# Experiences Porting Scientific Applications to the Intel (KNL) Xeon Phi Platform

Nicholas Malaya
Institute for Computational
Engineering and Sciences
201 East 24th St, Stop C0200
Austin, TX 78712-1229
nick@ices.utexas.edu

Damon McDougall
Institute for Computational
Engineering and Sciences
201 East 24th St, Stop C0200
Austin, TX 78712-1229
damon@ices.utexas.edu

Craig Michoski
Institute for Computational
Engineering and Sciences
201 East 24th St, Stop C0200
Austin, TX 78712-1229
michoski@ices.utexas.edu

Myoungkyu Lee
Institute for Computational
Engineering and Sciences
201 East 24th St, Stop C0200
Austin, TX 78712-1229
mk@ices.utexas.edu

Christopher S. Simmons
Institute for Computational
Engineering and Sciences
201 East 24th St, Stop C0200
Austin, TX 78712-1229
csim@ices.utexas.edu

## ABSTRACT

This paper presents experiences using Intel's KNL MIC platform on hardware that will be available in the Stampede 2 cluster launching in Summer 2017. We focus on 1) porting of existing scientific software; 2) observing performance of this software. Additionally, we comment on both the ease of use of KNL and observed performance of KNL as compared to previous generation "Knights Ferry" and "Knights Corner" Xeon Phi MICs [32]. Fortran, C, and C++ applications are chosen from a variety of scientific disciplines including computational fluid dynamics, numerical linear algebra, uncertainty quantification, finite element methods, and computational chemistry.

**ACM Reference format:**
Nicholas Malaya, Damon McDougall, Craig Michoski, Myoungkyu Lee, and Christopher S. Simmons. 2017. Experiences Porting Scientific Applications to the Intel (KNL) Xeon Phi Platform. In *Proceedings of Practice & Experience in Advanced Research Computing, New Orleans, Louisiana USA, July 2017 (PEARC'17),* 8 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

Accelerators such as current generation GPGPUs offer relatively high floating-point throughput and memory bandwidth with a lower relative power footprint than general-purpose compute platforms [15]. However, GPU-based acceleration commonly requires special programming constructs (e.g. NVIDIA's CUDA language [25] or OpenCL [11]) for the accelerator to work. Intel's Xeon Phi many-core architectures offer a balance with a smaller core count than GPGPUs but an x86 architecture and therefore do not need specialized programming paradigms. Existing MPI or OpenMP [27] threaded Fortran, C, or C++ codes can be compiled and run with relative ease using the Intel compilers.

The applications considered in this paper are taken from existing development efforts within the PECOS Center (http://pecos.ices.utexas.edu) at the University of Texas at Austin. This paper reports on two main areas: 1) the level of effort required in porting software applications from a variety of scientific disciplines written in commonly used procedural languages; and 2) observed performance of these applications. The applications cover Fortran, C, and C++ codes, and include an example with no prior thread-based parallelism as well as codes with existing OpenMP or MPI based threading from the following scientific disciplines: 1) computational fluid dynamics; 2) uncertainty quantification; 3) computational chemistry; and 4) finite element methods. We comment on both the ease of use and observed performance of KNL as compared to both previous generation "Knights Ferry" and "Knights Corner" Xeon Phi MICs [32] as well as traditional Intel Haswell CPUs.

The remainder of the paper is organized as follows: §2 describes the KNL hardware used as testing infrastructure, §3 describes cross compiling experiences, §4 presents results of the porting efforts for each application considered, and §5 summarizes the overall experiences.

## 2 TESTING INFRASTRUCTURE

The test hardware leveraged for this exercise were KNL nodes that form the Stampede Upgrade (Stampede-KNL) cluster. These nodes are due to be used for the upcoming Stampede 2 cluster at the Texas Advanced Computing Center in the Summer of 2017. This upgrade hardware leaves the original Stampede cluster hardware untouched and adds compute performance capabilities with the newest iteration of Intel's MIC architecture code-named "Knights Landing (KNL)".

The "Knights Landing" Xeon Phi 7250 compute nodes in the Stampede 2 cluster will boast 68 cores, each with 4 hardware threads, are bootable and each run a lightweight CentOS 7 Linux kernel. They also have 16GB of high-bandwidth Multi-Channel Dynamic Random Access Memory (MCDRAM) and 96GB of DDR4-2400.

*PEARC'17, New Orleans, Louisiana USA*

DOI: 10.1145/nnnnnnn.nnnnnnn

The login node of the Stampede-KNL cluster is a 14 core (28 thread) Haswell generation Intel Xeon E5-2695v3 processor with a clock speed of 2.30GHz. This was used to cross-compile scientific applications for the KNL compute nodes. Applications were built with version 17.0.0 20160721 of Intel's own C, C++, and Fortran compilers.

# 3 THIRD PARTY LIBRARY CROSS-COMPILING

As with many scientific research groups, application development at the PECOS center employs many open-source libraries. Fortunately, TACC's module system built on top of lmod provides many commonly used scientific packages already compiled with the necessary instructions supported by KNL. Moreover, since the Stampede 2 cluster will contain only bootable KNL nodes, the login Haswell node is not needed for offloading computation. We use it only for compiling and building software. Compilation is possible on KNL, but it is much slower.

The following software packages were compiled for the KNL MIC architecture: Poongback, QUESO, OCCA, ArcSyn3sis, CFOUR. Building these libraries for the login node environment is a well supported, common task, but building them for KNL required cross-compilation. Many of these libraries utilize a build system which supports cross-compilation. Builds that include KNL-specific instructions were configured by specifying two additional flags when compiling: -xCORE-AVX2 -axMIC-AVX512.

As an example, the following configure options were used to build version 0.55.0 of the QUESO library for KNL:

```
./configure CC=mpicc CXX=mpicxx  \
   CFLAGS="-xCORE-AVX2 -axMIC-AVX512"  \
   CXXFLAGS="-xCORE-AVX2 -axMIC-AVX512"
```

These strategies successfully built dynamic libraries and executables for KNL. Where appropriate, the kernels used for benchmarking in the subsequent sections have been publicly provided on GitHub: https://github.com/PECOS-KNL/kernels.

