Analyzing Performance of Selected NESAP Applications on the Cori HPC System

Thorsten Kurth[1], William Arndt[1], Taylor Barnes[1], Brandon Cook[1], Jack Deslippe[1], Doug Doerfler[1], Brian Friesen[1], Yun (Helen) He[1], Tuomas Koskela[1], Mathieu Lobet[1], Tareq Malas[1], Leonid Oliker[2], Andrey Ovsyannikov[1], Samuel Williams[2], Woo-Sun Yang[1], and Zhengji Zhao[1]

[1] National Energy Research Scientific Computing Center, Berkeley, CA, USA
[2] Computational Research Division, Lawrence Berkeley National Lab, Berkeley, CA, USA

Abstract. NERSC has partnered with over 20 representative application developer teams to evaluate and optimize their workloads on the Intel® Xeon Phi™ Knights Landing processor. In this paper, we present a summary of this two year effort and will present the lessons we learned in that process. We analyze the overall performance improvements of these codes quantifying impacts of both Xeon Phi™ architectural features as well as code optimization on application performance. We show that the architectural advantage, i.e. the average speedup of optimized code on KNL vs. optimized code on Haswell is about 1.1×. The average speedup obtained through application optimization, i.e. comparing optimized vs. original codes on KNL, is about 5×.

1 Introduction

The National Energy Research Scientific Computing Center (NERSC) [10] is the production HPC facility of the U.S. DOE Office of Science. It's mission is to enable and accelerate scientific discoveries through high performance computing and data analysis. The center supports over 6,000 users with more than 700 applications which cover a wide variety of science domains [7]. Therefore, HPC systems deployed at NERSC should not only support a diverse workload from a broad user base but also satisfy the increasing demand of computing cycles required to fulfill scientific goals. At the same time, power constraints for exascale computing are forcing major HPC and data centers to transition to more energy efficient-architectures.

At NERSC the transition to an energy-efficient pathway to exascale was realized via the procurement of the Cori system: a Cray XC40 powered by more than 9600 Intel® Xeon Phi™ 7250 (*Knights Landing*, KNL) based nodes which were added to an existing Cori phase I system powered by 1900+ Xeon™E5-2698 (*Haswell*) CPUs. The Xeon Phi™7250 is a self-hosted x86-64 compatible CPU. As such, in principal, all current NERSC users can immediately run their application without modification. In order to leverage the full capability of the Knights Landing architecture however, scientific applications often require some

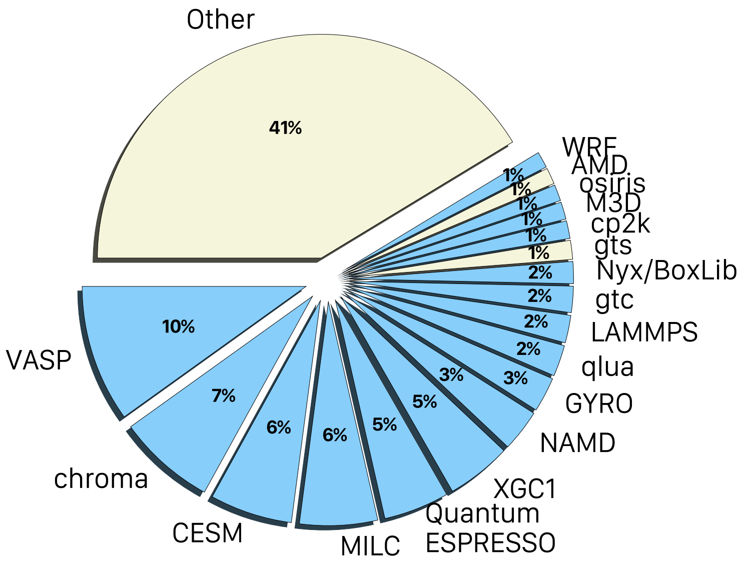

Fig. 1. Breakdown of NERSC workload in fractions of overall compute hour budget for 2015. NESAP applications are colored blue (note that the *Other* chunk includes four other NESAP apps).

non-trivial optimization. In order to facilitate this transition, NERSC has established the NERSC Exascale Science Application Program (NESAP) — a collaboration of NERSC staff along with experts at Cray and Intel, as well as the scientific application developers — with the goal of optimizing selected applications for the Xeon Phi$^{\text{TM}}$ architecture [8]. As shown by the blue regions of Fig. 1, the NESAP codes constitute about 60% of the overall NERSC workload.

