



HARDWARE AND SINGLE NODE / CORE OPTIMIZATION

Carlos Rosales-Fernandez

2019 ALCF Computational Performance Workshop

Legal Disclaimer & Optimization Notice

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Copyright © 2019, Intel Corporation. All rights reserved. Intel, the Intel logo, Pentium, Xeon, Core, VTune, OpenVINO, Cilk, are trademarks of Intel Corporation or its subsidiaries in the U.S. and other countries.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

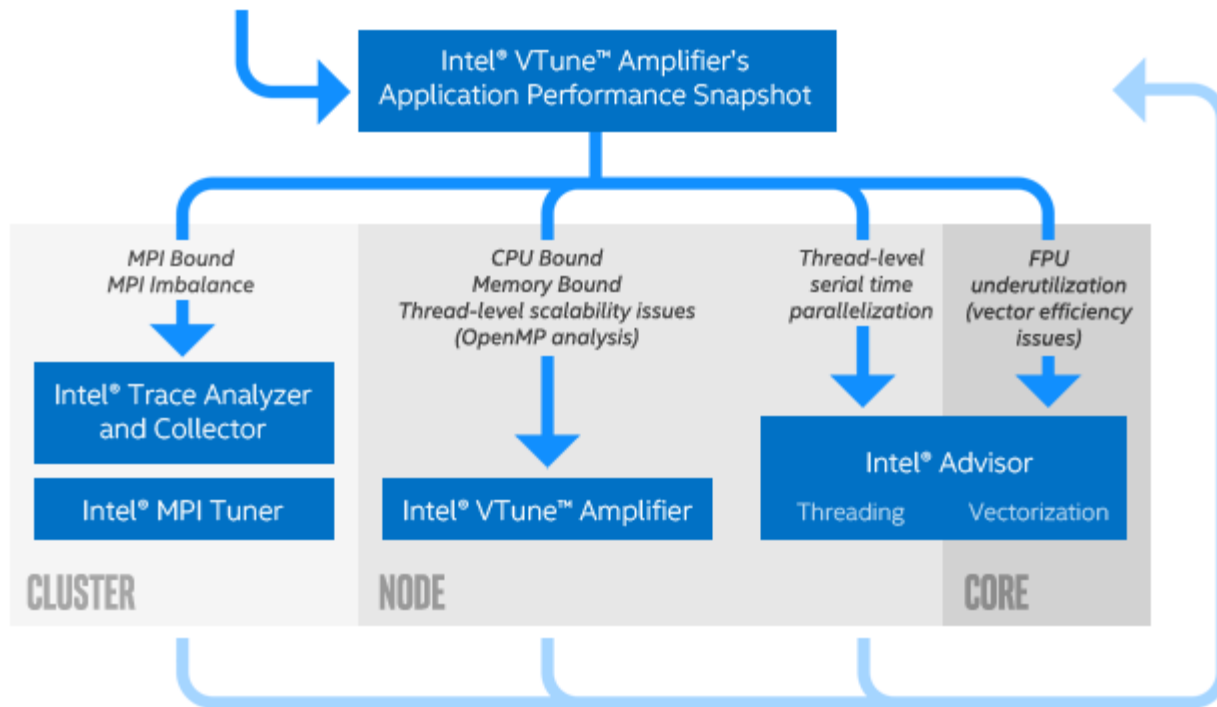
Notice revision #20110804

Tuning at Multiple Hardware Levels

Exploiting all features of modern processors requires good use of the available resources

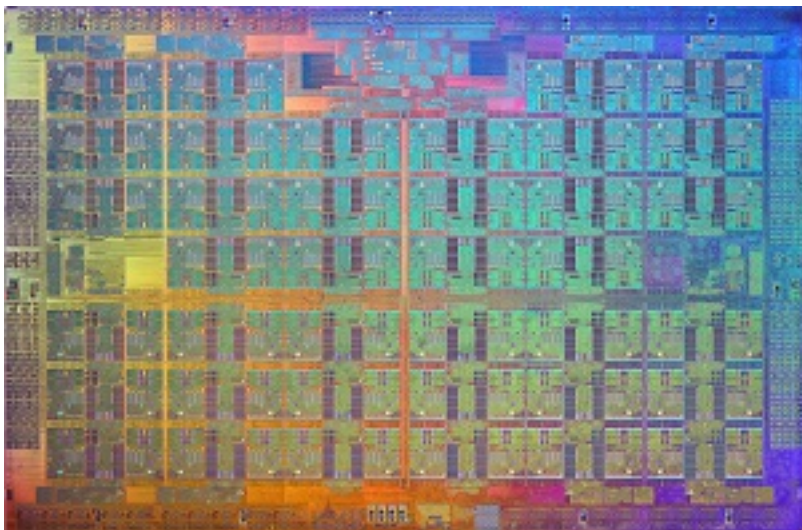
- Core
 - Vectorization is critical with 512bit FMA vector units (32 DP ops/cycle)
 - Targeting the current ISA is fundamental to fully exploit vectorization
- Socket
 - Using all cores in a processor requires parallelization (MPI, OMP, ...)
 - Up to 64 Physical cores and 256 logical processors per socket on Theta!
- Node
 - Minimize remote memory access (control memory affinity)
 - Minimize resource sharing (tune local memory access, disk IO and network traffic)