# 4 APPLICATIONS AND SCIENTIFIC KERNELS

The following subsections aim to highlight the two major targets of this work: 1) the level of effort required in porting software applications from a variety of scientific disciplines written in commonly used procedural languages; and 2) the observed performance of these applications. Applications are drawn from four areas of computational science:

- **Incompressible fluid dynamics**: PoongBack is a Fortran application targeted towards solving the incompressible Navier-Stokes equations in a periodic wall-bounded channel. PoongBack uses a Fourier spectral discretization in the periodic directions and a B-spline discretization in the wall-bounded direction.
- **Uncertainty quantification**: QUESO is a C++ library for the Quantifying Uncertainty in Estimation, Simulation, and Optimization. It provides a suite of algorithms for sampling unknown probability distributions and provides a parallel (MPI) environment to allow the user to leverage this in large-scale applications [7, 21, 29].
- **High order methods**: ArcSyn3sis is a C/C++ library that implements a blended isogeometric discontinuous Galerkin (BIDG) method that seamlessly blends the discontinuous Galerkin finite element method with isogeometric analysis.
- **Computational chemistry**: CFOUR is a collection of Fortran applications that leverage a suite of algorithms for solving coupled-cluster problems in quantum chemistry.

## 4.1 Incompressible DNS Application

Direct numerical simulation (DNS) plays an important role in understanding turbulent flows because DNS provides high fidelity data that is difficult to obtain experimentally. After Kim *et al.* first used the DNS for wall-bounded turbulence flow in 1987 [14], DNS has been used extensively to understand the turbulence phenomenon and to help develop models of turbulence. Turbulence is characterized by a non-dimensional parameter known as the Reynolds Number (*Re*). The vast majority of fluid flows in industry and nature occur at large *Re*. However, as *Re* increases, time and length scales become smaller. Thus, DNS at high *Re* naturally requires a fine meshes and time-steps to obtain meaningful data from flows of interest. Therefore, the use of state-of-art HPC systems is mandated to study turbulent flows of interest. To date, the highest *Re* of the wall-bounded turbulence DNS was at $\approx 250,000$, which required 242 billion degrees of freedom to resolve[18]. However, *Re* = 250,000 remains substantially lower than the *Re* for many practical engineering applications. To enable DNS at these higher *Re*, at least exascale capable HPC systems are required. These systems will require not only substantial hardware advanced, but also modifications to the underlying DNS codes.

In this section, we will show you the results of testing PoongBack on KNL nodes in Stampede, which is a channel flow DNS code optimized for the modern HPC systems. PoongBack, named after the ancient Korean god of winds, simulates a channel flow: the flow between two infinite parallel plates. The simulation code has already been scaled to several top-10 HPC systems, and has shown excellent performance during its use by several research groups[17]. In particular, it was used for generating the data for the NSF supported virtual flow laboratory in the Johns Hopkins Turbulence Data Base[10]. The code uses the Fourier-Galerkin spectral method in the streamwise and spanwise directions and a high-order basis spline method in the wall-normal direction. For time integration PoongBack uses a low-storage third-order Runge-Kutta method[33]. The simulation domain is partitioned by a two-dimensional decomposition, a.k.a. pencil-decomposition. PoongBack contains three major kernels; (1) solving the Navier-Stokes equations in a complex domain (2) one-dimensional fast Fourier transforms (3) data transpose in two dimension. The combination of (2) and (3) provides 3D FFTs and there are many existing libraries for it. We developed a customized 3D FFT library because of the needs for zero-padding for 3/2 dealiasing. Additionally, PoongBack uses a customized I/O library for the HDF-5 format, ESIO [19], which has been shown to be performant to large core counts, but the I/O performance is beyond the scope of the current work. See [17, 19] for more details about PoongBack.

For this study the grid size used was $1024 \times 128 \times 512$, which is comparable to the $Re_\tau = 180$ simulation from Kim[14]. For every

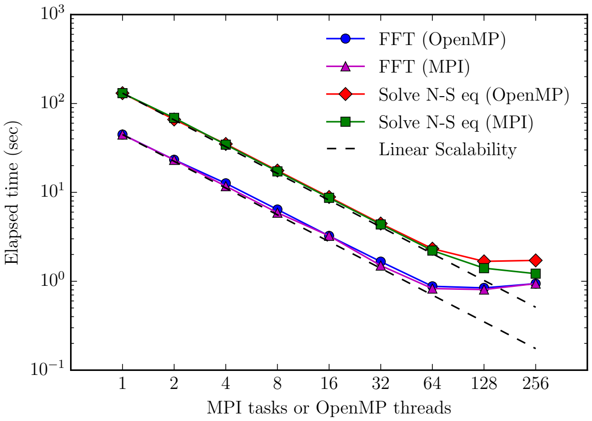

**Figure 1: Strong scaling result depicting 1D FFTs and the solution of the N-S equations in wavespace for a single timestep.**

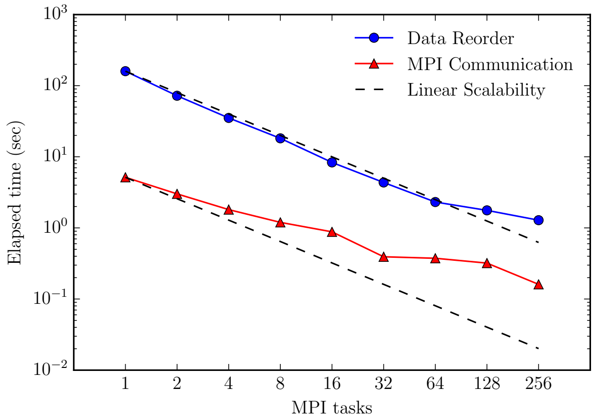

**Figure 2: Strong scaling result of data reorder and MPI communication; OpenMP is not used.**

benchmark cases, MCDRAM is used as a cache memory between processors and DRAM. The simulation code was compiled with the flag "`-xMIC-AVX512`." Also, the FFTW-3.3.5 library is installed with the options "`--enable-avx512`" and "`--enable-mpi`." [9]. We used double-precision operations for all the kernels and elapsed times were measured by "`mpi_wtime()`" with "`mpi_barrier()`."

