



Programming and Optimization for Intel[®] Architecture

The Hands-On Workshop (HOW) Series

Colfax International — @colfaxintl


August 2016 , Rev. 03b

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.

© Colfax International, 2013–2016

Parallel Programming Boot Camp (1-Day) / Workshop (4-Days)



Instructor-led 1-day or 4-days training, at your office or at Colfax facility in Sunnyvale, CA

[Click here to learn more](#)

1-Day Parallel Programming Boot Camp
 For software engineers and architects, providing an overview of parallel programming frameworks and optimization guidelines for multi-core CPUs (Intel® Xeon®) and many-core coprocessors (Intel® Xeon Phi™):

- Discussions about three layers of parallelism: SIMD, Threads, Cluster environment
- Tips for quick porting/development of HPC software applications
- Real-life examples of code and optimization techniques
- Hardware solution and corresponding software implementations, APIs, and frameworks

4-Days Parallel Programming Workshop
 For the developer who wants to hit the ground running with the modern multi-core CPUs (Intel® Xeon®), many-core coprocessors (Intel® Xeon Phi™) and leading software development tools:

- Hardware installation
- MPSS tools and the Linux environment on the Intel® Xeon Phi™ coprocessor
- Exploring differences in serial vs. parallel programming / processing / hardware usage
- Accelerated clusters
- Optimizations of vector arithmetics, memory traffic, thread parallelism and communication
- Using the Intel® Math Kernel Library

Register Now!

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

- HOW to Program Intel Architecture
 - ▶ 01. Parallelism, specialization, guided tour – Aug 29
 - ▶ 02. Programming Intel Xeon Phi (KNC, KNL) – Aug 30
- HOW to Express Parallelism
 - ▶ 03. Automatic vectorization – Aug 31
 - ▶ 04. Multi-threading with OpenMP – Sep 1
- HOW to Get Performance
 - ▶ 05. Comprehensive demo – Sep 2
 - ▶ 06. Scalar & vectorization tuning – Sep 5
 - ▶ 07. Multi-threading I – Sep 6
 - ▶ 08. Multi-threading II – Sep 7
 - ▶ 09. Memory traffic – Sep 8
- HOW to Scale
 - ▶ 10. Distributed Computing: MPI – Sep 9

August 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			

September 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	

 — Webinar+remote access

HOW Online

Course page: colfaxresearch.com/how-16-08

- 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-16-08



Forums (technical):

colfaxresearch.com/discussion

Log In/Register

COLFAX RESEARCH

CONTRIBUTING TO INNOVATIONS IN COMPUTING

/ READ WATCH LEARN **FORUMS** CONNECT JOIN

Join the Conversation

Welcome to Colfax Research forums, an online community for you to engage with HPC experts, software architects, developers, computational researchers, scientists, students and more—so you can acquire new knowledge, share ideas, and build new relationships.

Tap our experts and your peers to help meet the challenge of optimizing applications on modern hardware. This is the place to browse or post questions (and get answers) related to computational science, parallel programming and code modernization on Intel® Architecture.

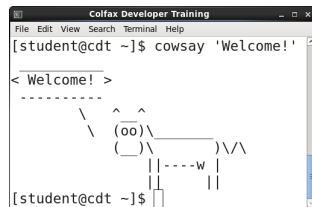
Welcome aboard. Post questions today!

Email (organizational):

training@colfax-intl.com

Hands-On Exercises and Remote Access

- 96 people receive a remote access token
 - Virtualized Intel Xeon CPU, real Intel Xeon Phi coprocessor (1st gen, KNC), SW tools
 - Can access the system the entire 2 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



```
Colfax Developer Training
File Edit View Search Terminal Help
[student@cdt ~]$ cowsay 'Welcome!'
< Welcome! >
-----
      \   ^__^
         (oo)\_____)
            (_____)
                ||----w |
                ||     ||

[student@cdt ~]$
```