Tuning Workflow



CPU BASICS

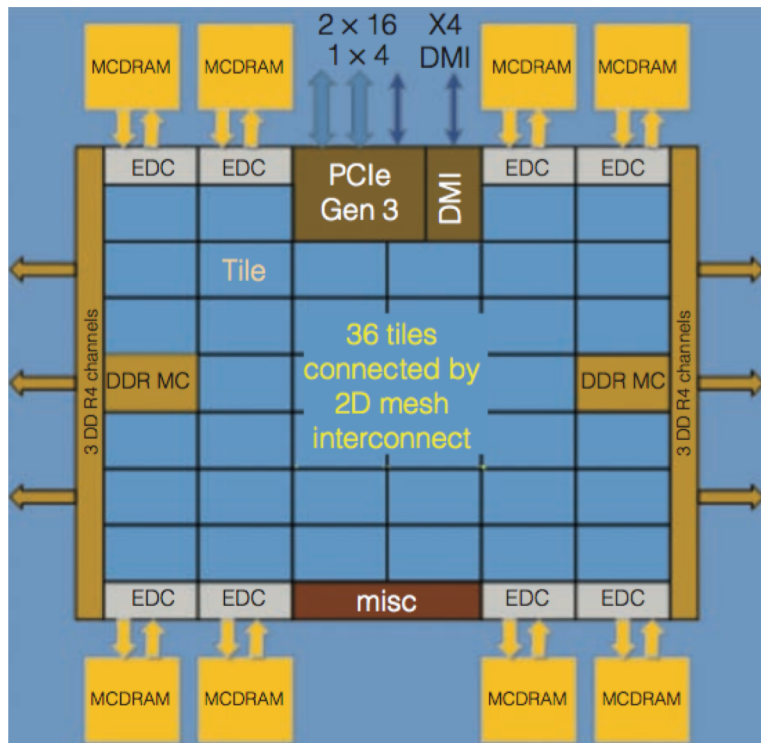
Intel® Xeon Phi™ Codenamed Knights Landing (KNL)



Theta uses Intel® Xeon Phi™ 7230

- 64 cores
 - 4 threads per core
 - Possible 256 threads of execution
- 16 GB MCDRAM
- 1 socket – self hosted (no PCI bottleneck!)
- 3+ TF DP peak performance
- 6+ TF SP peak performance
- 400+ GB/s STREAM performance

Intel® Xeon Phi™ Socket Diagram



Up to 36 tiles (32 in Theta)

- 2 cores/tile

2D mesh interconnect

2 DDR memory controllers

6 channels DDR4

- Up to 90 GB/s

16 GB MCDRAM

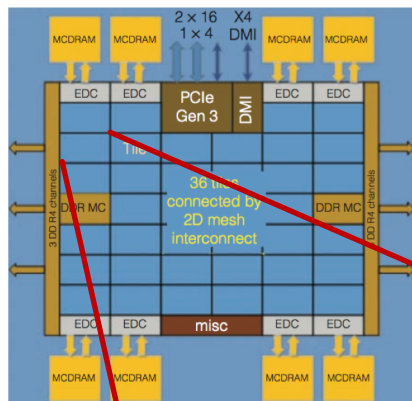
- 8 Embedded DRAM Controllers
- Up to 450 GB/s

Optimization Notice

Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Figure from : (KNIGHTS LANDING: SECOND- GENERATION INTEL XEON PHI PRODUCT ,
A. Sodani, et.al.,IEEE Micro March/April 2016)

Tile Details



- 2 cores/tile
- 2x 512-bit VPUs per core
- 1 MB shared 16-way L2 cache
- Shared 2D mesh connections handled by CHA
- CHA: Caching/Home Agent.
- Distributed Tag Directory to keep L2s coherent.
- MESIF protocol [Modified (M), Exclusive (E), Shared (S), Invalid (I) and Forward (F)]

