Noname manuscript No.
(will be inserted by the editor)

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

Nicholas Malaya* · **D. McDougall** · **C Michoski** · **M Lee**

the date of receipt and acceptance should be inserted later

Abstract This paper presents experiences using an early development environment release of the forthcoming Intel MIC platform, focusing on porting of existing scientific applications and micro-kernels. Fortran and C++ applications are chosen from disciplinesincluding numerical linear algebra, gaussian processes, finite-element analysis, and the core compute kernels used in the direct numerical simulation of turbulence.

Keywords co-processors · heterogeneous computing · scientific applications · many integrated cores

1 Introduction

Heterogeneous computing with multiple levels of exposed parallelism is a leading candidate under consideration for the design of future exascale systems.Indeed, accelerators like current generation GPGPUs offer relatively high floating-point rates and memory bandwidth with lower relative power footprints than general-purpose compute platforms [**?**]. However, GPU-based acceleration commonly requires special programming constructs (e.g. NVIDIA's CUDA language) for the accelerated work.

The application kernels considered herein are taken from existing development efforts within the PECOS Center at the University of Texas at Austin.

The applications include both Fortran and C++ codes, and include an example with no prior thread-based parallelism as well as codes with existing OpenMP or TBB based threading.

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

2 Testing Infrastructure

The test hardware used for this exercise was on a Stampede development cluster installed at the Texas Advanced Computing Center.

Each compute node was a host Linux (CentOS 6.1) server with two 3.47GHz Intel Westmere (X5690) six-core CPUs, 24GB of memory, and one Knights Ferry (KNF) co-processor with 30 active cores.

The KNF support on this hardware was provided with the *alpha9* release of Intel's MIC Software Development Platform, and all compilation was performed with an alpha release of Intel Composer for MIC (*029 Alpha Build 20120222*).

3 Third Party Library Cross-Compiling

As with many scientific research groups, application development at the PECOS center employs many open-source libraries.

Prior to performing any tests on the Westmere or KNF MIC, it was necessary to first build all auxiliary libraries required by each of the computational kernels considered.The following libraries were compiled for both the host CPU and KNF MIC architectures: *Boost, GSL, FFTW[?], GRVY,* and *libMesh*.

*Institute for Computational Engineering and Sciences (ICES), The University of Texas at Austin
E-mail: {nick,damon,michoski,mk}@ices.utexas.edu

†Texas Advanced Computing Center and ICES
The University of Texas at Austin
E-mail: karl@tacc.utexas.edu (*corresponding author*)

While building these libraries for the host environment is a common endeavor, building the libraries for MIC represents a new challenge as it necessitates cross-compilation techniques. Fortunately, many of these libraries utilize the Autotools build system, which can support cross-compilation. Native MIC builds were configured by specifying an existing non-native Linux host (e.g. *blackfin*) or by augmenting the autotools *config.sub* file with a new "mic" Linux target. To build native libraries for MIC, the "-mmic" flag was added to all relevant compiler flags. As an example, the following configure options were used to build a native MIC version of the FFTW 3.3 static library:

```
./configure CC=icc CXX=icpc FC=ifort       \
   CFLAGS="-mmic -O3" CXXFLAGS="-mmic -O3" \
   FCFLAGS="-mmic -O3" --host=blackfin
```

The libMesh library additionally required fixing configuration macros to support cross-compilation and autodetection of native TBB support.

These strategies successfully built native static libraries and executables for MIC. However, building shared libraries was more delicate and not always successful. For example, configuring Boost to build shared libraries caused the linker to crash. Other shared libraries, such as GSL, built without incident.

4 Applications and Scientific Kernels

The subsections which follow highlight the applications or scientific kernels which were ported to the KNL platform. When possible, we compare scaling achieved to that on the host platform and also examine the influence of various affinity settings when utilizing large thread counts.

4.1 Incompressible DNS FFT Kernel

In many fluid dynamics applications, turbulence plays a dominant role. Unfortunately, turbulence is notoriously complex and difficult to model. One reliable tool for analyzing turbulence is direct numerical simulation (DNS)[**?**], where Navier–Stokes numerical solutions are fully resolved in both time and space.

In the DNS application used by PECOS to simulate channel flow, a Fourier spectral representation is used in the streamwise and spanwise directions, while a high order compact finite difference is used in the wall-normal direction[**?**,**?**]. The Fourier method allows a natural decoupling between different modes in transformed space such that multiple tasks can operate on their own data independently. Our method uses a 2D decomposition which divides an $N_x \times N_y \times N_z$ domain into M pencils. Fourier transforms in 3D are performed one direction at a time, and transposes re-partition the domain into pencils aligned in appropriate directions, as depicted in Figure 1. This paper focuses on single-node performance. Thus, for the kernel discussed below, the final pencil depicted (x,z,y) is the portion of the domain considered for FFT analysis on MIC.