§2. Refresh

Performance Optimization

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*

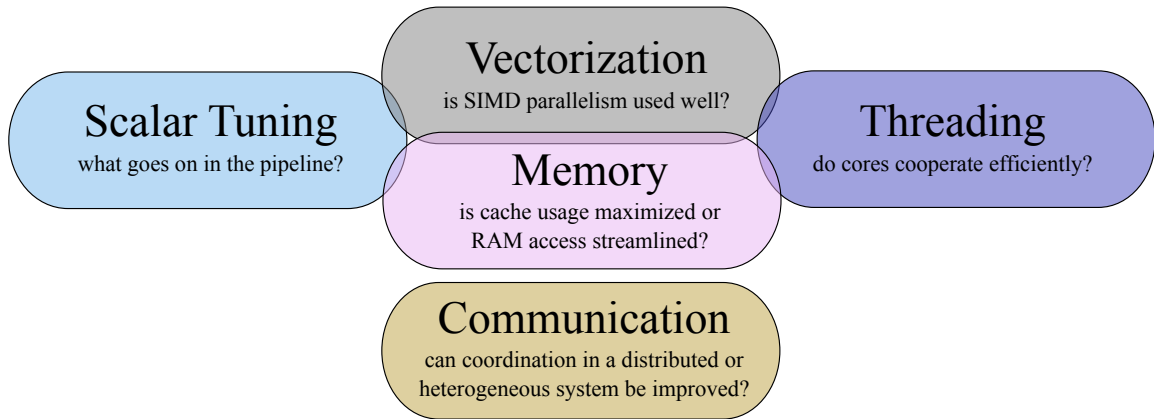


* socket and coprocessor versions

Knights Landing (KNL)

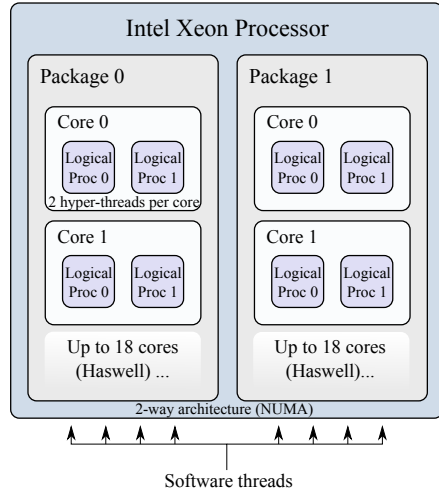
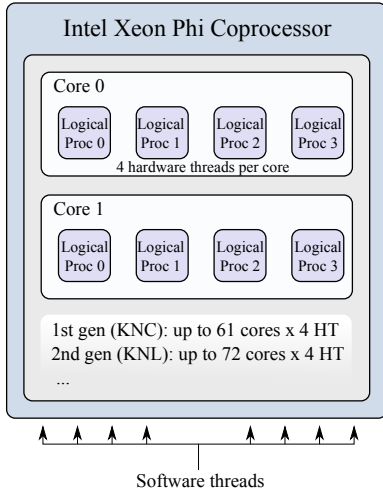
Intel Many Integrated Core (MIC) Architecture

Optimization Areas



Cores, Threads and OpenMP

Processor Hierarchy

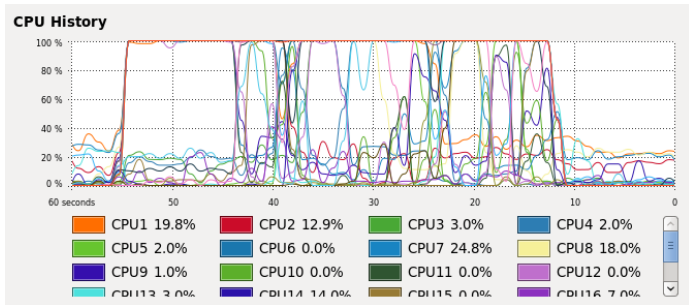


§3. Optimization of Multi-Threading II: Memory Access Aspect

Thread Affinity

What is Thread Affinity

- OpenMP threads may migrate between cores according to OS decisions.
- Forbid migration — improve locality — increase the performance.



The KMP_AFFINITY Environment Variable

```
KMP_AFFINITY=[<modifier>,...]<type>[,<permute>][, <offset>]
```

modifier:

- verbose/nonverbose
- respect/norespect
- warnings/nowarnings
- granularity=core or thread
- type=compact, scatter or balanced
- type=explicit, proclist=[<proc_list>]
- type=disabled or none.