Figure 1 shows the strong scaling performance of the kernels involving floating point operations; 1D FFTs and solving the Navier-Stokes equations. Four different FFTs are performed; real-to-complex (R2C), complex-to-real (C2R), forward complex-to-complex (fC2C) and backward complex-to-complex (bC2C). Before or after each FFT, 1d contiguous data lines are padded or truncated with zeros. Also, nonlinear products such as $A \times B \rightarrow C$ were performed between C2R and R2C transforms. The elapsed time for 1D FFTs, zero padding/truncation and nonlinear product are shown in Figure 1 in blue and magenta. In both cases using either MPI-only and OpenMP-only the performance is almost linear up to 64 processors. When two hardware threads were used per core (hyperthreading), the performance did not increase. Furthermore, the performance decreases when four hardware threads were used in both MPI and OpenMP cases. The performance was best with 64 processors and showed 270 GFlops, or approximately 9% of peak performance. This means that the performance of 1D FFTs on KNL nodes are memory bandwidth limited.

There are numerous floating point operations involved in the compute kernels for solving the Navier-Stokes equations. Particularly critical numerical operations include solving the linear system $A\mathbf{x} = \mathbf{b}$, along with matrix-vector multiplications. The matrix $A$ is a banded matrix with additional non-zero elements in several first and last rows. Also, the elements of $A$ are real while the elements of $\mathbf{x}$ and $\mathbf{b}$ are complex. To take advantage of these properties, we have implemented a customized linear algebra solver. As the Navier-Stokes equation is solved in a complex number domain, after the 1D FFTs in two directions, the 3D PDE is decoupled and becomes a series of 1D ODEs, one for each wavenumber. In this way, the Navier-Stokes equations are solved concurrently at each wavenumber without interaction between wavenumbers. As a result, this portion of the kernel is parallelized and requires no communication. The performance of the kernel for solving the Navier-Stokes equations is shown in Figure 1 in green and red. Similar to the FFTs, both MPI-only and OpenMP-only cases were performed and show almost ideal scalability up to 64 cores. There were small, but noticeable, performance increases observed with two hardware threads. Interestingly, the performance of OpenMP only case decreases while the performance of MPI only case increases when four hardware threads per core were applied. This could be due to different memory access patterns between MPI and OpenMP, or scheduling overhead in OpenMP, which was also observed in Intel's work on HPCG[28].

The last kernel of PoongBack is the data transpose in a two dimensionally partitioned domain. This kernel is to ensure data alignment for FFTs and linear problems when solving the Navier-Stokes equations and does not include any floating point operations. However, more than 50% of the simulation time occurs during this operation. The data transpose kernel includes two subsequent parts: 1) All-to-All type MPI communication in two sub-communicators; and 2) data reordering to maximize MPI message size and to finalize memory alignment after MPI communication. As the domain is partitioned in two dimensions, MPI communication needs to be performed in both dimensions. This is accomplished using MPI sub-communicators. The communication in each direction is logically All-to-All, but other communication patterns can be faster than "`mpi_alltoall`" depending on the 2D communication topology. We have used MPI-enabled FFTW (version 3.3 or higher), which dynamically determines the optimal communication patterns for each sub-communicator. Since MPI-enabled FFTW only supports a one dimensionally partitioned domain, a.k.a plane decomposition, we separately use FFTW for 1D FFTs and MPI communications with "`fftw_mpi_execute_r2r()`".[1] The part of code used for data reordering performs a tensor transpose: `B(i,j,k,l)` ← `A(l,i,k,j)` or `B(i,j,k)` ← `A(j,k,i)`. Unfortunately, cache-line

[1] The original idea of using FFTW for the global transpose is due to Dr. Rhys Ulerich.

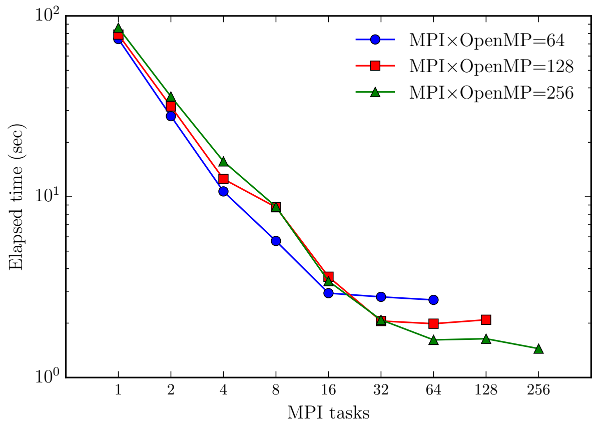

Figure 3: Comparison of MPI×OpenMP in data transpose

optimization cannot be achieved for both load and store at the same time. In this work we preferentially loaded the order of transpose. The performance of MPI communication and data reordering is shown in Figure 2. Similar to the FFT scaling and Navier-Stokes scaling in Figure 1, data reordering also shows nearly perfect linear scalability up to 64 cores. Indeed, the performance continues to increase even with hardware threading. On the other hand, profiling the MPI communication shows performance increases as the number of cores increases even with more than one MPI tasks for each hardware thread. Admittedly, the scalability is far from linear. This result is interesting because PoongBack benchmark results on other HPC systems show that MPI communication takes more than five times the elapsed time for data reordering [17]. This may imply that the performance could be increased by fine tuning the data reordering for KNL nodes.

For data re-ordering, OpenMP can be used in each MPI task. This hybrid MPI+OpenMP parallelism is tested and results are shown in Figure 3. Each line in Figure 3 details cases where the product of the number of MPI tasks and the number of OpenMP threads is constant. Using more MPI tasks and less OpenMP threads generally shows better performance. The best performance is achieved when 256 MPI tasks were used with four hardware threads. This is because using MPI shows better performance than OpenMP for data reordering. Also, using more MPI tasks performs better for MPI communication as shown in Figure 2.