Optimization Notice

Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Figure from : (KNIGHTS LANDING: SECOND- GENERATION INTEL XEON PHI PRODUCT ,
A. Sodani, et.al.,IEEE Micro March/April 2016)

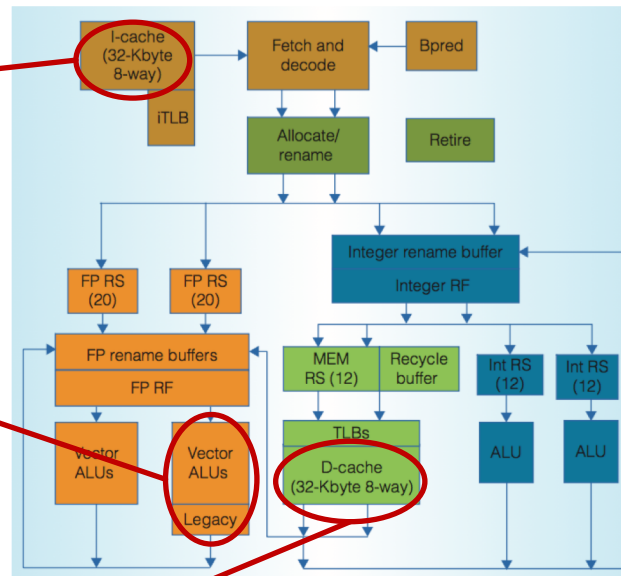


Core Details

8-way 32KB instruction cache

Although there are 2 VPU, only one has support for legacy floating point ops

- Must compile with AVX512 to use both VPUs



8-way 32KB data cache

Intel® Advanced Vector Extensions 512 (Intel® AVX-512)

- 512-bit wide vectors
- 32 operand registers
- 8 64b mask registers
- Embedded broadcast
- 7 Subsets

- 2 exclusive to Intel® Xeon Phi™
- 2 common
- 3 exclusive to Intel® Xeon®

Microarchitecture	Instruction Set	SP FLOPs per cycle	DP FLOPs per cycle
Skylake / KNL	AVX512 & FMA	64	32
Haswell / Broadwell	AVX2 & FMA	32	16
Sandybridge	AVX (256b)	16	8
Nehalem	SSE (128b)	8	4

Intel® AVX-512 Instruction Types	
AVX512-PF	Prefetch: multi-address prefetch using gather/scatter semantics
AVX512-ER	Exponential and Reciprocal: 'wide' approximation of Log and RCP/RSQRT
AVX512-F	AVX-512 Foundation Instructions
AVX512-CD	Conflict Detect : used in vectorizing loops with potential address conflicts
AVX512-VL	Vector Length Orthogonality : ability to operate on sub-512 vector sizes
AVX512-BW	512-bit Byte/Word support
AVX512-DQ	Additional D/Q/SP/DP instructions (converts, transcendental support, etc.)

Vectorization on KNL

Vectorized AVX512 code is much faster than legacy scalar code:

- ~16x in double precision
- ~32x in single precision

This is because legacy code (not using the AVX512 ISA) can only use one of the two vector units – a factor 2x – you miss on FMA – another factor 2x - and the vector length is 512 bits

So let's take this seriously and do some basic work to ensure we do not underuse the hardware

- Intel® Advisor is the best tool for detailed vectorization analysis
- But you can do a lot just by looking at compiler reports before you run

Nested Loop Example

```
#pragma omp parallel for private(j,k)
for( i = 1; i <= NX; i++ ){
    for( j = 1; j <= NY; j++ ){
        #pragma ivdep
        for( k = 1; k <= NZ; k++ ){
            phi[i][j][k] = f0[now][i][j][k] + f1[now][i][j][k] + f2[now][i][j][k]
                + f3[now][i][j][k] + f4[now][i][j][k] + f5[now][i][j][k]
                + f6[now][i][j][k];
        }
    }
}
```

