfax offers consulting services for enterprises, research help you to:

Fluid Dynamics with Fortran on Intel Xeon Phi coprocessors

Colfax offers consulting services for enterprises, research help you to:

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:

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

Colfax offers consulting services for enterprises, research help you to:

http://colfaxresearch.com/

colfaxresearch.com/how-17-05
REVIEW AND WHAT'S NEXT
© Colfax International, 2013–2017