



PROGRAMMING AND OPTIMIZATION FOR INTEL[®] ARCHITECTURE

The Hands-On Workshop (HOW) Series
Session 9

Colfax International — colfaxresearch.com

January 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 – Jan 16
 - 02. Xeon Phi, Coprocessors, Omni-Path – Jan 17
- ▶ **Module II. Expresssing Parallelism**
 - 03. Expressing Parallelism with Vectors – Jan 18
 - 04. Multi-threading with OpenMP – Jan 19
 - 06. Distributed Computing, MPI – Jan 20
- ▶ **Module III. Optimization**
 - 06. Optimization Overview: N-body – Jan 23
 - 07. Scalar tuning, Vectorization – Jan 24
 - 08. Common Multi-threading Problems – Jan 25
 - 09. Multi-threading, Memory Aspect – Jan 26
 - 10. Access to Caches and Memory – Jan 27

January 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				
— Webinar+remote access						

Course page:

colfaxresearch.com/how-17-01

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

More workshops:

colfaxresearch.com/training



GET YOUR QUESTIONS ANSWERED

Chat (current):

colfaxresearch.com/how-17-01



Forums (technical):

colfaxresearch.com/discussion

COLFAX RESEARCH

CONTRIBUTING TO INNOVATIONS IN COMPUTING

[Log In/Register](#)

[/](#) [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@colfaxresearch.com

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. REFRESH



PERFORMANCE OPTIMIZATION

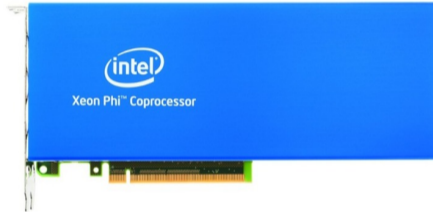
Intel Xeon Processor



Current: Broadwell
Upcoming: Skylake

Multi-Core Architecture

Intel Xeon Phi Coprocessor, 1st generation



Knights Corner (KNC)

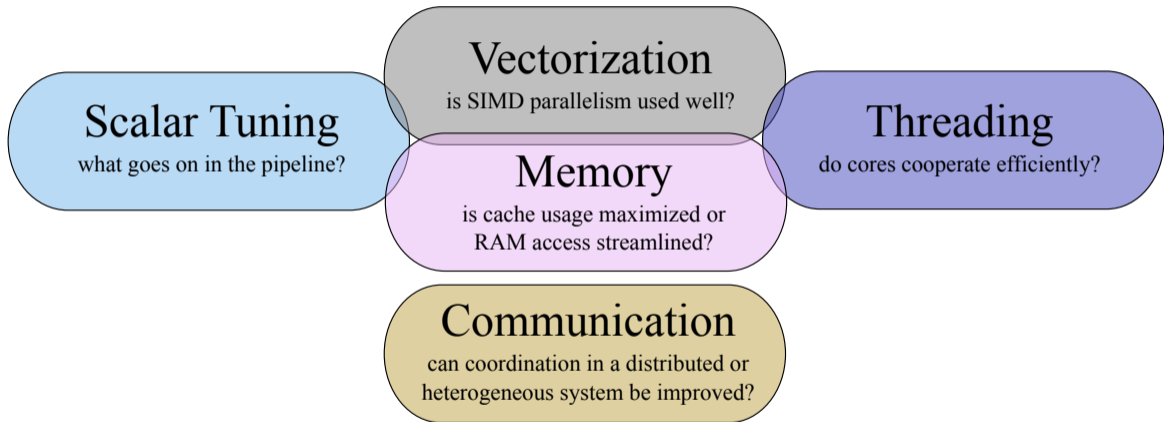
Intel Xeon Phi Processor, 2nd generation*



