Using Cray Systems with Knights Landing processors

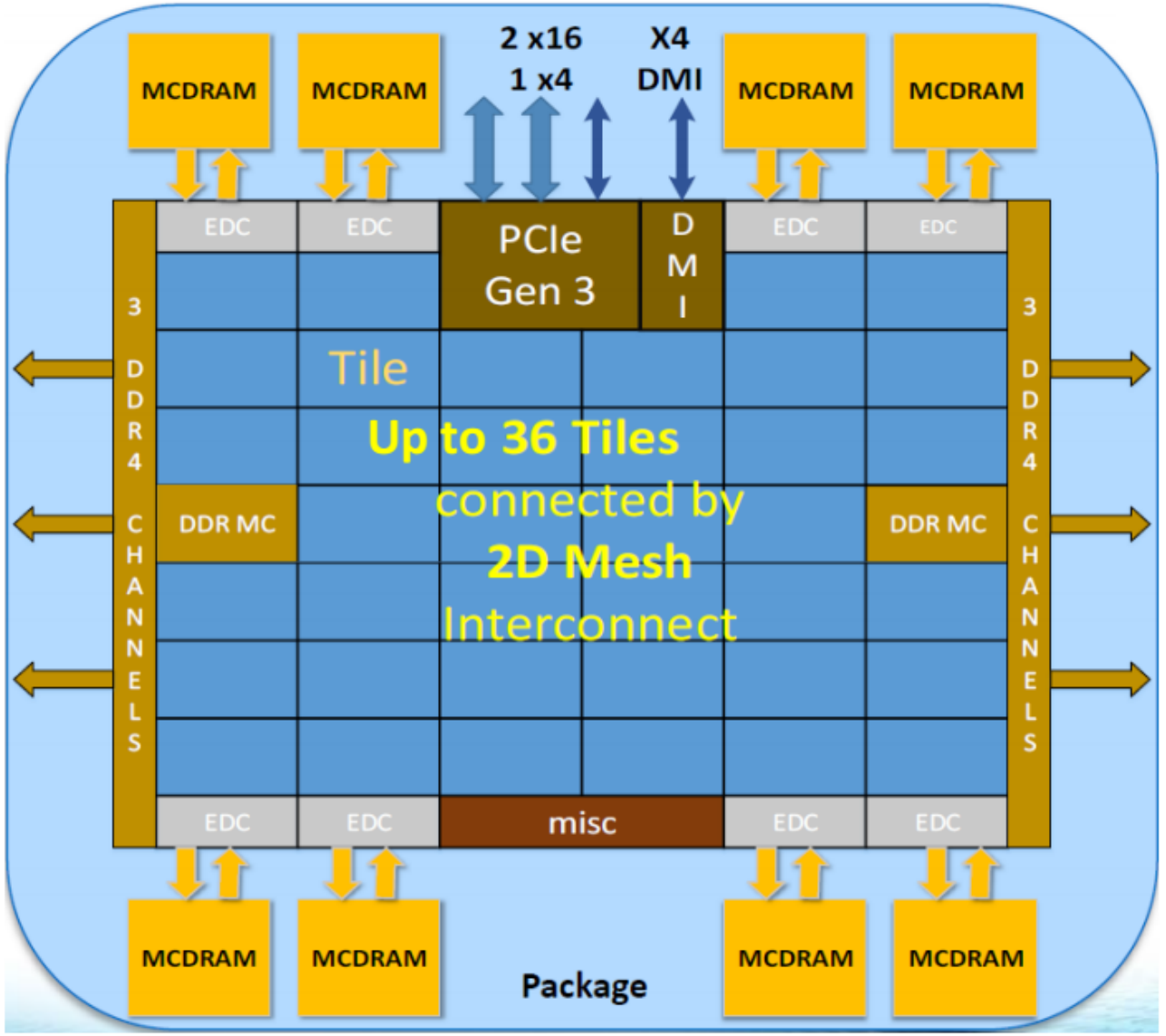
Kevin Thomas

Cray Inc.

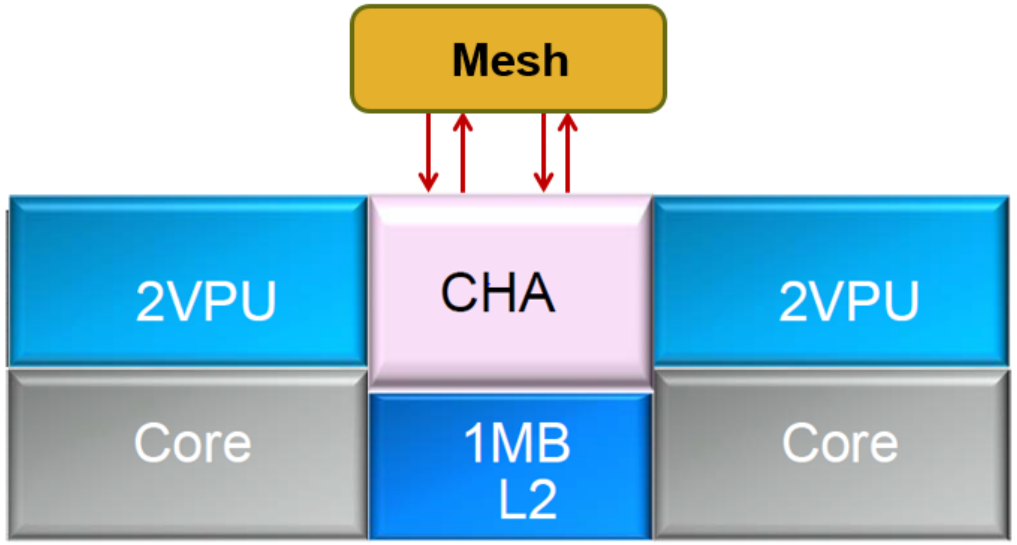
Outline

- **Part 1**
  - Background
  - Node Comparison
  - How to use
- **Part 2**
  - Strategies
  - Best Known Methods
  - Resources