The performance of one full timestep (without I/O) is shown in Figure 4. The best performing set of MPI × OpenMP concurrency for each number of processors was chosen. The benchmark with the same grid size is performed on a single node of Lonestar 5 which has two sockets with Xeon E5-2690 v3 (Haswell) processors . We also performed another benchmark with slightly bigger grid size, $1536 \times 192 \times 768$. Both cases with different grid sizes shows good scalability on Lonestar 5. On the KNL node, it also shows almost perfect scalability up to 64 processors. The performance continues to increase after using hardware threads. In most cases, using only MPI tasks shows the best performance, excepts at 64 processors with 2 hardware threads in the small grid size case. However, the difference between hybrid parallelism and only MPI tasks are minimal at 128 processors. Comparing the best performance on Lonestar 5 and the KNL node with large grid size benchmark, the KNL node shows approximately 21% better performance. A conclusion from this study is that generally speaking, using only MPI seems to be the best option for any problem size. Despite this, we also note that hybrid parallelism could be more important due to the off-node impact. When the problem size is bigger than the size of DRAM on a KNL node, one must use multiple KNL nodes. Using many MPI tasks can easily saturate the capability of the interconnect device. Under these conditions, reducing the number of MPI tasks by using OpenMP would likely improve performance.

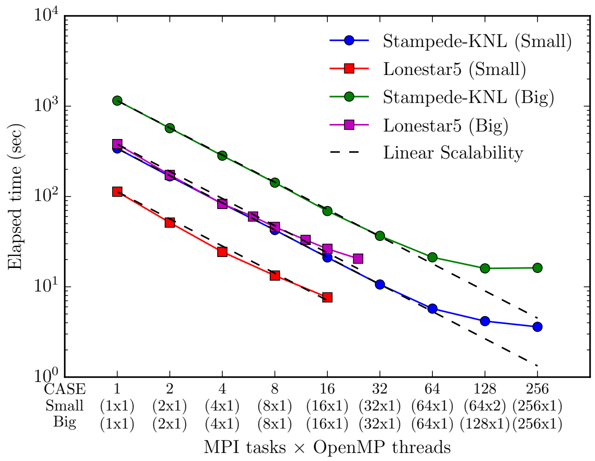

**Figure 4: Strong scaling results for the total elapsed time for single timestep. The parentheses denote: (MPI tasks × OpenMP threads).; (Small)** $1024 \times 128$**; (Big)** $1536 \times 192 \times 768$

## 4.2 Gaussian processes

Many uncertainty quantification applications replace an expensive model, such as a discretized partial differential equation, with a cheap surrogate. This is done so that during a Markov chain Monte Carlo procedure, likelihood evaluations are cheap. There are a plethora of different surrogate choices and strategies for choosing which is appropriate for a particular application. Gaussian process surrogates are a common choice [2–4, 13, 26, 30], and we explore their scaling on the KNL architecture here.

Gaussian processes are a probabilistic model for generating random fields characterized by a mean $\mu$ function and a covariance function $c$,

$$f(x) \sim \mathcal{N}(\mu(x), c(x, x')). \tag{1}$$

They have the property that evaluation of the field at some finite set of points $(x_1, \ldots, x_n)$ yields a multivariate normal vector with a known mean vector and symmetric and positive-definite covariance matrix

$$\begin{pmatrix} f(x_1) \\ \ldots \\ f(x_n) \end{pmatrix} \sim \mathcal{N}(\mu, \Sigma). \tag{2}$$

Thus, laying down a grid of points at which we wish to model the field yields the usual Gaussian probability distribution function we are familiar with,

$$p(x) \propto |\Sigma|^{-1/2} \exp(-\frac{1}{2} x^\top \Sigma^{-1} x). \tag{3}$$

Notice that evaluation of this probability density function necessitates a linear solve $\Sigma u = x$. Since $\Sigma$ is symmetric and positive-definite a Cholesky factorization is appropriate. Iterative approaches exist too, but we concentrate on the direct approach for this work. It is also worth noting that the covariance function typically takes an exponential form

$$c(x, x') = \sigma^2 \exp(-\frac{1}{2} \|x - x'\|^2 / l^2) \tag{4}$$

and so the resulting covariance matrix

$$\Sigma_{ij} = c(x_i, x_j) \tag{5}$$

is full and dense. The covariance function can also contain unknown hyperparameters that need to be inferred, for example, a hierarchical Bayesian setting. The usual approach to evaluate the probability density function is:

1. Compute $L$ such that $\Sigma = LL^\top$.
2. Solve $\Sigma u = x$ using (1).
3. Compute the inner-product $x^\top u$.
4. Divide by the square root of the determinant of $\Sigma$.
5. Return $p$
6. Go to (1) for the next state in the Markov chain.

QUESO was used to manage the Gaussian process and Markov chain infrastructure. QUESO is a C++ library for doing uncertainty quantification. QUESO itself stands for Quantification of Uncertainty for Estimation, Simulation and Optimization. It is free and open source software and is available on GitHub: http://libqueso.com. The bulk of the computational cost lies inside the likelihood function which necessitates the evaluation of the Gaussian probability density function. Extensive use of Intel's MKL was used for both the Cholesky factorization, subsequent linear solve, and determinant calculation.

Figure 5 shows timings of a Gaussian process likelihood evaluation as a function of number of OpenMP threads. The timings only include the assembly and the factorization, since these two procedures are the most expensive at each iteration of a Markov chain Monte Carlo procedure. The assembly was parallelized with the usual OpenMP preprocessor directives, and the factorization was executed by Intel's MKL. Two cases were run on the KNL node, one entirely in MCDRAM and the other entirely in DDR4. Timings for both of these show that for problems that are not memory-bandwidth limited, MCDRAM does not offer a significant payoff. A third case was run on a Xeon E52670v3 for comparison. We note that KNL and Xeon scalability are similar, but that the KNL time-to-solution is more favorable beyond 16 threads.

## 4.3 Blended Isogeometric Discontinuous Galerkin (BIDG) Method

The blended isogeometric discontinuous Galerkin (BIDG) method is a hybrid finite element method that seamlessly blends together isogeometric analysis (IGA) with discontinuous Galerkin (DG) methods in order to exploit the strengths of each, while mitigating their respective drawbacks. Classical DG methods are finite element methods (FEMs) that are appealing in aspects of their computational efficiency on modern architectures, particularly in the context of explicit wave propagation. In this context, DG allows for localization of finite element stencils (e.g nearest edge/face neighbors) in such a way that dynamic problems lead to block diagonal matrix systems, providing the capability to compute solutions at high order accuracy on sparse meshes with large memory throughput [16].