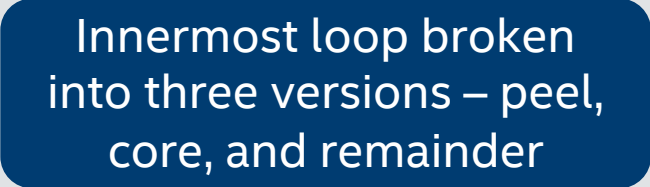
Simple example with traditional setup

- Outer loop parallelized with OpenMP threads
- Inner loop vectorized

Compiler Reports on Vectorization

Compiled with options `-qopt-report-phase=vec -qopt-report=5`

```
LOOP BEGIN at hydro.cpp(32,2)
remark #15542: loop was not vectorized: inner loop was already vectorized
  LOOP BEGIN at hydro.cpp(33,3)
    remark #15542: loop was not vectorized: inner loop was already vectorized
      LOOP BEGIN at hydro.cpp(37,4)
        <Peeled loop for vectorization>
        remark #15301: PEEL LOOP WAS VECTORIZED
      LOOP END
      LOOP BEGIN at hydro.cpp(37,4)
        remark #15300: LOOP WAS VECTORIZED
      LOOP END
      LOOP BEGIN at hydro.cpp(37,4)
        <Remainder loop for vectorization>
        remark #15301: REMAINDER LOOP WAS VECTORIZED
      LOOP END
    LOOP END
  LOOP END
LOOP END
```



Innermost loop broken into three versions – peel, core, and remainder

A Better Look at the Vectorized Loop

```
LOOP BEGIN at hydro.cpp(37,4)
  remark #15388: vec support: reference XX has aligned access [hydro.cpp(38,5)]
  remark #15389: vec support: reference YY has unaligned access [hydro.cpp(39,67)]
  remark #15381: vec support: unaligned access used inside loop body
  remark #15305: vec support: vector length 8
  remark #15399: vec support: unroll factor set to 2
  remark #15309: vec support: normalized vectorization overhead 0.707
  remark #15300: LOOP WAS VECTORIZED
  remark #15449: unmasked aligned unit stride stores: 1
  remark #15450: unmasked unaligned unit stride loads: 7
  remark #15475: --- begin vector cost summary ---
  remark #15476: scalar cost: 23
  remark #15477: vector cost: 3.620
  remark #15478: estimated potential speedup: 5.740
  remark #15488: --- end vector cost summary ---
LOOP END
```

Lots of information
on memory access
and vectorization
efficiency

Vectorization Reports : Alignment

```
LOOP BEGIN at hydro.cpp(37,4)
```

```
remark #15388: vec support: reference XX has aligned access [hydro.cpp(38,5)]
remark #15389: vec support: reference YY has unaligned access [hydro.cpp(39,67)]
remark #15381: vec support: unaligned access used inside loop body
remark #15305: vec support: vector length 8
remark #15399: vec support: unroll factor set to 2
remark #15309: vec support: normalized vectorization overhead 0.707
remark #15300: LOOP WAS VECTORIZED
remark #15449: unmasked aligned unit stride stores: 1
remark #15450: unmasked unaligned unit stride loads: 7
remark #15475: --- begin vector cost summary ---
remark #15476: scalar cost: 23
remark #15477: vector cost: 3.620
remark #15478: estimated potential speedup: 5.740
remark #15488: --- end vector cost summary ---
```

```
LOOP END
```

Loop has unaligned access, which likely reduce efficiency

Vectorization Reports: Vectorization Details

LOOP BEGIN at hydro.cpp(37,4)

```
remark #15388: vec support: reference XX has aligned access [hydro.cpp(38,5)]
remark #15389: vec support: reference YY has unaligned access [hydro.cpp(39,67)]
remark #15381: vec support: unaligned access used inside loop body
remark #15305: vec support: vector length 8
remark #15399: vec support: unroll factor set to 2
remark #15309: vec support: normalized vectorization overhead 0.707
remark #15300: LOOP WAS VECTORIZED
remark #15449: unmasked aligned unit stride stores: 1
remark #15450: unmasked unaligned unit stride loads: 7
remark #15475: --- begin vector cost summary ---
remark #15476: scalar cost: 23
remark #15477: vector cost: 3.620
remark #15478: estimated potential speedup: 5.740
remark #15488: --- end vector cost summary ---
```

LOOP END

Expected vector length
(512bit / 64bit = 8)

