PROGRAMMING AND OPTIMIZATION FOR INTEL® ARCHITECTURE

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

Session 9

Colfax International — *colfaxresearch.com*

February 2017

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

- Module I. Programming
  - 01. Intel Architecture and Modern Code – Feb 13
  - 02. Xeon Phi, Coprocessors, Omni-Path – Feb 14
- Module II. Expressing Parallelism
  - 03. Automatic vectorization – Feb 15
  - 04. Multi-threading with OpenMP – Feb 16
  - 06. Distributed Computing, MPI – Feb 17
- Module III. Optimization
  - 06. Optimization Overview: N-body – Feb 20
  - 07. Scalar tuning, Vectorization – Feb 21
  - 08. Common Multi-threading Problems – Feb 22
  - 09. Multi-threading, Memory Aspect – Feb 23
  - 10. Access to Caches and Memory – Feb 24

February 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				

— Webinar+remote access

HOW SERIES ONLINE

Course page:
colfaxresearch.com/how-17-02

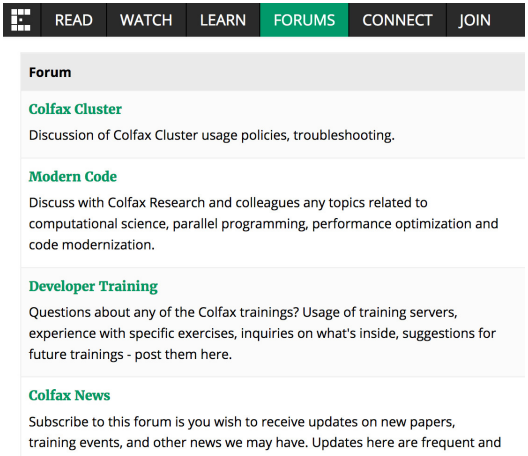
- Slides
- Code
- Video
- Chat

More workshops:
colfaxresearch.com/training

GET YOUR QUESTIONS ANSWERED: CHAT

colfaxresearch.com/how-17-02

GET YOUR QUESTIONS ANSWERED: FORUMS



colfaxresearch.com/discussion

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

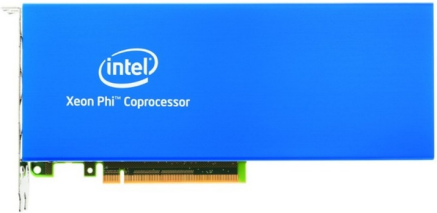
COMPUTING PLATFORMS

Intel Xeon Processor

Current: Broadwell
Upcoming: Skylake

Multi-Core Architecture

Intel Xeon Phi Coprocessor, 1st generation



Knights Corner (KNC)

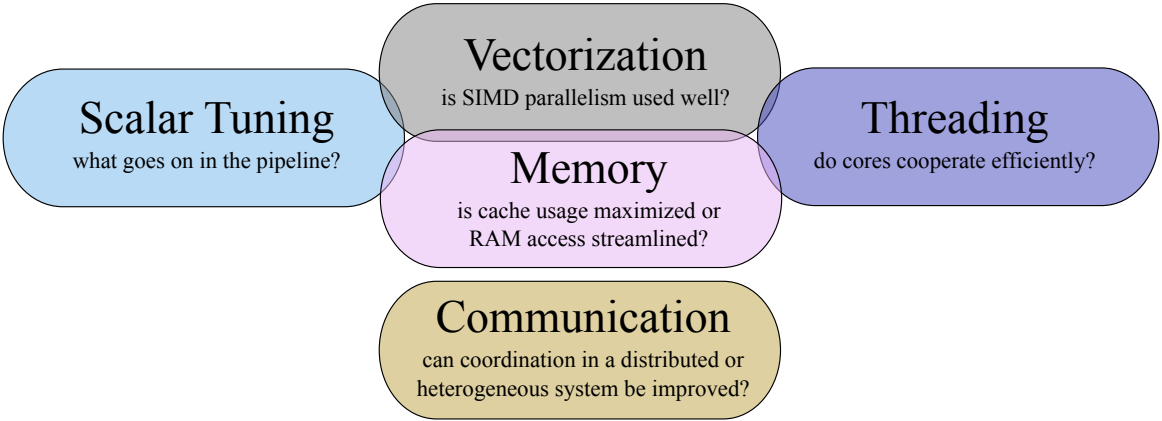
Intel Xeon Phi Processor, 2nd generation*