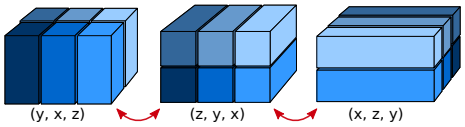


Fig. 1 The two dimensional data decomposition. The localized FFT on the pencil shown on the far right is ported to the KNF MIC platform.

Algorithmically, the final inverse Fourier transform, physical space calculations, and the first Fourier transform into wavespace are performed completely on-node and do not require MPI subroutine calls. Previously the inverse Fourier transform, realspace calculations, and forward transform were performed separately. This section of code was re-factored to perform all three steps on each line of the pencil individually. These changes improved cache coherency and optimized the algorithm for multithreading, the performance and scaling of which is the basis for this section.

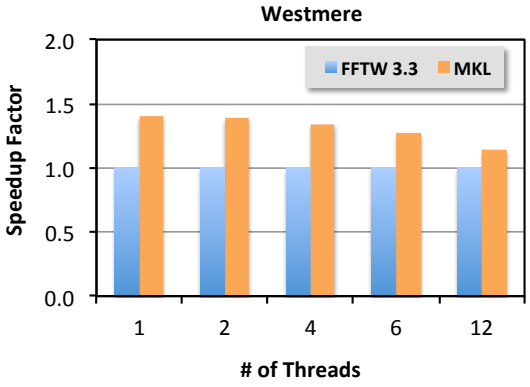
The resulting FFT kernel was run with a pencil size of $\{N_x, N_z, N_y\}$ = (8192,1280,1), with N_x corresponding to the direction being Fourier transformed. This was iterated over six pencils to emulate the real DNS code, which requires operating on three velocity components (u,v,w) as well as the non-linear terms (uu,uv,ww) required to resolve the velocity at the subsequent timestep. The size of the pencil was chosen to correspond to a realistic pencil size for a a single compute node during a large scale DNS simulation, where the global mesh size would be (8192,12288,1024).

The kernel application code is written entirely in Fortran-90, and uses OpenMP for multithreading to perform multiple 2D FFTs simultaneously.

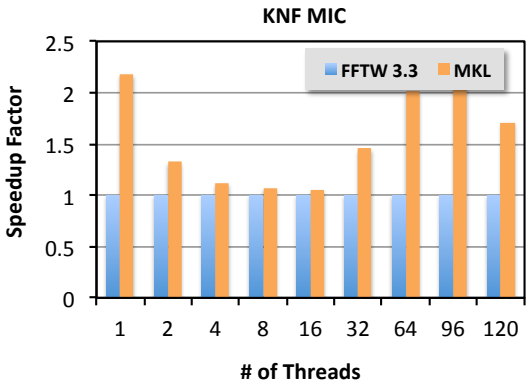
We compare results from FFTW 3.3 to the FFTW interface in MKL. Runs were made at various thread counts on both the host Westmere and KNF MIC platforms and the results were verified analytically. Timings were averaged over two successive runs, and results comparing FFTW to MKL performance are shown in Figure 2. Figure 2(a) compares the speedup factor obtained using MKL on Westmere with observed performance increases ranging from ∼20-40%. Figure 2(b) shows results on MIC, where the performance increases with MKL are more dramatic, suggesting that the FFT

implementation within MKL enables more vectorization than the FFTW counterpart. This performance difference also impacts scalability on MIC, as shown in Figure 3. In this case, the speedup factor is based on a baseline case with four threads. The MKL implementation scales much better than the FFTW version. With 120 threads, an overall scaling efficiency of 72% was achieved using MKL, whereas only 35% was achieved with FFTW.

To quantify the impact of various runtime affinity settings available on MIC, Figure 4 presents the real-to-complex (Fourier transform into frequency space) performance gains for several thread affinity options: scatter, compact or explicit. Results in Figure 4 are all compared against the default case: no KMP_AFFINITY setting. For both MKL and FFTW linkage, the *scatter* setting provided the best overall performance across a wide thread-count range. Note that this setting was seen to have a significant impact on observed runtimes.



(a)



(b)

Fig. 2 Relative comparison between threaded FFT kernel used in DNS applications on (a) dual-socket Westmere server and (b) KNF MIC co-processor.

