# TACC Technical Report TR-16-03

# KNL Utilization Guidelines

Document Revision 1.0
November 13, 2016

C. Rosales, D. James, A. Gómez-Iglesias, J. Cazes, L. Huang, H. Liu, S. Liu and W. Barth
carlos@tacc.utexas.edu

Texas Advanced Computing Center
The University of Texas at Austin
www.tacc.utexas.edu

# Executive Summary

This work presents a set of recommendations for running effectively on Intel's second-generation Xeon Phi Knights Landing (KNL) processors. Unlike its predecessor, the KNL we tested is a stand-alone, self-booting processor. This makes it very easy to build and run software originally developed on other processors. The KNL features up to 72 low-power cores running at reduced clock rates. Thus, achieving high performance on the KNL depends on exposing a high degree of parallelism.

The practices we recommend include the following:

- **Target AVX-512:** Compile your code using `-xMIC-AVX512`.
- **Parallelize:** Parallel codes may scale very well with increasing core count. Serial codes will likely perform at levels consistent with the KNL's reduced clock rate.
- **Vectorize:** Vectorized codes may achieve up to 16x double precision speedup.
- **Use MCDRAM:** Unless your code is latency bound use MCDRAM. If total memory is less than the available MCDRAM, this happens automatically. In flat mode, launch your code with launch your code with `numactl --membind=1 <exe>`.
- **Manage thread count:** Optimal performance typically occurs at 1-2 threads per core.
- **Manage MPI task count:** A minimum of 2 MPI tasks per KNL helps achieve OPA bandwidth saturation.
- **Leave 2-4 cores free:** This helps manage OPA traffic and improves scalability.

# 1 Introduction

This work presents a set of recommendations to run effectively on the processors from the second generation of Intel Xeon Phi (Knights Landing, or KNL). These recommendations have been chosen after extensive testing[1] and have broad applicability across multiple code types. In order to keep the document to a reasonable length we have chosen to give justifications for our choices without providing low level details of our tests, but much of the data has been published or presented in public venues and is referenced throughout the text. We will be happy to share additional details with any interested readers.

## 1.1 KNL Architecture

The $2^{nd}$ generation KNL is a substantial change from the first generation Knights Corner (KNC) co-processor. It has architectural features that allow it to run as a stand-alone system, removing the PCIe data exchange bottleneck of the prior generation, it is based on a significantly more powerful core, and it also has two different types of memory: MCDRAM and DDR4. The on-module Multi-Channel DRAM (MCDRAM) [12] provides the high-bandwidth memory that accelerated applications have become accustomed to; and the usual DDR4 memory provides the capacity storage many HPC applications require.

KNL cores are organized as tiles, where each tile is comprised of two cores. While L1 cache is private to the core, L2 cache is shared at the tile level. Tiles are interconnected using a mesh (as opposed to the bidirectional ring in KNC). All the cores on the chip are cache coherent, so that the tile with a specific data can supply that data to another tile in the chip. This mesh can be clustered to achieve a higher performance for specific memory access patterns in applications. The three modes of operation are: All-to-All, Quadrant and Sub-NUMA clustering [12]. KNL is configurable in two important ways at boot time, with BIOS options to select a memory mode and a clustering mode. The memory mode determines whether the fast MCDRAM operates as RAM, as direct-mapped L3 cache, or as a mixture of the two. The cluster mode determines the mechanisms for achieving cache coherency, which in turn determines latency: roughly speaking, this amounts to specifying whether and how one can think of some memory addresses as closer to a given core than others. Following Intel's recommendations and our own testing for performance and usability, the nodes in the KNL cluster's normal and development queues are configured as Cache-Quadrant (memory mode set to Cache, cluster mode set to Quadrant).

# 2 Methodology

Developing a set of guidelines for general use of KNL is not a simple endeavor. We have used a combination of microbenchmarks and applications selected from the ten most used

[1] These guidelines reflect our experience with the 508-node KNL Cluster that is now a part of TACC's Stampede system. Each Stampede Xeon Phi 7250 KNL compute node has 68 active cores per node, 16GB of high-speed on-chip Multi-Channel DRAM (MCDRAM), an additional 96GB of DDR4 RAM, and a 112GB of local solid state drive (SSD). The interconnect is a 100Gb/sec OmniPath network: a fat tree topology of eight core-switches and 320 leaf switches with 5/4 oversubscription.

scientific application in Stampede to analyze performance under a wide variety of conditions.