* socket and coprocessor versions

Knights Landing (KNL)

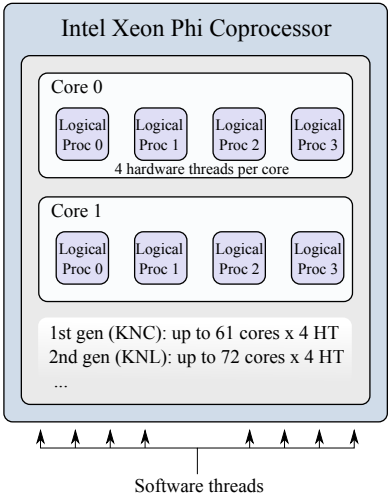
Intel Many Integrated Core (MIC) Architecture

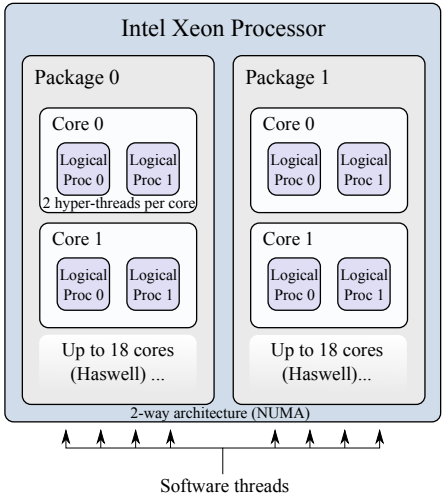
OPTIMIZATION AREAS



CORES, THREADS AND OPENMP

PROCESSOR HIERARCHY



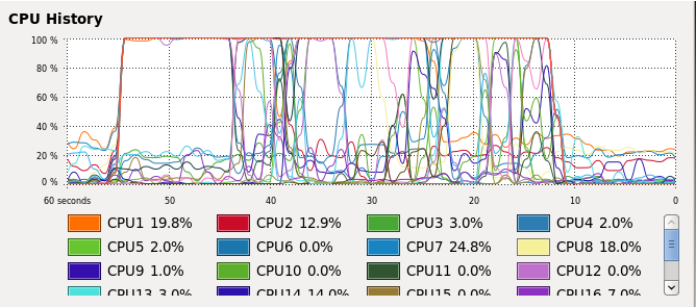


§3. MULTI-THREADING II: MEMORY ASPECT

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_HW_SUBSET ENVIRONMENT VARIABLE

Control the # of cores and # of threads per core:

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

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

or

```
vega@lyra% export MIC_ENV_PREFIX=XEONPHI
vega@lyra% export KMP_HW_SUBSET=1t # 1 thread per core on host
vega@lyra% export XEONPHI_KMP_HW_SUBSET=2t # 2 threads per core on Xeon Phi
vega@lyra% ./my-offload-app
```

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

OMP_PROC_BIND AND OMP_PLACES VARIABLES

Control the binding pattern, including nested parallelism:

```
OMP_PROC_BIND=type[,type[,...]]
```

Here `type=true`, `false`, `spread`, `close` or `master`.
Comma separates settings for different levels of nesting (`OMP_NESTED` must be enabled).

