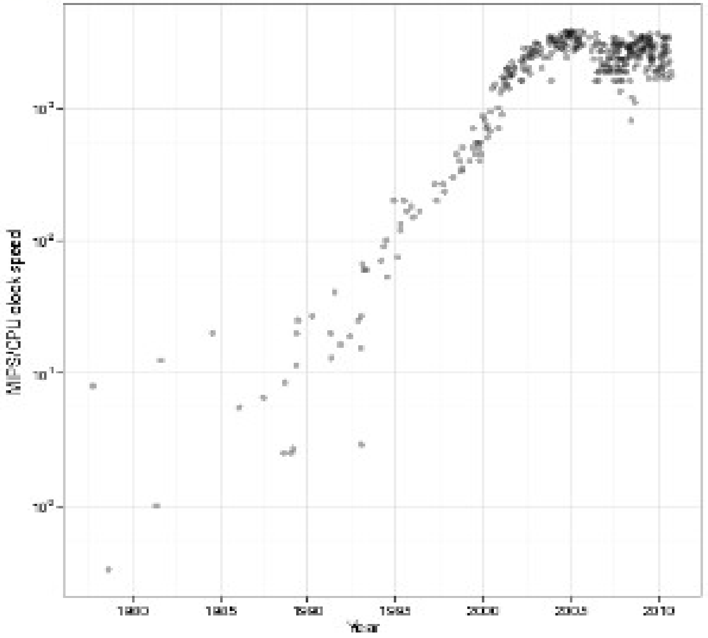
Knights Landing Architecture at Cineca (KNL)

Andrew Emerson, Fabio Affinito

HPC Trends

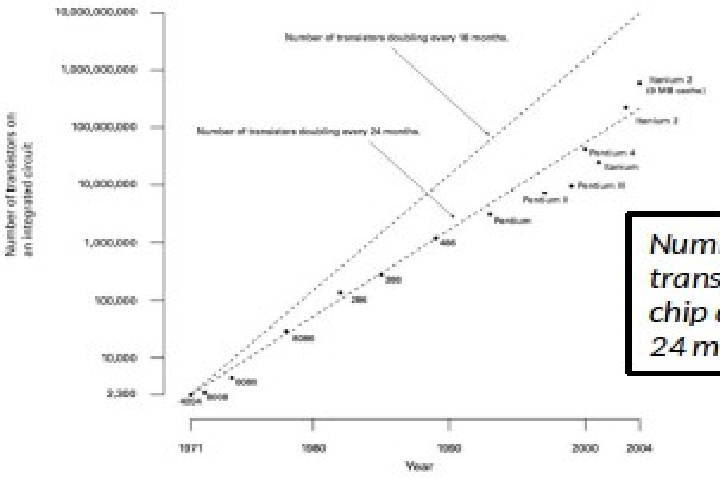
Dennard scaling law (downscaling)



The core frequency and performance do not grow following the Moore's law any longer

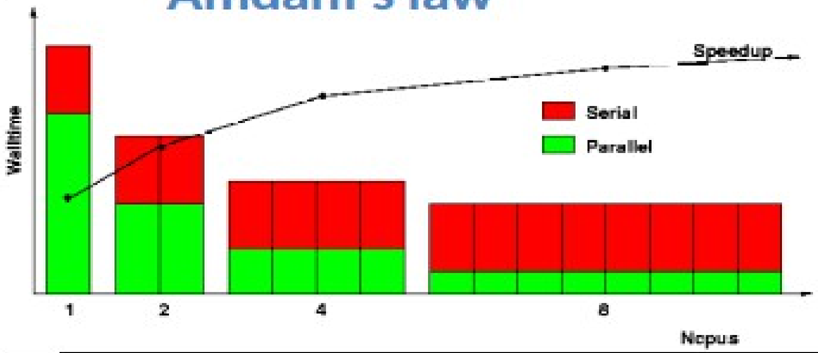
Increase the number of cores to maintain the architectures evolution on the Moore's law

Moore's Law



Number of transistors per chip double every 24 month

Amdahl's law



maximum speedup tends to 1 / (1 − P) P= parallel fraction

The upper limit for the scalability of parallel applications is determined by the fraction of the overall execution time spent in non-parallel operations.

HPC Trends

opportunity

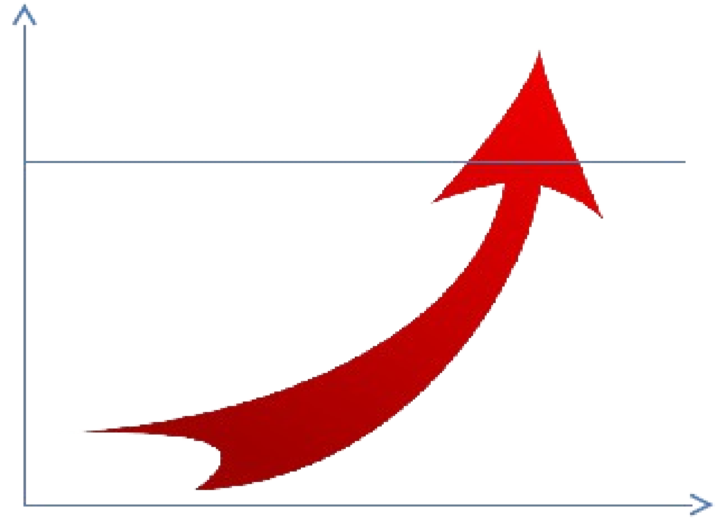
Peak Performance	Exaflops 10^18	Moore law
FPU Performance	Gigaflops 10^9	Dennard law
Number of FPUs	10^9	Moore + Dennard
App. Parallelism	serial fraction 1/10^9	Amdahl's law
System TCO	Power >10^6 Watt	Moore + Dennard

challenge

Energy trends

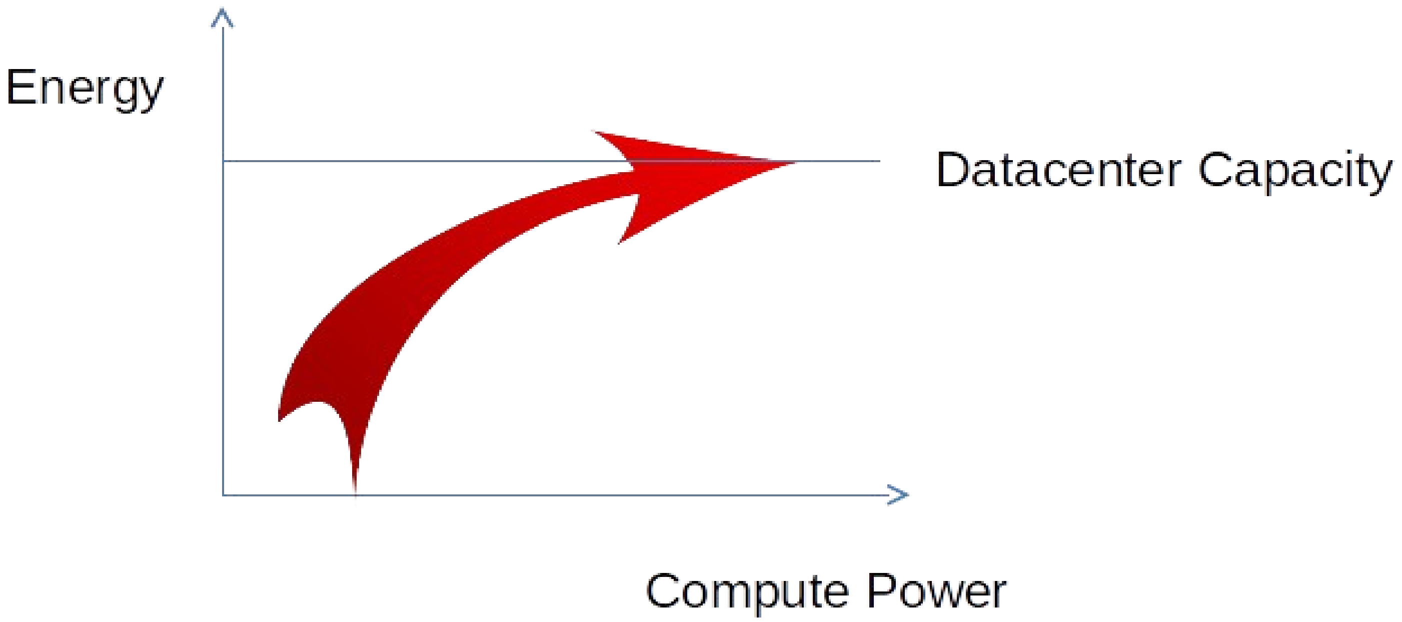
"traditional" RISK and CISC chips are designed for maximum performance for all possible workloads

A lot of silicon to maximize single thread performace

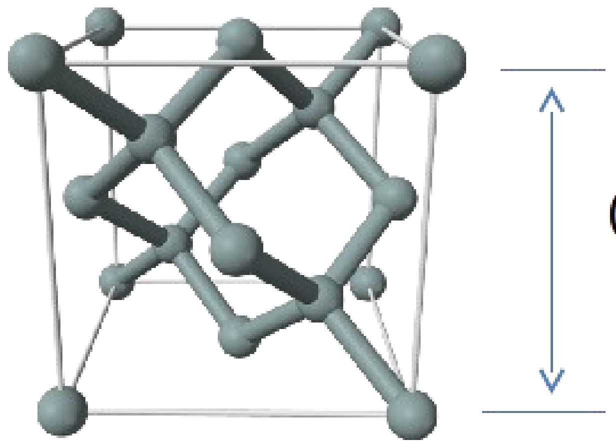


Change of paradigm

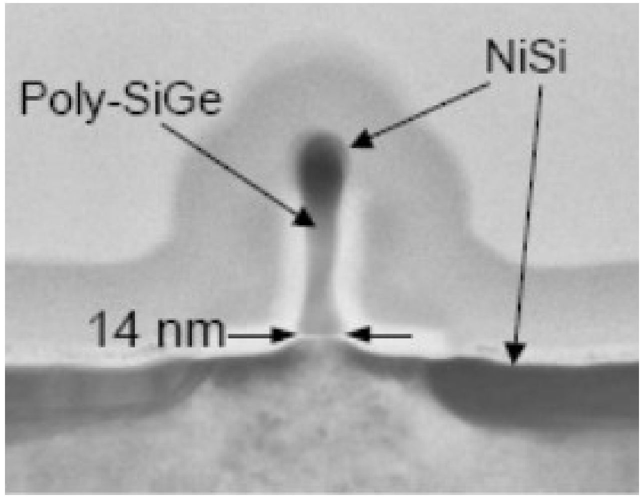
New chips designed for maximum performance in a small set of workloads → Simple functional units, poor single thread performance, but maximum throughput



The silicon lattice



Si lattice



50 atoms!