The most important argument is type:

- compact: place threads as *close to each* other as possible
- scatter: place threads as *far from each* other as possible

Thread Affinity: Scatter Pattern

Generally beneficial for bandwidth-bound applications.

`KMP_AFFINITY=scatter,granularity=fine`

Threads:

0

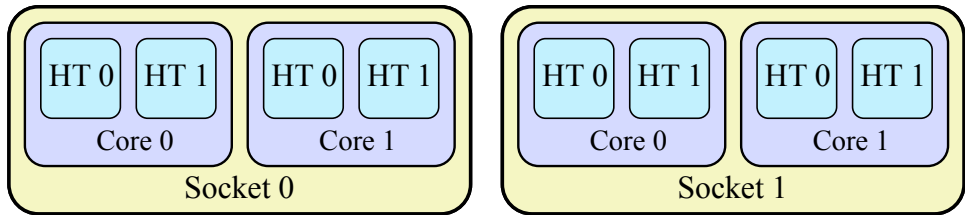
2

1

3



Cores:



Thread Affinity: Compact Pattern

Generally beneficial for compute-bound applications.

`KMP_AFFINITY=compact,granularity=fine`

Threads:

0

1

2

3

4

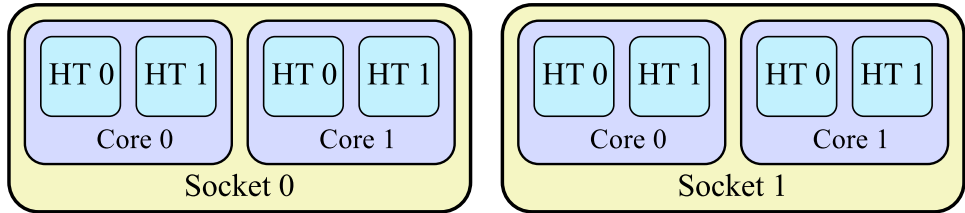
5

6

7



Cores:



Thread Affinity: Compact Pattern with Permutation

Same effect as “compact” without hyper-threading.

`KMP_AFFINITY=compact,granularity=fine,1`

Threads:

0

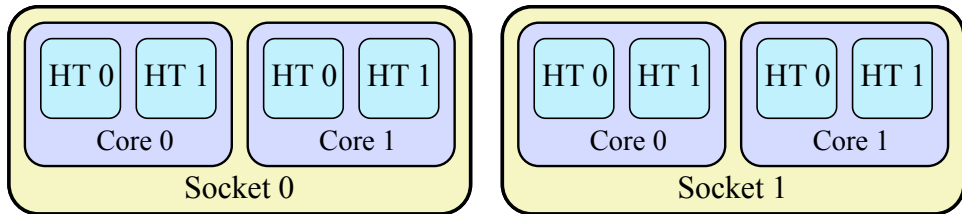
1

2

3



Cores:



Thread Affinity: Compact Pattern with an Offset

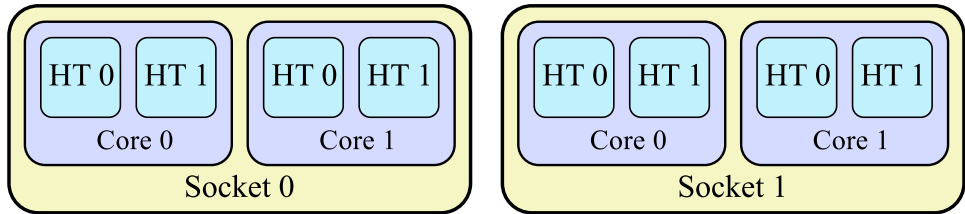
Useful for partitioning a system between multiple processes.