KNL Processor Architecture



KNL Tile Architecture



KNL Tile Frequencies and Turbo Mode

- **Two turbo tile frequencies implemented**
  - "All tiles active" turbo, +100 MHz
  - "Single tile active" turbo, +200 MHz
- **Two below-base frequencies**
  - Heavy AVX instructions, -200 MHz
  - Under some conditions -100 MHz also possible
- **Xeon Phi 7250 tile frequencies**
  - 1.6 GHz single tile turbo
  - 1.5 GHz all tiles turbo
  - 1.4 GHz base frequency
  - 1.3 GHz
  - 1.2 GHz AVX

Xeon Phi "*Knights Landing*" Compatibility

- **Runs existing Xeon x86 64-bit executables**
  - Linux commands
  - ISV applications
  - Applications built for Xeon processors
- **Existing Xeon-targeted libraries will work**
  - If library is not a critical compute component, recompiling not needed
  - Intel 16 MKL has AVX-512 support enabled at run time
- **Xeon executables can take advantage of all KNL features**
  - Except AVX-512 (requires recompiling)
  - Except moving selected data to MCDRAM (requires source changes)
  - Optimal instruction selection and organization is different
- **Recompiling will probably improve performance**
  - HPGMG-FV - High-Performance Geometric Multi-Grid benchmark
  - Run on 64 KNL nodes, 64 cores per node, quad/cache
  - CCE 8.5, craype-sandybridge: 1.264 billion DOF/s
  - CCE 8.5, craype-haswell: 1.447 billion DOF/s
  - CCE 8.5, craype-mic-knl: 1.866 billion DOF/s

Acronym and Terminology Reference

- **DDR - Double Data Rate**
  - Refers to the 6 channels of DDR4-2400 DIMM main memory
- **MCDRAM - Multi-Channel DRAM**
  - High-bandwidth on-package memory
- **MCDRAM Cache**
  - MCDRAM configured as a last-level memory-side cache
- **Flat MCDRAM**
  - MCDRAM configured as addressable memory
  - User-visible as a NUMA node with memory but no cpus
- **EDC - Embedded DRAM Controller**
  - Interface to MCDRAM, 8 controllers per processor
- **Tile - A logic block including two cores sharing an L2 cache**
  - Includes an on-chip mesh interface and CHA
- **CHA - Caching Home Agent**
  - Per-tile block which manages cache coherence (L2 and MCDRAM)
- **MC or IMC - Integrated (DDR) Memory Controller**
- **OPIO - On-Package I/O**
  - Interface from KNL processor to MCDRAM
- **HBM - High Bandwidth Memory**
  - HBM is a memory hardware technology developed by AMD and partners
  - Sometimes used informally to refer to flat MCDRAM on KNL
- **VPU - Vector Processing Unit**
  - AVX-512 SIMD execution unit, 2 per core
- **SNC - Sub-NUMA Cluster**
  - Processor mode which divides memory capacity and bandwidth into 2 or 4 NUMA nodes per memory type
  - Also divides the cores and MCDRAM cache among the DDR NUMA nodes

Core to Core: Comparing Xeon Phi to Xeon

Feature	Haswell	Knights Landing	How KNL compares
Number of cores	16	68	A lot more cores (4X)
Core frequency	2.3 to 3.6 GHz	1.4 to 1.6	Lower frequency (2X)
Serial scalar rate	Lorenz=3048	874	3.5X slower
L1 cache size	32KB	32KB	Same
L1 load bandwidth	2X 32 bytes	2X 64 bytes	Higher per cycle (2X)
L1 load rate	7 billion/sec	3 billion/sec	Same per cycle, but lower clock
L2 cache size	256KB	1MB/2 cores	Much larger (2X per core)
L2 bandwidth	64 bytes/cyc	64 bytes/cyc	Same per cycle, but lower clock
L3 cache size	2.5 MB/core	N/A	Many kernels bandwidth limited

Node to Node: Comparing Xeon Phi to Xeon

Feature	Haswell	Knights Landing	How KNL compares
Number of cores	32	68	More cores (2X)
DDR	8 channels	6 channels	25% less bandwidth, capacity
MCDRAM	N/A	8 channels, 16 GB	Unique feature
Memory Bandwidth	~120 GB/s	490 GB/s	MCDRAM rate (4X)
FP Peak (vector)	~1.2 TF/s	~2.6 TF/s	Higher peak (2X)
FP Peak (scalar)	294 GF/s	326 GF/s	Slightly higher
Instruction Peak	387 Ginst/s	190 Ginst/s	Half peak rate
Package Power	270 W	215 W	Less power