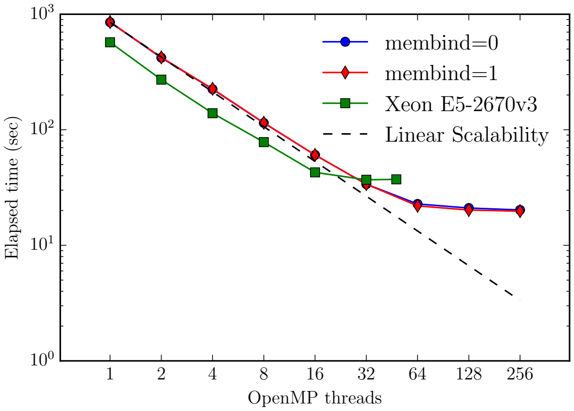

**Figure 5: Strong scaling of time taken to execute both a Gaussian process assembly and factorization. The red line shows the case where the problem was allocated entirely in MCDRAM. The blue line shows the case where the problem was allocated entirely in DDR4. The green line shows a Xeon E52670v3 scaling for comparison.**

In contrast, IGA typically informs a class of continuous Galerkin (CG) type methods where the geometric shape functions and finite element test functions are chosen as isogeometric basis functions (e.g. NURBS or splines) [31]. These choices are made to address the critical "design-to-analysis" bottleneck, where the engineering design community utilizes computer aided design (CAD) technologies to parameterize complicated geometries, but in such a way that resulting seams and gaps in the meshes make them impossible to perform finite element analysis (FEA) on. Recent progress in the IGA community has provided a comprehensive solution to this problem by building into CAD programs isogeometric basis functions that lead to CAD meshes that are able to generate analysis suitable designs (i.e. can be used for FEA). Moreover, it is known that traditional efforts to deal with complicated geometric domains, such as heavily refining linear meshes or using isoparametric elements, can lead to "geometric variational crimes" [23]. These numerical errors in fact can ultimately dominate the error behavior in many applications, where it has been shown that geometric sensitivities generate largely inadmissible (or unreliable) solutions [8, 34, 36]; thus necessitating the need for geometrically consistent meshes for accurate solutions at high polynomial order.

The BIDG algorithm developed in [23] merges key capabilities offered by both IGA and DG methods. In BIDG different basis functions are used to parameterize the geometry than to solve for the DG method. For the geometry piecewise rational Bernstein-Bézier

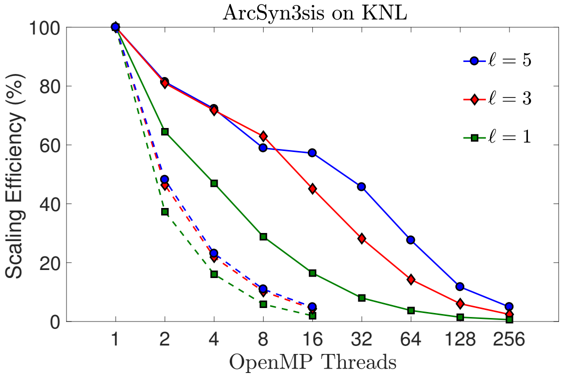

**Figure 6: Strong scaling of the ArcSyn3sis BIDG kernel on KNL nodes run on isogeometric 168 element mesh, and shown at different polynomial order $\ell$. Dashed lines are the comparable runs on Xeon E5 processors.**

basis functions are used, and for the DG basis piecewise discontinuous polynomials are used. What this provides is the ability to exactly parameterize complex geometries, while still being able to utilize the small localized stencils from DG methods. A set of geometric transformations between the DG and IGA bases is then easily established, up to a constraint on the conditioning of the IGA elements of the mesh. Recently automation tools for complex geometric domains have also been developed for BIDG on triangles in two dimensions [5] and generalized elements in three dimensions [6]. This allows for meshes that exactly preserve underlying geometry in real-world designs, including both arbitrarily smooth and discontinuous features. The resulting BIDG method is an inherently high-order method that is amenable to *hp*-adaptive schemes, and shows requisite and theoretically rigorous super-exponential convergent behavior.

The C/C++ software package ArcSyn3sis implements the BIDG method and it is used here for the scaling study on the KNL architecture. For our strong scaling test we solve the first-order acoustic wave equation:

$$\frac{\partial p}{\partial t} + \nabla \cdot \boldsymbol{u} = 0, \quad \frac{\partial \boldsymbol{u}}{\partial t} + \nabla p = 0, \tag{6}$$

where $\boldsymbol{u} = (u_x, u_y)$ is the velocity, and $p$ the pressure. The following semidiscrete block diagonal system results:

$$\left(v, \frac{\partial A}{\partial t}\right)_{\Omega_i} = V_{\Omega_i} + S_{\partial\Omega_i}$$

for $A = (p, \boldsymbol{u})$, $V_{\Omega_i}$ a volume kernel that has only elementwise dependencies, and $S_{\partial\Omega_i}$ a surface kernel that depends on inter-element communication through classical upwinding [24]. The resulting system can be understood as a globally sparse block-matrix with locally dense blocks, thus motivating the use of a many-core architecture for execution. The implementation utilizes the Open Concurrent Compute Abstraction (OCCA) library [22]. The OCCA library abstracts multiple threading back-ends (OpenMP, OpenCL, and CUDA) using C preprocessor macros

The results of the basic scaling study using OpenMP threads on a single node are shown in Fig. 6, where the strong scaling efficiency $e_n$ for $n$ OpenMP threads is defined as $e_n = [t_1/(nt_n)] \times 100\%$, and $t_n$ is the time to completion. The implementation was relatively straightforward using the OCCA library, with fairly little effort put into code optimization. The Intel icpc compiler was used on Stampede and Stampede-KNL, where on Stampede-KNL C and C++ compiler flags `-xCORE-AVX2` -axMIC-AVX512 we also set. The code was run using explicit fourth-order Runge-Kutta over a short time window (< 300 timesteps), and the average over timesteps is reported.