## 2.1 Microbenchmarks

Memory bandwidth was analyzed using STREAM [7], floating point performance using a DGEMM test with Intel MKL (Math Kernel Library), and network performance was explored using a variety of point to point and collective benchmarks. In all cases these microbenchmarks produced optimal results when using one process per logical core o less.

We determined that the sustained memory bandwidth to MCDRAM varies significantly with memory and clustering modes (from 320 GB/s to 479 GB/s). However the performance difference is much smaller when looking at full applications.

Looking at DGEMM performance (based on the threaded MKL implementation) allowed us to explore performance variability and sustained flop rates. DGEMM achieved 2.07 TF per node, which is equivalent to over 79% of the peak for minimum AVX frequency. Variability was relatively high, up to 9% when comparing runs on different nodes.

MPI performance was characterized using both the OSU MPI Benchmark [1] and the Performance Assessment Workbench [2]. Particular attention was given to single MPI task per node performance and to the effect of running tasks on every core, particularly for collective operations.

## 2.2 Codes

Five codes were chosen to perform exhaustive run configuration tests. These five codes are within the ten most used applications in Stampede with the exception of Quantum Espresso, which is a bit lower in the ranks, but has similar characteristics with other popular ab-initio codes in Stampede.

**FLASH** [10] is a multi-physics framework often used for the simulation of turbulent fluid flow and transport in plasmas, including exploding stars, laser energy deposition, and nuclear burning. The benchmark case computes a driven turbulence simulation on a fixed, $64^3$-point mesh per node. Version 4.2.2 of the code was used in the tests with minor modifications to improve threading and vectorization added to a single file in the `Stir` subdirectory.

**NAMD** [9] is a parallel molecular dynamics code for high-performance simulation of large molecular systems used by many research teams around the world and the 2002 Gordon Bell Prize winner. NAMD is also the most used application on Stampede. This study uses the APOA1 and STMV20 NAMD benchmarks with modified input for the single and multi node studies respectively. The APOA1 benchmark is a 92-thousand atom molecular dynamics simulation of Apolipoprotein A-1 which is the primary component of the high-density lipoprotein cholesterol molecule. The STMV20 benchmarks is a 20 million atom simulation of the satellite tobacco mosaic virus, which is a small icosahedral plant virus which worsens the symptoms of infection by tobacco mosaic virus (TMV). A multicore version of NAMD CVS 2016-11-01 was built for the single node test, and a hybrid memory-optimized version was built for the large STMV20 case.

**WRF** The Weather Research and Forecasting (WRF) Model [5] is a widely used numerical weather prediction system used for both research and operational forecasts. WRF is primarily a Fortran code implemented using MPI and OpenMP for distributed computing. The problem space on each process is divided into tiles that are processed by OpenMP threads. Ideally, the best performance is achieved when the size of the tile fits into the smallest cache. Having multiple application tiles allows WRF to obtain high levels of memory bandwidth utilization. A substantial effort was made to optimize WRF for the first generation Xeon Phi, Knights Corner [8]. The current version of WRF, 3.8.1, supports a configuration option for the KNC. For this investigation two benchmark cases are used, the Continental US (CONUS) 12km input is used for single node tests, and the larger CONUS 2.5km case is used for scalability analysis.

**Quantum ESPRESSO** (QE) [4] is an integrated suite of Open-source codes for electronic-structure calculations and materials modeling at the nanoscale. It is based on density-functional theory, plane waves, and pseudopotentials. For our experiments, we have chosen a medium size benchmark input for PWscf, AUSURF112. We use Quantum Espresso V5.4.0.

**GROMACS** is a versatile package to perform molecular dynamics, i.e. simulate the Newtonian equations of motion for systems with hundreds to millions of particles [3]. It is primarily designed for biochemical molecules like proteins, lipids and nucleic acids that have a lot of complicated bonded interactions. In this study, Hen egg white lysozyme (PDB code 1AKI) and water (SPC model) solutions were simulated using version 2016. The simulated systems consist a total of 140124 atoms for the single node test, and 1.8 million atoms for the scalability tests. All simulations were performed in the isothermal isobaric (NpT) ensemble at 300 K and 1 atm.

# 3 Performance Measurements

This section presents performance measurements for the selected applications on KNL, Sandy Bridge,and Haswell processors. First the single node performance numbers are introduced and then scalability data for the applications running in the Stampede KNL upgrade is discussed.