Overhead is normalized
by vector iteration cost

Vectorization Reports: Cost Estimation

LOOP BEGIN at hydro.cpp(37,4)

```
remark #15388: vec support: reference XX has aligned access [hydro.cpp(38,5)]
remark #15389: vec support: reference YY has unaligned access [hydro.cpp(39,67)]
remark #15381: vec support: unaligned access used inside loop body
remark #15305: vec support: vector length 8
remark #15399: vec support: unroll factor set to 2
remark #15309: vec support: normalized vectorization overhead 0.707
```

remark #15300: LOOP WAS VECTORIZED

```
remark #15449: unmasked aligned unit stride stores: 1
remark #15450: unmasked unaligned unit stride loads: 7
```

remark #15475: --- begin vector cost summary ---

remark #15476: scalar cost: 23

remark #15477: vector cost: 3.620

remark #15478: estimated potential speedup: 5.740

remark #15488: --- end vector cost summary ---

LOOP END

5.74x speedup
~ 71% efficiency

Threading on KNL

SMT with 4 threads per core

- Some resources are shared (Caches, TLB) and some dynamically partitioned (Rename Buffers)
- Single thread gets all resources, multiple threads a fraction
- Peak throughput can be achieved with a single thread per core, unlike in the previous generation processor with required two threads per core
- Not all applications will benefit from execution on all four threads per core
- Pure MPI possible, but remember each rank requires a minimum amount of memory and that can increase your memory footprint quickly

Intel® Advisor and Threading

Add annotations as shown on the left sample

Complex sites may be analyzed in more detail using task sections if needed

- ANNOTATE_SITE_BEGIN / ANNOTATE_SITE_END
- ANNOTATE_TASK_BEGIN / ANNOTATE_TASK_END

Recompile including annotation definitions:

```
-I/opt/intel/advisor/include
```

Collect suitability data

```
aprun -n 1 -N 1 advixe-cl --collect suitability --project-dir ./adv_res \  
--search-dir src:=./ --search-dir bin:=./ -- ./application
```

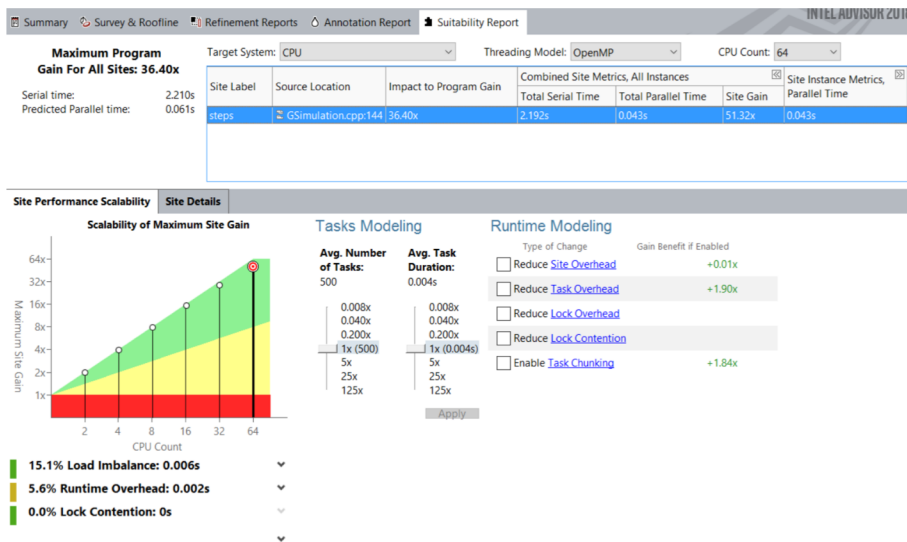
```
#include "advisor-annotate.h"  
...  
ANNOTATE_SITE_BEGIN(steps)  
for (int s=1; s<=get_nsteps(); ++s)  
{  
    ...  
    ANNOTATE_TASK_BEGIN(particles)  
    for (i = 0; i < n; i++)  
    {  
        ...  
    }  
    ANNOTATE_TASK_END(particles)  
}  
ANNOTATE_SITE_END(steps)
```

Suitability Report

Good speedup expected, but far from ideal (~56% efficiency).

Modeling shows that increasing the task length would improve efficiency.

Next step: add omp parallel region to code and re-test



Optimization Notice

THAT MEMORY BUSINESS

Tiles, Quadrants, and Directories

Memory hierarchy (configurable MCDRAM)

- DDR4 (192 GB) / MCDRAM (16 GB)
- Tile L2 (1 MB) / Core L1 (32 KB)

A **tile** is a set of two cores sharing a 1MB L2 cache and connectivity to the mesh.

A **quadrant** is a virtual concept and not a hardware property. It is a way to divide the tiles at a logical level.

A **tag directory** tracks cache line locations in all L2s. It provides the block of data or (if not available in L2) a memory address to the memory controller.