Cannot get much smaller ……

Current solutions for HPC

1. IBM + NVIDIA/GPU (PASCAL)
2. INTEL Xeon/XeonPHI

Q: Why use Intel Xeon Phi instead of GPU?
A: No need to write in CUDA: standard FORTRAN or C/C++ will work (KNL is even binary compatible)

What is Intel Xeon Phi ?

The brand name given to the chips using Intel's Many Integrated Core (MIC) technology.

For this reason Xeon Phi's are confusingly also called MICs.. (mikes? Micks?).

Different design principle to a standard Xeon → idea to have many cores (60+) and many threads (e.g. 240+).

Current Xeons (Haswell, Broadwell, etc) do not go beyond 44 threads.

Knight's Landing (KNL) is the second generation Xeon Phi and is used in the A2 partition of Marconi (the A1 partition is Intel Broadwell).

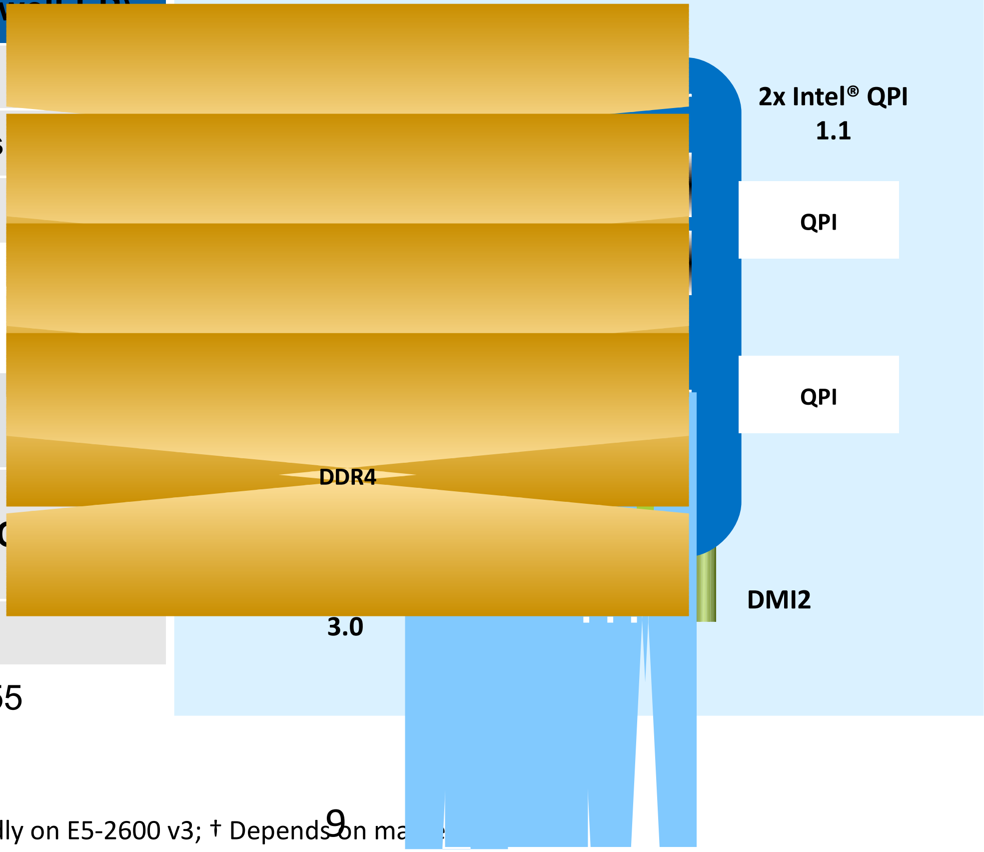
Intel® Xeon® E5-2600 v4 Product Family Overview

New Features:
- Broadwell microarchitecture
- Built on 14nm process technology
- Socket compatible◊ replacement/ upgrade on Grantley-EP platforms

New Performance Technologies:
- Optimized Intel® AVX Turbo mode
- Intel TSX instructions^

Other Enhancements:
- Virtualization speedup
- Orchestration control
- Security improvements

Features	Xeon E5-2600 v3 (Haswell-EP)	Xeon E5-2600 v4 (Broad
Cores Per Socket	Up to 18	**Up to 22**
Threads Per Socket	Up to 36 threads	**Up to 44 threads**
Last-level Cache (LLC)	Up to 45 MB	**Up to 55 MB**
QPI Speed (GT/s)	2x QPI 1.1 channels 6.4, 8.0, 9.6 GT/s	
PCIe* Lanes / Speed(GT/s)	40 / 10 / PCIe* 3.0 (2.5, 5, 8 GT/s)	
Memory Population	4 channels of up to 3 RDIMMs or 3 LRDIMMs	**+ 3DS LRDIMM†**
Memory RAS	ECC, Patrol Scrubbing, Demand Scrubbing, Sparing, Mirroring, Lockstep Mode, x4/x8 SDDC	**+ DDR4 Write CR**
Max Memory Speed	Up to 2133	**Up to 2400**
TDP (W)	160 (Workstation only), 145, 135, 120, 105, 90, 85, 65, 55	



◊ Requires BIOS and firmware update; ^ not available broadly on E5-2600 v3; † Depends on ma

Intel® Xeon Phi™ Product Family

based on Intel® Many Integrated Core (MIC) Architecture

2010
Intel® Xeon Phi
Knights Ferry
prototype
45 nm process
32 cores

2013:
Intel® Xeon Phi™ Coprocessor x100 Product Family

"Knights Corner"
22 nm process
Up to 61 Cores
Up to 16GB Memory

2016:
Second Generation Intel® Xeon Phi™

"Knights Landing"

14 nm

Processor & Coprocessor

+60 cores

On Package, High-Bandwidth Memory

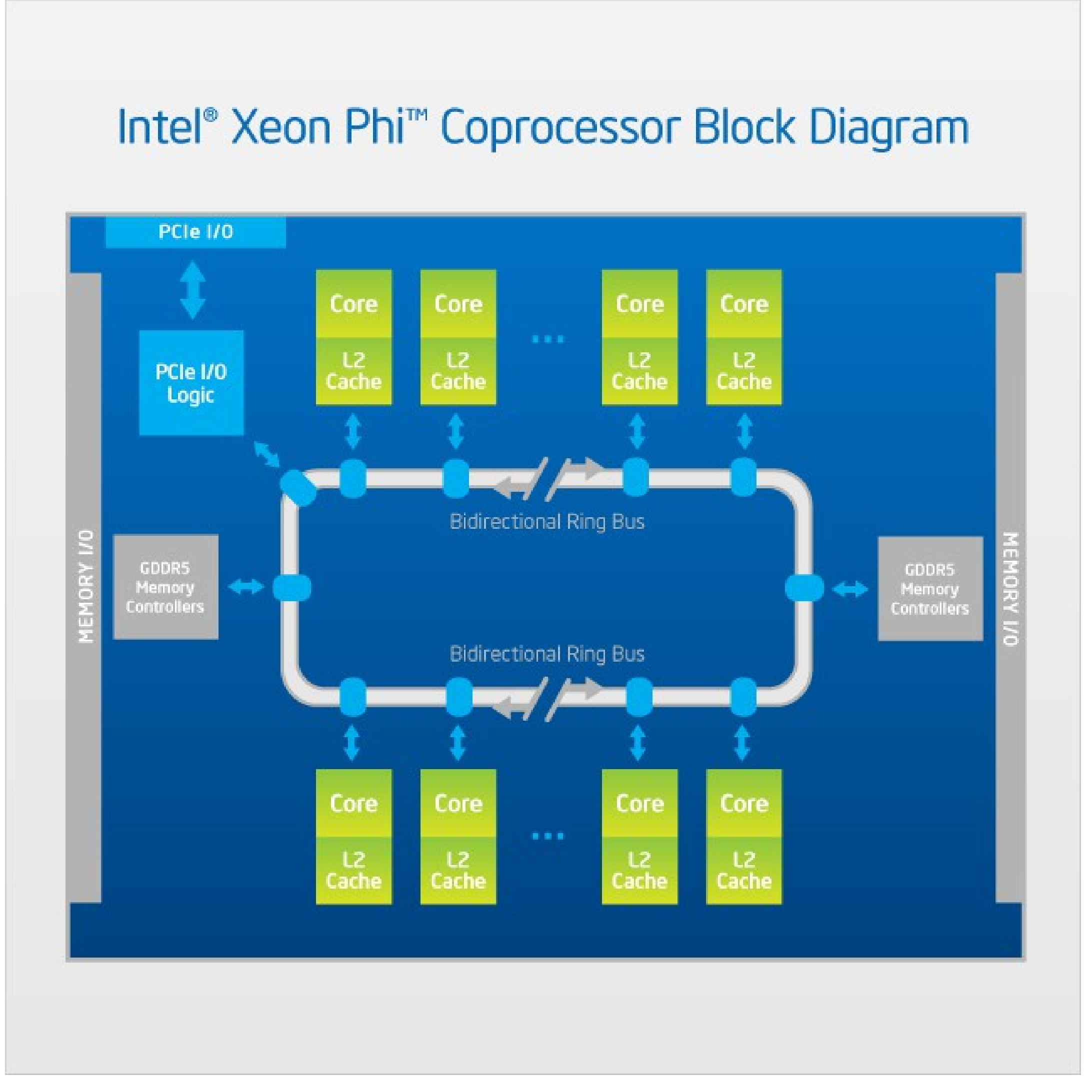
Future Knights:
Upcoming Gen of the Intel® MIC Architecture (Knights Hill)