## 3.1 Single Node

We use the execution time on a single node of Stampede (Sandy Bridge – SNB) as baseline for each application. We then run the same application on a single node of Stampede KNL Upgrade (KNL) using Cache-Quadrant as the memory mode. Table 1 provides details regarding the processors.

Table 1: Processors on Tested Systems

| System | Processor | Clock (GHz) | Core Count |
|---|---|---|---|
| Stampede | Xeon E5-2680 (x2) | 2.7 | 16 |
| Stampede KNL | Xeon Phi 7250 | 1.4 | 68 |

Figure 1 shows a direct comparison of performance for each of the applications across the three systems, with each application being normalized to the value for a single Stampede node. The improvement in performance with respect to Sandy Bridge ranges from 1.3x (Gromacs) to 3.2x (WRF). It is important to note that each application has been run on the optimal configuration for each architecture, as determined by multiple runs, so that no bias in favor or against the KNL system was present. The direct average improvements with respect to Sandy Bridge is 2.1x.

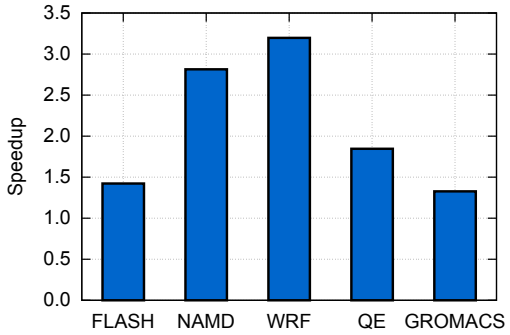
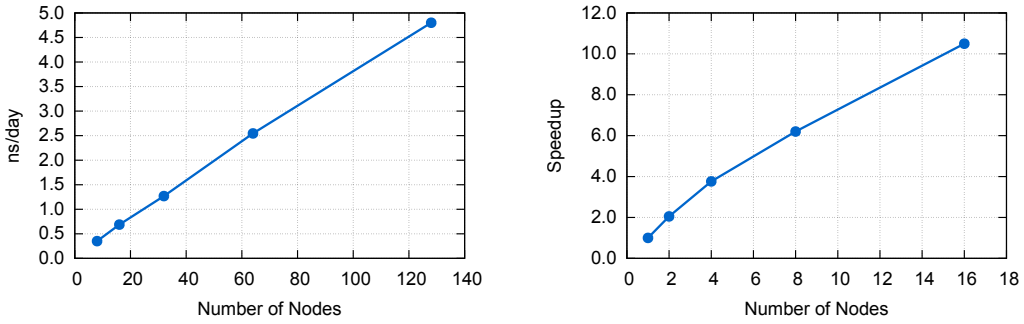



Figure 1: Application performance normalized by a single Stampede node

## 3.2 Scalability

In this section we present results for application scalability on KNL using an Omni-Path fabric. As with our microbenchmarks results we found that using more than a single MPI task per node increased performance and scalability for the applications under study, with the optimal number varying from 2 to 8 MPI tasks per node. Part of this effect is due to issues intrinsic to the code designs and part of it due to a single task not being capable of saturating the network bandwidth. Besides this detail applications scaled as expected on a 100Gb/s network, so we have chosen just NAMD and WRF as representative cases as shown in figures 2(a) and 2(b). In both cases we established that a 4 MPI task setup with 16 threads per MPI tasks provided the best results.




(a) NAMD (b) WRF

Figure 2: Scalability on the Stampede KNL Upgrade

# 4 Utilization Guidelines

This section contains a set of simple guidelines we have found to provide good performance when using KNL processors. They should be considered of general applicability, and a good starting point for anyone running codes on KNL. It should be understood that specific workloads may need some adjustment to achieve their optimal performance on KNL, just like they would in any new processor.

## 4.1 Default Memory Configuration

**Use Cache-Quadrant as a default**. Our first efforts were devoted to analyzing the performance of the different modes of the KNL processor to identify the most suitable configuration to be deployed [11]. Based on the results that we achieved, we decided to use Cache memory mode and Quadrant clustering mode as our default and largest queue. TACC's users can find in all our systems that the `normal` queue is the largest in terms of computational resources. We still offer a set of queues that allow users to analyze the performance of their applications on using different memory and clustering modes.

**Enable transparent large pages**. Some applications saw a performance decrease of a factor 5X when transparent large pages were disabled. No benefit was observed for any of the applications tested when transparent large pages were disabled.

