Preparing to Optimize for Intel Xeon Phi "Knights Landing"

Steven Warren

swarren@cray.com

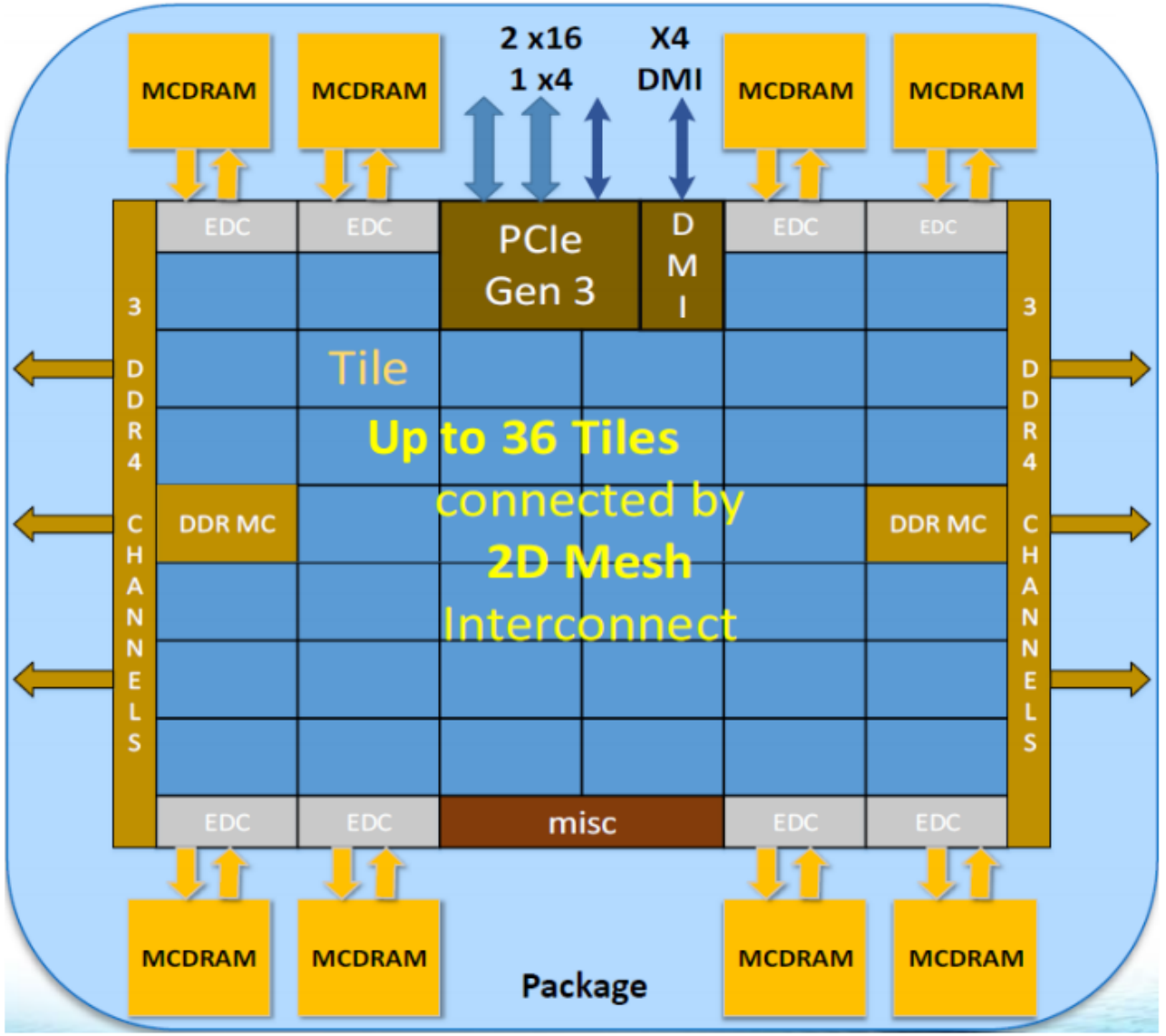
Discussion topics

- Processor architecture overview
- Notes about MCDRAM in cache-mode
- Preparing to optimize your application for KNL
  - Target Science
  - Scaling and Communication
  - Memory and cache footprint analysis
  - Creating a test case

