Early Experiences Porting Scientific Applications to the Many Integrated Core (MIC) Platform

Karl W. Schulz † · Rhys Ulerich * · Nicholas Malaya * · Paul T. Bauman * · Roy Stogner * · Chris Simmons *

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 disciplines including quantum mechanics, hypersonics, rarefied gas dynamics, finite-element analysis, and FFT and linear algebra kernels used in the direct numerical simulation of turbulence. To characterize the porting efforts required for applications at different stages of development, codes considered include both serial applications (no previous threading), codes with high BLAS usage, and codes with prior shared memory parallelism.

Keywords co-processors \cdot heterogeneous computing \cdot scientific applications \cdot 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 [6]. However, GPUbased acceleration commonly requires special programming constructs (e.g. NVIDIA's CUDA language) for the accelerated work. With the release of Intel's Many Integrated Core (MIC) architecture, an additional coprocessor technology will become available to the scientific community. This paper reports on several early porting experiences to the Knights Ferry MIC platform. An attractive feature of this architecture is support for standard threading models like OpenMP [13] which are

E-mail: {rhys,nick,pbauman,roystgnr,csim}@ices.utexas.edu

 $^\dagger \text{Texas}$ Advanced Computing Center and ICES The University of Texas at Austin

E-mail: karl@tacc.utexas.edu (corresponding author)

already used by many scientific applications. In addition, the MIC platform is based on an x86 architecture, and C/C++ and Fortran kernels can be easily compiled for direct native execution on MIC.

The application kernels considered herein are taken from existing development efforts within the PECOS Center at the University of Texas at Austin. The PECOS Center is a DOE-funded Center of Excellence working to develop the next generation of advanced computational methods for predictive simulation and uncertainty quantification of multiscale, multiphysics phenomena, focusing on the analysis of hypersonic atmospheric entry. Consequently, porting efforts include relevant kernels from the direct numerical simulation (DNS) of turbulence, quantum mechanics, hypersonics, rarefied gas dynamics, and general-purpose finiteelement assembly. 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 an eightnode 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. KNF parts are intended to provide
early access to the software development environment
of forthcoming Knights Corner cards, but do not have
the same floating-point performance. Consequently, we
report only relative performance and software porting
experiences with Knights Ferry. The KNF support on
this hardware was provided with the alpha9 release of
Intel's MIC Software Development Platform, and all

^{*}Institute for Computational Engineering and Sciences (ICES), The University of Texas at Austin

