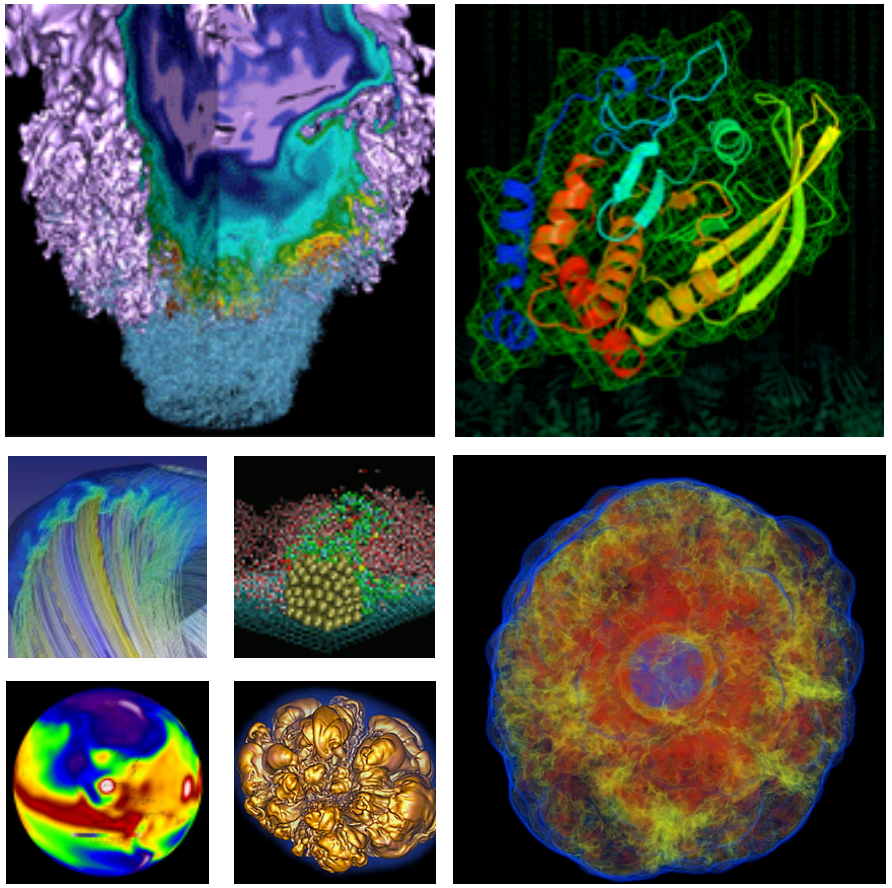
Process and Thread Affinity with MPI/OpenMP on KNL



Yun (Helen) He
NERSC/LBNL

IXPUG 2016 Annual Meeting
Sept 19-22, 2016

NERSC 40 YEARS at the FOREFRONT 1974-2014

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

Affinity Goal

- Correct process and thread affinity for hybrid MPI/OpenMP programs is the basis for getting optimal performance on KNL. It is also essential for guiding further performance optimizations.
- Our goal is to promote OpenMP4 standard settings for portability. For example, OMP_PROC_BIND and OMP_PLACES are preferred to Intel specific KMP_AFFINITY and KMP_PLACE_THREADS settings.
- Besides CPU binding, there are also memory affinity concerns.