## 4.2 Number of Threads per Core

**Use 1 or 2 threads per core**. We have found that one or two threads per core often produce optimal performance. While some codes - particularly those instruction-issue bound - may continue to increase in performance with 3 or 4 threads per core the majority of the codes tests show excellent scaling until each core is running one thread, and moderate performance improvements up to two threads per core. Increasing the number of threads to three

or four per code shows minor performance improvements in some codes and performance decrease in others, likely doe to increased conflicts in memory accesses.

## 4.3 MPI configuration

**Use a maximum of one MPI task per core**. While it is possible to run more than one MPI process on each core of a KNL processor, our experience indicates this is inadvisable. MPI overhead and memory footprint increase with the number of tasks and for manycore processors such as KNL, which provide the capability of running hundreds of simultaneous processes, the use of hybrid implementations is highly recommended.In fact, leaving a few cores free of user processes helps overall throughput. Wen running on Stampede KNL Upgrade our recommendation is to never run more than 64 MPI tasks per node.

**Use a minimum of 2 MPI tasks per node** One single MPI process per node can achieve approximately 9 GB/s of network bandwidth. However, when using 2 MPI processes the bandwidth achieved increases over 11 GB/s. This occurs because the Omni-Path Architecture makes use of CPU cores to perform many operations related to communication. Since individual KNL cores do no provide the same computational capabilities than individual cores of the Intel Xeon family, a single process is typically unable to saturate the network bandwidth. This is mitigated by increasing the number of MPI processes.

**Intel MPI configuration** Intel MPI is the default MPI library on Stampede KNL Upgrade. In order to efficiently use the Omni-Path Fabric and to reduce intranode communication we have setup the environment with `I_MPI_FABRICS=shm:tmi` and `I_MPI_TMI_PROVIDER=psm2`. In order to reduce the initialization time we also set `I_MPI_HYDRA_MPI_CONNECT=alltoall` as part of our default configuration.

## 4.4 Use Both Vector Units

**Recompile code targeting AVX-512**. While KNL is compatible with older instruction sets, only one of the two vector units in each core supports legacy instructions. This means that binaries compiled without AVX-512 support will potentially miss on half the floating point computational power of the processor. As with any new architecture, it is strongly advised to recompile code to target the AVX-512 instruction set. This is easily achieved with the Intel compiler by adding the flag `-xMIC-AVX512`.

## 4.5 Parallel IO

Preliminary tests of MPIIO collective write performance suggest that Lustre striping, file size, and number of aggregators (writers) affect performance in expected ways. In particular, collective write performance generally increases with stripe count until reaching a maximum, at which point it levels off. Single node write performance (a single independent stream writing to a single Lustre stripe) compared to Sandy Bridge is slightly faster than the ratio of clock speeds. Peak performance (at optimum choices of stripe count and aggregators) is somewhat slower than the ratio of clock speeds, and seems to occur at lower stripe counts

than is it does on Sandy Bridge. Like the Sandy Bridge side of Stampede, stripe size seems to have no effect on write performance on Stampedes KNL nodes.

## 4.6 Expect Increased Variability

Due to the advanced power saving features present in KNL, run times with a fully utilized hardware will show variation higher than what is normally expected on a regular Xeon processor. The hardware prefetcher may also affect performance variability. The level of variability is unlikely to be visible for production runs on scientific applications, but it could be surprising to somebody running microbenchmarks. Plan your performance studies with this effect in mind.

## 4.7 Porting Codes to KNL

While this section goes beyond the main focus of this report, there are a few elements that should be highlighted. For a good programming guide to KNL please refer to the recent book by Jeffers, Reinders and Sodani [12].

**Target AVX-512 instructions in the build** We have not seen any special requirements when porting codes to KNL. Only in the case of codes that use intrinsics[6] it is necessary, as expected, to rewrite the code to take advantage of the new AVX-512 extensions. For codes that do not use intrinsics, using the appropriate flag to generate code that uses the AVX-512 extensions is enough to use the two vector units available on each core.

**Parallelize the code** This step will be critical not only for KNL but for any upcoming HPC system, since the level of parallelism required to achieve good performance on a single node continues increasing.

**Vectorize the code** Deeper vector pipelines are becoming commonplace in the struggle to improve performance while maintaining power consumption contained. As with parallelization, this is a modification that will benefit code not only on KNL but on other platforms as well.