Control the granularity of binding:

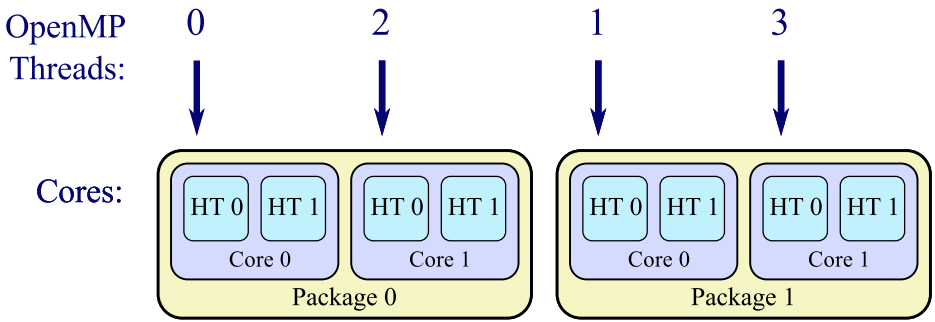
```
OMP_PLACES=<threads|cores|sockets|(explicit)>
```

THREAD AFFINITY: SCATTER PATTERN

Generally beneficial for bandwidth-bound applications.

`OMP_NUM_THREADS={1 thread/core}` or `KMP_HW_SUBSET=1t`

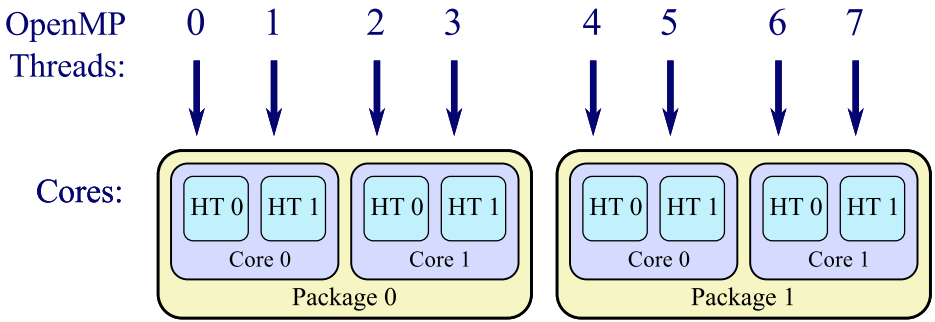
`KMP_AFFINITY=scatter,granularity=fine`



THREAD AFFINITY: COMPACT PATTERN

Generally beneficial for compute-bound applications.

```
OMP_NUM_THREADS={2(4) threads/core on Xeon (Xeon Phi)}
KMP_AFFINITY=compact,granularity=fine
```



PARALLELISM AND AFFINITY INTERFACES

Intel-specific (in order of priority):

- Functions (e.g., `kmp_set_affinity()`)
- Compiler arguments (e.g., `-par-affinity`)
- Environment variables (e.g., `KMP_AFFINITY`)

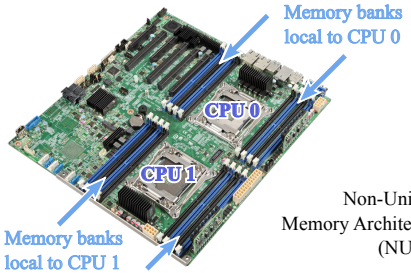
Defined by the OpenMP standard (in order of priority):

- Clauses in pragmas (e.g., `proc_bind`)
- Functions (e.g., `omp_set_num_threads()`)
- Environment variables (e.g., `OMP_PROC_BIND`)

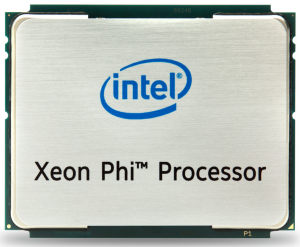
NUMA LOCALITY

NUMA ARCHITECTURES

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



Non-Uniform
Memory Architecture
(NUMA)



Examples:

- Multi-socket Intel Xeon processors
- Second generation Intel Xeon Phi in sub-NUMA clustering mode