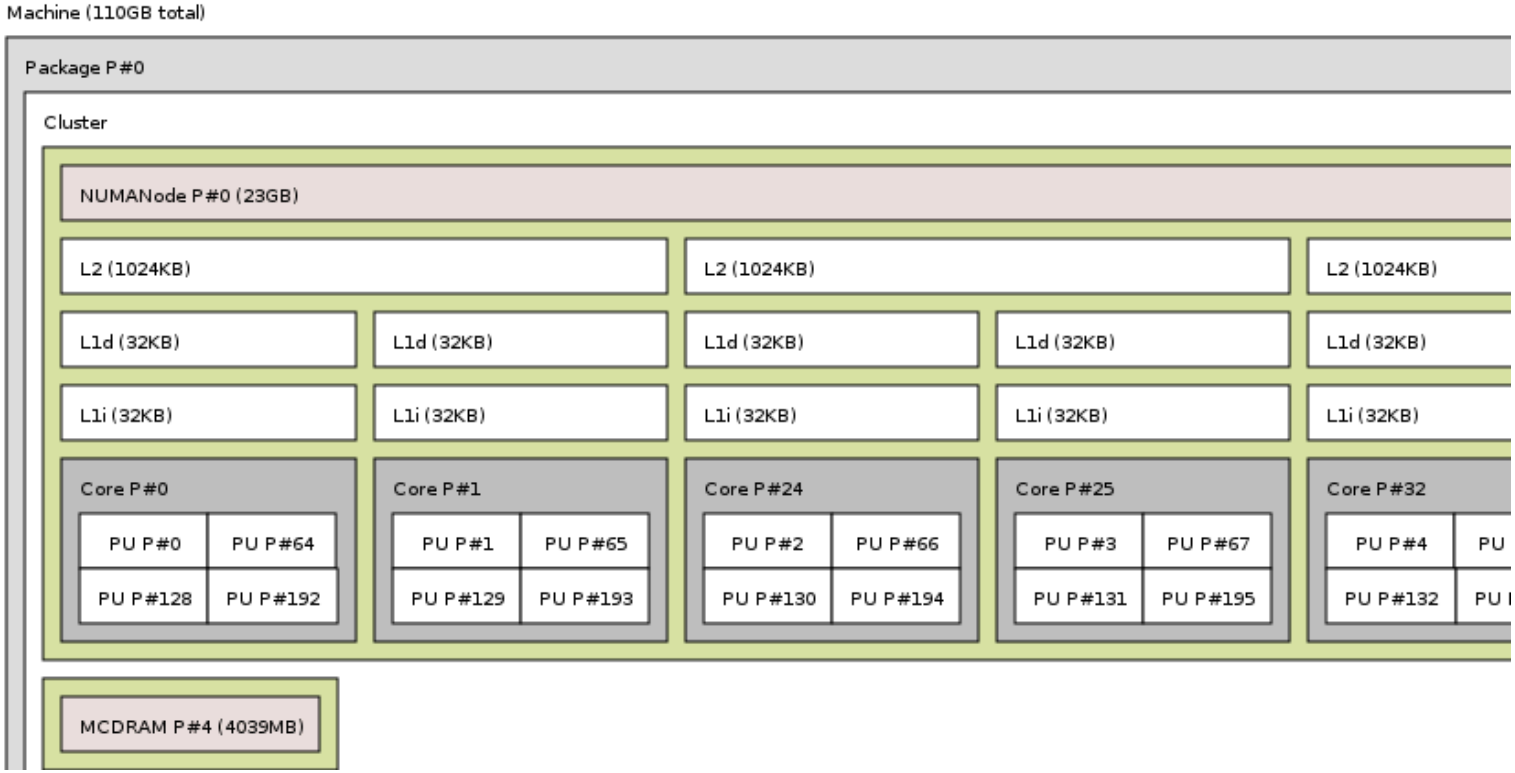
KNL Features Affecting Affinity

- **Cluster modes**
  - No SNC: All2All, Hemisphere, Quadrant
  - Sub NUMA domains: SNC4, SNC2
- **Memory modes**
  - No extra NUMA domains: Cache
  - Extra NUMA domains: Flat, Hybrid
- **Number of cores per node**
  - 64 cores
  - 68 cores

2 Physical Cores Share a Tile on KNL

In hybrid MPI/OpenMP, we need to avoid multiple MPI tasks sharing a tile.

% hwloc-ls



“numactl -H” Displays NUMA Domain Info

68-core SNC4 Cache node:

NUMA Domains 0 & 1: 18 physical cores each.