* socket and coprocessor versions

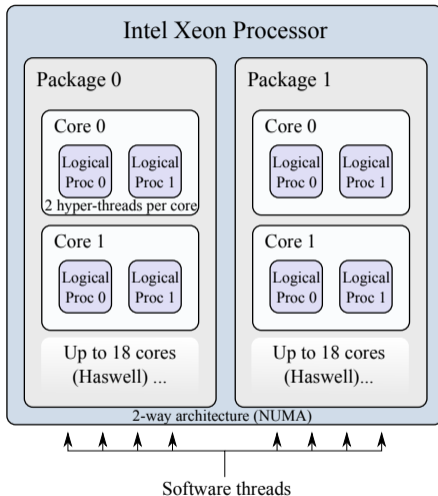
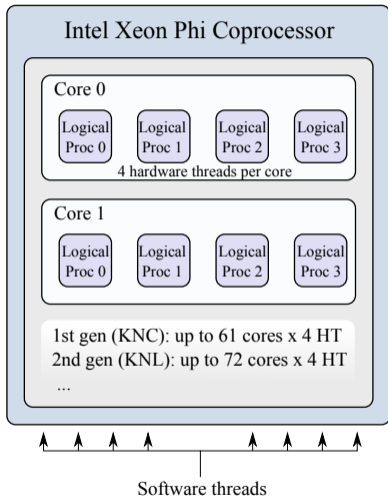
Knights Landing (KNL)

Intel Many Integrated Core (MIC) Architecture





CORES, THREADS AND OPENMP



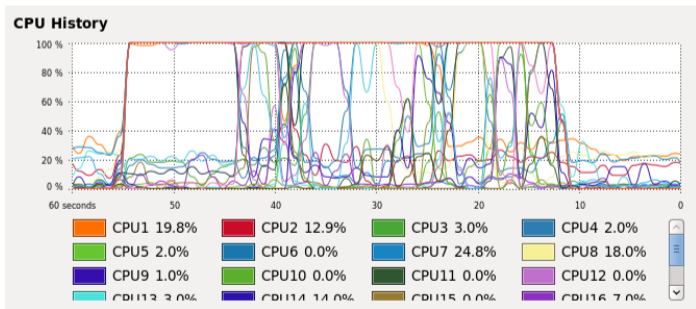
§3. MULTI-THREADING II: MEMORY ASPECT



THREAD AFFINITY

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



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

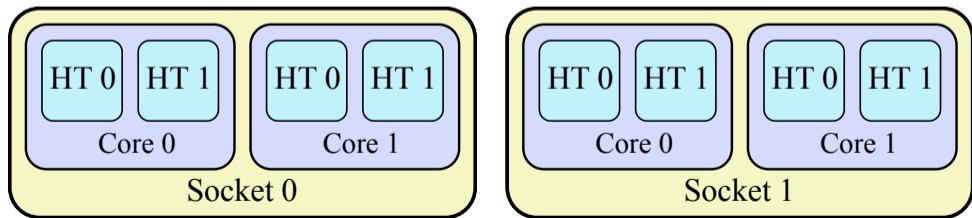
2

1

3



Cores:



THREAD AFFINITY: COMPACT PATTERN

Generally beneficial for compute-bound applications.

`KMP_AFFINITY=compact,granularity=fine`

Threads:

0

1

2

3

4

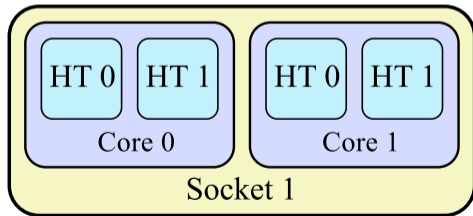
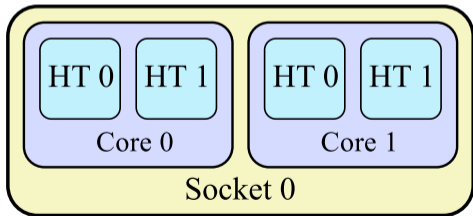
5

6

7



Cores:



THREAD AFFINITY: COMPACT PATTERN WITH PERMUTATION

Same effect as “compact” without hyper-threading.