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 opensource 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[2], GRVY, and lib-Mesh. The libMesh library employs 3rd party libraries which were also compiled: ExodusII, Laspack, lib-Hilbert, Metis, Nemesis, NetCDF, Parmetis, sfcurves, Tetgen, and Triangle. 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 nonnative 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 -03" CXXFLAGS="-mmic -03" \
   FCFLAGS="-mmic -03" --host=blackfin
```

The libMesh library additionally required fixing configuration macros to support cross-compilation and auto-detection 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 KNF MIC 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 on MIC.

4.1 CFOUR Quantum Chemistry

CFOUR [3] (Coupled-Cluster techniques for Computational Chemistry) is an application for performing high-level quantum chemical calculations which 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.

CFOUR is a very large code base with 1.4 million lines of Fortran (a mix of 77 and 90) and it was chosen to test the native compilation capabilities of MIC. Only minimal changes to the code's build system were required to port CFOUR and utilize BLAS functionality from Intel's Math Kernel Library (MKL). A suite of regression tests were then run and the results obtained were the same as those on the host CPU. Due to memory and I/O limitations imposed by the current hardware available, only small problem sizes could be examined with limited scalability achieved. However, the fact that we were able to get a large legacy-style application to compile and run successfully on the MIC is an impressive feat not normally attainable on accelerators. Future work will explore offload models with MPI to leverage MIC in a co-processor mode.

4.2 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)[4], 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[5,10]. 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.

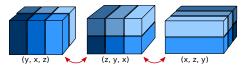


Fig. 1 The two dimensional data decomposition. The localized FFT on the pencil shown on the far right is ported to the KNF MIC platform.

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 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 $\sim 20\text{-}40\%$. 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.

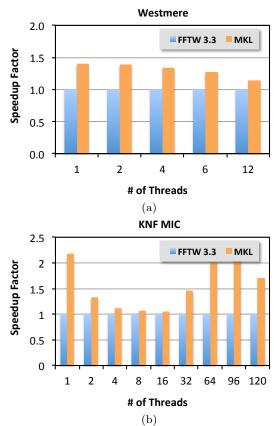


Fig. 2 Relative comparison between threaded FFT kernel used in DNS applications on (a) dual-socket Westmere server and (b) KNF MIC co-processor.

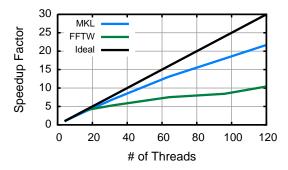


Fig. 3 Scaling of DNS FFT kernel on KNF MIC accelerator (KMP_AFFINITY=scatter).

4.3 Compressible DNS Banded Factorization Kernel

Just as DNS of the incompressible Navier—Stokes equations sheds light on incompressible turbulence, DNS of the compressible equations can illuminate compressible turbulence[11]. Unlike incompressible codes, compressible codes must mitigate severe time step restrictions stemming from the rapid propagation of acoustic waves.

The PECOS compressible DNS code, "Suzerain", has much in common with the incompressible code described in section 4.2, including partially implicit time

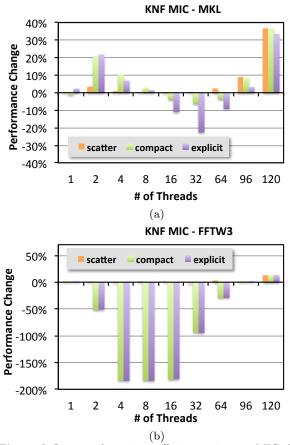


Fig. 4 Influence of runtime affinity setting on MIC. Relative performance change is based on comparison to runs with no affinity setting provided. A positive performance change indicates faster runtimes. Results are presented for FFTs performed with the (a) MKL and (b) FFTW3 libraries.

steps and the use of a Fourier spectral representation in two periodic dimensions. Differing slightly, Suzerain uses a B-spline representation in the wall-normal direction [9]. The spatial treatment and linear implicit time advance, necessary for coping with acoustics, give rise to a large number of dense banded-matrix solve operations. On modest problem sizes, the dominant cost is the conveniently parallel factorization of these dense banded matrices using the complex-valued factorization routine "ZGBTRF" from the MKL. Performing many solves was a deliberate design choice which enables the largest possible time step for a fixed number of FFT and ALLTOALL operations.

We assessed the strong scalability of ZGBTRF-like routines on MIC with a C++ benchmark. We natively timed the sequential MKL's single, double, complex single, and complex double banded factorization routines (SGBTRF, DGBTRF, CGBTRF, and ZGBTRF) for a problem size matching Suzerain operator assembly using piecewise septic B-splines. Each run factorized a large number (2¹⁵) of matrices so we could observe the

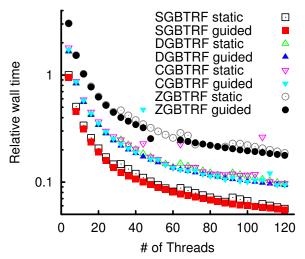