NUMA Domains 2 & 3: 16 physical cores each.

```
yunhe@nid00046:~> numactl -H
available: 4 nodes (0-3)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 136 137 138 139 140 141
142 143 144 145 146 147 148 149 150 151 152 153 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221
node 0 size: 24023 MB
node 0 free: 21803 MB
node 1 cpus: 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 154 155
156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236
237 238 239
node 1 size: 24232 MB
node 1 free: 23669 MB
node 2 cpus: 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 172
173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255
node 2 size: 24233 MB
node 2 free: 23925 MB
node 3 cpus: 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 188
189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271
node 3 size: 24230 MB
node 3 free: 23951 MB
node distances:
node   0   1   2   3
  0:  10  21  21  21
  1:  21  10  21  21
  2:  21  21  10  21
  3:  21  21  21  10
```

“numactl -H” Displays NUMA Domain Info (2)

68-core Quad Flat node:

NUMA Domain 0: all 68 cores (272 logic cores)

NUMA Domain 1: HBM with no CPUs

```
yunhe@nid00034:~> numactl -H
available: 2 nodes (0-1)
node 0 cpus: 0 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 32 33 34 35 36 37 38 39 40 41 42 43
44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87
88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122
123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153
154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184
185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215
216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246
247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271
node 0 size: 96723 MB
node 0 free: 93924 MB
node 1 cpus:
node 1 size: 16157 MB
node 1 free: 16088 MB
node distances:
node   0   1
  0:  10  31
  1:  31  10
```

Example Commands: Is this Correct?

For 16 MPI tasks x 8 OpenMP threads per task on a single 68-core KNL node:

```
% export OMP_NUM_THREADS=8
% export OMP_PROC_BIND=spread    (other choiceare “close”,”master”,”true”,”false”)
% export OMP_PLACES=threads      (other choices are: cores, sockets, and various ways to specify explicit lists, etc.)

% mpirun -n 16  ./xthi |sort -k4n,6n
Hello from rank 0, thread 0, on cc10. (core affinity = 0)
Hello from rank 0, thread 1, on cc10. (core affinity = 204)
Hello from rank 0, thread 2, on cc10. (core affinity = 69)
Hello from rank 0, thread 3, on cc10. (core affinity = 205)
Hello from rank 0, thread 4, on cc10. (core affinity = 70)
Hello from rank 0, thread 5, on cc10. (core affinity = 206)
Hello from rank 0, thread 6, on cc10. (core affinity = 71)
Hello from rank 0, thread 7, on cc10. (core affinity = 207)
Hello from rank 1, thread 0, on cc10. (core affinity = 72)
Hello from rank 1, thread 1, on cc10. (core affinity = 5)
```

68-core Quad Flat: Initial Try, OMG

Multiple MPI tasks are sharing cores within a same tile. (Marked first 6 MPI tasks only)

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**				
68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83				
136					141	142	143	144					149	150	151				
204	205	206	207						213	214	215	216			219				
16	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**				
84		86	87	88	89										99				
152						158	159	160							167				
220	221	222	223	224											235				
32	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**				
100															115				
168															183				
236															251				
48	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**	**61**	**62**	**63**	**64**	**65**	**66**	**67**
116															131				135
184															199				203
252															267				272

And so on for other MPI tasks and threads ….

Example Commands: How to Fix the Problem?

The reason is #MPI tasks is not divisible by 68!

- Each MPI task is getting 68x4/#MPI tasks of logical cores as the domain size
- MPI tasks are crossing tile boundaries
- Let's set number of logical cores per MPI task manually by wasting extra 4 cores on purpose, which is 256/#MPI tasks.

- **Intel MPI with Omni Path using mpirun:**

  % export I_MPI_PIN_DOMAIN=16

  Note: No need to set explicitly if #MPI is divisible by 68 or for 64-core KNL nodes when default setting of "auto:compact" is OK.

  % mpirun -n 16 ./app

- **Cray MPICH with Aries network using native SLURM:**

  % srun -n 16 -c 16 --cpu_bind=threads ./app

  Notes: Here the value for -c is also set to number of logical cores, which is 256/#MPI tasks. This is needed for both 64-core or 68-core KNL nodes.