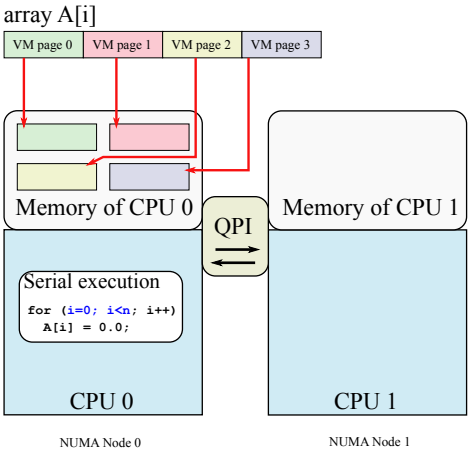
ALLOCATION ON FIRST TOUCH

- 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

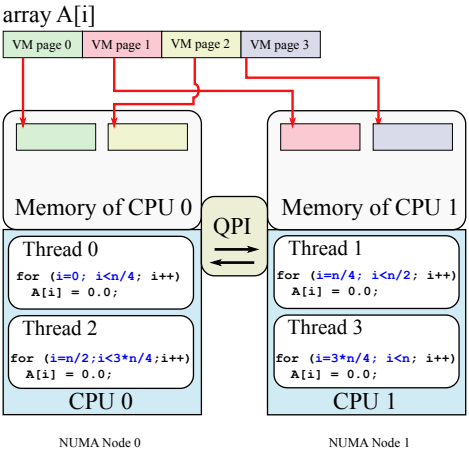
```
1  float* A = (float*)_mm_malloc(n*m*sizeof(float), 64);
2
3  // Initializing from parallel region for better performance
4  #pragma omp parallel for
5  for (int i = 0; i < n; i++)
6    for (int j = 0; j < m; j++)
7      A[i*m + j] = 0.0f;
```