`KMP_AFFINITY=compact,granularity=fine,1`

Threads:

0

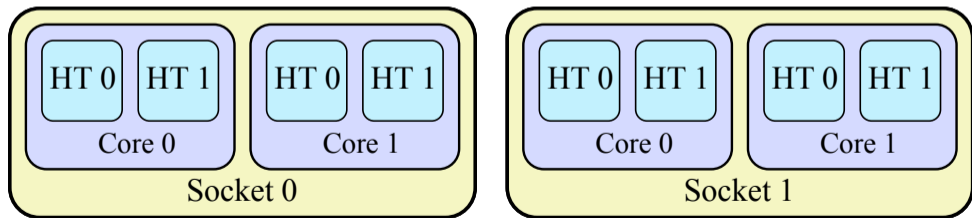
1

2

3



Cores:




THREAD AFFINITY: COMPACT PATTERN WITH AN OFFSET

Useful for partitioning a system between multiple processes.

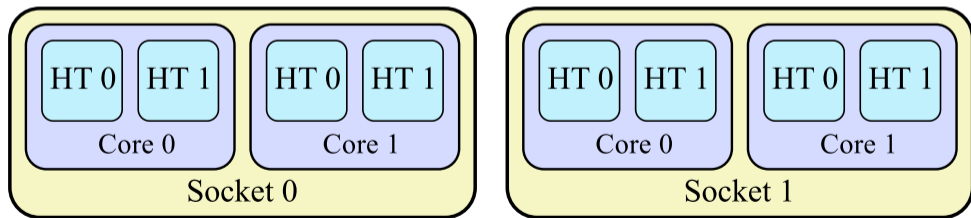
`KMP_AFFINITY=compact,granularity=fine,0,4`

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_HW_SUBSET ENVIRONMENT VARIABLE

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

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

Complements KMP_AFFINITY:

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

OR

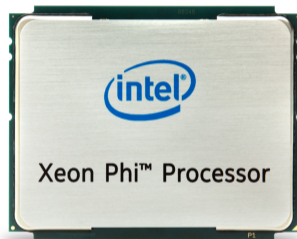
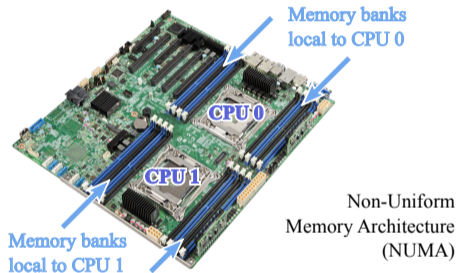
```
vega@lyra% export MIC_ENV_PREFIX=XEONPHI
vega@lyra% export XEONPHI_KMP_HW_SUBSET=60c,3t # 3 threads per core
vega@lyra% export XEONPHI_KMP_AFFINITY=balanced
vega@lyra% ./my-offload-app
```



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 in **sub-NUMA clustering mode**