`KMP_AFFINITY=compact,granularity=fine,0,4`

Threads:

0 1 2 3
↓ ↓ ↓ ↓

Cores:



Bandwidth-bound, KMP_AFFINITY=scatter

```
vega@lyra% export OMP_NUM_THREADS=32
vega@lyra% export KMP_AFFINITY=none
vega@lyra% for i in {1..4} ; do ./rowsum_stripmine | tail -1; done
Problem size: 2.980 GB, outer dimension: 4, threads: 32
Strip-mine and collapse: 0.061 +/- 0.002 seconds (52.89 +/- 1.31 GB/s)
Strip-mine and collapse: 0.059 +/- 0.002 seconds (54.11 +/- 1.56 GB/s)
Strip-mine and collapse: 0.077 +/- 0.001 seconds (41.71 +/- 0.69 GB/s)
Strip-mine and collapse: 0.070 +/- 0.005 seconds (45.59 +/- 3.14 GB/s)
vega@lyra% export OMP_NUM_THREADS=16
vega@lyra% export KMP_AFFINITY=scatter
vega@lyra% for i in {1..4}; do ./rowsum_stripmine | tail -1 ; done
Problem size: 2.980 GB, outer dimension: 4, threads: 16
Strip-mine and collapse: 0.059 +/- 0.004 seconds (54.47 +/- 3.25 GB/s)
Strip-mine and collapse: 0.061 +/- 0.004 seconds (52.30 +/- 3.30 GB/s)
Strip-mine and collapse: 0.062 +/- 0.005 seconds (51.37 +/- 4.29 GB/s)
Strip-mine and collapse: 0.058 +/- 0.001 seconds (55.48 +/- 1.27 GB/s)
```

Compute-Bound, KMP_AFFINITY=compact/balanced

```
1 double* A = (double*)_mm_malloc(sizeof(double)*N*Nld, 64);
2 double* B = (double*)_mm_malloc(sizeof(double)*N*Nld, 64);
3 double* C = (double*)_mm_malloc(sizeof(double)*N*Nld, 64);
4
5 for(int k = 0; k < nIter; k++) {
6
7     dgemm(&tr, &tr, &N, &N, &N, &v, A, &Nld, B, &Nld, &v, C, &N);
8
9     double flopsNow = (2.0*N*N*N+1.0*N*N)*1e-9/(t2-t1);
10    printf("Iteration %d: %.1f GFLOP/s\n", k+1, flopsNow);
11 }
12 _mm_free(A); _mm_free(B); _mm_free(C);
```

Compute-Bound, KMP_AFFINITY=compact/balanced

```
vega@lyra% icpc -o bench-dgemm -mkl -mmic bench-dgemm.cc
```

```
vega@lyra% micnativeloadex ./bench-dgemm
```

```
Iteration 1: 312.7 GFLOP/s
```

```
Iteration 2: 346.5 GFLOP/s
```

```
Iteration 3: 348.5 GFLOP/s
```

```
Iteration 4: 347.2 GFLOP/s
```

```
Iteration 5: 348.3 GFLOP/s
```

```
vega@lyra% micnativeloadex ./bench-dgemm -e "KMP_AFFINITY=compact"
```

```
Iteration 1: 626.8 GFLOP/s
```

```
Iteration 2: 769.1 GFLOP/s
```

```
Iteration 3: 769.4 GFLOP/s
```

```
Iteration 4: 769.3 GFLOP/s
```

```
Iteration 5: 769.4 GFLOP/s
```


The KMP_PLACE_THREADS Environment Variable

Only for Xeon Phi, control the # of cores and # of threads per core:

```
KMP_PLACE_THREADS=[<cores>c,]<threads-per-core>t
```

Complements KMP_AFFINITY:

```
vega@lyra-mic0% export KMP_PLACE_THREADS=61c,3t # 3 threads per core
vega@lyra-mic0% export KMP_AFFINITY=balanced
vega@lyra-mic0% ./my-native-app
```