68-core Quad Flat: with "I_MPI_PIN_DOMAIN=*num_logical_cores*"

Now the process/thread affinity are good! (Marked first 6 and last MPI tasks only)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15				
68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83				
136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151				
204															219				
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31				
84															99				
152	153	154	155	156	157	158	159								167				
220															235				
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47				
100															115				
168															183				
236															251				
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67
116															131				135
184												196	197	198	199				203
252															267				272

And so on for other MPI tasks and threads ….

68-core SNC4 Cache: with "-c *num_logical_cores*" for srun

Process/thread affinity are good! (Marked first 6 and last MPI tasks only)

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85
136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153
204																220	221
18	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**
86																102	103
154	155	156	157	158	159	160	161									170	171
222																238	239
36	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**	**48**	**49**	**50**	**51**		
104																	
172																	
240																	
52	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**	**61**	**62**	**63**	**64**	**65**	**66**	**67**		
120															135		
188												200	201	202	203		
256															271		

And so on for other MPI tasks and threads ….

An Alternative from TACC

- **To exclude two extra codes each in NUMA domains 0 and 1 for SNC4:**

  % export I_MPI_PIN_PROCESSOR_EXCLUDE_LIST=16,17,34,35,84,85,102,103,152,153,170,171,220,221,238,239

  Can achieve same affinity as in the previous slide.

- **A similar list can be used to exclude explicit cores in Quad mode (say two each from first two quadrants) instead of not using last 4 physical cores with the I_MPI_PIN_DOMAIN setting. Always make sure to exclude cores in pairs (on same tile).**

Memory Binding

For Flat or Hybrid Memory Modes.

- **"numactl" works for both mpirun or srun.**
  - Quad example:

    % mpirun -n 16 numactl -m 1 ./app

    % srun -n 16 -c 16 --cpu_bind=cores numactl -m 1 ./app
  - SNC4 example:

    % mpirun -n 4 numactl -m 4 ./app: -n 4 numactl -m 5 ./app : -n 4 numactl -m 6 ./app: -n 4 numactl -m 7 ./app
  - "-m" flag forces binding to HBM. Good if application memory footprint fits in HBM. If not, use "--preferred" instead.
- **SLURM also has a --mem_bind option (force binding). No "preferred" binding option in SLURM.**
  - Quad example:

    % srun -n 16 -c 16 --cpu_bind=cores --mem_bind=mem_map:1 ./app
  - SNC4 example:

    % srun -n 16 -c 16 --cpu_bind=cores --mem_bind=mem_map:4,4,4,4,5,5,5,5,6,6,6,6,7,7,7,7 ./app

Nested OpenMP Thread Affinity

- Again, I_MPI_PIN_DOMAIN (or I_MPI_PIN_PROCESSOR_EXCLUDE_LIST) is important for 68-core KNL nodes.
- Sample settings for 2 MPI tasks, 4 outer OpenMP threads, and 4 inner OpenMP threads:

```
% export OMP_NUM_THREADS=4,4
% export OMP_PROC_BIND=spread,close
% export OMP_PLACES=threads
% export OMP_NESTED=true
% export I_MPI_PIN_DOMAIN=128        # 32 physical cores
% export KMP_HOT_TEAMS=1
% export KMP_HOT_TEAMS_MAX_LEVELS=2
```

- Tip: Use “num_threads” clause in source codes to set threads for nested regions. For most other non-nested regions, use OMP_NUM_THREADS environment variable for simplicity and flexibility.