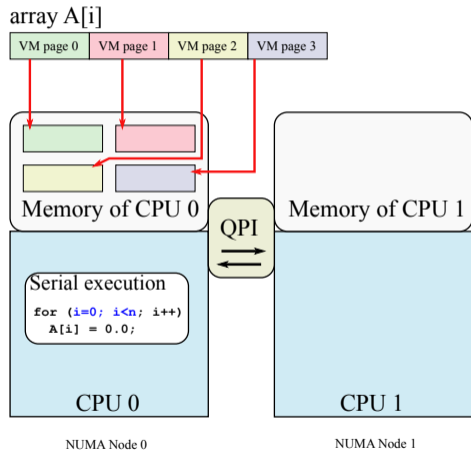
ALLOCATION ON FIRST TOUCH

- ▶ 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

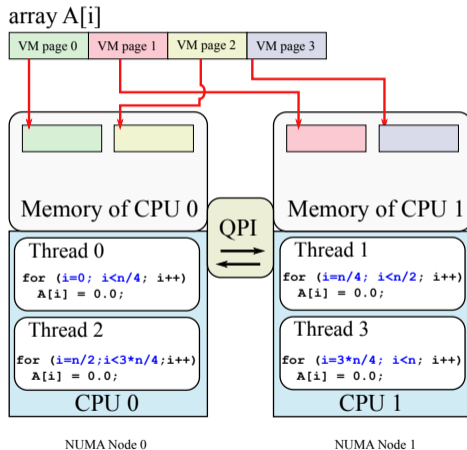
```
1 float* A = (float*)_mm_malloc(n*m*sizeof(float), 64);
2
3 // Initializing from parallel region for better performance
4 #pragma omp parallel for
5 for (int i = 0; i < n; i++)
6     for (int j = 0; j < m; j++)
7         A[i*m + j] = 0.0f;
```

FIRST-TOUCH ALLOCATION POLICY

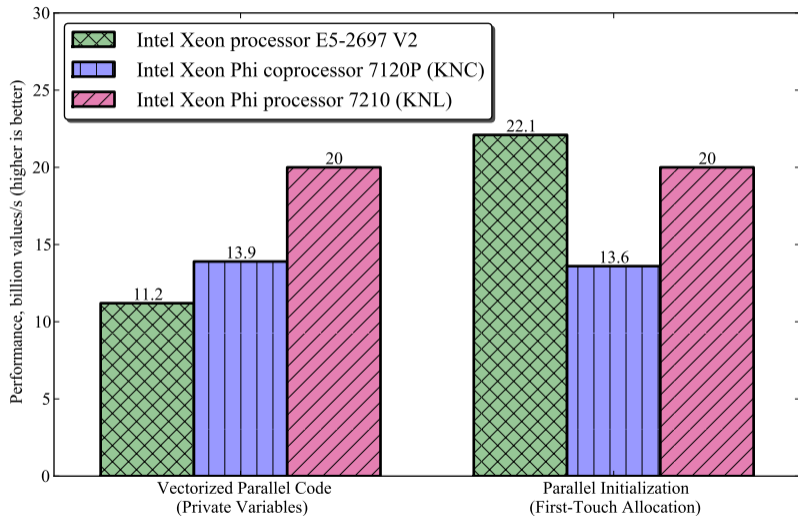
Poor First-Touch Allocation



Good First-Touch Allocation



IMPACT OF FIRST-TOUCH ALLOCATION



BINDING TO NUMA NODES WITH `numactl`

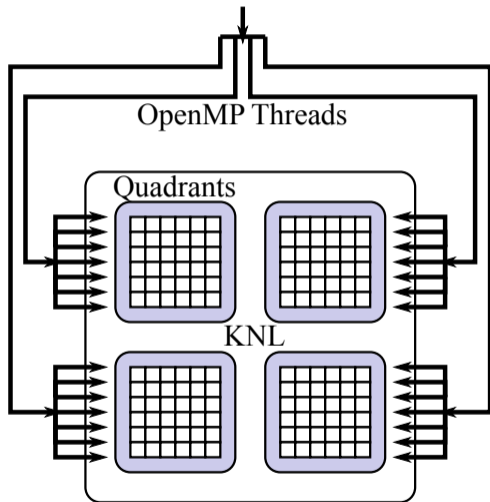
- ▶ `libnuma` – a Linux library for fine-grained control over NUMA policy
- ▶ `numactl` – a tool for global NUMA policy control

```
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  }

```

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

OPENMP HOT TEAMS

Xeon

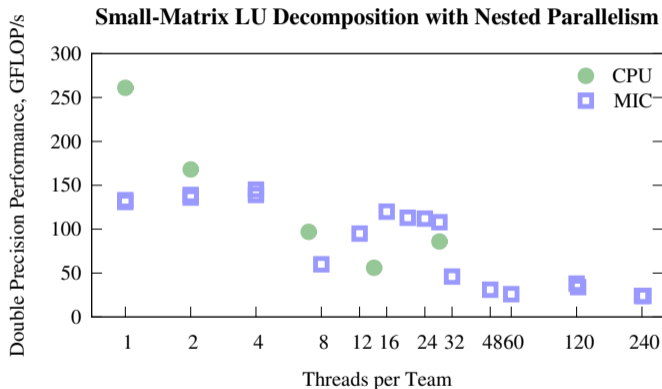
- ▶ `OMP_NUM_THREADS=2,14`
- ▶ `OMP_NESTED=1`
`OMP_PROC_BIND=spread,close`
`OMP_PLACES=cores`
- ▶ `KMP_HOT_TEAMS_MODE=1`
`KMP_HOT_TEAMS_MAX_LEVEL=2`
`OMP_MAX_ACTIVE_LEVELS=2`