Fig. 5 Strong scalability of MKL banded factorization routines running natively on MIC for n=640 and kl=ku=44. Time shown relative to the single precision statically scheduled result ("SGBTRF static") on 4 threads.

steady state behavior of many threads vying for memory bandwidth and cache under scatter affinity. We examined both static and guided OpenMP scheduling.

Figure 5 depicts the observed results averaged across twenty-five consecutive runs. The results were highly repeatable — bars showing two standard deviations were omitted because they would be barely visible. We always saw a speedup as the number of threads increased. Double precision edged out complex single prevision which is consistent with memory bandwidth concerns dominating performance and floating point operation counts being a secondary consideration. We observed some possible memory hierarchy effects at 44, 64, and 112 threads with a puzzling dependence on static versus guided OpenMP scheduling. Aside from those peculiar data points, we found that guided scheduling was slightly faster in nearly every other case. Overall, we find these results to be encouraging for using KNF MIC to speedup Suzerain's banded factorization workloads.

4.4 Finite Element Method (FEM) System Assembly

The libMesh library[7] supports object-oriented FEM simulations of multiphysics on adaptive unstructured meshes, using hybrid parallel computing with message passing between nodes and multithreading on a node. libMesh is a relatively large package, but its cross-platform threading support made a rapid MIC port practical. Several libMesh examples, including the fem_system_ex1 incompressible flow solver we used, are easily configured for multithreaded assembly.

To isolate the scalability of FEM system assembly, the algebraic solvers were allowed zero iterations per solve. Each time step included one residual and Jacobian assembly, two algebraic constraint passes, and two residual vector norms. Assembly was multithreaded; other operations were serialized but were expected to be a small fraction of the total cost. Initialization was amortized over a hundred time steps in each test.

Initial tests revealed scalability limits due to the standard libMesh threading parameters. TBB partitions ranges, in this case ranges of elements, into subranges to evaluate on each thread. Allowing subranges with fewer elements allows for better load-balancing, but subranges with more elements require less overhead during partitioning and evaluation. The libMesh assembly granularity of 1000 elements per subrange, although efficient for large problems on two-core through six-core processors, leaves nearly every core idle for small- and medium-sized problems on MIC. Reducing the granularity to allow 10-element subranges repaired scalability on even the smallest test cases. Although fine granularity did not hurt total throughput in our tests, it could conceivably slow down threaded assembly with lowerorder elements or fewer unknown variables. Dynamic granularity selection at runtime will be needed to deliver an unconditional improvement in performance.

The next obstacle to scalable assembly was more subtle. Evaluation of small dense element residual vectors and element Jacobian matrices is a conveniently parallel problem, but summing these into a global vector and sparse global matrix is not. To accommodate non-thread-safe solver APIs, libMesh uses a spin-lock mutex to serialize this addition. For relatively simple problems such as incompressible Newtonian flow, which includes nonlinearities but no transcendental function evaluations, this addition can take up a sizable fraction of runtime. On the MIC, speedup over single-threaded assembly was limited to roughly 5 (on 2D problems) and 7 (on 3D problems).

Even if this mutex is removed, however, speedup was limited to under 14 (in 2D) and 20 (in 3D). Neither increased problem size nor TBB partitioning affinity improved those results. It may be that the single-threaded code (global vector norms, constraint application, initial mesh and sparse matrix allocation, degree of freedom ordering, etc.) simply takes 5-7% of runtime, in which case we run afoul of Amdahl's Law before utilizing more than a fraction of the MIC.

4.5 Rarefied Gas Dynamics Kernel

This Discrete Velocity Method (DVM) kernel solves the Boltzmann equation [12]. The kernel is currently

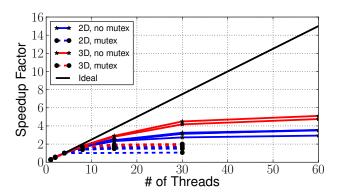


Fig. 6 Scaling of libMesh fem_system_ex1 system assembly on MIC, relative to four threads.

designed for solving spatially homogeneous relaxation problems, e.g. Bobylev-Krook-Wu distribution [8,1], and one-dimensional shock problems. The latter application is a prime candidate for threaded parallelism, as the spatial decomposition, for certain collision integral algorithms, is conveniently parallel.