KNL Memory Modes

- **Also known as MCDRAM configuration**
- **Memory Mode describes how MCDRAM is used**
  - As memory-side cache, as addressable memory, or some of both
- **Cache Mode**
  - All MCDRAM configured as direct-mapped cache
- **Flat Mode**
  - All MCDRAM configured as addressable memory
  - Only mode latency-optimized for DDR
  - Cores are associated with DDR NUMA node
- **Hybrid Mode**
  - MCDRAM split between addressable and cache
  - Allowed ratios are: 75%:25% or 50%:50% (aka split and equal)
- **Configured at node boot time**

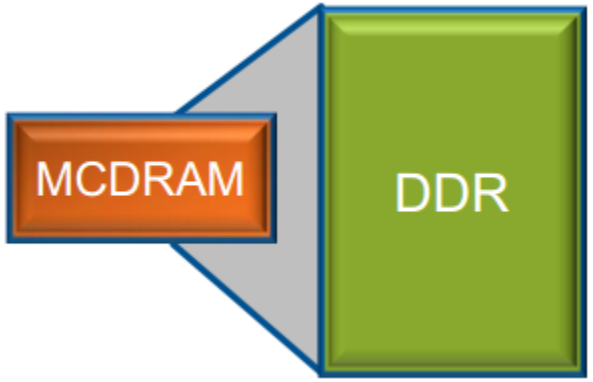
KNL Memory Modes

Flat

MCDRAM is NUMA node 1

DDR is NUMA node 0

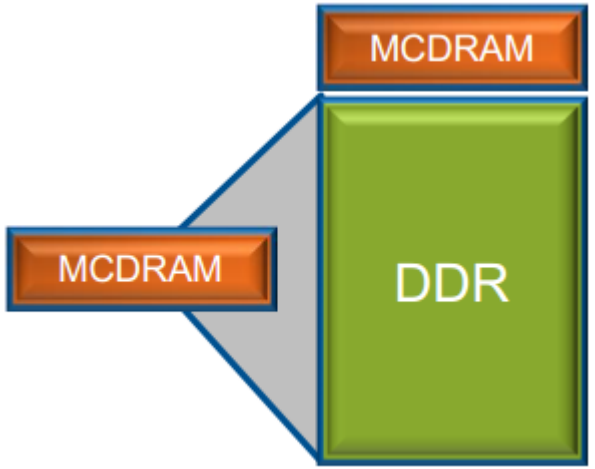
Cache



MCDRAM acts as memory-side cache for DDR

DDR is NUMA node 0

Hybrid



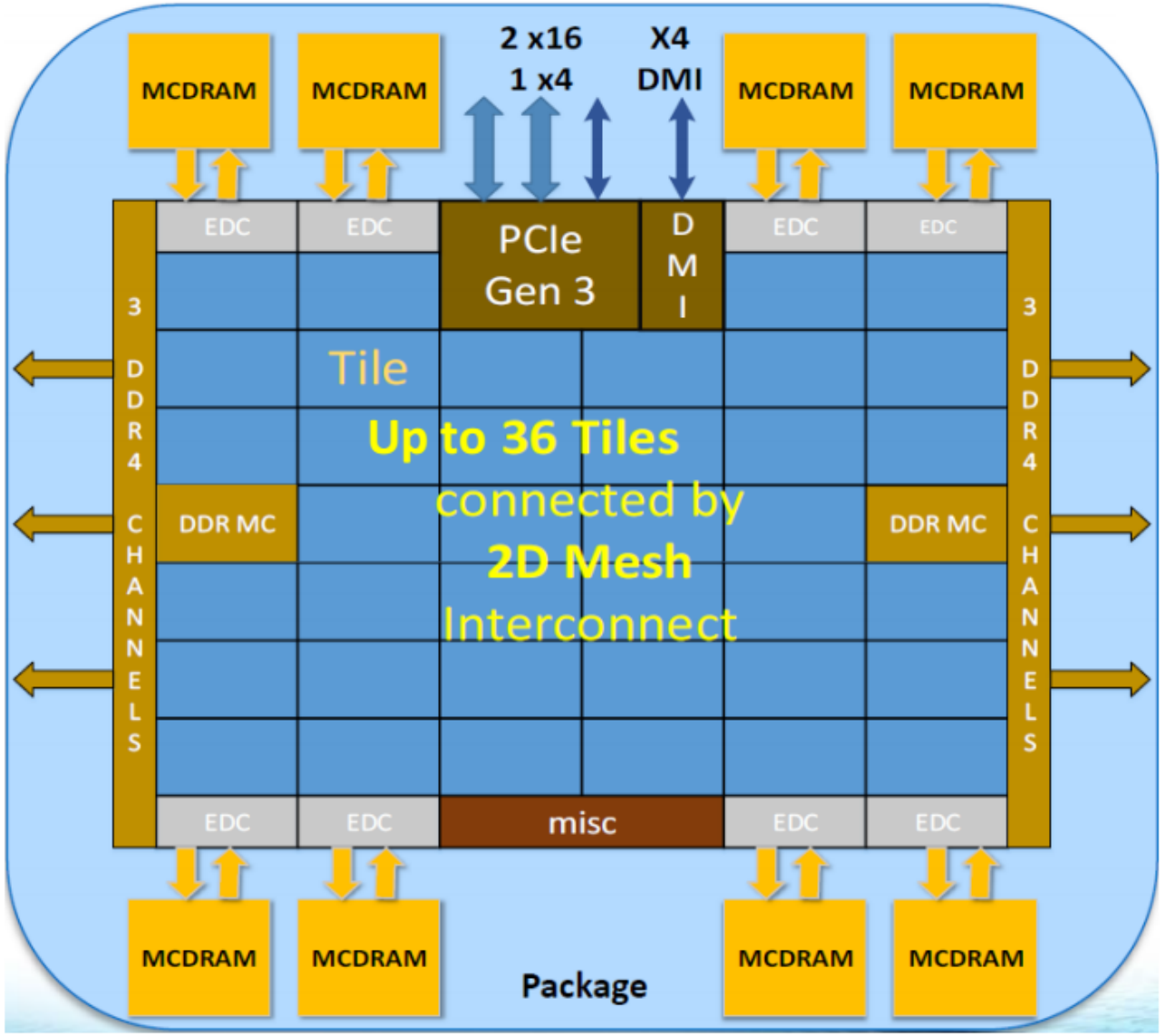
Part of MCDRAM is cache, part is NUMA node 1