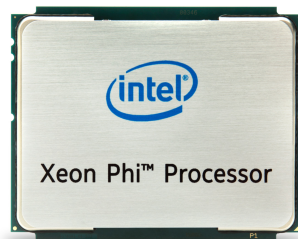
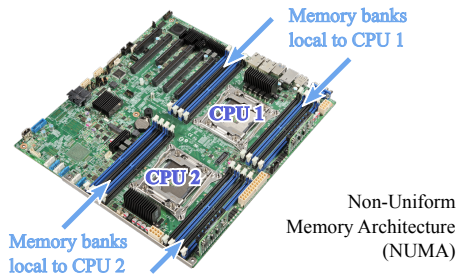
or

```
vega@lyra% export MIC_ENV_PREFIX=XEONPHI
vega@lyra% export XEONPHI_KMP_PLACE_THREADS=60c,3t # 3 threads per core
vega@lyra% export XEONPHI_KMP_AFFINITY=balanced
vega@lyra% ./my-offload-app
```

NUMA Locality

NUMA Architectures

NUMA = Non-Uniform Memory Access. Cores have fast access to local memory, slow access to remote memory.



Examples:

- Multi-socket Intel Xeon processors
- Second generation Intel Xeon Phi

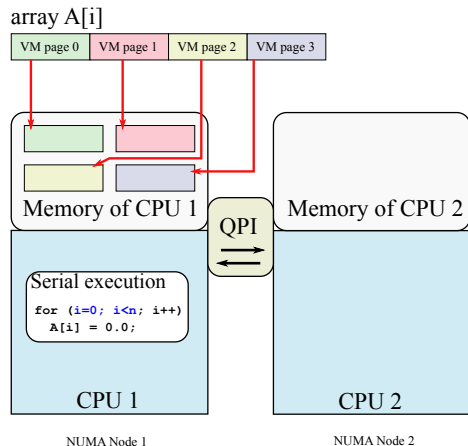
Allocation on First Touch

- Memory allocation occurs not during `_mm_malloc()`, but upon the first write to the buffer (“first touch”)
- Default NUMA allocation policy is “on first touch”
- For better performance in NUMA systems, initialize data with the same parallel pattern as during data usage

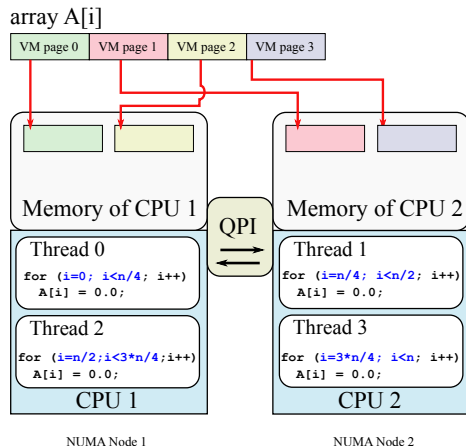
```
1 float* A = (float*)_mm_malloc(n*m*sizeof(float), 64);
2
3 // Initializing from parallel region for better performance
4 #pragma omp parallel for
5 for (int i = 0; i < n; i++)
6     for (int j = 0; j < m; j++)
7         A[i*m + j] = 0.0f;
```