DVM is implemented using Fortran 90. Prior to this work, it was a pure serial application. OpenMP parallelism was added as part of this work. Thus, this kernel represents experiences adding MIC-compatible parallelism to a scientific application from scratch.

For this work, we only consider the "N-squared" algorithm for the collision integral. The collision integral calculation represents over 99% of the runtime for this application and, for each spatial cell, the N-squared algorithm is independent of all other spatial cells resulting in convenient parallelism over the spatial cells.

Figure 7 shows results for speedup on the host Westmere CPUs. One interesting aspect here is that we compare the typical "parallel do" paradigm with the new "task" paradigm in OpenMP 3, which is attractive for load balancing in other collision integral algorithms. Both methods scale well, achieving 90% of ideal speedup at the maximum number of threads. Task parallelism seems to achieve slightly better speedup.

Next, we cross-compiled DVM to study scalability on MIC. Here, we only consider "parallel-do" threading. We study both scaling over cores (assuming four threads per core) and even division of the number of space cells (there were 100 spatial cells considered for this study). We compare the default and the "scatter" affinity settings. Results are shown in Figure 8. Scaling is excellent through 25 threads, beyond which performance degrades. Interestingly, the superior load balancing for 50 threads can yield better scaling than with 32 threads. Default affinity on MIC almost uniformly performs better than scatter affinity for large numbers of threads, with the exception of 50 threads

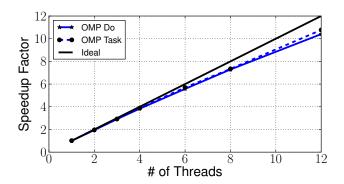


Fig. 7 Scaling of DVM on Westmere CPU. Scaling is relative to minimum of one thread. We compare both "parallel do" parallelism with "task" parallelism.

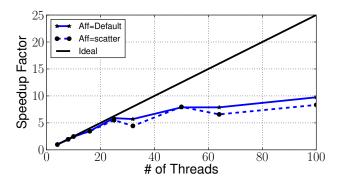


Fig. 8 Scaling of DVM on MIC. Scaling is relative to minimum of four threads.

where performance was comparable. A few experiments with "compact" affinity were not pursued after initial results showed worse performance than scatter affinity.

5 Summary

Overall, porting a number of different scientific applications and kernels to the KNF MIC architecture was relatively straightforward. 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 work-

loads 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 This work was supported in part by the Department of Energy [National Nuclear Security Administration] under Award Number [DE-FC52-08NA28615] and the National Science Foundation. 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

- Bobylev, A.V.: Exact solutions of the Boltzmann equation. Soviet Physics Doklady (Translation) 20(12), 822–824 (1976)
- Frigo, M., Johnson, S.G.: The design and implementation of FFTW3. Proceedings of the IEEE 93(2), 216–231 (2005). Special issue on "Program Generation, Optimization, and Platform Adaptation"
- Harding, M.E., Metzroth, T., Gauss, J., Auer, A.A.: Parallel calculation of ccsd and ccsd(t) analytic first and second derivatives. Journal of Chemical Theory and Computation 4(1), 64–74 (2008)
- 4. Jiménez, J., Moser, R.D.: What are we learning from simulating wall turbulence? Phil. Trans. Royal Soc. A **365**, 715–732 (2007)
- Kim, J., Moin, P., Moser, R.D.: Turbulence statistics in fully developed channel flow at low Reynolds number. J. Fluid Mech. 177, 133–166 (1987)
- Kindratenko, V., Enos, J., Shi, G., Showerman, M., Arnold, G., Stone, J., Phillips, J., Hwu, W.: Gpu clusters for high-performance computing. In: IEEE Cluster 2009 (2009)
- Kirk, B.S., Peterson, J.W., Stogner, R.H., Carey, G.F.:
 libMesh: A C++ Library for Parallel Adaptive Mesh
 Refinement/Coarsening Simulations. Engineering With
 Computers 22(3), 237–254 (2006)
- 8. Krook, M., Wu, T.T.: Exact solutions of the Boltzmann equation. Phys Fluids **20**(10), 1589–1595 (1977)
- Kwok, W.Y., Moser, R.D., Jiménez, J.: A critical evaluation of the resolution properties of B-Spline and compact finite difference methods. Journal of Computational Physics 174(2), 510–551 (2001)
- Lele, S.K.: Compact finite difference schemes with spectral-like resolution. Journal of Computational Physics 103(1), 16 – 42 (1992)
- Moin, P., Mahesh, K.: Direct Numerical Simulation: A tool in turbulence research. Annual Review of Fluid Mechanics 30(1), 539–578 (1998)
- Morris, A.B., Varghese, P.L., Goldstein, D.B.: Monte Carlo solution of the Boltzmann equation via a discrete velocity model. Journal of Computational Physics 230(4), 1265–1280 (2011)
- OpenMP: The OpenMP API Specification for Parallel Programming. http://openmp.org/wp/openmpspecifications/