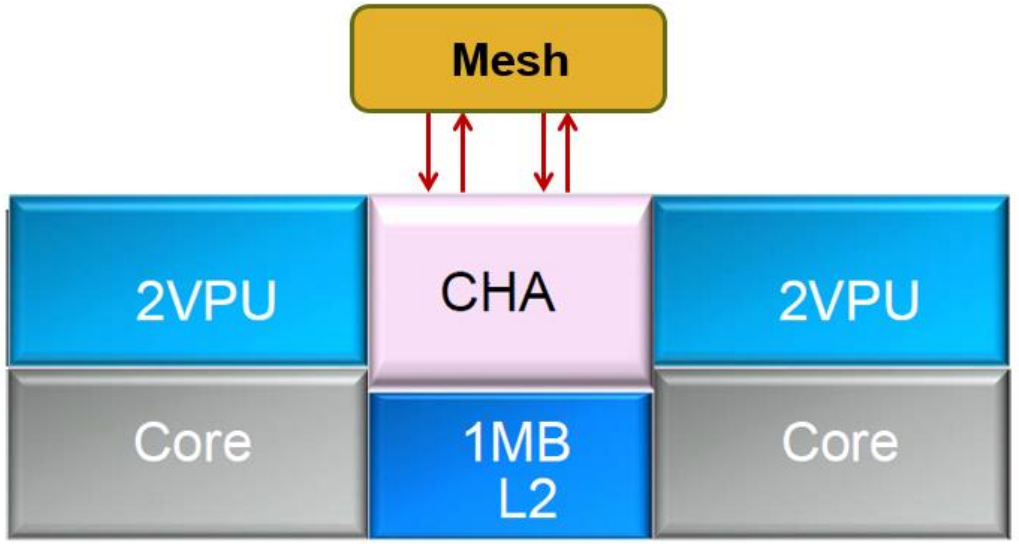
Intel Xeon Phi “Knights Landing”

Processor architecture overview

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 in flat mode (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

A closer look at using MCDRAM in cache mode

MCDRAM

- MCDRAM is a configurable memory that is smaller than main memory but has much higher bandwidth
  - Can be configured as a “memory cache”
  - Can be configured at 100% explicitly managed (Flat mode)
  - Can be configured 50% cache and 50% explicitly managed
- MCDRAM can sustain 300-450 GB/s of bandwidth
  - KNL DRAM can sustain only about 90 GB/s
  - Haswell DRAM can sustain about 120 GB/s
- When configured Flat mode the programmer must decide what goes into MCDRAM and what does not
- When configured as a cache, the hardware attempts to keep most recently used data in the cache
  - However the cache is a “direct-mapped” cache

Basic picture of memory and a direct mapped cache

- There are 6 memory locations that map to the same location in cache
  - Assumes a 96 GB DRAM config. Larger memory will have more
- Cache is direct mapped, i.e., there is only 1 “way”

Pros of using MCDRAM as a cache

- Because the hardware is automatically bringing data into the MCDRAM, the user does not have to do anything to the code to start benefiting from MCDRAM

- The MCDRAM is much larger than the Haswell caches, so some structures that never fit before suddenly will fit

- However, there is a word of caution

Start filling pages in memory

					PE3's A		
						PE5's A	
	PE4's A						
			PE2's A				
PE1's A							
$PE1's A	$PE4's A	$	$PE2's A	$	$PE3's A	$PE5's A	$

- OS starts to place pages that contain the variable "A" for each PE as those PEs reach the allocation statement
  - Each page placement is more or less "random" based on when various PEs arrive at the allocate, and the order in which the pages were free, perhaps even by a previous program

Conflicts are bound to happen

			PE6's A				
					PE3's A		
						PE5's A	
	PE4's A						
			PE2's A				
PE1's A							
$PE1's A	$PE4's A	$	$ conflict	$	$PE3's A	$PE5's A	$

- If two PE's A are aliased to the same cache location, then thrashing may occur if those PEs both reuse A at "about" the same time

Cache thrashing will occur, but how often?

- Any job that uses more than 16 GB/node is guaranteed to see some amount of cache thrashing, but how much?

- Good news: The cache is really large, so there is a lot of potential for reuse and there is a relatively low probability of a bad aliasing conflict occurring
  - Conflict probability is also a function of size of the data being reused
  - This is why we often don’t observe this on single node runs

- Bad news: All of these are NON-ZERO probabilities, and the dice are rolled on every node in the job
  - As the number of nodes in the job increases, the probability that aliasing problems will occur approaches 100%

Direct mapped cache causing scaling problems

- If your performance is impacted by the effectiveness of the MCDRAM cache, you may experience scaling problems
  - This will likely show up in communication, but will be because of synchronization, not bandwidth or latency constraints
- Normal profiling may not point to the offending compute region
  - Only a few PEs might be slow, and thus that signal could be drowned out by the other PEs in the job