DDR is NUMA node 0

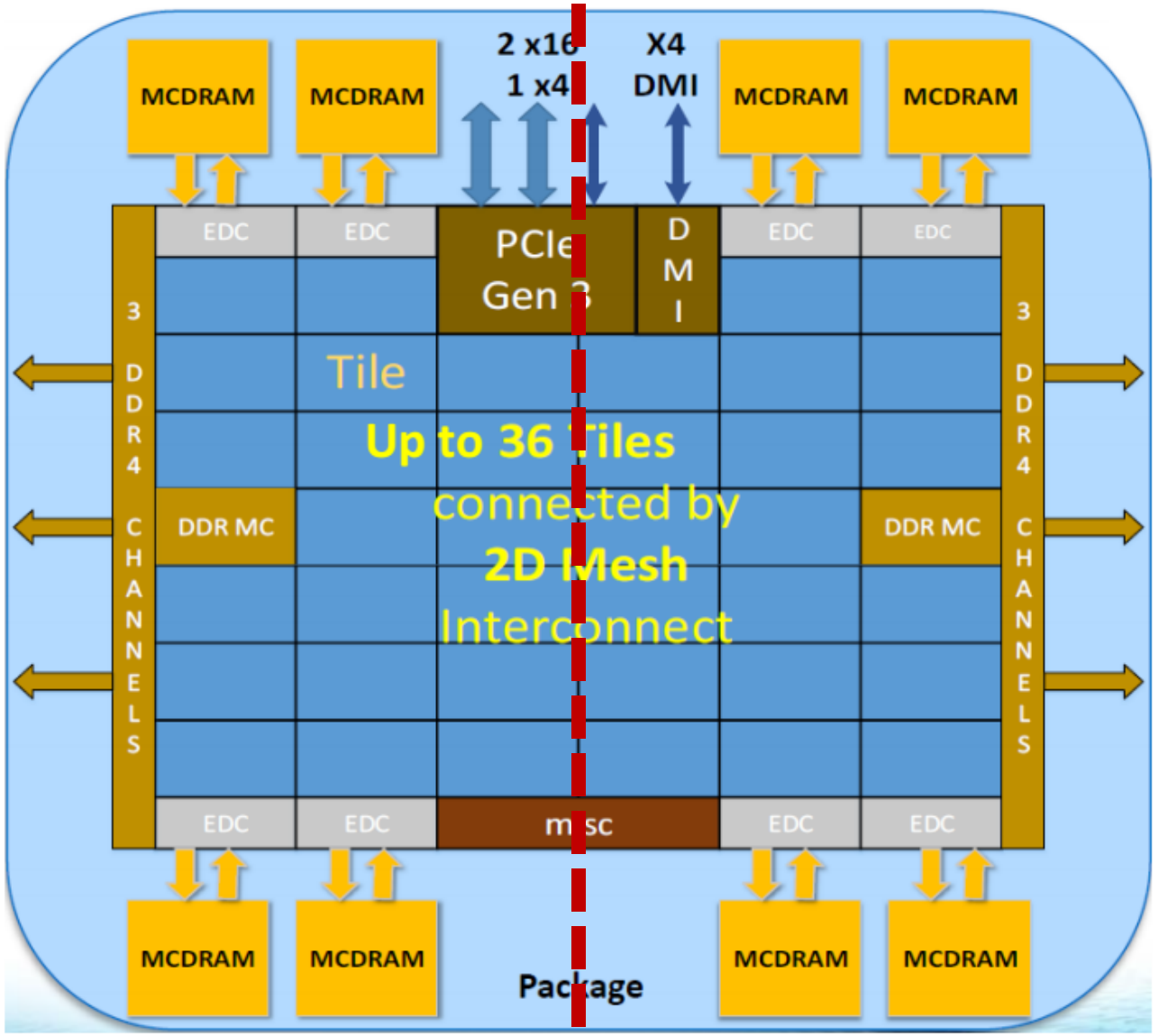
KNL Cluster Modes

- **Also known as NUMA configuration**
- **Cluster Mode sets cache coherency configuration**
  - Each tile has a caching home agent (CHA)
- **Quadrant Mode**
  - CHAs divided into 4 groups with locality to memory controllers
  - Each page of memory striped over all CHAs
  - Also Hemisphere mode to divide tiles into halves
  - And AlltoAll mode, with no locality between CHAs and memory
- **Sub-NUMA Clustering (SNC) mode**
  - SNC2, Memory and tiles are divided into NUMA domains
    - 2 NUMA nodes for each addressable memory type (DDR4, MCDRAM)
    - Tiles (cores) divided between the DDR domains
    - MCDRAM cache also partitioned
  - Also SNC4 mode to divide memory into quarters
- **Configured at node boot time**