Memory Architecture

Two main memory types

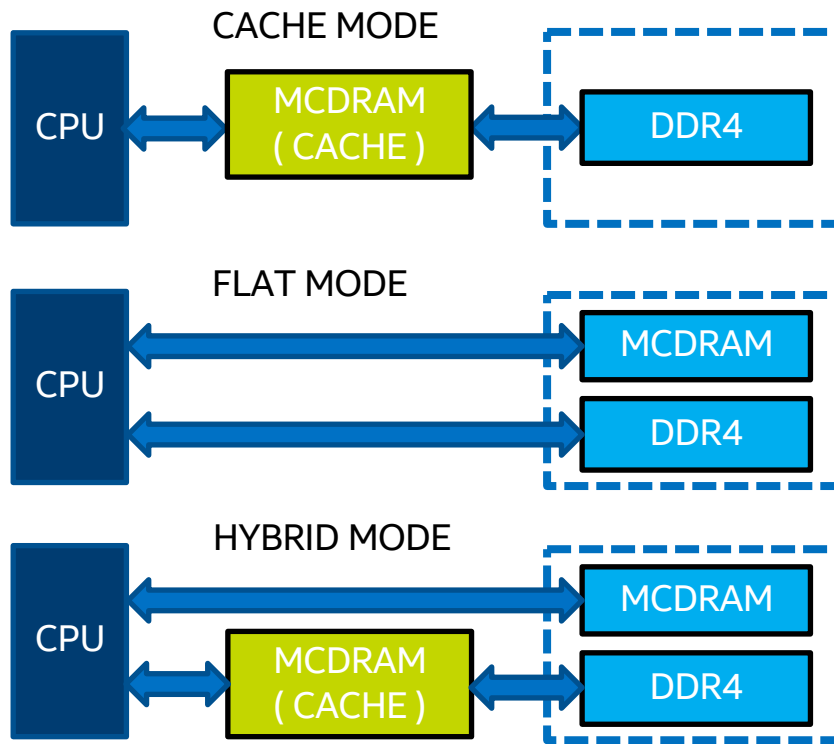
- DDR4
- MCDRAM

Three memory modes

- Cache
- Flat
- Hybrid

Hybrid mode

- Three choices
- 25% / 50% / 75%
- 4GB / 8 GB / 12GB



Memory Modes

Memory Modes determine overall memory layout

Cache

- MCDRAM acts as cache for transactions to DDR4 memory
- Direct-mapped, but large so conflicts usually not an issue
- Completely transparent to the user

Flat

- Flat address space
- Different NUMA nodes for DDR4 and MCDRAM
- Use numactl or memkind library to manage allocations

Hybrid

- Choice of 25% / 50 % / 75 % of MCDRAM set up as cache
- Potentially useful for some applications

Cluster Modes

Cluster Modes determine L2 coherency traffic flow

Three supported modes

- All-to-all (A2A)
- Quadrant (Quad)
- Sub-NUMA Cluster (SNC)

Cluster modes change memory latency by modifying the distance that coherency related traffic flows go through in the mesh

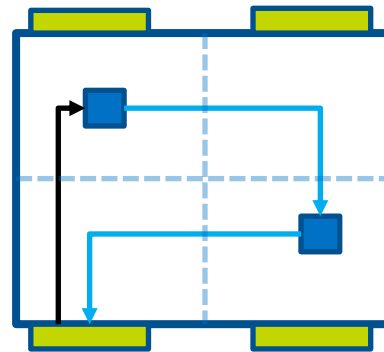
Remember these **quadrants** we mention are logical entities, not hardware features.

All-to-all Clustering

No affinity between tile, directory, and memory

An L2 miss may have to traverse the mesh in order to read the required 64-byte line from memory

Simple configuration, but potentially worst latency of all modes



Quadrant Clustering

Chip divided in 4 quadrants

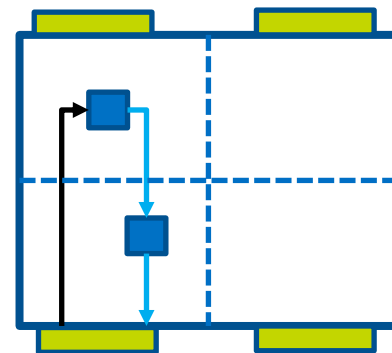
No affinity between tile and directory

- When there is an L2 miss in a tile the directory tag may be anywhere on the chip

Affinity between directory and memory

- Data associated to a directory tag will be in the same quadrant that the directory tag is located

Potentially reduces the length of the trip and makes memory latency lower than All-to-all