MCDRAM Usage Conclusions

- Flat mode is good if your entire data set can fit into 16GB
  - However if your code uses more than 16 GB of data, it may be difficult to find just the right arrays to place there
  - Flat mode seems likely to be unforgiving if some bandwidth data does not fit
- Cache mode seems to capture reuse well on many apps
  - And it requires no work on the part of the user
  - But is susceptible to thrashing if important data aliases to the same location
  - This becomes more likely as node counts increase
- Each mode has pros and cons

Preparing to Optimize for KNL

What Science do you want to run on KNL

- Identify science problems that you anticipate running on KNL
  - The science problems will help focus efforts on what routines and issues are important
- Estimate how many nodes you will use during the run
  - Does the code already scale this high?
  - What can we say about communication
- The combination of science problem and number of nodes will allow one to estimate memory footprints, array sizes, and trip count sizes
  - This information is critical

Scaling and communication

- How high does the code scale

- Does your code use both OpenMP and MPI?
  - How many OpenMP threads can you utilize

- What is limiting your scaling?
  - Communication overhead?
  - Lack of parallelism on a given science problem?

- Understanding and optimizing scaling is critical
  - KNL requires scaling to higher numbers of cores to achieve the same level of performance
  - Scaling impacts loop trip counts, memory footprints, and more

Understanding your memory footprint is critical

- Do you expect your problem to consume a significant amount of main memory?
  - Main memory is about 96 Gbytes

- Is it possible that your problem will fit into fast memory?
  - Fast memory is 16 Gbytes per node
    - Can be configured as a “memory cache”
    - Can be configured at 100% explicitly managed

- What is the memory access pattern for the routines and loops identified as important?
  - What are the trip counts in that loop nest?
  - How much data is accessed?
  - How much is used more than once?

Create test case that represents a real science run

- Use all of the information about your target science problem to develop a test case that can be optimized
- Want that test case to be as representative as possible, but without using 100s of nodes
- Adjust time step if possible, not problem size
  - Want to capture the memory footprint, bandwidth and scaling attributes but still limit run time
- Should use multiple nodes, 4-32 nodes might be ideal
  - If you have communication, you want to make sure that behavior is represented in the test case
  - You want to run on enough nodes to capture some communication and scaling characteristics, but few enough to allow for more rapid turn around and not burn up allocation

Where is the time being spent

- Are you sure? Verify
  - Cray has come across many examples where performance was limited by something in some place that was not expected
- Use statistical profilers to determine where the time is being spent
  - Are there obvious key routines using a significant percentage of run time?
  - Are there key loops or code sections?
  - How many routines before you hit 80% of the run time
- Is the profile different for different science problems?
- If you start heavy optimization efforts before you get a representative profile you risk wasting a significant amount of your time and effort

Vectorization

- Do the loops vectorize?
- Vectorization is very important to achieving high performance rates
  - Edison vectors are 4 DP words, KNL has 8 DP words
  - Cannot take full advantage of functional units without vectorization
  - Unlikely to take full advantage of memory bandwidth
  - Scalar performance on KNL core is approximately 1/3rd the speed of a Haswell core
- Common inhibitors
  - Dependencies
  - Indirect addressing may prevent vectorization or make it less efficient
    - i.e., A(indx(i)) =
  - Function / subroutine calls
  - “IF” tests inside of inner loops may slow execution and prevent vectorization
  - More…

How can you tell if you are memory bandwidth bound?

- Sometimes it is easy
  - One or more loop nests are streaming through a huge amount of data
  - Little to no reuse
- Sometimes it is difficult
  - Some trip counts are large
  - But some data are reused
  - Not obvious what the compiler did
  - Not obvious if the data remains in cache
- Counters can be difficult to interpret
  - Difficult to keep track of different levels of cache
- Try to run kernel using 1 or 2 fewer cores
  - Adjust the number of OMP threads
  - Use srun --ntasks-per-socket= option to spread mpi ranks across more sockets
  - If performance per core increases, kernel may be bandwidth bound
- Try and examine trip counts and reference patterns

Summary

- Identify the target science problem and the number of nodes you plan on using on KNL
- Understand your memory footprint and how to utilize MCDRAM
- Create a representative test case that runs on multiple nodes
- Verify where the time is being spent using a statistical profiler
- Vectorization and Memory bandwidth optimizations are likely to be your primary means of compute-based optimizations

THANK YOU

QUESTIONS?

swarren@cray.com

cray.com

@cray_inc

linkedin.com/company/cray-inc-/