FIRST-TOUCH ALLOCATION POLICY

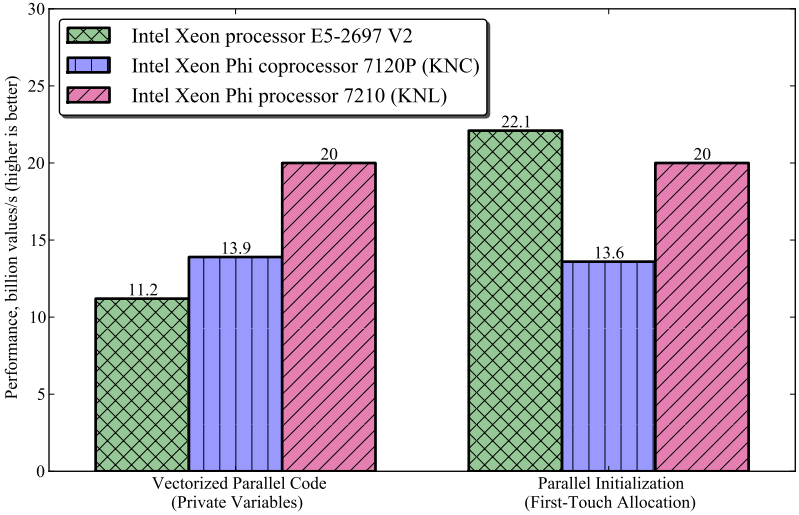
Poor First-Touch Allocation



Good First-Touch Allocation



IMPACT OF FIRST-TOUCH ALLOCATION



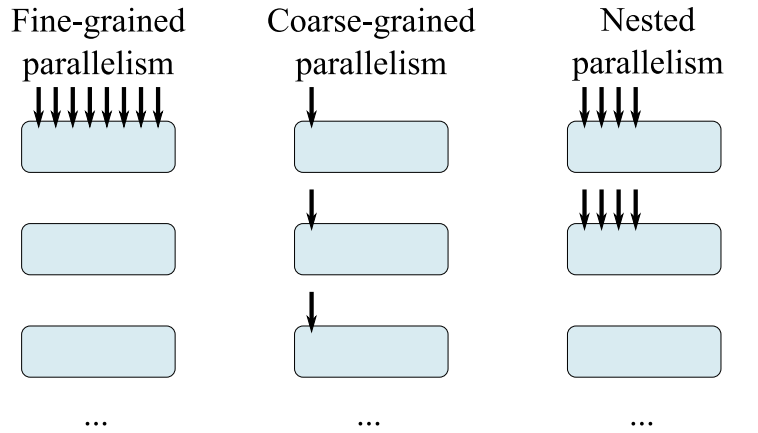
BINDING TO NUMA NODES WITH `numactl`

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

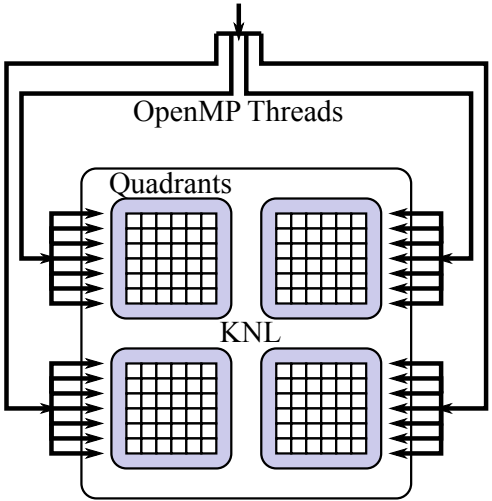
```
vega@lyra% numactl --hardware
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 12 13 14 15 16 17
node 0 size: 65457 MB
node 0 free: 24426 MB
node 1 cpus: 6 7 8 9 10 11 18 19 20 21 22 23
node 1 size: 65536 MB
node 1 free: 53725 MB
node distances:
node   0   1
  0:  10  21
  1:  21  10
vega@lyra%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

NESTED PARALLELISM

MOTIVATION FOR 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

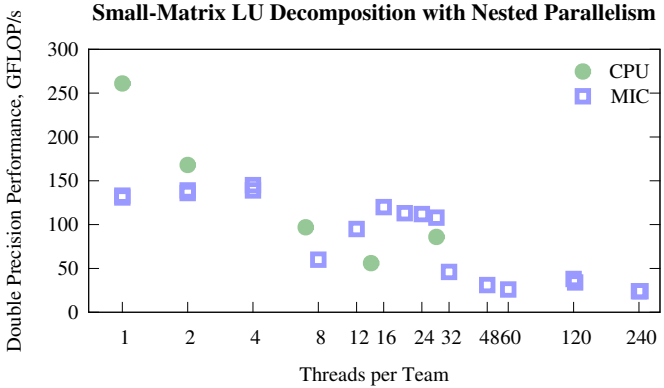
Xeon	Xeon Phi
▷ `OMP_NUM_THREADS=2,14`	▷ `OMP_NUM_THREADS=60,4`

▷ `OMP_NESTED=1`
`OMP_PROC_BIND=spread,close`
`OMP_PLACES=cores`
	▷ `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`
	▷ `KMP_HOT_TEAMS_MODE=1`
	`KMP_HOT_TEAMS_MAX_LEVEL=2`
	`OMP_MAX_ACTIVE_LEVELS=2`

SMALL MATRIX LU DECOMPOSITION WITH NESTED PARALLELISM

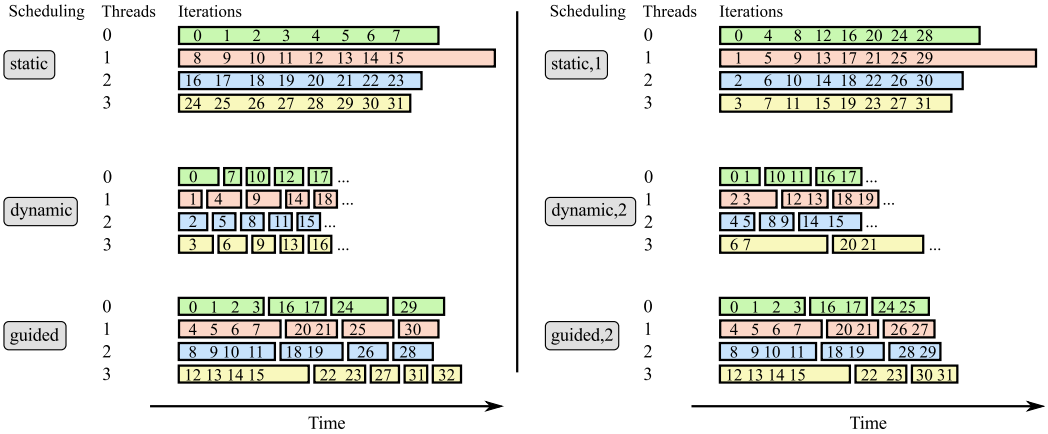
Decomposing 10^3 small square matrices of size 1024×1024.



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   // ...
```