First-Touch Allocation Policy

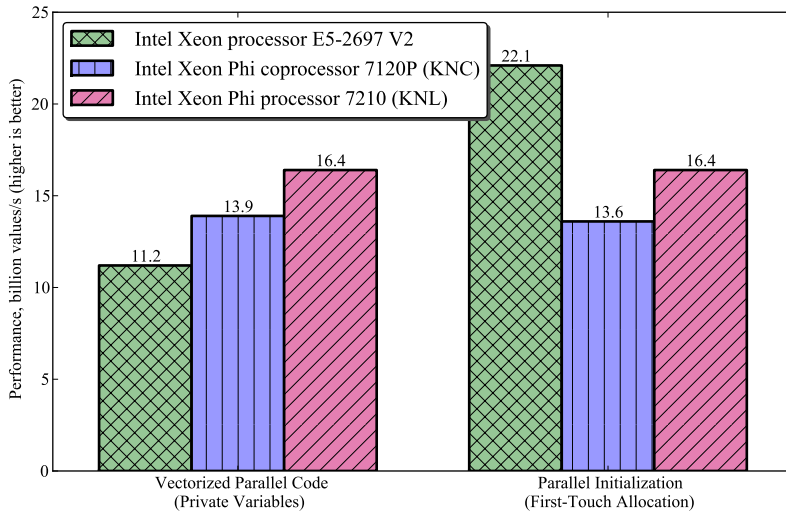
Poor First-Touch Allocation



Good First-Touch Allocation



Impact of First-Touch Allocation



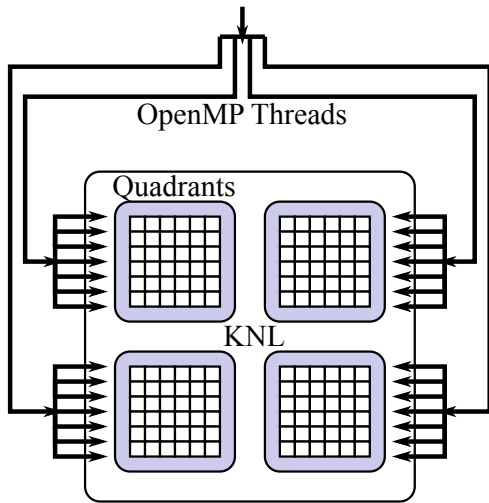
Binding to NUMA Nodes with numactl

- numactl – a Linux tool for controlling NUMA policy for processes

```
vega@lyra% numactl --hardware
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 12 13 14 15 16 17
node 0 size: 65457 MB
node 0 free: 24426 MB
node 1 cpus: 6 7 8 9 10 11 18 19 20 21 22 23
node 1 size: 65536 MB
node 1 free: 53725 MB
node distances:
node  0  1
  0:  10  21
  1:  21  10
vega@lyra% numactl --membind=<nodes> --cpunodebind=<nodes> ./myApplication
```

Nested Parallelism

Nested Parallelism with OpenMP



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

OpenMP Hot Teams

Xeon

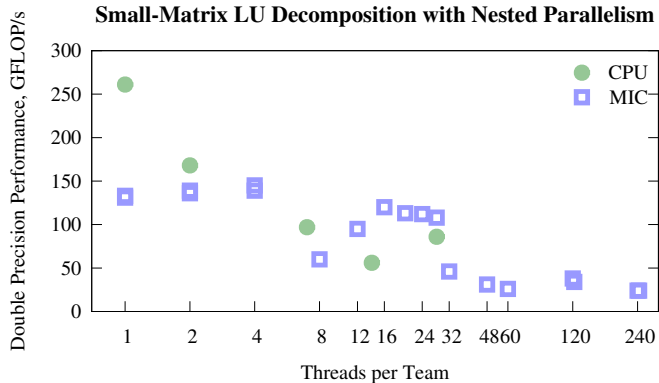
- OMP_NUM_THREADS=2,14
- OMP_NESTED=1
OMP_PROC_BIND=spread,close
OMP_PLACES=cores
- KMP_HOT_TEAMS_MODE=1
KMP_HOT_TEAMS_MAX_LEVEL=2
OMP_MAX_ACTIVE_LEVELS=2

Xeon Phi

- OMP_NUM_THREADS=60,4
 - OMP_NESTED=1
OMP_PROC_BIND=spread,close
OMP_PLACES=threads
 - KMP_HOT_TEAMS_MODE=1
KMP_HOT_TEAMS_MAX_LEVEL=2
OMP_MAX_ACTIVE_LEVELS=2
-
-

Small Matrix LU Decomposition with Nested Parallelism

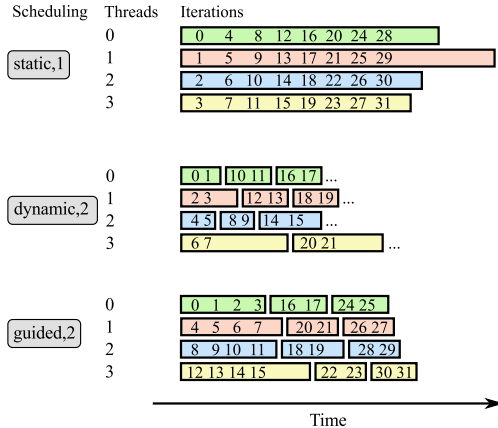
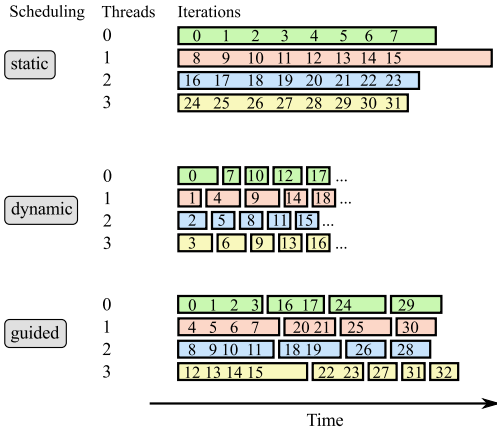
Decomposing 10^3 small square matrices of size 1024×1024 .