These preliminary results seem to indicate improved scaling as a function of polynomial order $\ell$, which is consistent with the theoretical behavior of the algorithmic intensity of DG algorithms. However, this is clearly not enough evidence to be certain that such scaling is being observed. In particular, on this mesh we see an efficiency saturation above $\ell > 5$. Moreover, since the mesh itself has only 168 elements, the number of threads eventually exceeds the number of elements, making for some intriguing questions regarding the observed behavior. In order to more fully understand the observed behavior more extensive study is required. For example the standard roofline diagnostic [35] can be used to ascertain the relationship of locality walls and bandwidth ceilings to the performance of the current implementation, which will be used to help guide optimization.

## 4.4 EOM-CCSDT single-node performance

For the calculations used in this work, the CFOUR [12] program was used. CFOUR (Coupled-Cluster techniques for Computational Chemistry) is an application for performing high-level quantum chemical calculations that is under active development by research groups at UT Austin and Universität Mainz, Germany. CFOUR's major strength is its arsenal of high-level *ab initio* methods for the calculation of atomic and molecular properties. Virtually all approaches based on Møller-Plesset (MP) perturbation theory and the coupled-cluster approximation (CC) are available.

This section will focus on a comparison of optimal single-node performance between traditional (Haswell) Intel CPUs and KNL compute nodes due to be used in Stampede 2. While the scalability of CFOUR on current and emerging architectures will be the focus of future work, the results presented here will detail overall runtime performance gains from migrating to this new architecture.

For this work, the performance of a single calculation, the EOM-CCSDT energy of the first excited singlet state of $C_2H_2$, was studied. This excited state is the focus of currently ongoing work being conducted by authors of this paper in which over 1 million single point energies on a potential energy surface will be calculated. As such, ensuring the best performance for this run type conserves important national compute resources available at TACC. All EOM-CCSDT calculations were performed using the NCC module[20] of the CFOUR program system and the ANO1 basis set[1] with core electrons uncorrelated. The nonequilibrium geometry used for this excited singlet state has C1 symmetry. This lowest symmetry point group available to $C_2H_2$ was used because it requires the most memory and compute time. This establishes an upper bound per computation.

Porting CFOUR to the KNL architecture was straightforward. No modification of the code or the build system was required. The

**Table 1: Comparison of runtimes for EOM-CCSDT energy calculation with 16 OpenMP threads for 3 different KNL configurations**

| | Mem Mode | Mem Used | Runtime (s) |
|---|---|---|---|
| Config 1 | Flat | DDR RAM | 828 |
| Config 2 | Flat | MCDRAM | 789 |
| Config 3 | Cache | MCDRAM (cache)<br>DDR RAM | 793 |

compile and link flags previously discussed in Section 3 were added to corresponding Fortran, C, and C++ environmental variables.

The NCC module in CFOUR is a recent contribution to the code and, as such, has been designed to take advantage of many-core architectures. It uses OpenMP for thread-based parallelism and in this work, the built-in parallelism of Intel's MKL is also exploited. As discussed previously [20], on traditional dual Intel CPU compute node systems, best performance is obtained by using OMP_NUM_THREADS of 16 (8 threads per socket). This relies on the default behavior that MKL will use OMP_NUM_THREADS if outside an OpenMP block but run single-threaded within an OpenMP block. In the NCC module, there are MKL-library calls both within and outside OpenMP blocks, and best performance is achieved by using only serial MKL calls within OpenMP blocks. This ensures that thread oversubscription never occurs.

For benchmark comparison purposes, the traditional CPUs used for reference were two Intel Xeon E5-2670 v3 CPUs (each with 12 cores) running at 2.30GHz. These CPUs were first available on market in late 2014 and is representative of typical HPC compute nodes in current generation supercomputers. Best performance on this system was achieved with 16 OpenMP threads (8 per socket) and the total compute time for the $C_2H_2$ excited state energy discussed previously was 607s. If more than 16 threads were used, the total walltime for the calculation increases.

While the KNL architecture offers three different memory modes, only two of those will be supported on Stampede 2, cache mode and Flat mode. Cache mode preallocates all of the "fast" MCDRAM as direct-mapped L3 cache and as such only 96 GB of DDR4 RAM is presented to the operating system. Flat mode presents both the MCDRAM and the DDR RAM to the operating system and the user can use the "numactl" utility to decide at run time if memory allocations should be directed to the 16 GB of MCDRAM or the 96 GB of DDR RAM. For these comparisons, 16 OpenMP threads (8 per socket) and Intel's recommended default cluster mode of "Quadrant" are used. This results in three different runtime configurations: Flat mode with DDR RAM in Quadrant mode, Flat mode with MCDRAM in Quadrant mode, and Cache-Quadrant mode. These results are presented in Table 1.

The total memory required for this EOM-CCSDT calculation is 10 GB. As such, the entire calculation can be fit into the fast MCDRAM. While this does result in a small performance increase (789s vs 828s), it is important to note that the Cache-Quadrant mode (793s) performed almost identically. This is important for several reasons. First, the majority of the KNL compute nodes are set up in Cache-Quadrant mode. There are only a handful of Flat nodes available. Second, many CFOUR calculations will not fit solely in MCDRAM. The Cache-Quadrant mode offers the performance benefits of the fast MCDRAM that also applies when a processes uses more than 16 GB. This is something that the Flat nodes can not do. Finally, this performance benefit is transparent to the enduser. No runtime configuration is required. Because of this, the remainder of this section will focus on maximizing performance for Cache-Quadrant systems.

On traditional CPUs, best performance is achieved when using 16 OpenMP threads and not allowing threaded MKL calls from within an OpenMP block. This is not the case on KNL-based systems. For the EOM-CCSDT energy calculation, run time improvements were seen up to 128 OpenMP threads for the NCC module. Additional improvements were seen when allowing threaded MKL calls from within an OpenMP parallel region. This was done by setting both environmental variables OMP_NESTED and MKL_DYNAMIC to true. The first variable is what enables threaded MKL calls from OpenMP parallel blocks while the second variable reduces oversubscription from MKL threading within our NCC OpenMP blocks. This is accomplished by analyzing system workload and dynamically changing the number of MKL threads. This combination of environmental variables along with our NCC module results in 128 threads used for NCC OpenMP parallel regions, 128 MKL threads outside OpenMP blocks and a dynamic number of MKL threads when called within OpenMP blocks. With these changes, total run time for our EOM-CCSDT calculation was 395s.