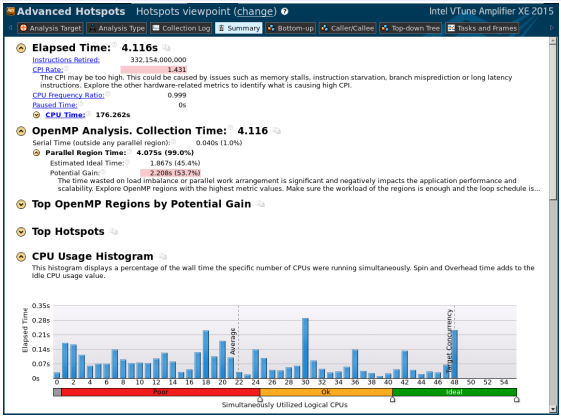
```
vega@lyra% export OMP_SCHEDULE=dynamic,4
vega@lyra% ./run-my-app
```

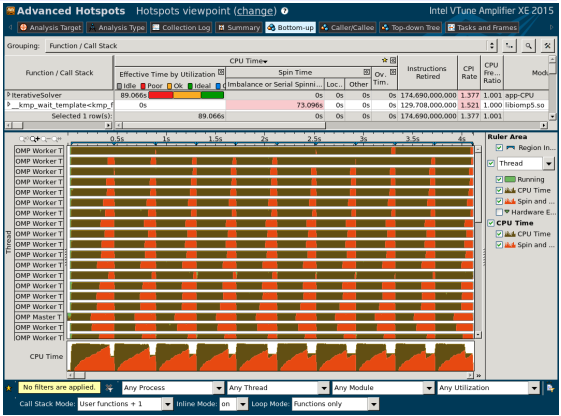
ITERATIVE JACOBI SOLVER

```
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

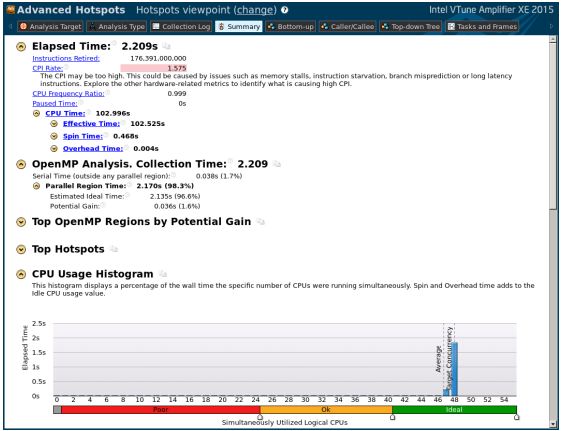
```
1  #pragma omp parallel for
2      for (int c = 0; c < nBVectors; c++)
3          IterativeSolver(n, M, &b[c*n], &x[c*n], accuracy[c]);
```

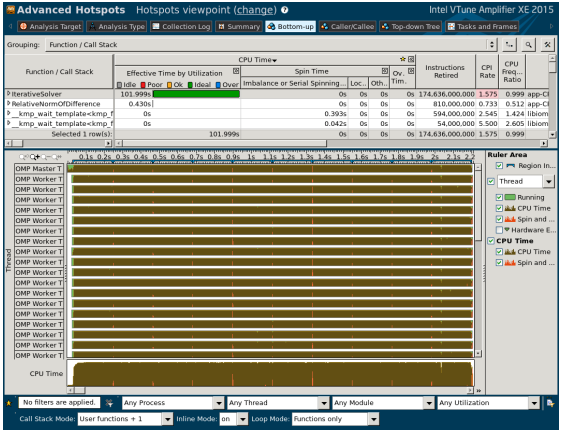




