Preparing to optimize for KNL

Section Outline

- **Characterization and Multi-node Considerations**
  - Target Science
  - Profiles and Hotspots
  - Scaling and Communication
  - Memory and cache footprint analysis
  - Creating a test case
- **Single node optimizations**
  - Vectorization
  - Memory bandwidth requirements

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
- **Understand 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 to 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 that time up a significant percentage of 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 is less efficient
    - e.g. A(indx(i)) =
  - Function / subroutine calls
  - If tests inside of inner loops may slow execution and prevent vectorization
  - More…

Are your kernels memory bandwidth bound

- **Do you expect to 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 MCDRAM memory**
  - Fast memory is 16 Gbytes per node
    - Can be configured as a “memory cache”
    - Can be configured 50% cache and 50% explicitly managed
    - 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 reused more than once?

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 single node optimizations**