In planning

Continued roadmap commitment

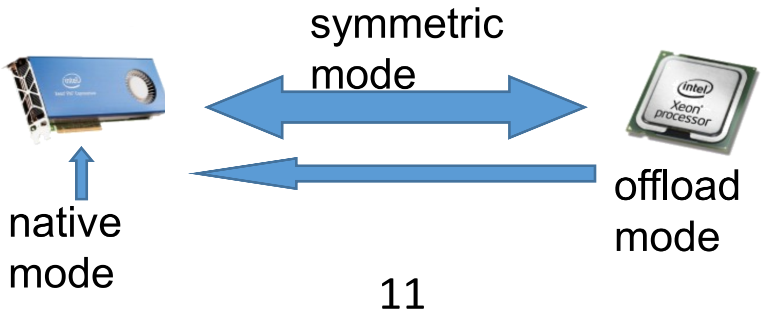
*Per Intel's announced products or planning process for future products

Xeon PHI architecture (KNC)



- 22nm technology
- Up to 61 cores (Pentium-like), ~1.1 Ghz (dep. model)
- 352 Gb/s memory bandwidth (*fast*).
- Upto 244 threads (i.e. 4 threads/core)
- 512 bit SIMD (vector) unit
- 8-16 Gb on board memory,
- ~ 1Tflop peak performance

Includes also sensors for monitoring temperature and power consumption.

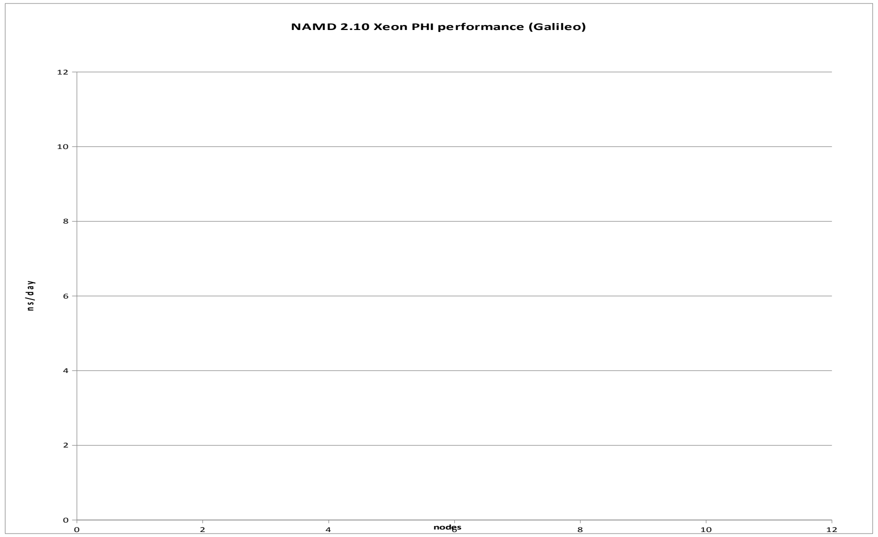


Programming for KNC

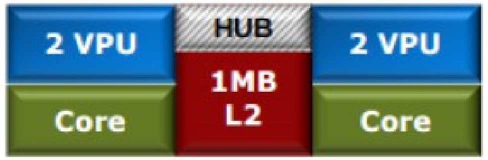
- Because of low power cores and ring network, MPI-only programs will run slowly on KNC.
- Need instead to find programs which can exploit all the cores with OpenMP threads (at least 120).
- The PCI-Express is slow compared to on-board memory so memory transfers should be kept to a minimum – reuse data on the card as much as possible.
- To reach the peak performance need also to exploit the 512bit vector registers.
- Note also that Intel MKL has been optimised for MIC.
- Applications modified to offload to GPUs could be good candidates for offloading to KNCs.

KNC experience

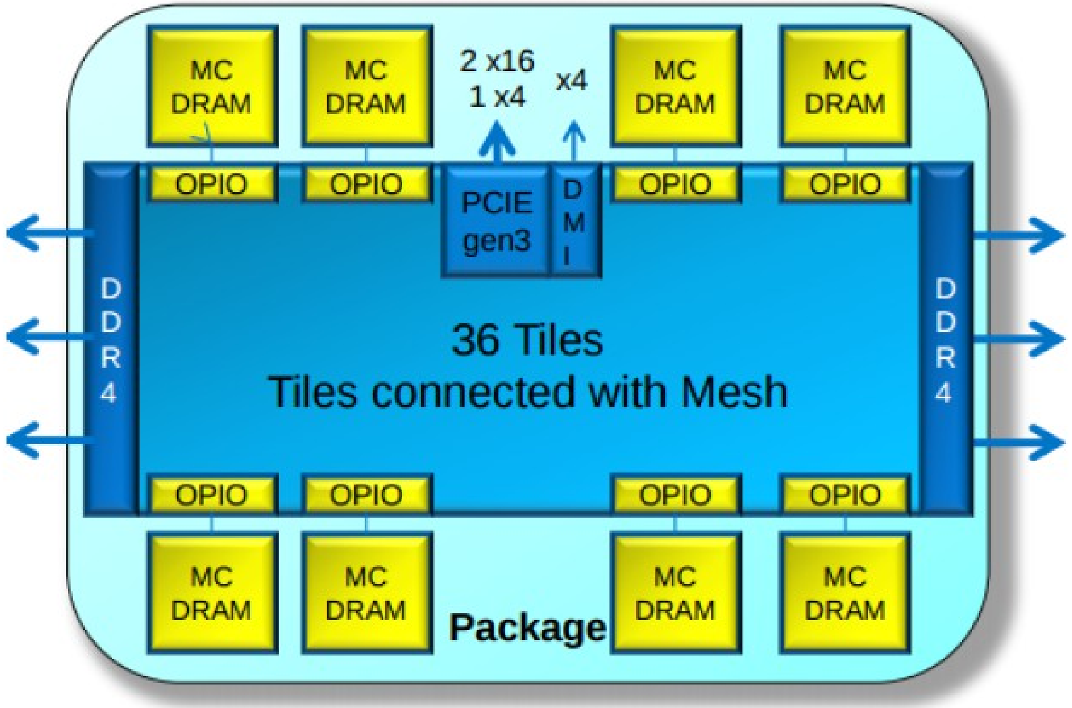
- Cineca’s Galileo cluster has 2 Intel Phi 7120p per node on 384 nodes (768 KNCs in total)
- Do not have hard figures but usage has probably been quite low.
- Main problem is that the KNC cores are not powerful so you need to work hard to get performance. MPI programs can be 10X slower (ring communication network).
- Also KNC is only a co-processor, so unless you already have an offload parallelisation model (e.g. for GPU/CUDA) code needs to be re-worked.



Knights Landing Overview



TILE



Stand-alone, Self-boot CPU

Up to 72 new Silvermont-based cores

4 Threads per core. 2 AVX 512 vector units

Binary Compatible[1] with Intel® Xeon® processor

2-dimensional Mesh on-die interconnect

MCDRAM: On-Package memory: 400+ GB/s of BW[2]

DDR memory

Intel® Omni-path Fabric

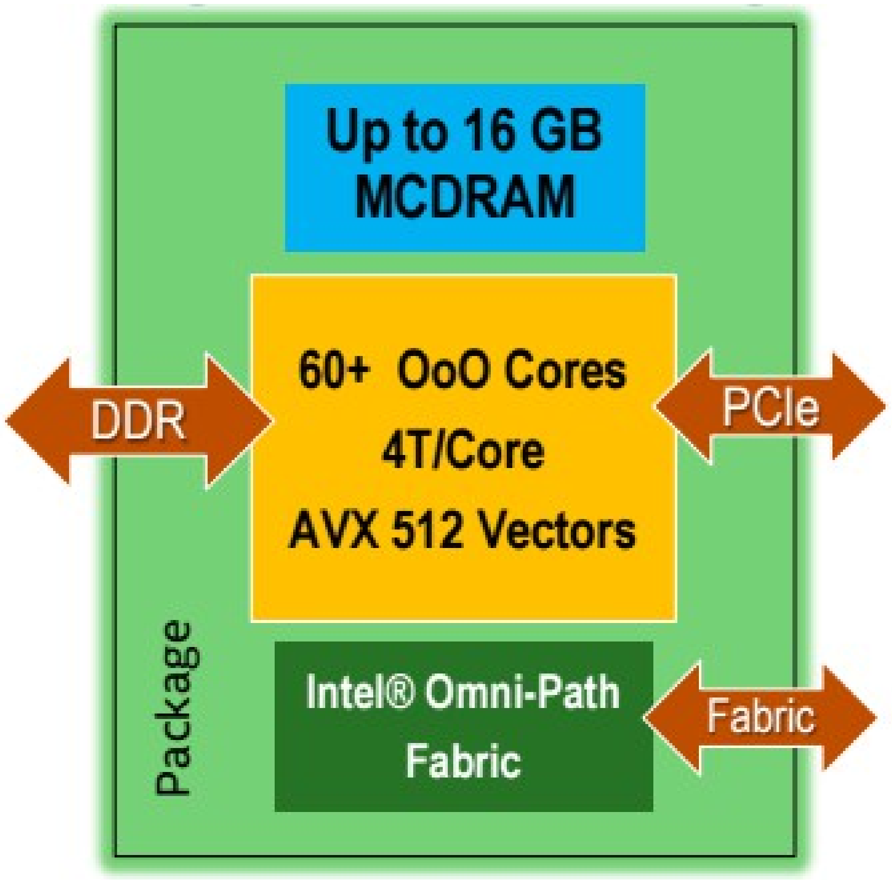
3+ TFLops (DP) peak per package

~3x ST performance over KNC

It's not a GPU. It's not an accelerator.
It's very different from a KNC.