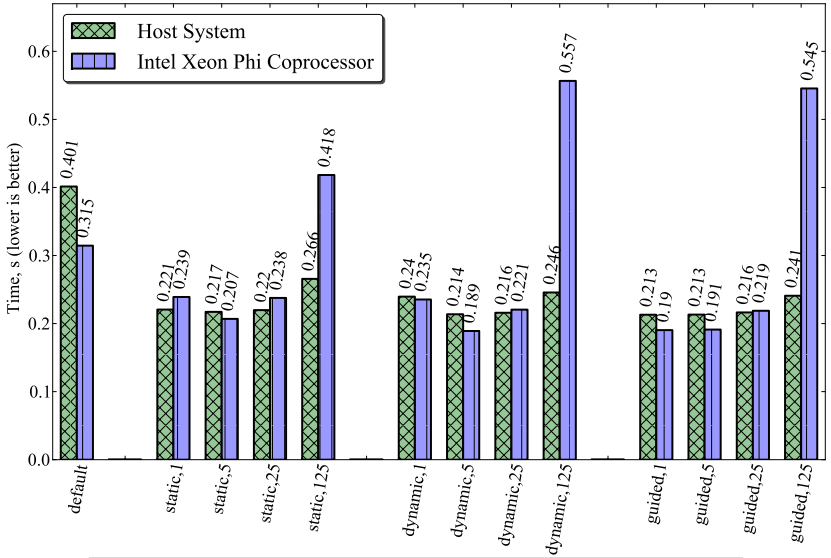
AN ITERATIVE JACOBI SOLVER WITH DYNAMIC SCHEDULING

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

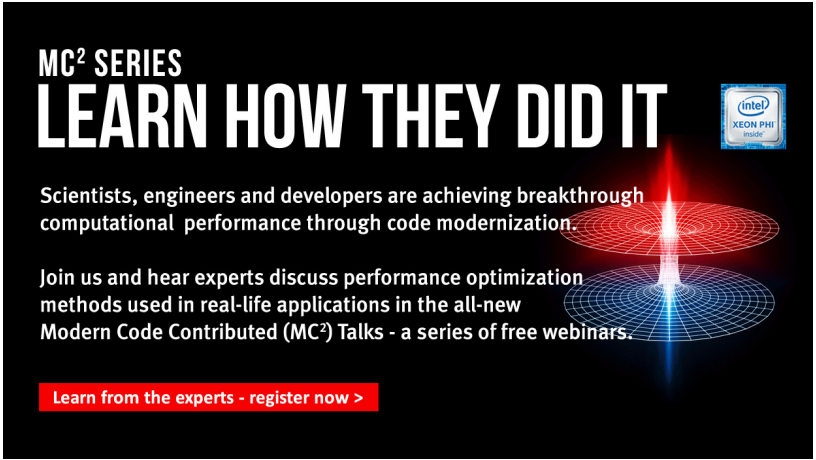
SUMMARY

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 + control NUMA locality

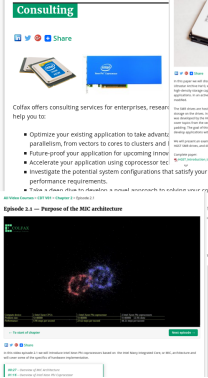
Next session: optimization of memory traffic.

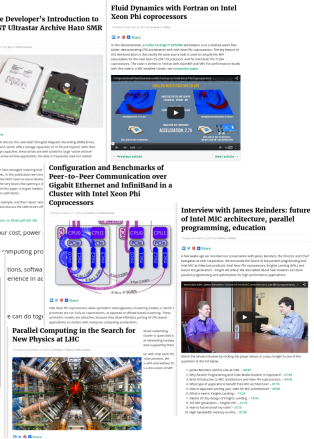
NEW PROJECT: MC^2 SERIES



mc2series.com

COLFAX RESEARCH





http://colfaxresearch.com/