In Quadrant mode, all tiles are in NUMA node 0



With SNC2, tiles are split between NUMA nodes



NUMA and MCDRAM Configuration

- **Node configuration selected by user**
- **If no configuration specified, any node is allocated**
- **Managed by the workload manager**
  - If enabled, WLM can initiate node reconfiguration
  - Some or all of the nodes reserved for the job may be rebooted
  - About 20 minutes delay before job starts execution
- **NUMA configuration**
  - {a2a, hemi, quad, snc2, snc4}
- **MCDRAM configuration**
  - {cache, equal, split, flat}
  - equal is 50% cache, split is 25% cache

SLURM Node Configuration

- **Selected by --constraint option (or -C)**
  - On sbatch, salloc, or srun
  - --constraint={numa},{mcdram}
- **Normally specified together, but not required**
  - --constraint=quad,cache
- **Other syntax is also allowed**
  - See --constraint in the sbatch man page
- **sinfo with active features output shows configurations**

```
$ sinfo -o "%D %t %b"
NODES  STATE ACTIVE_FEATURES
4  alloc# quad,flat
152  idle cache,quad
```

  - alloc# means “allocated but rebooting”
- **Job time limit does not begin until nodes are rebooted**

SLURM CPU Affinity (1)

- **Default CPU affinity depends upon run options**
  - "auto binding"
  - Depends upon --ntasks-per-node and --cpus-per-task
  - Enabled if ntasks-per-node * cpus-per-task equals
    - number of sockets
    - number of cores
    - number of threads
  - Otherwise, no cpu affinity is set
- **--cpu_bind can be used to set cpu affinity**
  - Recommended for Cori
  - --cpu_bind=cores (task affinity to hardware cores)
  - --cpu_bind=threads (task affinity to hardware threads)
  - For KNL, 4 hardware threads per core

SLURM CPU Affinity (2)

- **SLURM does not provide CPU affinity for software threads**
  - SLURM sets a cpu mask for each process (MPI rank)
  - By default, threads can float among the cpus in the mask
  - This is similar to “aprun -d XX -cc depth”
- **Use OMP_PROC_BIND and OMP_PLACES**
  - OMP_PROC_BIND={true,close,spread}
  - OMP_PLACES={cores,threads}
  - OMP_PLACES=cores for behavior like “aprun -j 1 -cc cpu”
  - OMP_PLACES=threads for “aprun -j {2,3,4} -cc cpu”
- **For OMP_NUM_THREADS > 4 and --hint=multithread**
  - Use --cpus-per-task to get all threads of cores needed
    - e.g., use --cpus-per-task=8 and OMP_NUM_THREADS=4 for 2 cores, 2 threads per core
  - Set OMP_PROC_BIND=true

Compilers

- **With Cray PE: module load craype-mic-knl**
  - Targets KNL instruction set
  - AVX-512
  - Will only run on KNL nodes
- **Without craype-mic-knl, use compiler flags for KNL**
  - Intel: -xMIC-AVX512
  - CCE: -h cpu=mic-knl
  - GCC: -march=knl
- **Intel 16 has KNL support**
- **Intel 17 has additional KNL enhancements**
- **CCE 8.5 has initial KNL support**
- **CCE 8.6 has enhancements**
- **-march=knl was introduced with GCC 5.1**

Vectorize for best performance

- **Take advantage of AVX-512**
  - 8 results per instruction versus 1 result per instruction
- **Helps compensate for low core frequency**
  - More work each cycle with vectorized code
- **Helps compensate for low instruction dispatch width**
  - 2 instructions/cycle
- **Vectorization techniques the same as Xeon**
  - Use CCE loopmark listing or compiler messages
    - -rm (Fortran) or -h list=m (C)
    - -h msgs
  - Use Intel compiler reporting
    - -qopt-report
    - -qopt-report-phase=vec
- **Vectorization helps if data is accessible**
  - Contiguous memory access (stride-1)
  - Cache block for L2

Strategies For Using High-Bandwidth MCDRAM

- **If 16 GB is big enough**
  - Configure as flat, use default MCDRAM
  - numactl --membind=1
- **If data touched within key loops is known, < 16 GB**
  - Configure flat, use default DDR and memkind for MCDRAM
  - hbw_malloc / hbw_free
  - For Intel: !DIR$ ATTRIBUTES FASTMEM
  - For CCE: !DIR$ MEMORY(BANDWIDTH)
  - Or try numactl --preferred=1
- **Otherwise**
  - Configure as cache
  - Can do both and configure as equal or split
  - With equal and split, memory bandwidth is reduced

Using numactl to use MCDRAM

- **To get all memory allocated in MCDRAM, use numactl**
  - Configure MCDRAM as flat
  - MCDRAM will be NUMA node 1
  - Per-node memory limit is 16 GB (MCDRAM size)
  - Run using numactl

```
srun -n 320 --ntasks-per-node=64 numactl --membind=1 a.out
```

- **To use MCDRAM first, but overflow into DDR**
  - Configure flat as above
  - Use --preferred=1 instead of --membind=1
  - Per node memory limit includes all memory (MCDRAM+DDR)
  - Often does not help much
    - Only first allocations will be to MCDRAM
    - Later allocations will overflow into DDR

Memory Allocation Examples

Allocate from DDR

```
float   *fv;
fv = (float *)malloc(sizeof(float) * 1000);
```