NB: No L3 cache.

Knights Landing: Next-Generation Intel® Xeon Phi™



Stand-alone, Self-boot CPU

60+ new Silvermont-based cores

4 Threads per core

AVX 512 vector units

Binary Compatible[1] with Intel® Xeon® processor

2-dimensional Mesh on-die interconnect

MCDRAM: On-Package memory: 400+ GB/s of BW[2]

DDR memory

Intel® Omni-path Fabric

3+ TFLops (DP) peak per package

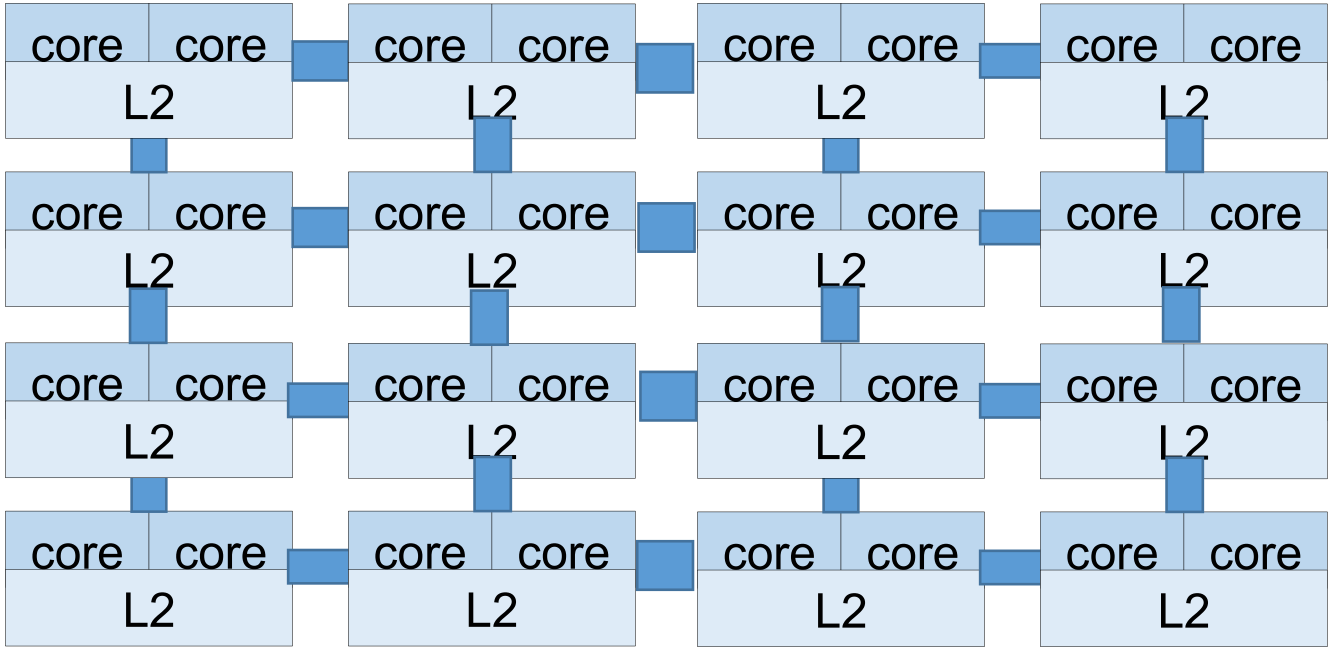
~3x ST performance over KNC

KNC →KNL key differences

Feature	KNC	KNL
Cores	<=61 cores Pentium, 1.1 GHz [in-order]	<=72 Silvermont, 1.4GHz (KNL 7250) [out-of-order]
Boot-up	**Co-processor so needs host CPU**	**Standalone, self boot**
Internal Network	Bi-directional ring	2D Mesh
Connections	PCIe	PCIe, Intel OmniPath or other vendor.
Memory	8-16GB on board	16 GB MCDRAM (High Bandwidth Memory) on board Supports upto 384Gb DDR
Vectorisation	512 bit SIMD/core	2x AVX2 512 units/core
Xeon Compatibility	For Native mode recompile with –mic flag.	Binary compatible, although recompilation recommended (for vectorisation)
Peak Performance	~1 Tflops (DP)	~3 Tflops (DP)
Power consumption	300W	215W (KNL 7250)*

See https://www.nextplatform.com/2016/06/20/intel-knights-landing-yields-big-bang-buck-jump/

Cache organization in KNL



Each core has its own L1 cache (32K)

All the caches are kept coherent in the mesh with the *MESIF* protocol.
Each tile has a directory (*tag directory*) which together make up the DTD (*distributed tag directory)* which identify on the chip the location of any cache line.
If a tile cannot find some data from its local cache then it must query the DTD to find the data.

See, for example, https://colfaxresearch.com/knl-numa/

Cache Clustering Modes

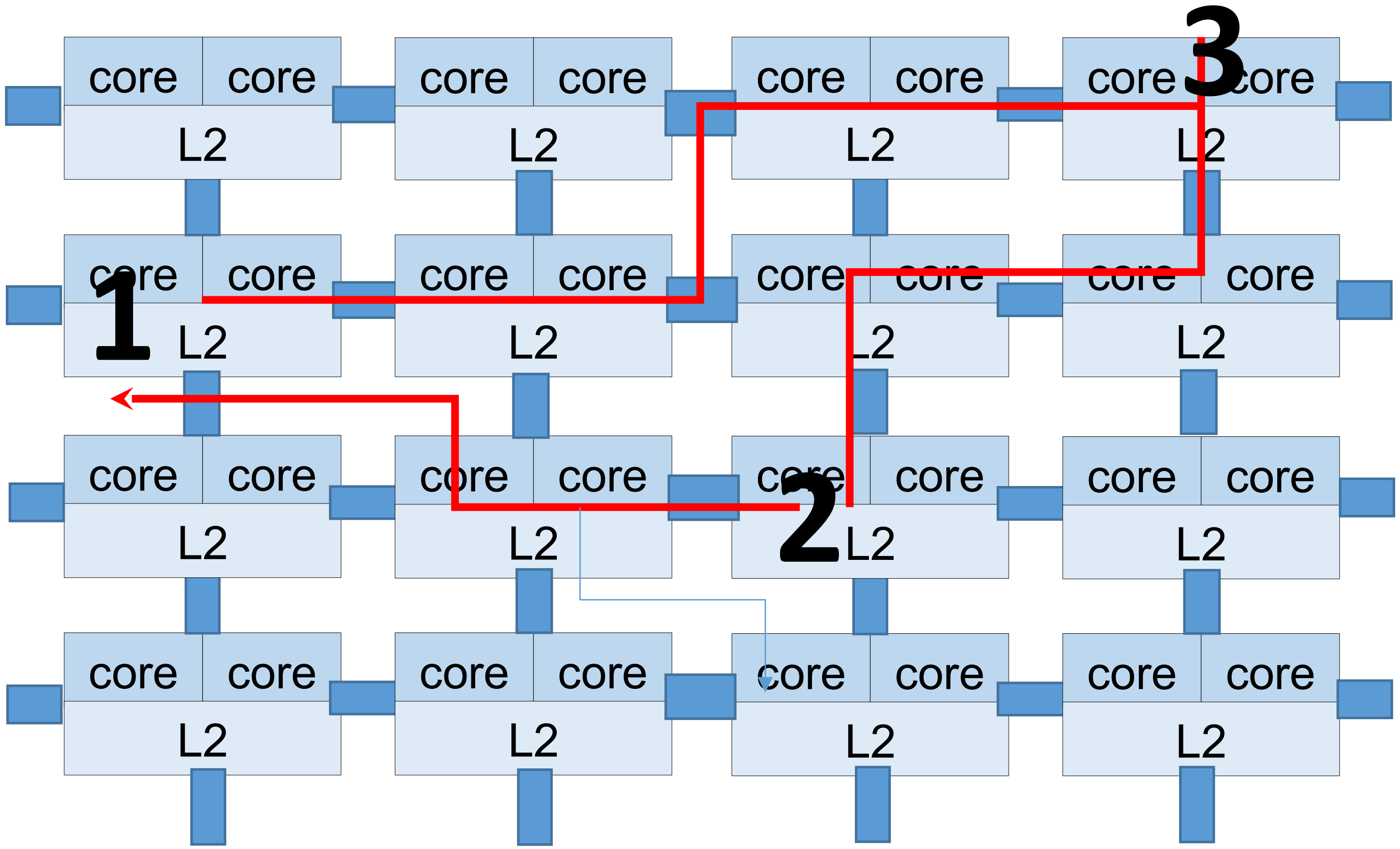
To reduce latency and maximise bandwidth, try to keep the data as local as possible to where it is needed.

This can be done by *clustering modes*. The KNL supports three:

1. All-to-All
2. Quadrant/Hemisphere
3. SNC-4/SNC-2

Clustering is a boot-time decision and can't be changed without restarting the KNL.