**Use MCDRAM** Few codes will require any source modification to achieve this. When running on Cache memory mode, MCDRAM is transparent to the end user and nothing needs to be done. Codes which can be contained in 16 GB of memory per node do not require source modification even in Flat mode, since a simple `numactl` command is sufficient to restrict memory allocation to the MCDRAM space. DDR4 is the default memory, so this must be explicitly done when launching a job, but it still requires no code changes. For workloads that can't be contained in 16 GB per node, the programmer can still use the `memkind` library to explicitly allocate the most heavily user arrays in MCDRAM to optimize performance.

While using MCDRAM will be of benefit for a large fraction of applications it must be noted that latency-bound codes should, in general, use DDR4 instead of MCDRAM.

**Knights Corner offload codes** The offload model is not available on Stampedes self-

hosted KNLs. To port a KNC offload code to KNL, begin by compiling the code with offload disabled (the `-no-offload` flag is an easy way to do this). If the code uses synchronous offload (host idle while offloaded computation takes place) this may be all that is needed. If the code uses asynchronous offload (host active while offloaded computation takes place), one approach is to use OpenMP `omp parallel section` directives. In the simplest case define two sections: one corresponding to the activity that used to take place on the KNC, the other corresponding to the activity formerly assigned to the host processor. Begin the optimization work by experimenting with thread counts for the two sections.

# 5 Final Notes

We have presented a set of general guidelines for running applications on the new Intel Knights Landing processor. We believe these guidelines to be of general applicability and of benefit to most scientific applications typically executed on HPC systems. Our performance analysis indicates that following these guidelines leads to execution times on KNL that represent a substantial improvement over both Sandy Bridge and Haswell processors.

# References

[1] OSU MPI Benchmark. http://mvapich.cse.ohio-state.edu/benchmarks/. Accessed: 2016-10-31.

[2] Performance Assessment Workbench. https://github.com/carlosrosales/paw. Accessed: 2016-10-31.

[3] H.J.C. Berendsen, D. van der Spoel, and R. van Drunen. GROMACS: A message-passing parallel molecular dynamics implementation. *Computer Physics Communications*, 91(1):43 – 56, 1995.

[4] Paolo Giannozzi et al. Quantum espresso: a modular and open-source software project for quantum simulations of materials. *Journal of Physics: Condensed Matter*, 21(39):395502, 2009.

[5] W. C. Skamarock et al. A description of the advanced research WRF version 3. Technical report, National Center for Atmospheric Research, 2008.

[6] Chris Lomont. Introduction to intel advanced vector extensions. software.intel.com/en-us/articles/introduction-to-intel-advanced-vector-extensions. Accessed: 2016-11-2.

[7] John D. McCalpin. STREAM: Sustainable memory bandwidth in high performance computers. Technical report, University of Virginia, 1991-2016. A continually updated technical report http://www.cs.virginia.edu/stream/.

[8] J. Michalakes. Optimizing weather models for intel xeon phi. Intel Theater Presentation SC'13, 2013.

[9] James C. Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D. Skeel, Laxmikant Kal, and Klaus Schulten. Scalable molecular dynamics with namd. *Journal of Computational Chemistry*, 26(16):1781–1802, 2005.

[10] David Radice, Sean M. Couch, and Christian D. Ott. Implicit large eddy simulations of anisotropic weakly compressible turbulence with application to core-collapse supernovae. *Computational Astrophysics and Cosmology*, 2(1):1–17, 2015.

[11] Carlos Rosales, John Cazes, Kent Milfeld, Antonio Gómez-Iglesias, Lars Koesterke, Lei Huang, and Jérôme Vienne. A comparative study of application performance and scalability on the intel knights landing processor. In Michela Taufer, Bernd Mohr, and Julian M. Kunkel, editors, *High Performance Computing - ISC High Performance 2016 International Workshops, ExaComm, E-MuCoCoS, HPC-IODC, IXPUG, IWOPH, P^3MA, VHPC, WOPSSS, Frankfurt, Germany, June 19-23, 2016, Revised Selected Papers*, volume 9945 of *Lecture Notes in Computer Science*, pages 307–318, 2016.

[12] A. Sodani, R. Gramunt, J. Corbal, H. S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, and Y. C. Liu. Knights landing: Second-generation intel xeon phi product. *IEEE Micro*, 36(2):34–46, Mar 2016.