Allocate from MCDRAM

```
float   *fv;
fv = (float *)hbw_malloc(sizeof(float) * 1000);
```

Allocate arrays from MCDRAM & DDR in Intel Fortran

```
c      Declare arrays to be dynamic
       REAL, ALLOCATABLE :: A(:), B(:), C(:)
!DIR$ ATTRIBUTES FASTMEM :: A
       NSIZE=1024
c
c      allocate array 'A' from MCDRAM
c
       ALLOCATE (A(NSIZE))
c
c      Allocate arrays that will come from DDR
c
       ALLOCATE  (B(NSIZE), C(NSIZE))
```

CCE Memory Allocation Examples

Allocate from MCDRAM in CCE C

```
#pragma memory(bandwidth)
float *fv = (float *)malloc(sizeof(float) * 1000);
```

Allocate from MCDRAM in CCE C++

```
#pragma memory(bandwidth)
float *fv = new float[1000];
```

Allocate arrays from MCDRAM & DDR in CCE Fortran

```
c      Declare arrays to be dynamic
       REAL, ALLOCATABLE :: A(:), B(:), C(:)
       NSIZE=1024
c
c      allocate array 'A' from MCDRAM
c
!DIR$ MEMORY(BANDWIDTH)
       ALLOCATE (A(NSIZE))
c
c      Allocate arrays that will come from DDR
c
       ALLOCATE  (B(NSIZE), C(NSIZE))
```

MCDRAM as Cache - Cache Conflicts

- **Direct mapped cache, DDR is 6 times larger**
  - 6 DDR pages map to the same locations in MCDRAM cache
- **Two physical pages which conflict will cause thrashing**
  - Performance impact depends upon how often this occurs
- **Conflicts cause cache misses, lower performance**
  - Performance limited by DDR, not MCDRAM cache bandwidth
- **Most often seen when using a large number of nodes**
  - Chance of any node having a conflict increases
  - Physical pages assigned independently on each node
  - A page sorting feature in the OS reduces randomness

Alternative MCDRAM Strategies - Reverse NUMA Binding

- Makes MCDRAM default, but places selected arrays in DDR
  - Works when a few large arrays are not part of the main compute
- Add hbwmalloc calls or directives on data to be placed into DDR
- Configure MCDRAM as flat
- export MEMKIND_HBW_NODES=0
- Launch program using “numactl --membind=1”
- Can be combined with the autohbw strategy as “reverse autohbw”

Alternative MCDRAM Strategies (2) - Autohbw

- **Place allocations into MCDRAM without source changes**
  - Selected by allocation size

  export AUTO_HBW_SIZE=min_size[:max_size]
  - Run the program with MCDRAM configured as flat
  - Can be combined with reverse NUMA binding
- **Cori: module load autohbm**
- **On other systems, build memkind from source**

  https://github.com/memkind/memkind
  - Link with autohbw:

  -L $MEMKIND_DIR/memkind/lib -Wl,--whole-archive -lautohbw -Wl,--no-whole-archive
- **For prelinked dynamic executables, use LD_PRELOAD**

  export LD_PRELOAD=$MEMKIND_DIR/lib/libautohbw.so

  https://github.com/memkind/memkind/blob/dev/autohbw/autohbw_README

With CCE OpenMP, try wait policy passive

- **Many programs run faster with OpenMP and 2-way HT**
  - export OMP_NUM_THREADS=2
  - srun --ntasks-per-node=64 -c 4 --cpu-bind=cores
- **With active wait policy, worker threads spin-wait when idle**
  - Spin-waiting consumes instruction issue bandwidth
- **With passive wait policy, worker threads halt when idle (after a short spin-wait)**
  - CCE default wait policy is active when ncores=nthreads
  - CCE default wait policy is passive when ncores<nthreads
  - Intel default wait policy is passive
- **To reduce issue bottleneck, set passive wait policy**
  - export OMP_WAIT_POLICY=passive
  - May also increase OpenMP parallel overhead due to thread wakeup time when executing many small parallel regions

SLURM CPU affinity recipes for OpenMP

- export OMP_NUM_THREADS=xx
- export OMP_PROC_BIND=true
- export OMP_PLACES=threads

- srun -n yyy -c xx --threads-per-core=1 a.out
  - For one thread per core, xx OpenMP threads per process

- Same environment settings can be used on ALPS systems
  - Useful when “aprun -cc depth” is used.

Use Core Specialization

- **MPI synchronization time can rise due to OS noise**
  - Sync time is time until the last process enters the collective
  - Usually when a large number of nodes is used
- **May show up as large times for collectives**
  - MPI_Allreduce
- **To reduce OS noise by 50%, use core specialization**
  - Impact on collective calls even larger than 50%
- **Core specialization reserves hardware to handle OS work**
  - Highest numbered available cpu is selected first
- **SLURM: srun --thread-spec=1**
  - Can also set SLURM_THREAD_SPEC=1

For best performance, avoid dynamic linking

- **Dynamically linked executables usually run slower**
  - More overhead for library calls
  - Not all libraries are performance-critical
- **Dynamic linking penalty mainly impacts**
  - Calls of short duration
  - Which occur frequently
- **Use of memkind requires dynamic linking**
  - Work-around is to build a local static memkind library
- **Consider static linking for these types of libraries**
  - Compiler runtime (intrinsics, pattern-matched code)
  - Math libraries
  - OpenMP runtime
  - Memory allocation (if occurs frequently)
  - MPI (if many small messages or if latency is important)

KNL Features and Usage Summary

- More cores - 68 per processor
- Wider SIMD instructions - AVX-512
- 4 hardware threads per core
- L2 cache is shared between two cores
- Can execute Xeon programs
- High-bandwidth memory - MCDRAM
- MCDRAM is often used as a 16GB cache

Using Cray Systems with Knights Landing processors

Questions?