All –to – All



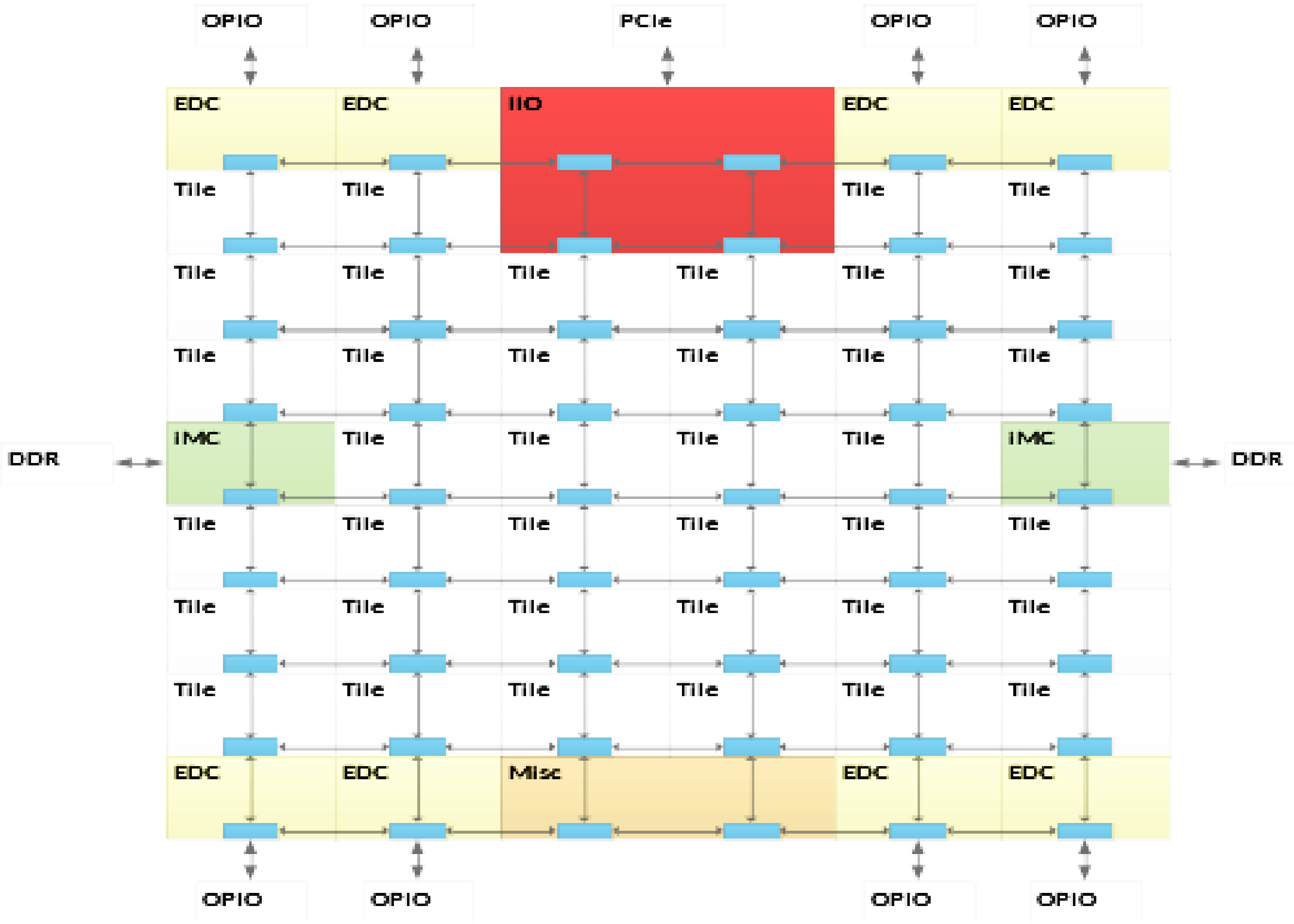
Memory address are uniformly distributed across the TDs on the chip.
An L2 cache miss typically consists of the following steps:

1. L2 miss encountered
2. Send request to the distributed directory
3. Miss in the directory. Forward to memory
4. Memory sends the data to the requestor

Result is a low-performance memory request.

Not normally used

KNL Mesh Interconnect



Mesh of Rings

- Every row and column is a (half) ring
- YX routing: Go in Y → Turn → Go in X
- Messages arbitrate at injection and on turn

Cache Coherent Interconnect

- MESIF protocol (F = Forward)
- Distributed directory to filter snoops

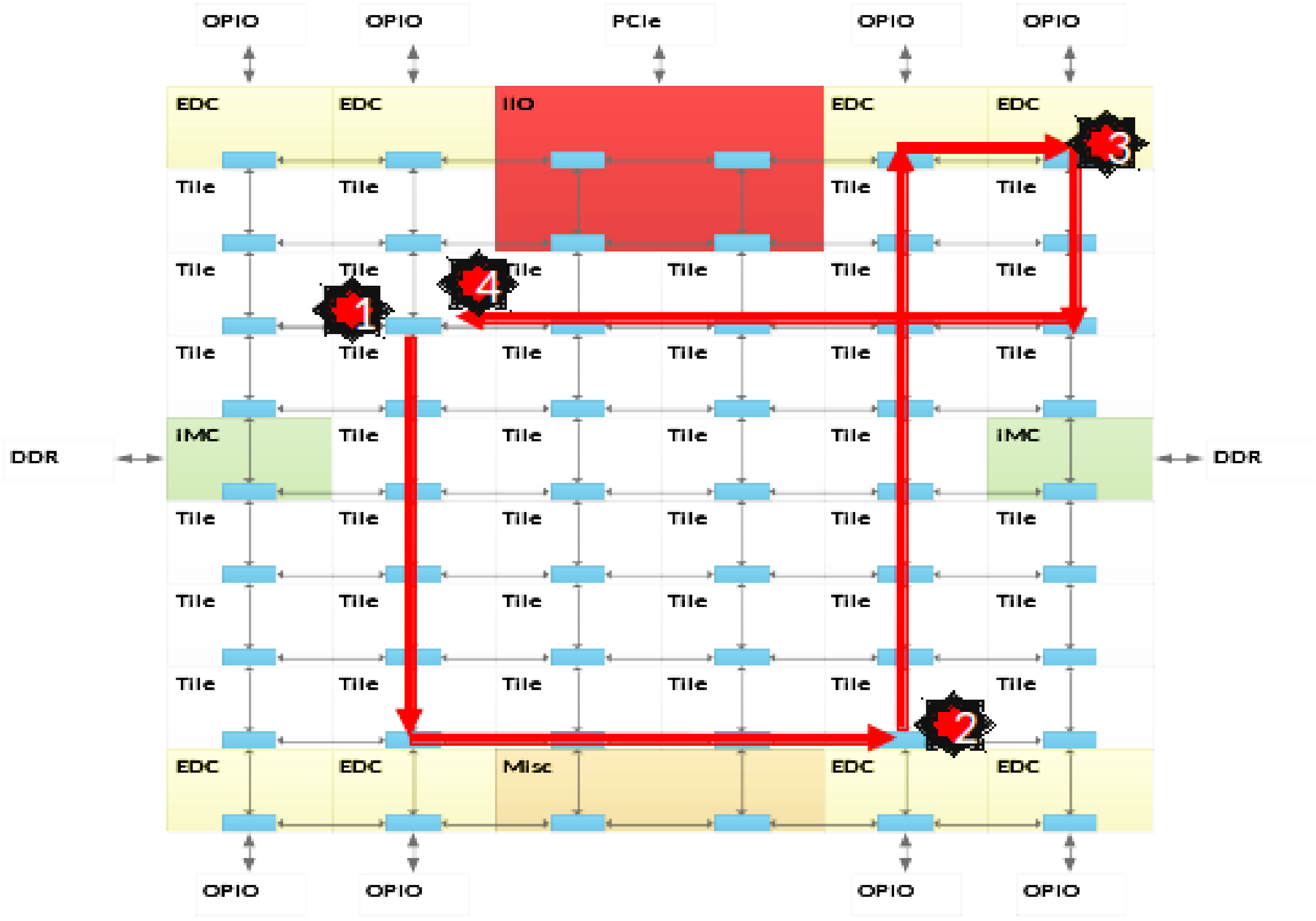
Three Cluster Modes

(1) All-to-All

(2) Quadrant

(3) Sub-NUMA Clustering (SNC)