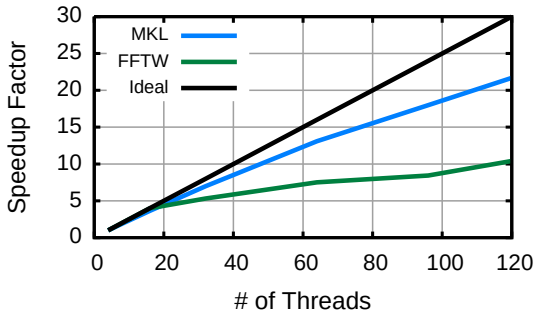
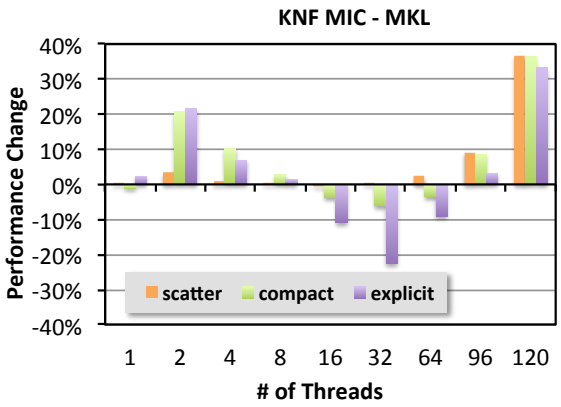
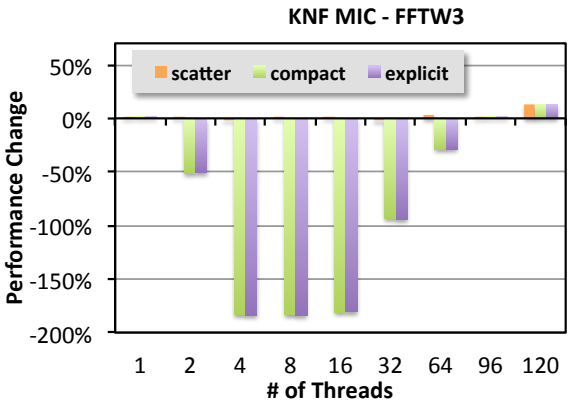


Fig. 3 Scaling of DNS FFT kernel on KNF MIC accelerator (KMP_AFFINITY=scatter).



(a)



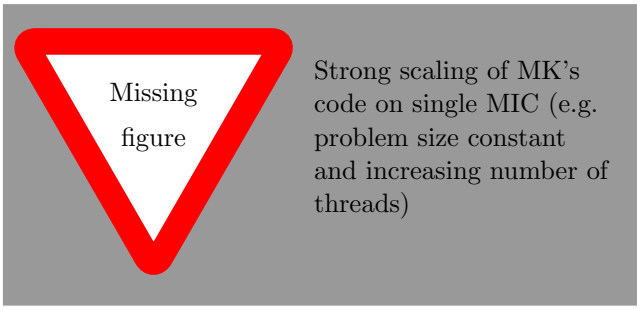
(b)

Fig. 4 Influence of runtime affinity setting on MIC. Relative performance change is based on comparison to runs with no affinity setting provided. A positive performance change indicates faster runtimes. Results are presented for FFTs performed with the (a) MKL and (b) FFTW3 libraries.

4.2 Incompressible DNS Application

In many fluid dynamics applications, turbulence plays a dominant role. Unfortunately, turbulence is notoriously complex and difficult to model.

if we merge this with the previous section, we can have an

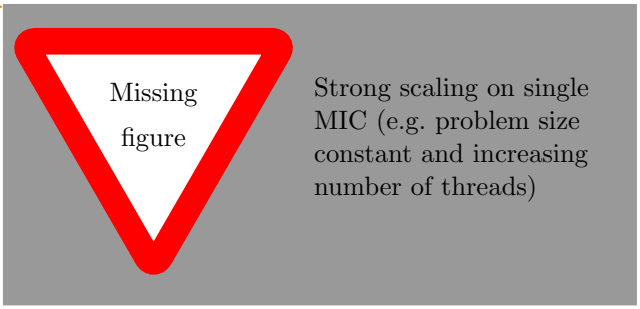


Strong scaling of MK's code on single MIC (e.g. problem size constant and increasing number of threads)

4.3 Cholesky Factorization

Decomposition of PD matrix into the product of a lower triangular matrix and its conjugate transpose.

ne sic ro the ear gebra u are ing

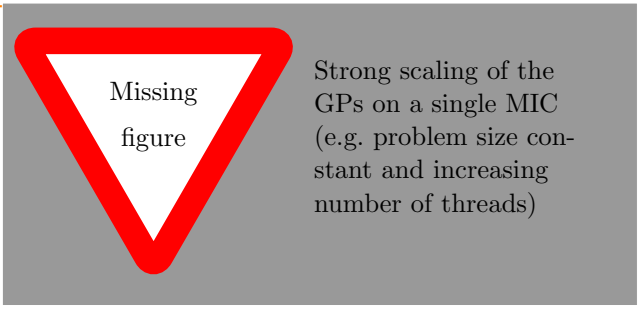


Strong scaling on single MIC (e.g. problem size constant and increasing number of threads)

4.4 Gaussian Processes

GPs

ne sic ro the ear gebra u are ing



Strong scaling of the GPs on a single MIC (e.g. problem size constant and increasing number of threads)

Some discussion is warrented: is this ameniable to MICs? Vs typical CPUs?