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

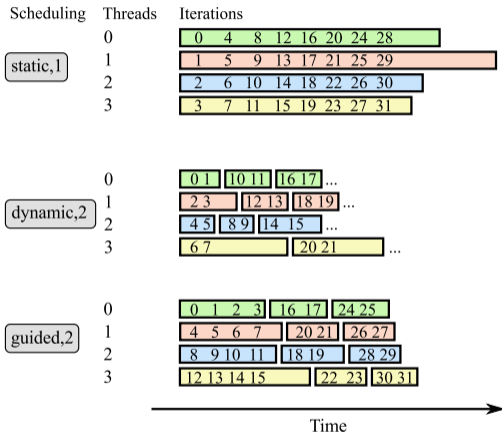
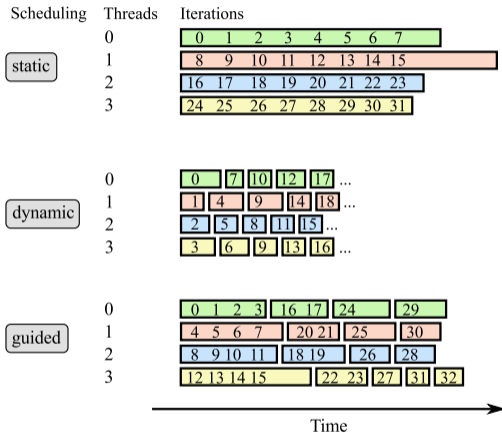
Decomposing 10^3 small square matrices of size 1024×1024 .



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; }

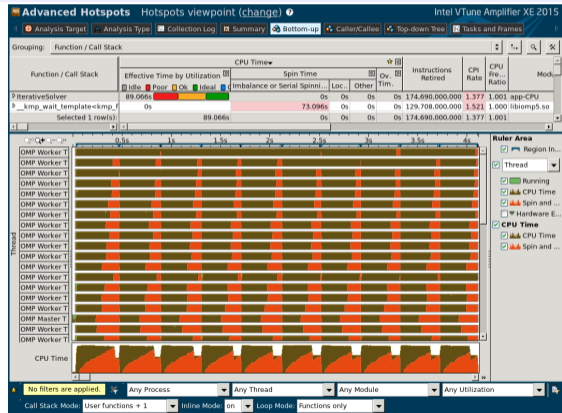
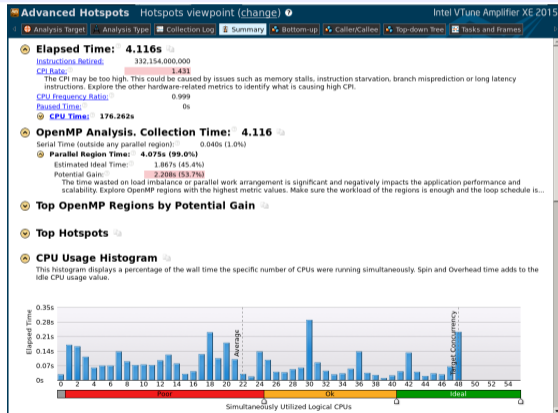
```

AN ITERATIVE JACOBI SOLVER

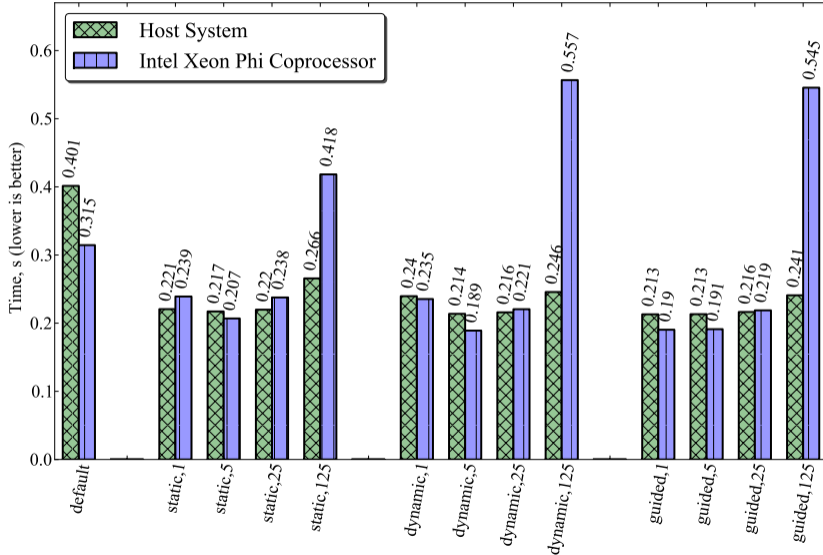
```