Cluster Mode: All-to-All



Address uniformly hashed across all distributed directories

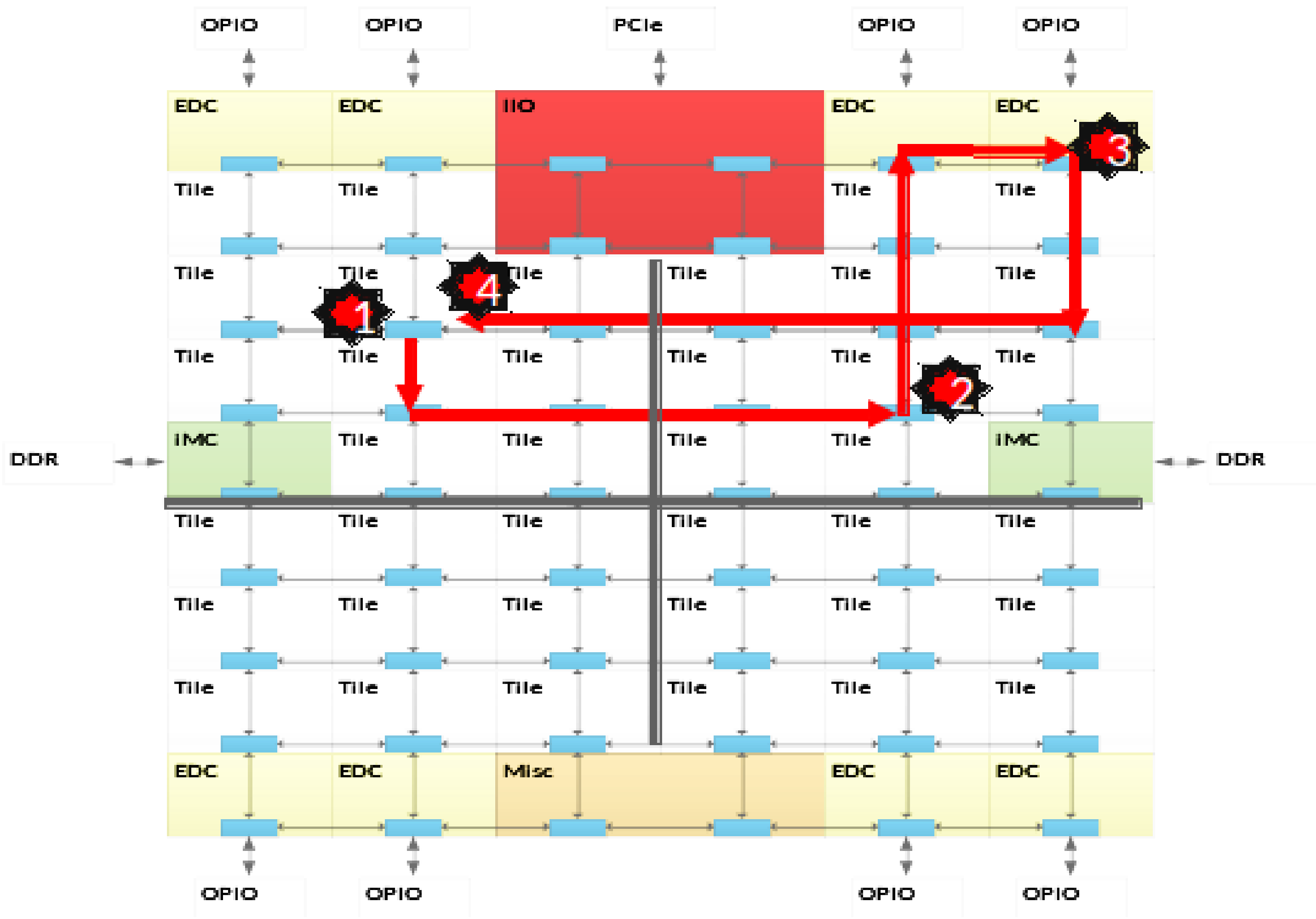
No affinity between Tile, Directory and Memory

Lower performance mode, compared to other modes. Mainly for fall-back

<u>Typical Read L2 miss</u>

1. L2 miss encountered
2. Send request to the distributed directory
3. Miss in the directory. Forward to memory
4. Memory sends the data to the requestor

Cluster Mode: Quadrant



Chip divided into four virtual Quadrants

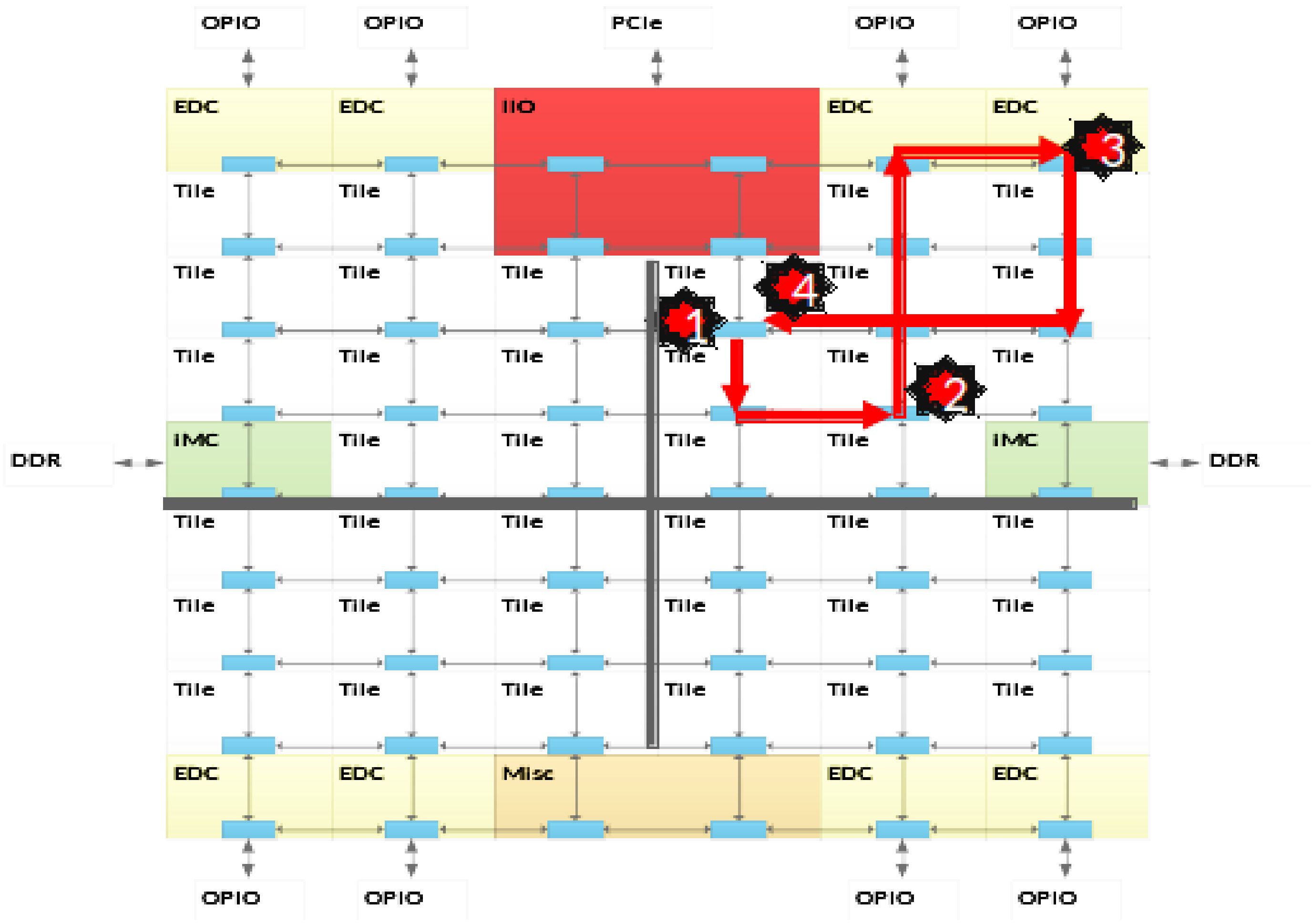
Address hashed to a Directory in the same quadrant as the Memory

Affinity between the Directory and Memory

1. L2 miss, 2. Directory access, 3. Memory access, 4. Data return

Lower latency and higher BW than all-to-all. Software transparent.

Cluster Mode: Sub-NUMA Clustering (SNC)



1. L2 miss, 2. Directory access, 3. Memory access, 4. Data return

Each Quadrant (Cluster) exposed as a separate NUMA domain to OS

Looks analogous to 4-Socket Xeon

Affinity between Tile, Directory and Memory

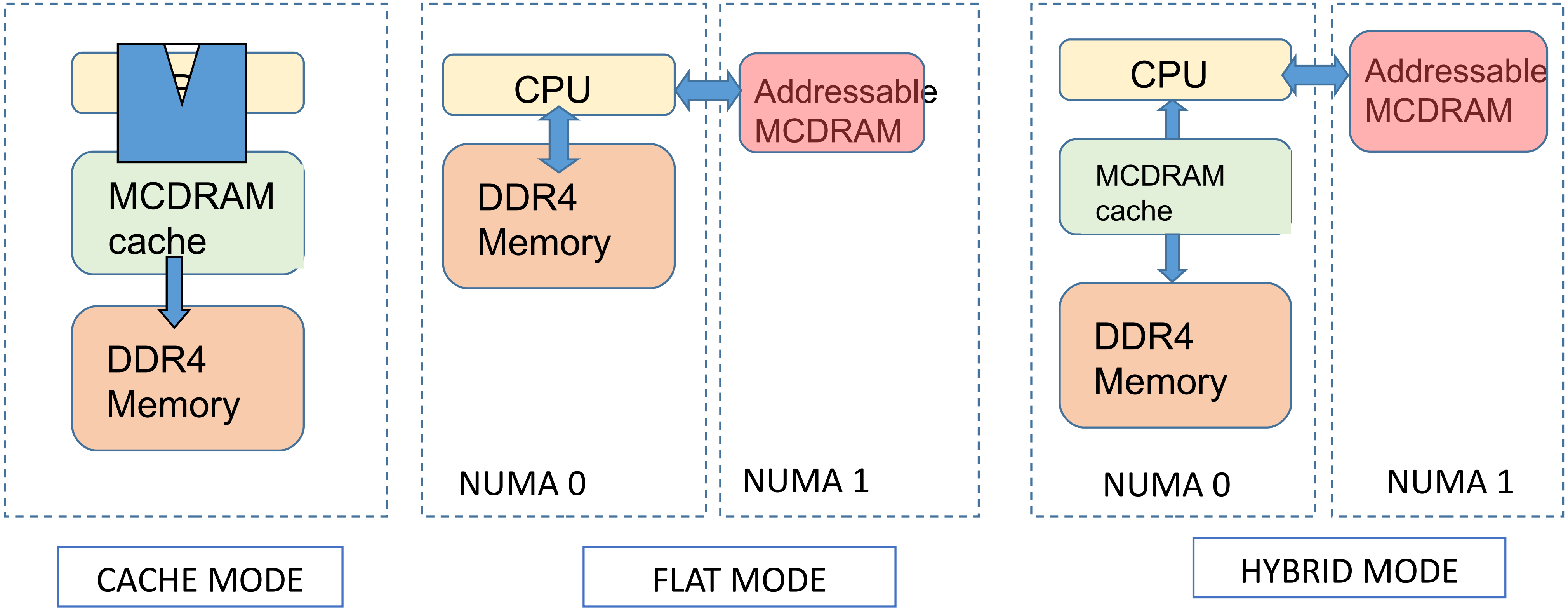
Local communication. Lowest latency of all modes

Software needs to be NUMA-aware to get benefit

Using MCDRAM (High Bandwidth Memory)

CINECA SCAI
SuperComputing Applications and Innovation

- Multi-Channel Dynamic Random Access (MCDRAM) is a high bandwidth memory (~ 400 GB/s) roughly 5x faster than standard DDR4 memory (~90 GB/s).
- On the KNL can be used in three ways:



MCDRAM modes

Cache mode

- The MCDRAM is used as cache so may give performance benefits if DDR memory accesses are reduced.
- Transparent to users so no modifications required.
- But increases latency if data not found in cache (DDR → MCDRAM → L2).