See [HOW Series "Tools"](#) (MKL webinar).

Loop Scheduling

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

```

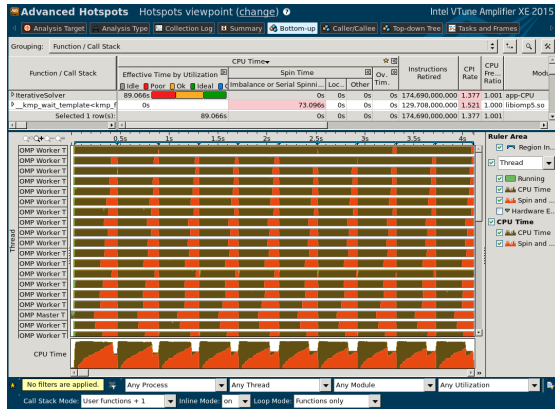
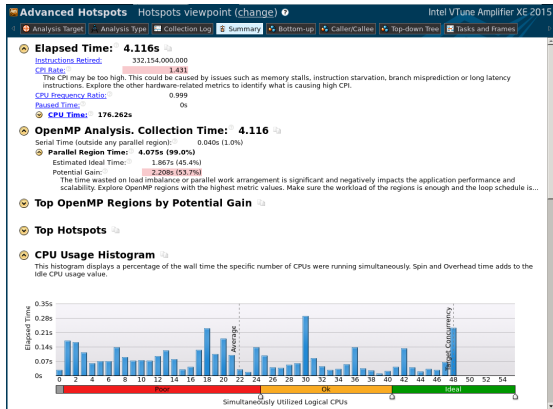
1 int IterativeSolver(int n, double* M, double* b, double* x, double minAccuracy){
2     double accuracy; int iters=0; double bTrial[n] __attribute__((aligned(64)));
3     x[0:n] = 0.0; // Initial guess
4     do { iters++; // The Jacobi method - iterate until convergence
5         for (int i = 0; i < 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; }

```

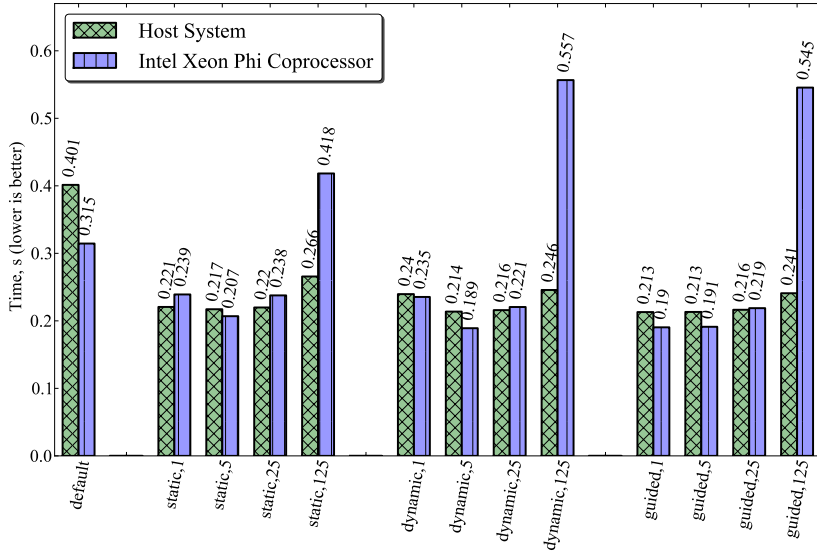
An Iterative Jacobi Solver

```

1 #pragma omp parallel for
2   for (int c = 0; c < nVectors; c++)
3     IterativeSolver(n, M, &b[c*n], &x[c*n], accuracy[c]);
  