Bonus Slide - Additional Resources

Using Cori - from the November 2016 NESAP Hackathon
https://www.nersc.gov/assets/Uploads/Using-Cori-20161129-NESAP-HACKATHON.pdf

Cori Intel Xeon Phi Nodes
http://www.nersc.gov/users/computational-systems/cori/cori-intel-xeon-phi-nodes/

Example batch scripts for Cori KNL Nodes
https://www.nersc.gov/users/computational-systems/cori/running-jobs/example-batch-scripts-for-knl/

PRACE Best Practice Guide – Knights Landing, January 2017
http://www.prace-ri.eu/best-practice-guide-knights-landing-january-2017/

Explicit Vector Programming – Best Known Methods
https://software.intel.com/en-us/articles/explicit-vector-programming-best-known-methods

Bonus Slide - Core Block Diagram

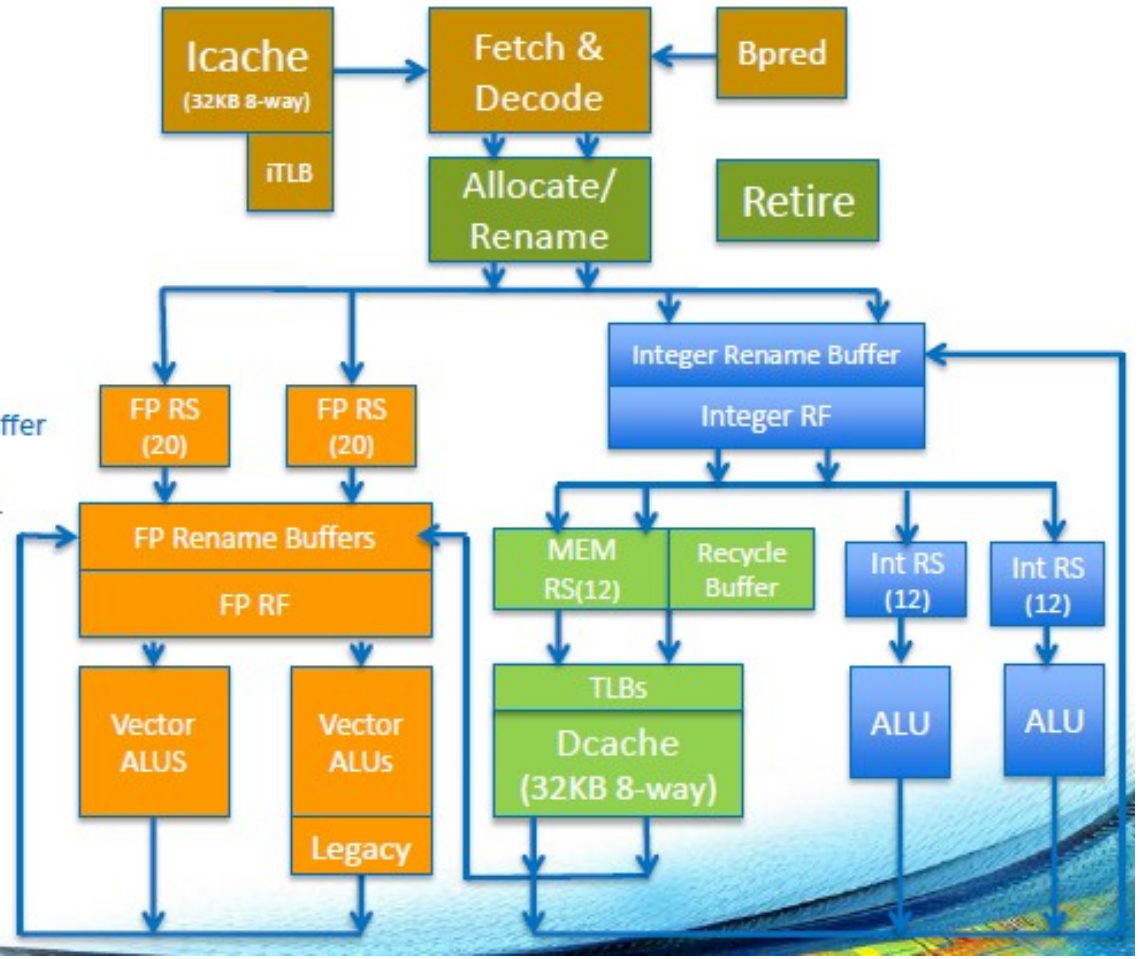
Core & VPU

- Out-of-order core w/ 4 SMT threads
- VPU tightly integrated with core pipeline

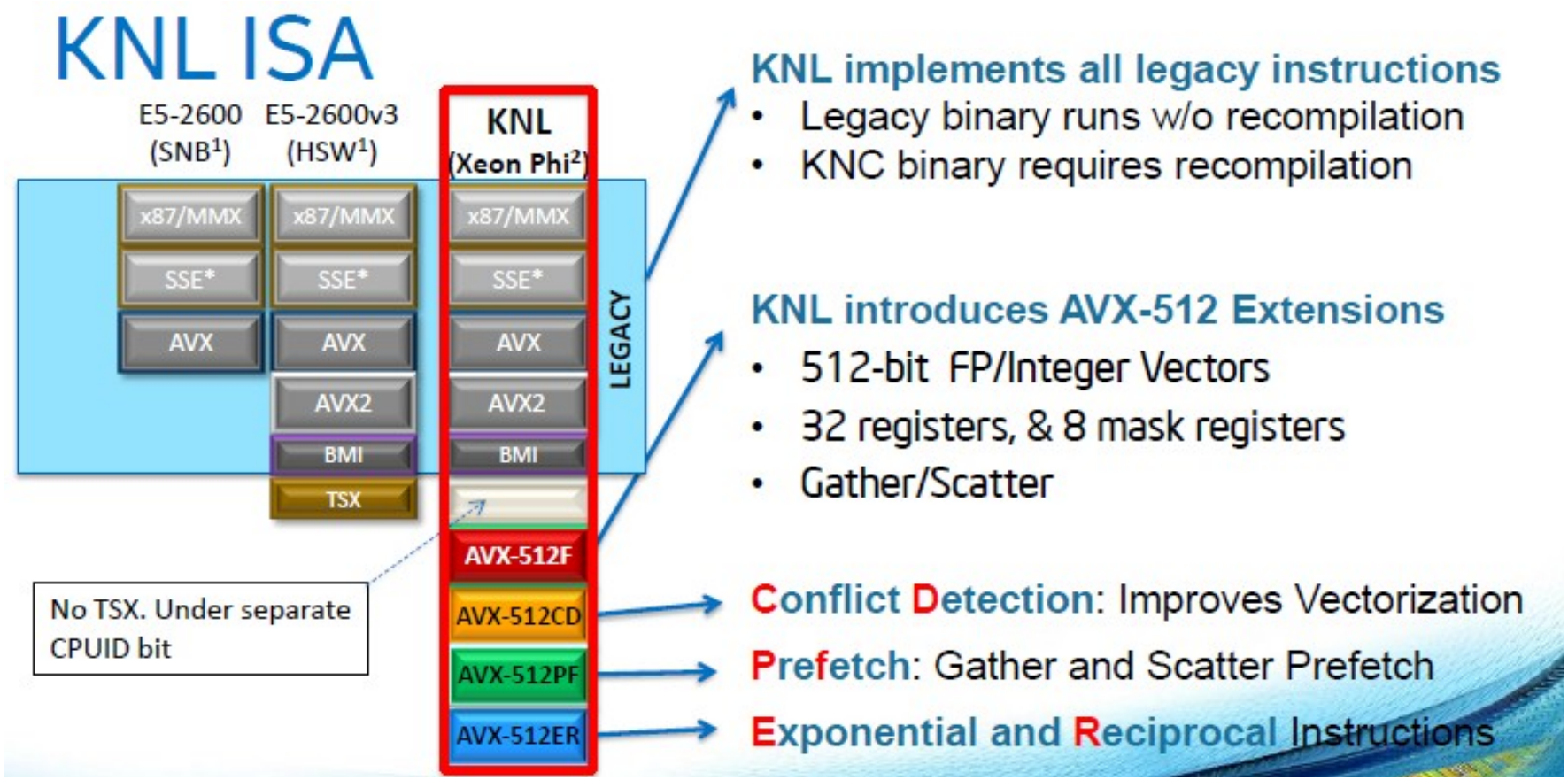
- 2-wide Decode/Rename/Retire
- ROB-based renaming. 72-entry ROB & Rename Buffers
- Up to 6-wide at execution
- Int and FP RS OoO.
- MEM RS inorder with OoO completion. Recycle Buffer holds memory ops waiting for completion.
- Int and Mem RS hold source data. FP RS does not.

- 2x 64B Load & 1 64B Store ports in Dcache.
- 1st level uTLB: 64 entries
- 2nd level dTLB: 256 4K, 128 2M, 16 1G pages