Flat mode

- High bandwidth, low latency.
- More complicated to use – requires software or environmental changes.

Hybrid

- Benefits of both, but smaller sizes.

> The MCDRAM mode is normally at chosen at boot-up of the KNL. In principle should be possible in a batch job to choose which mode but seems more common to select nodes already booted in the desired mode.

Using MCDRAM in flat mode

Even if the KNL has been booted in flat mode, the DDR4 memory is used by default – MCDRAM needs to be explicitly requested.

This can be done in two ways:

1. By launching the application with the numactl command (if executable <16Gb).
2. Modifying the source code to allocate variables in the MCDRAM using, for example, the Memkind library.

With the numactl command, first find the numa device number and then launch with that number.

MCDRAM should be good for for large and often-used arrays.

Using MCDRAM in flat mode - numactl

```
deep70@knl08]:~ > 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

node 0 size: 98200 MB

node 0 free: 88704 MB

node 1 cpus:

node 1 size: 16384 MB

node 1 free: 15909 MB

node distances:

node   0   1

  0:  10  31

  1:  31  10

[deep70@knl08]:~ > mpiexec -n 256 numactl -m 1 ./executable
```

Deeper-sdv JSC

Using MCDRAM in flat mode - numactl

```
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 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 68 69 70 71 72 73 74 75 76 77
…
node 0 size: 49055 MB
node 0 free: 46228 MB
node 1 cpus: 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 67 102 103 104 105
…..
252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271
node 1 size: 49152 MB
node 1 free: 46369 MB
node 2 cpus:
node 2 size: 8192 MB
node 2 free: 7547 MB
node 3 cpus:
node 3 size: 8192 MB
node 3 free: 7538 MB
…..
```

Marconi Cineca

```
numactl --membind 2,3    ./executable
```

MCDRAM and memkind

Allows memory to be allocated on any NUMA device.

Has two interfaces:

hbwmalloc

memkind

but both use the same “backend”.

man hbwmalloc and *man memkind* give more information.

```
// C memkind  interface Example
#include <memkind.h>
hbw_str = (char *)memkind_malloc(MEMKIND_HBW,
size);
if (hbw_str == NULL) {
        perror("memkind_malloc()");
        fprintf(stderr, "Unable to allocate
hbw string\n");
        return errno ? -errno : 1;
    }
// use hbw_star
memkind_free(MEMKIND_HBW, default_str);
```

```
! Fortran memkind  interface Example
Real, allocatable :: a(:), b(:)
! FASTMEM attribute
!DEC$ ATTRIBUTES FASTMEM :: A
! A is allocated in HBM
ALLOCATE (A(1:1024))

! B is allocated in DDR4
ALLOCATE (B(1:1024))
```

Thread and task affinity (pinning)

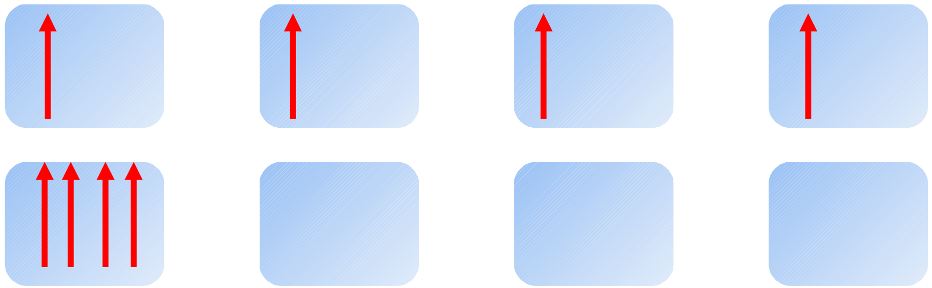
With the number of tasks or threads/node increasing it is important to know to which cores they have been assigned. This is true for all NUMA HPC systems but particularly relevant for KNL. The performance difference can be significant !

A complicated topic because the pinning is controlled by a variety of options or environment options, many of which are intel-specific and liable to change. Also the clustering mode of the KNL adds further complexity.

Recommended 2 or 4 threads/core (but never 3).

Documentation for Intel KNL is given here:

https://software.intel.com/en-us/articles/process-and-thread-affinity-for-intel-xeon-phi-processors-x200



KMP_AFFINITY=scatter

KMP_AFFINITY=compact

Thread and task pinning – some

Variable	Example	Description
I_MPI_DEBUG=[0-5]	export I_MPI_DEBUG=5	Shows the logical cores owned by each MPI rank (the affinity). Default affinity=scatter
KMP_AFFINITY=[scatter,compact, proclist={..}]	export KMP_AFFINITY=compact, verbose.	Changes the affinity to, e.g compact. The verbose option shows the result of the change.
KMP_PLACE_THREADS, KMP_HW_SUBSET=<t>T (new)	export KMP_HW_SUBSET=2T	threads per core (by default all 4 thds/core are used)
OMP_NUM_THREADS=n	export OMP_NUM_THREADS=4	For an OpenMP p… threads, for hybr…
OMP_PLACES=[cores,threads]	export OMP_PLACES=0,1,2,3,4,…271,272	Specifies hardware resource. Used with OMP_PROC_BIND
OMP_PROC_BIND=[close, spread]	export OMP_PROC_BIND=spread,close	How OpenMP threads are bound to resources

Difficult to use correctly.
Better use KMP_HW_SUBSET

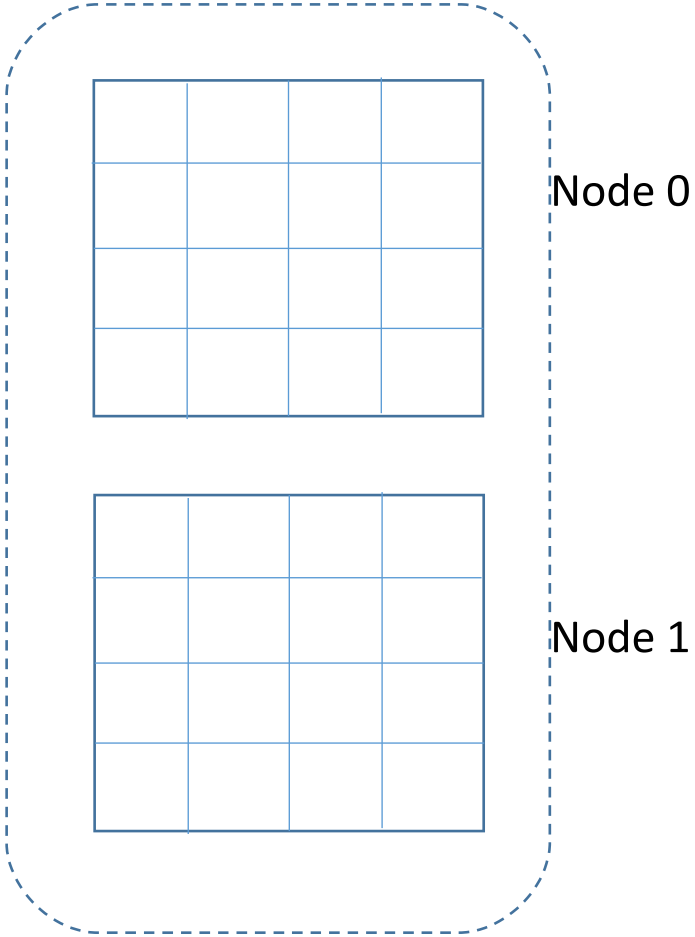
Pinning MPI tasks and OpenMP threads in SNC mode

On Marconi default cluster mode is SNC-2, i.e. two numa nodes each with 34 cores (max 136 threads).

For a pure OpenMP program, one possibility is to use nested OpenMP, creating two teams of 136 threads.

```
$export OMP_NESTED=1
$export OMP_NUM_THREADS=2,136
$export OMP_PLACES=0,1,2,3,…33,68,..237,
34,35,67,102,135,170, 171,..203,238,271
$export OMP_PROC_BIND=spread,close
```

numa 0 thds
numa 1 thds



Threads in each team will be close to each other.

(But OpenMP rarely scales up to 136 threads so need to try different variants)

Pinning MPI tasks and OpenMP threads in SNC mode

The MPI task pinning can be queried by setting

I_MPI_DEBUG=5

For NUMA-aware MPI (e.g. Intel) the default affinity should be “sensible”, e.g. for SNC-2 the ranks should be spread evenly between the nodes.

Finer control (e.g at the tile level) can be controlled with variables such as I_MPI_PIN_PROCESSOR_LIST or I_MPI_PIN_DOMAIN.