4.5 Blended Isogeometric Discontinuous Galerkin Application

For the scaling study the blended isogeometric discontinuous Galerkin (BIDG) code, ArcSyn3sis was used. The BIDG algorithm was developed in cite, to exploit and seamlessly merge key capabilities offered by both isogeometric analysis and discontinuous Galerkin methods. This method is an inherently high-order method, that is amenable to *hp*-adaptive schemes, and shows super-exponential convergence behaviors.

In traditional isogeometric analysis tight coupling between computer aided design (CAD) mesh automation tools for complex geometric domains, such as jet engines and tokamak fusion reactors, has been developed for triangles in two dimensions cite, and semi-arbitrary elements in three dimensions cite. This allows for meshes that exactly preserve the underlying geometry of real-world mesh designs, including arbitrarily smooth and discontinuous features cite. In many application, geoemtric sensitivities are known to dominate errors, and as such having high-order accurate solutions on geometrically consistent meshes is essential.

The discontinuous Galerkin method, on the other hand, excels in aspects of its computational efficiency in the context of modern architectures, particular in convection-dominated flows that allow for localization of finite element stencils in such a way that dynamic problems lead to block diagonal matrix systems. The locality of the memory footprint in these application models leads to the ability to compute at high local order at unusally high arithmetic intensity cite.

The BIDG method merges these two methods seamlessly way cite, by utilizing an peicewise rational isogeometric basis for the geometric representation, a peicewise discontinuous po9lynomial representation for the model, and an exact (and properly conditioned) geometric transformation between the two spaces.

For our scaling test here we solve the first-order acoustic wave equation:

$$\frac{\partial p}{\partial t} + \nabla \cdot \boldsymbol{u} = 0, \quad \frac{\partial \boldsymbol{u}}{\partial t} + \nabla p = 0, \tag{1}$$

where $\boldsymbol{u} = (u_x, u_y)$ is the velocity, and p the pressure.

Broadly such a system is discretized by solving the semidiscrete system

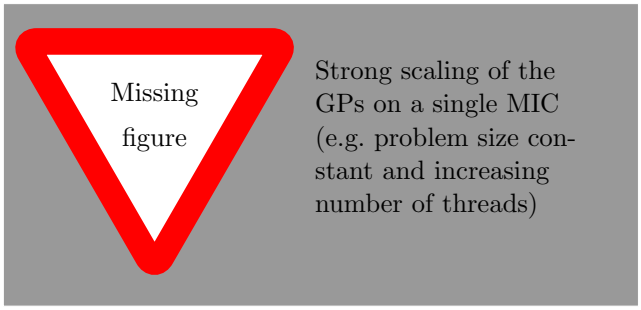
$$\left(v, \frac{\partial A}{\partial t}\right)_{\Omega_i} = V_{\Omega_i} + S_{\partial\Omega_i}$$

for $A = (\boldsymbol{u}, p)$, V_{Ω_i} a volume kernel that has only elementwise dependencies, and $S_{\partial\Omega_i}$ a surface kernel that depends on interelement communication through a classical DG flux cite.

yar2
yar3
yar4

some basic intro to the linear algebra you are doing

any nuances of



Strong scaling of the GPs on a single MIC (e.g. problem size constant and increasing number of threads)

Finally, some discussion is needed: is DG ameniable to MICs? Vs typical CPUs? I naively guess so, since you can essentially vectorize several of the activities.

5 Summary

Overall, porting a number of different scientific applications and kernels to the KNL architecture was relativelystraightforward. Successfully ported applications (written in either Fortran or C++) include serial codes as well as codes with existing OpenMP- or TBB-based threading. Even native compilation of 3rd-party libraries was reasonably straightforward, with only a few configuration options left unsupported and dynamic library builds not successful across all of the libraries considered.

Performance results vary more widely.We have demonstrated good strong scaling to large thread counts for some of our computational kernels, but others only effectively used a fraction of the MIC's potential capability. However, runtime performance improved for large thread counts on all workloads, and simple changes to thread algorithms or affinity settings often delivered further improvements. Different workloads require different settings in order to achieve peak performance. Finally, vectorization is important in order to fully exploit the MIC architecture. When comparing FFT results obtained with MKL and FFTW, the improved vectorization in MKL accounts for better performance and scalability.

Acknowledgements The Author's 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.