- L1 Prefetcher (IPP) and L2 Prefetcher.
- 46/48 PA/VA bits
- Fast unaligned and cache-line split support.
- Fast Gather/Scatter support



Bonus Slide - Instruction Set Architecture



Bonus Slide - AVX-512 Extensions

- Intel AVX-512 Prefetch Instructions (PFI)
- Intel AVX-512 Exponential and Reciprocal Instructions (ERI)
- Intel AVX-512 Conflict Detection Instructions (CDI)

CPUID	Instructions	Description
AVX512PF	PREFETCHWT1	Prefetch cache line into the L2 cache with intent to write
	VGATHERPF{D,Q}{0,1}PS	Prefetch vector of D/Qword indexes into the L1/L2 cache
	VSCATTERPF{D,Q}{0,1}PS	Prefetch vector of D/Qword indexes into the L1/L2 cache with intent to write
AVX512ER	VEXP2{PS,PD}	Computes approximation of 2^x with maximum relative error of 2^{-23}
	VRCP28{PS,PD}	Computes approximation of reciprocal with max relative error of 2^{-28} before rounding
	VRSQRT28{PS,PD}	Computes approximation of reciprocal square root with max relative error of 2^{-28} before rounding
AVX512CD	VPCONFLICT{D,Q}	Detect duplicate values within a vector and create conflict-free subsets
	VPLZCNT{D,Q}	Count the number of leading zero bits in each element
	VPBROADCASTM{B2Q,W2D}	Broadcast vector mask into vector elements

Bonus Slide - AVX-512 Extensions