For multi-nodes, the –perhost option indicates how many tasks/knl.

```
$ export I_MPI_DEBUG=5
$ mpirun -np 2 ./simple
…
[0] MPI startup(): Rank    Pid       Node name
 Pin cpu
[0] MPI startup(): 0       698092    r086c15s03

{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,1
8
30,31,32,33,68,69,70,71,72,73,74,75,76,77,78,7
9,80,81,82,83,84,85,86,87,88,89,90

221,222,223,224,225,226,227,228,229,230,231,23
2,233,234,235,236,237}
[0] MPI startup(): 1       698093    r086c15s03

{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
                      ,
61,62,63,64,65,66,67,102,103,104,105,106,107,1
08,109,110,…,265,266,267,268,269,270,271}
```

Binding memory to NUMA nodes

Once the thread or task has been pinned to a sub-NUMA cluster it makes sense also to bind the memory objects.

As with any NUMA device the default allocation policy in Linux is "first touch", when a thread first writes into a newly allocated array (touches it) the memory page is allocated on the thread's NUMA node.

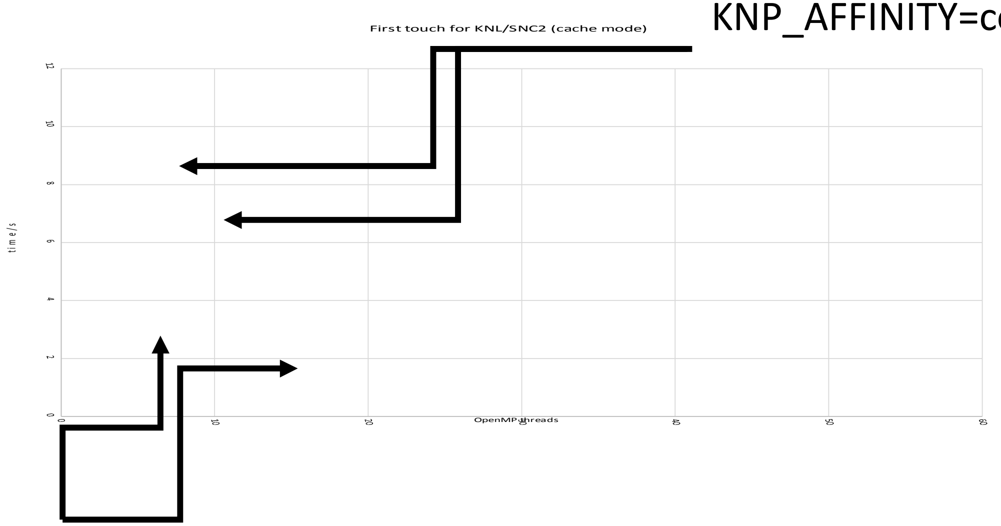
For this reason also array initialisation should be done in the parallel region: otherwise the array is allocated on the numa mode of the master thread. [assuming the shared array is large]

```
// C++ version
int main() {
   float *A = new float[N];
#pragma parallel for
 for (int i=0; i < N; i++)
    A[i] = 0.0f;
}
..
! FORTRAN
program numa

real, allocatable :: a(:)
allocate(a(N))
!$omp parallel do
do i=1,N
   a(i)=0.0
end do
!$omp end parallel do
..
```

physical memory allocation occurs here

Binding memory to numa nodes - Example

KNP_AFFINITY=compact



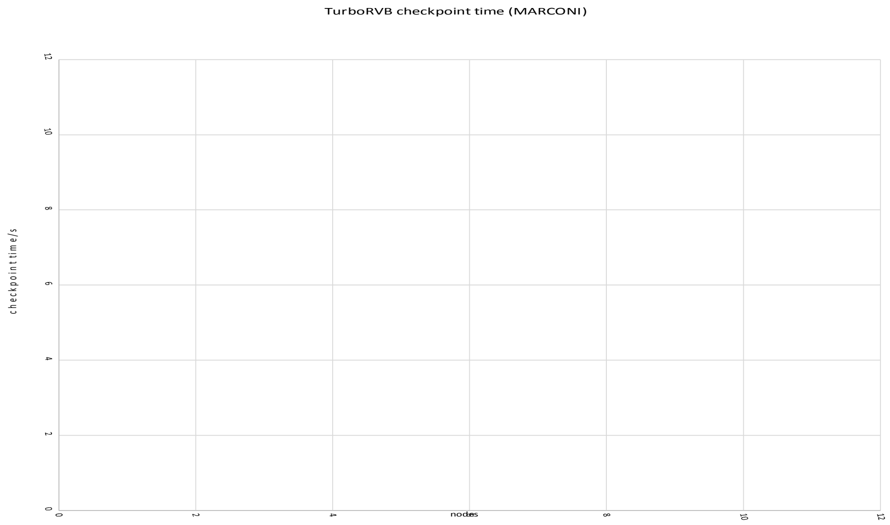
KNP_AFFINITY=scattered

```
integer :: n=1000000000
allocate(a(n),b(n),c(n))

!!$omp parallel do
      do i=1,n
         a(i)=10.0
         b(i)=2.0
         c(i)=1.0
      enddo

!!$omp end parallel do
      t=mysecond()
   do j=1,10
!$OMP parallel do

      do i=1,n
         a(i)=b(i)+d*c(i)
      enddo
!$OMP end parallel do
```

KNLs and I/O

Little hard data but likely to be slow:

- Relatively slow cores.
- Many threads and processes.
- Kernel I/O (e.g. C/Fortran or POSIX I/O) in parallel to single files is not threadsafe. Need to use HDF5, MPI-I/O or other parallel library
- GPFS and other parallel filesystems do not scale well for task-local files.



each MPI rank writes its own checkpoint file

Using KNLs

Although very different to KNCs the trends are the same…

Data parallelism

- Lots of threads, spent on MPI ranks or OpenMP/TBB/pthreads
- Improving support for both peak throughput and modest/single thread

Bigger, better, faster memory

- High capacity, high bandwidth, low latency DRAM
- Effective caching and paging
- Increasing support for irregular memory refs, modest tuning

Vectorisation

- Increasing support for vectorisation

Using KNLs

Even for non-developers, a number of options need to be considered in order to optimise performance for KNL:

- How many MPI ranks and/or OpenMP threads per node (at least 2 per core to hide hardware latency). With higher DRAM we can use more MPI ranks and perhaps < OpenMP threads.
- Quandrant, hemisphere, SNC2 or SNC4
- MCDRAM: Flat mode or Cache mode?
- Thread or task pinning? (IMPORTANT)
- If linked with MKL, how many threads for MKL ? (MKL_NUM_THREADS)

For developers the first step is to re-compile with –xMIC-AVX512 but a further analysis with e.g. Vtune would be a good idea.

Intel advises cache/quadrant as the preferred configuration for KNLs but should be possible to test other configurations

Which applications are good for KNL?

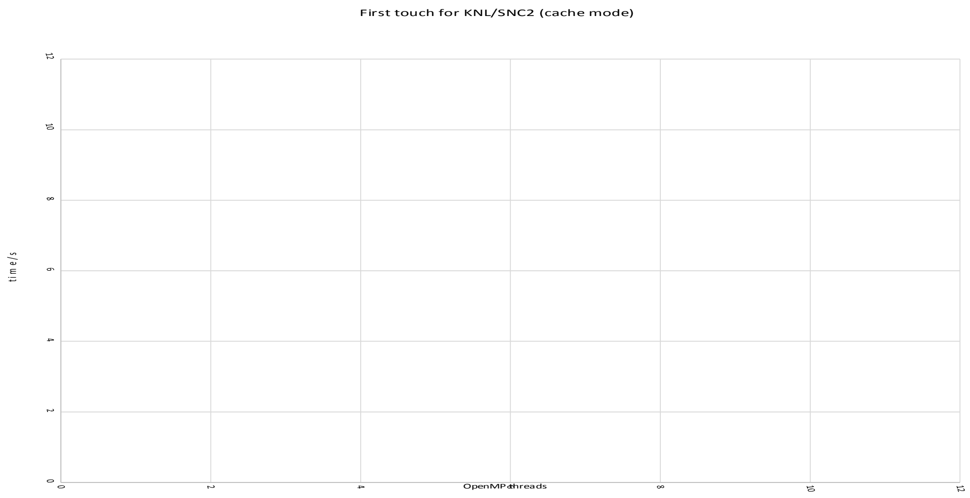
Not an easy question to answer because also depends on input.

But “KNL-friendly” application+input combinations should have the following features:

- Highly parallel, many ranks and threads.
- Low memory/thread
- Highly vectorised
- Low I/O overheads

KNLs are new, Marconi is new, so help us find out what works best!

MARCONI experiences so far




```
integer :: n=1000000000
allocate(a(n),b(n),c(n))


!!$omp parallel do
      do i=1,n
         a(i)=10.0
         b(i)=2.0
         c(i)=1.0
      enddo
!!$omp end parallel do

      t=mysecond()
   do j=1,10
!$OMP parallel do
      do i=1,n
         a(i)=b(i)+d*c(i)
      enddo
!$OMP end parallel do
```

- KNL
- -xMIC-AVX512

Function	Rate (MB/s)	Avg time	Min time	Max time
Copy:	19044.1886	0.0169	0.0168	0.0169
Scale:	13998.8035	0.0230	0.0229	0.0231
Add:	15505.8643	0.0311	0.0310	0.0312
Triad:	15681.7174	0.0307	0.0306	0.0308

Function	Rate (MB/s)	Avg time	Min time	Max time
Copy:	18974.1900	0.0196	0.0169	0.0229
Scale:	8648.6625	0.0407	0.0370	0.0454
Add:	9712.8780	0.0516	0.0494	0.0551
Triad:	9478.8762	0.0535	0.0506	0.0600