Sub-NUMA Clustering

Affinity between tile and directory

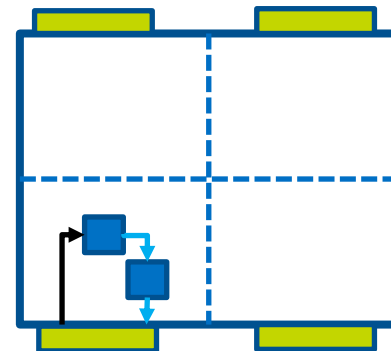
- When there is an L2 miss in a tile the directory tag will be in the same quadrant

Affinity between directory and memory

- Data associated to a directory tag will be in the same quadrant that the directory tag is located

Two options: SNC-2 / SNC-4

Potentially lowest memory latency for properly configured runs



numactl

- **DDR4 is the default memory**

- To see NUMA node layout:

```
$ numactl -H
```

- On **Cache** mode, MCDRAM acts as a L3 cache, transparent to numactl
- On **SNC2 / SNC4** modes, there are 2 / 4 MCDRAM NUMA nodes and 2 / 4 DDR4 NUMA nodes (DDR4 first)
- On the other Memory Modes, DDR is NUMA node 0 and MCDRAM is NUMA node 1

Cache (All-to-all / Quadrant)

Requesting cache-quadrant mode in Theta: **--queue debug-cache-quad / --attrs mcdram=cache:numa=quad**

Output of numactl -H shows single NUMA domain with 192 GB of memory available:

```
available: 1 nodes (0)
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
node 0 size: 193297 MB
```

Cache (SNC-4)

Requesting cache-snc4 mode in Theta: **--attrs mcdram=cache:numa=snc4**

Output of numactl -H shows four NUMA domains (0-3) in DDR4 (48 GB each):

```
available: 4 nodes (0-3)

node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 64 65 66 67 68 69 70 71 72 73 74 75 76
77 78 79 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 192 193 194 195
196 197 198 199 200 201 202 203 204 205 206 207

node 0 size: 48172 MB

node 1 cpus: 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 80 81 82 83 84 85 86 87 88
89 90 91 92 93 94 95 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 208
209 210 211 212 213 214 215 216 217 218 219 220 221 222 223

node 1 size: 48375 MB

[...]

node 3 cpus: 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 112 113 114 115 116 117 118
119 120 121 122 123 124 125 126 127 176 177 178 179 180 181 182 183 184 185 186 187 188
189 190 191 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255

node 3 size: 48375 MB
```

Flat (All-to-all / Quadrant)

Requesting flat-quadrant mode in Theta: **--queue debug-flat-quad / --attrs mcdram=flat:numa=quad**

Output of numactl -H shows two NUMA domains: DDR4 (192 GB, node 0) and MCDRAM (16GB, node 1):

```
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
node 0 size: 193300 MB
-----
node 1 cpus:
node 1 size: 16125 MB
```


Flat (SNC-4)

Requesting flat-snc4 mode in Theta: **--attrs mcdram=flat:numa=snc4**

Output of numactl -H shows eight NUMA domains: 0-3 in DDR4 (48 GB each) and 4-7 in MCDRAM (4 GB each):

```
available: 8 nodes (0-7)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78
79 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 192 193 194 195 196 197 198
199 200 201 202 203 204 205 206 207
node 0 size: 48172 MB
[...]
node 3 cpus: 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 112 113 114 115 116 117 118 119 120
121 122 123 124 125 126 127 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 240
241 242 243 244 245 246 247 248 249 250 251 252 253 254 255
node 3 size: 48375 MB
-----
node 4 cpus:
node 4 size: 4031 MB
[...]
node 7 cpus:
node 7 size: 4029 MB
```

Split (All-to-all/Quadrant)

Requesting flat-quadrant mode in Theta: **--attrs mcdram=split:numa=quad**

Output of numactl -H shows two NUMA domains: DDR4 (192 GB, node 0) and MCDRAM (12GB, node 1). The remaining 25% of MCDRAM (4GB) is a transparent cache to DDR4.

```
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
node 0 size: 193300 MB
-----
node 1 cpus:
node 1 size: 12093 MB
```

Memory allocation control with numactl

DDR4 is the default (Node 0)

To only use DDR:

```
$ numactl --membind=0 ./a.out
```

To only use MCDRAM:

```
$ numactl --membind=1 ./a.out
```

If the code uses more than 16 GB but you prefer to use MCDRAM as main memory:

```
$ numactl --preferred=1 ./a.out
```

The problem with **preferred** is that you have no certainty MCDRAM is being used for the arrays that matter – more in a couple of slides.