A Real Application Gotcha

- **Discovered in an Intel Dungeon session with an application that OMP settings ran slower than KMP settings.**
- **What can be the cause? Investigation started with 7 example cases:**
  - Case 1: mpirun -n 32 -env OMP_NUM_THREADS 2 -env KMP_AFFINITY compact,verbose -env KMP_PLACE_THREADS 1T numactl -m 1 ./app
  - Case 2: mpirun -n 32 -env KMP_AFFINITY compact,verbose -env KMP_PLACE_THREADS 2C, 1T numactl -m 1 ./app
  - Case 3: mpirun -n 32 -env OMP_NUM_THREADS 2 -env KMP_AFFINITY scatter,verbose numactl -m 1 ./app
  - Case 4: mpirun -n 32 -env OMP_NUM_THREADS 2 -env OMP_PROC_BIND spread -env OMP_PLACES threads numactl -m 1 ./app
  - Case 5: mpirun -n 32 -env OMP_NUM_THREADS 2 -env OMP_PROC_BIND close -env OMP_PLACES cores numactl -m 1 ./app
  - Case 6: mpirun -n 32 -env KMP_AFFINITY scatter,verbose -env KMP_PLACE_THREADS=2C,1T numactl -m 1 ../app
  - Case 7: mpirun -n 32 -env KMP_AFFINITY scatter,verbose -env KMP_PLACE_THREADS=2C,1T -env OMP_NUM_THREADS 2 numactl -m 1 ./app

Affinity Analysis

- Expect to see same performance from all 7 cases on a quad flat node
- Confirmed with another application this is the case
- Our goal is to promote OpenMP4 standard settings for portability
- Confirmed with my simple affinity test code that core bindings are all equivalent
- However, initial runs see different results from the 7 test cases. Some cases are >2X slower. Also quad cache performance is about 5% slower.
- Further investigations showed even though asking for 2 threads only, the code is running with 4. It was discovered later nested OpenMP is set in the code by default!
- Using the modified code with explicit num_threads clauses specified for nested OpenMP regions, all 7 cases then performed the same on quad flat and quad cache nodes.
- Good nested OpenMP scaling was also achieved after code bug was fixed

Thread Affinity Verification Methods

- **I routinely use simple codes to display process thread affinity**
  - xthi.c (from Cray) and xthi-nested.c (modified by me based on xthi.c)

```
% mpirun -n 8 -env OMP_PROC_BIND spread -env OMP_PLACES threads -env
OMP_NUM_THREADS 4 -env I_MPI_PIN_DOMAIN 32 ./xthi |sort -k4n,6n
 Hello from rank 0, thread 0, on ekm118. (core affinity = 0)
 Hello from rank 0, thread 1, on ekm118. (core affinity = 2)
 Hello from rank 0, thread 2, on ekm118. (core affinity = 4) …
```

- **Intel compiler has a run time environment variable KMP_AFFINITY, when set to "verbose”:**

```
OMP: Info #242: KMP_AFFINITY: pid 255705 thread 0 bound to OS proc set {55}
OMP: Info #242: KMP_AFFINITY: pid 255660 thread 1 bound to OS proc set {10,78}
OMP: Info #242: OMP_PROC_BIND: pid 255660 thread 1 bound to OS proc set {78} …
```

- **Cray compiler has a similar env CRAY_OMP_CHECK_AFFINITY, when set to "TRUE”:**

```
[CCE OMP: host=nid00033 pid=14506 tid=17606 id=1] thread 1 affinity:  90
[CCE OMP: host=nid00033 pid=14510 tid=17597 id=1] thread 1 affinity:  94  …
```

- **It is highly desirable that OpenMP standard can provide an equivalent run time env to display affinity binding info. I have initiated the discussion within the OpenMP Standard Committee.**

Summary

- Correct process and thread affinity are critical for getting optimal performance.
- Promote OpenMP standard settings for settings over specific compiler settings for portability.
- Various KNL cluster and memory modes, and number of cores per node being 68 make it harder to achieve this.
- Setting number of logical cores per MPI task is essential if #MPI tasks is not divisible by 68.
- xthi.c and xthi-nested.c codes are available upon request to check run time affinity.
- Requested to OpenMP Standard to consider adding run time environment (and/or scripts if needed) to provide process and thread affinity info.

Acknowledgement

- NCAR: John Dennis, Christopher Kerr
- Intel: Karthik Raman, Larry Meadows, Richard Mills
- NERSC: Doug Jacobsen, Zhengji Zhao, Doug Doerfler
- SchedMD: Moe Jette
- TACC: John Cazes, Lars Koesterke

Thank you.