1 #pragma omp parallel for
2   for (int c = 0; c < nBVectors; 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.

COLFAX RESEARCH
CONTRIBUTING TO INNOVATIONS IN COMPUTING

READ WATCH LEARN CONNECT JOIN

[Log In/Out or Register](#)

To search, type and hit enter



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

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.
© 2015, Colfax International, 508 pages.

Featured Video

See Research material on reconstruction to a streaming code

[View Video](#)

Research and Educational Publications

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

[View Article](#)

Optimization Techniques for the Intel MIC Architecture, Part 3 of 3: False Sharing and Padding

[View Article](#)

Software Developer's Introduction to the HGST Ultrastar Archive H800 SMR Drives

[View Article](#)

Optimization Techniques for the Intel MIC Architecture, Part 2 of 3: Strip-Mining for Vectorization

[View Article](#)

Optimization Techniques for the Intel MIC Architecture, Part 1 of 3: Multi-Threading and Parallel Reduction

[View Article](#)

Performance to Power and Performance to Cost Ratios with Intel Xeon Phi Coprocessors (and why ix Acceleration May Be Enough)

[View Article](#)

Consulting


Facebook

Twitter

LinkedIn

Google+

Share



Colfax offers consulting services for enterprises, research help you:


- 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 reduce your computing pro

Episode 2.1 — Purpose of the MIC architecture

[View Video](#)

Software Developer's Introduction to the HGST Ultrastar Archive H800 SMR Drives


[View Article](#)



In this paper we will discuss the new HGST Ultrastar Archive H800 SMR drives, which are the first 8TB hard drives to be available in a 3.5-inch form factor. These drives are well suited for high capacity and high performance applications. These drives are well suited for high capacity and high performance applications. These drives are well suited for high capacity and high performance applications.

Fluid Dynamics with Fortran on Intel Xeon Phi coprocessors

[View Article](#)



In this demonstration, a Fortran program solves a partial differential equation on a 3D domain. The program is written in Fortran and runs on an Intel Xeon Phi coprocessor. The program is written in Fortran and runs on an Intel Xeon Phi coprocessor. The program is written in Fortran and runs on an Intel Xeon Phi coprocessor.

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

[View Article](#)



This paper reports on the configuration and benchmarks of a cluster with Intel Xeon Phi coprocessors. The cluster is configured with Gigabit Ethernet and InfiniBand. The benchmarks are run on a variety of applications. The benchmarks are run on a variety of applications. The benchmarks are run on a variety of applications.

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

[View Article](#)



A few weeks ago we conducted an interview with James Reinders, the Director of Intel's Parallel Computing Group. In this interview, we discuss the future of Intel MIC architecture, parallel programming, and education. In this interview, we discuss the future of Intel MIC architecture, parallel programming, and education.

http://colfaxresearch.com/