Overall, decent performance with CFOUR on KNL systems was easily achieved and resulted in a performance gain compared to traditional CPUs. By moving from Haswell to KNL, run time performance for the EOM-CCSDT energy of the first excited singlet state of $C_2H_2$ was improved by 35%. It is interesting to note, that the parallel scalability of KNL was better than that of a Haswell-based system: haswell compute nodes were only able to see performance improvements up to 16 threads, while on KNL, improvements were seen up to 128 threads. This could be attributed to either the higher memory bandwidth available on these systems or to improvements introduced with the AVX512 instruction set. Future work will focus on a new implementation of the parallelization of NCC to take further advantage of these new systems.

## 5 SUMMARY

Using hardware that will appear in the upcoming Stampede 2 cluster, we studied the ease of portability of a diverse set of scientific software packages to KNL. The software packages used in this paper were written in a variety of the most commonly used procedural languages and are workhorses for solving problems in many disciplines of computational science.

Overall, our experiences porting scientific applications to the KNL architecture were positive. Software at PECOS typically ships with a flexible build system and customizing compiler flags for a KNL-specific instruction set is simple. Furthermore, we observed that KNL compute nodes can deliver impressive compute power and scalability for relatively little human effort. Contrasting this with experiences on previous generations of Intel Xeon Phi MICs [32], it is clear bootable MICs that do not require code intervention for offloading onto co-processor MICs such as Knights Ferry yield a much larger performance-to-effort ratio for computational scientists.

We note that although KNL has several boot-time options that specify memory hierarchy and layout, Intel's recommended default cluster mode, Quadrant, along with the Cache memory mode brings most of the benefits of performant memory access and bandwidth without the drawbacks of explicitly management through "numactl".

Lastly, in addition to strong scaling results on KNL, we have presented some promising runtime comparisons with traditional Haswell Xeon E5-2670 CPUs.

## ACKNOWLEDGMENTS

The authors acknowledge the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing HPC resources that have contributed to the research results reported within this paper.

## REFERENCES

[1] J. Almlof and P. R. Taylor. 1987. *Journal of Chemical Physics* 86 (1987), 4070.

[2] MJ Bayarri, James O Berger, Eliza S Calder, Keith Dalbey, Simon Lunagomez, Abani K Patra, E Bruce Pitman, Elaine T Spiller, and Robert L Wolpert. 2009. Using statistical and computer models to quantify volcanic hazards. *Technometrics* 51, 4 (2009), 402–413.

[3] Stefano Conti, John Paul Gosling, Jeremy E Oakley, and Anthony O'Hagan. 2009. Gaussian process emulation of dynamic computer codes. *Biometrika* (2009), asp028.

[4] Stefano Conti and Anthony O'Hagan. 2010. Bayesian emulation of complex multi-output and dynamic computer models. *Journal of statistical planning and inference* 140, 3 (2010), 640–651.

[5] Luke Engvall and John A. Evans. 2016. Isogeometric triangular Bernstein-Bézier discretizations: Automatic mesh generation and geometrically exact finite element analysis. *Computer Methods in Applied Mechanics and Engineering* 304 (2016), 378 – 407. DOI: http://dx.doi.org/10.1016/j.cma.2016.02.012

[6] Luke Engvall and John A. Evans. 2017. Isogeometric unstructed tetrahedral and mixed-element Bernstein-Bézier discretizations. *In Review* (2017).

[7] K C Estacio-Hiroms, E E Prudencio, Nicholas Malaya, M Vohra, and Damon McDougall. 2016. *Quantification of Uncertainty for Estimation, Simulation, and Optimization (QUESO).*

[8] H. Fahs. 2011. Improving Accuracy of High-order Discontinuous Galerkin Method for Time-domain Electromagnetics on Curvilinear Domains. *Int. J. Comput. Math.* 88, 10 (July 2011), 2124–2153. DOI: http://dx.doi.org/10.1080/00207160.2010.527960

[9] Matteo Frigo and Steven G. Johnson. 2005. The Design and Implementation of FFTW3. *Proc. IEEE* 93, 2 (2005), 216–231.

[10] J. Graham, K. Kanov, X. I. A. Yang, M. Lee, N. Malaya, C. C. Lalescu, R. Burns, G. Eyink, A. Szalay, R. D. Moser, and C. Meneveau. 2015. A Web services accessible database of turbulent channel flow and its use for testing a new integral wall model for LES. *Journal of Turbulence* 17, 2 (2015), 181–215.

[11] Khronos OpenCL Working Group and others. 2008. The opencl specification. *version* 1, 29 (2008), 8.

[12] Michael E. Harding, Thorsten Metzroth, Jurgen Gauss, and Alexander A. Auer. 2008. Parallel Calculation of CCSD and CCSD(T) Analytic First and Second Derivatives. *Journal of Chemical Theory and Computation* 4, 1 (2008), 64–74. DOI: http://dx.doi.org/10.1021/ct700152c arXiv:http://pubs.acs.org/doi/pdf/10.1021/ct700152c

[13] Dave Higdon, James Gattiker, Brian Williams, and Maria Rightley. 2008. Computer model calibration using high-dimensional output. *J. Amer. Statist. Assoc.* 103, 482 (2008), 570–583.

[14] John Kim, Parviz Moin, and Robert Moser. 1987. Turbulence statistics in fully developed channel flow at low Reynolds number. *Journal of Fluid Mechanics* 177 (1987), 133–166.

[15] Volodymyr V Kindratenko, Jeremy J Enos, Guochun Shi, Michael T Showerman, Galen W Arnold, John E Stone, James C Phillips, and Wen-mei Hwu. 2009. GPU clusters for high-performance computing. In *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on.* IEEE, 1–8.