```



Performance of Iterative Jacobi Solver



§4. Review and What's Next

Summary

This session:

- 1 Setting affinity prevents thread migration
- 2 Affinity pattern “scatter” for bandwidth-bound
- 3 Affinity pattern “compact” for compute-bound
- 4 NUMA locality: use parallel first touch
- 5 Nested parallelism: reduce memory overhead/expose more work-items
- 6 Tune scheduling: tradeoff between load balancing and overhead

Next session: optimization of memory traffic.

[Learn More](#)

HOW Series: Knights Landing

OPTIMIZATION FOR INTEL® XEON PHI™ PROCESSOR DEVELOPER'S GUIDE TO KNIGHTS LANDING

Free 2-Hour Webinar

SEPTEMBER 13, 2016

Register Today >



colfaxresearch.com/how-knl/

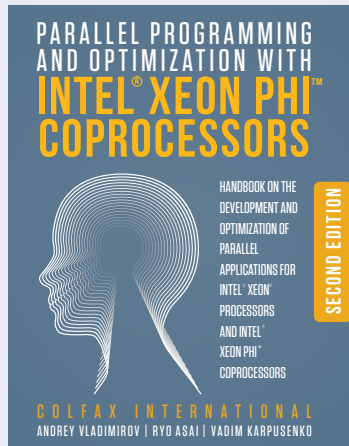
Textbook

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

Parallel Programming and Optimization with Intel® Xeon Phi™ Coproprocessors

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

The screenshot shows the Colfax Research website with a search bar at the top right and navigation links (READ, WATCH, LEARN, CONNECT, JOIN) on the left. The main content area features several articles:

- Popular:** "The Hands-On Tutorials (HOT) webinars: details an efficient programming for Intel architecture" and "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." with a book cover image.
- Research and Educational Publications:**
 - "Introduction to Intel DAAL, Part 1: 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 HaaS 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 TX Acceleration May Be Enough)"
- Featured Video:** "Additional Reading" with a video player showing a presentation slide.

The 'Consulting' section features a blue header and social media share buttons. Below is a video player showing a server rack. The text describes consulting services for enterprises, research, and education. A list of services includes:

- Optimize your existing application to take advantage of parallelism, from vectors to cores to clusters and clouds.
- Future-proof your application for upcoming innovations.
- Accelerate your application using coprocessor technologies.
- Investigate the potential system configurations that satisfy your cost, power, and performance requirements.
- Take a clean-room look at hardware, a neutral approach to reduce vendor lock-in.

Below the list is a video player titled "Episode 2.1 - Purpose of the MIC architecture" with a thumbnail showing a server rack.

This block contains two article thumbnails:

- Software Developer's Introduction to the HGST Ultrastar Archive HaaS SMR Drives:** Features an image of two hard drives. The text mentions that the paper will discuss the HGST Ultrastar Archive HaaS SMR drives, which offer a new storage solution for high-capacity, high-performance applications.
- Configuration and Benchmarks of Peer-to-Peer Communication over Gigabit Ethernet and InfiniBand in a Cluster with Intel Xeon Phi Coprocessors:** Features a diagram of a network cluster with nodes labeled 'CPU' and 'Phi'. The text discusses the configuration and benchmarks of peer-to-peer communication over Gigabit Ethernet and InfiniBand in a cluster with Intel Xeon Phi coprocessors.

This block contains two article thumbnails:

- Fluid Dynamics with Fortran on Intel Xeon Phi coprocessors:** Features a 3D visualization of fluid dynamics. The text discusses the use of Fortran on Intel Xeon Phi coprocessors for fluid dynamics simulations.
- Interview with James Reinders: future of Intel MIC architecture, parallel programming, education:** Features a video player showing an interview with James Reinders. The text discusses the future of Intel MIC architecture, parallel programming, and education.

<http://colfaxresearch.com/>