Flat SNC-2/SNC-4 NUMA node selection

When running on SNC-2 / SNC-4, MCDRAM nodes start in NUMA node 2 / 4.

```
SNC-2: $ numactl --membind=2,3 ./application
```

```
SNC-4: $ numactl --membind=4,5,6,7 ./application
```

But this is not optimal because in SNC different cores are associated to quadrants / halves of the chip

Optimal affinity in SNC-2 / SNC-4 requires that you identify the core(s) a process is associated with in order to choose the closest MCDRAM NUMA domain (No, you can't use `aprun -S` to do this for you)

memkind

- memkind is a library that enables control of memory
- It allows to specify the location of each element in the heap memory
- Built on top of libnuma and jemalloc
- It provides two different interfaces:
 - `hbwmalloc.h` (recommended)
 - `memkind.h`
- **Add `-lmemkind` to the linking step**

Use Vtune™ Amplifier's Memory Access analysis to determine arrays suitable for MCDRAM

NOTE: Fortran will not issue a warning if you forget to link with the library, it will simply default to using DDR4

Changes in the code (C)

ORIGINAL

```
#include <stdlib.h>

...

double *A;
A = (double*) malloc(sizeof(double) * N);

...

free(A);
```

NEW

```
#include <hbwmalloc.h>

...

double *A;
A = (double*) hbw_malloc(sizeof(double) * N);

...

hbw_free(A);
```

- Very straightforward change
- Beware different spelling of header and API calls

Changes in the code (Fortran)

ORIGINAL

```
...  
real, allocatable :: A(:)  
...  
ALLOCATE (A(1:1024))
```

NEW

```
...  
real, allocatable :: A(:)  
!DIR$ ATTRIBUTES FASTMEM :: A  
...  
ALLOCATE (A(1:1024))
```

- Very straightforward change
- Simple attribute achieves desired effect

Aligned memory

ORIGINAL

```
#include <stdlib.h>
...
int ret;
double *A;
ret = posix_memalign((void *)A, 64, sizeof(double)*N);
if (ret!=0) //error
...
free(A);
```

NEW

```
#include <hbwmalloc.h>
...
int ret;
double *A;
ret = hbw_posix_memalign((void*) A, 64, sizeof(double)*N);
if (ret!=0) //error
...
hbw_free (A);
```


Memkind Functionality

```
int  hbw_check_available(void);  
void* hbw_malloc(size_t size);  
void* hbw_calloc(size_t nmemb, size_t size);  
void* hbw_realloc(void *ptr, size_t size);  
void  hbw_free(void *ptr);  
  
int  hbw_posix_memalign(void **memptr, size_t alignment, size_t size);  
int  hbw_posix_memalign_psize(void **memptr, size_t alignment, size_t size, hbw_pagesize_t pagesize);  
hbw_policy_t hbw_get_policy(void);  
int  hbw_set_policy(hbw_policy_t mode);
```

MCDRAM usage cheatsheet

To run on MCDRAM for Flat + (A2A/Quad):

```
$ numactl --membind=1 ./executable
```

Using memkind to control allocations:

```
#include <hbwmalloc.h>  
  
double *A;  
  
A=(double *)hbw_malloc(<size>);
```

```
REAL, ALLOCATABLE :: A(:)  
  
!DIR$ ATTRIBUTES FASTMEM :: A  
  
ALLOCATE( A(<range>) )
```

Link with **-lmemkind** (Fortran does NOT give an error if it is missing!!)

Summary

Three basic memory layouts

- Flat, Cache, Hybrid

Three basic clustering modes

- All-to-all, Quadrant, Sub-NUMA

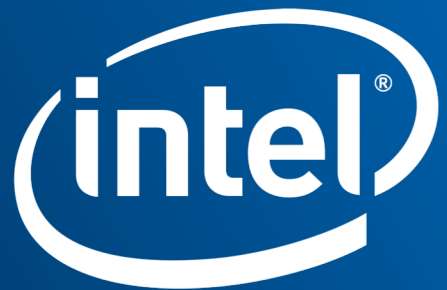
A staggering total of 20 combinations which can only be changed at boot

Cache-Quadrant and Flat-Alltoall provide high performance and simplicity of use

Flat mode allows for fine-grained memory allocation control using the memkind library

SNC-4 should only be employed by experienced users

- Requires additional process binding and memory affinity settings to be effective



Software