[16] A. Klöckner, T. Warburton, J. Bridge, and J.S. Hesthaven. 2009. Nodal discontinuous Galerkin methods on graphics processors. *J. Comput. Phys.* 228, 21 (2009), 7863 – 7882. DOI: http://dx.doi.org/10.1016/j.jcp.2009.06.041

[17] Myoungkyu Lee, Nicholas Malaya, and Robert D. Moser. 2013. Petascale direct numerical simulation of turbulent channel flow on up to 786K cores. In *the International Conference for High Performance Computing, Networking, Storage and Analysis.* ACM Press, New York, New York, USA, 1–11.

[18] Myoungkyu Lee and Robert D. Moser. 2015. Direct numerical simulation of turbulent channel flow up to $\mathrm{Re}_\tau = 5200$. *Journal of Fluid Mechanics* 774 (June 2015), 395–415.

[19] Myoungkyu Lee, Rhys Ulerich, R, Nicholas Malaya, and Robert D. Moser. 2014. Experiences from Leadership Computing in Simulations of Turbulent Fluid Flows. *Computing in Science Engineering* 16, 5 (Sept. 2014), 24–31.

[20] D. Matthews and J. F. Stanton. 2015. *Journal of Chemical Physics* 142 (2015), 06410.

[21] Damon McDougall, Nicholas Malaya, and Robert D Moser. 2015. The Parallel C++ Statistical Library for Bayesian Inference: QUESO. In *Handbook of Uncertainty Quantification.* Springer International Publishing, Cham, 1–38.

[22] David Medina, Amik St-Cyr, and Tim Warburton. 2017. OCCA: A unified approach to multi-threading languages. *In Review* (2017). https://arxiv.org/pdf/1403.0968.pdf

[23] C. Michoski, J. Chan, L. Engvall, and J.A. Evans. 2016. Foundations of the blended isogeometric discontinuous Galerkin (BIDG) method. *Computer Methods in Applied Mechanics and Engineering* 305 (2016), 658 – 681. DOI: http://dx.doi.org/10.1016/j.cma.2016.02.015

[24] C. Michoski, D. Meyerson, T. Isaac, and F. Waelbroeck. 2014. Discontinuous Galerkin methods for plasma physics in the scrape-off layer of tokamaks. *J. Comput. Phys.* 274, 0 (2014), 898–919. DOI: http://dx.doi.org/10.1016/j.jcp.2014.06.058

[25] CUDA Nvidia. 2010. Programming guide. (2010).

[26] Jeremy Oakley and Anthony O'Hagan. 2002. Bayesian Inference for the Uncertainty Distribution of Computer Model Outputs. *Biometrika* 89, 4 (2002), 769–784. http://www.jstor.org/stable/4140536

[27] ARB OpenMP. 2010. The OpenMP API specification for parallel programming. *URL http://openmp. org* (2010).

[28] Jongsoo Park, Mikhail Smelyanskiy, Karthikeyan Vaidyanathan, Alexander Heinecke, Dhiraj D. Kalamkar, Xing Liu, Md. Mosotofa Ali Patwary, Yutong Lu, and Pradeep Dubey. 2014. Efficient Shared-memory Implementation of High-performance Conjugate Gradient Benchmark and Its Application to Unstructured Matrices. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '14).* IEEE Press, Piscataway, NJ, USA, 945–955. DOI: http://dx.doi.org/10.1109/SC.2014.82

[29] Ernesto E Prudencio and Karl W Schulz. 2012. The Parallel C++ Statistical Library 'QUESO': Quantification of Uncertainty for Estimation, Simulation and Optimization. In *Euro-Par 2011: Parallel Processing Workshops: CCPI, CGWS, HeteroPar, HiBB, HPCVirt, HPPC, HPSS, MDGS, ProPer, Resilience, UCHPC, VHPC, Bordeaux, France, August 29 – September 2, 2011, Revised Selected Papers, Part I,* Michael Alexander, Pasqua D'Ambra, Adam Belloum, George Bosilca, Mario Cannataro, Marco Danelutto, Beniamino Di Martino, Michael Gerndt, Emmanuel Jeannot, Raymond Namyst, Jean Roman, Stephen L Scott, Jesper Larsson Traff, Geoffroy Vallée, and Josef Weidendorfer (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 398–407.

[30] Carl Edward Rasmussen. 2006. Gaussian processes for machine learning. (2006).

[31] Richard F. Riesenfeld, Robert Haimes, and Elaine Cohen. 2015. Initiating a CAD renaissance: Multidisciplinary analysis driven design: Framework for a new generation of advanced computational design, engineering and manufacturing environments. *Computer Methods in Applied Mechanics and Engineering* 284, 0 (2015), 1054–1072. DOI: http://dx.doi.org/10.1016/j.cma.2014.11.024 Isogeometric Analysis Special Issue.

[32] Karl W Schulz, Rhys Ulerich, Nicholas Malaya, Paul T Bauman, Roy Stogner, and Chris Simmons. 2012. Early experiences porting scientific applications to the Many Integrated Core (MIC) platform. In *TACC-Intel Highly Parallel Computing Symposium, Austin, Texas*, Vol. 44.

[33] Philippe R. Spalart, Robert D. Moser, and Michael M. Rogers. 1991. Spectral Methods for the Navier-Stokes Equations with One Infinite and Two Periodic Directions. *J. Comput. Phys.* 96 (1991), 297–324.

[34] Thomas Toulorge, Yves Reymen, and Wim Desmet. 2008. A 2D discontinuous Galerkin method for aeroacoustics with curved boundary treatment. In *Proceedings of International Conference on Noise and Vibration Engineering (ISMA2008).* Citeseer, 565–578.

[35] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Commun. ACM* 52, 4 (April 2009), 65–76. DOI: http://dx.doi.org/10.1145/1498765.1498785

[36] D. Wirasaet, S.R. Brus, C.E. Michoski, E.J. Kubatko, J.J. Westerink, and C. Dawson. 2015. Artificial boundary layers in discontinuous Galerkin solutions to shallow water equations in channels. *J. Comput. Phys.* 299 (2015), 597 – 612. DOI: http://dx.doi.org/10.1016/j.jcp.2015.07.015