In this paper, we present the results of the NESAP effort by discussing achieved speedups, lessons learned, and multi-node specific challenges developers might face when they aim at running their applications on KNL-based Cray XC40 systems at scale.

2 HPC Systems at NERSC

We will briefly describe the three major HPC systems at NERSC as well as compare the performance of (un-)optimized NESAP codes on all three systems later on. We consider the following systems:

– **Edison** is a Cray XC30 supercomputer with peak performance of about 2.57 PFLOP/s. It is comprised of 5,586 compute nodes with two 12-core Intel® Xeon$^{\text{TM}}$E5-4603 CPUs per node. Each of the 12 superscalar out-of-order cores runs at 2.4GHz and is capable of hosting 2 threads per core. Each cores supports the AVX instruction set and includes a 32KB L1 and 256KB

L2 cache, and each socket includes a shared 30MiB L3 cache and 32GiB of DDR3 memory (64GiB/node).

– **Cori-Haswell** represents the 1.92 PFLOP/s Haswell partition of the Cori Cray XC40 supercomputer. It is comprised of 2,004 compute nodes with two 16-core Intel® Xeon™E5-2698 CPUs per node. Each superscalar out-of-order core runs at 2.3GHz, has a similar cache architecture to those in Edison, but supports the AVX2 instruction set. Unlike Edison, each socket has a 40MiB L3 cache and has 64GiB of DDR4 main memory (128GiB/node).
– **Cori-KNL** is the KNL partition of the Cori Cray XC40 supercomputer. It has a peak performance of about 29.1 PFLOPS/s and is comprised of 9,688 self-hosted KNL compute nodes. Each KNL processor includes 68 cores running at 1.3GHz and capable of hosting 4 HyperThreads (272 HyperThreads per node). Each out-of-order superscalar core has a private 32KiB L1 cache and two 512-bit wide vector processing units (supporting the AVX-512 instruction set[3]). Each pair of cores (called "tile") shares a 1MiB L2 cache and each node has 96GiB of DDR4 memory and 16GiB of on-package high bandwidth (MCDRAM) memory. The MCDRAM memory can be configured into different modes, where the most interesting being *cache* mode in which the MCDRAM acts as a 16GiB L3 cache for DRAM. Additionally, MCDRAM can be configured in *flat* mode in which the user can address the MCDRAM as a second NUMA node. The on-chip directory can be configured into a number of modes, but in this publication we only consider *quad* mode, i.e. in *quad-cache* mode where all cores are in a single NUMA domain with MCDRAM acting as a *cache* for DDR, and in *quad-flat* mode where MCDRAM acts as a separate, *flat* memory domain.

All three systems feature the Cray Aries low-latency, high-bandwidth interconnect utilizing the dragonfly topology.

There are a number of challenges associated with optimizing codes for Xeon Phi™ . Perhaps the most obvious is that new sources of parallelism must be identified. This is not limited to only thread parallelism, but also includes vectorization opportunities. The latter imposes restrictions on data layouts (i.e. data should be preferably contiguous and 64-bit aligned) and data dependencies between loop iterations should be avoided. Furthermore, maximizing cache locality is more important as there is no on-chip L3 cache to capture misses. Finally, and this is important for multi-node scalability, a single Xeon Phi™ core can not saturate the injection rate of the Aries interconnect. Therefore, multiple cores (multiple threads or multiple MPI ranks per node) should be employed in order to achieve good performance. The detailed analysis of this is beyond the scope of this paper and can be found in another reference [20].

3 NESAP Results Overview

In this paper, we present the results from a variety of NESAP codes or their proxies. Table 1 displays an overview of these codes along with categorizations

[3] This includes the subsets F, CD, ER, PF but not VL, BW, DQ, IFMA, VBMI.

of their scientific field and, if applicable, the application they act as proxy for. The table further shows the most performance-critical kernels. Many of these kernels are representative for kernels in modern scientific codes used on a variety of HPC systems worldwide. The selection of codes further encompasses a broad variety of communication patterns (nearest neighbor exchanges or other point-to-point patterns, global reductions, all-to-all exchange, etc...) representative of those found in a wide range of applications.

Name	Scientific Field	Description	Kernels	Proxy
BerkeleyGW [19, 1]	Materials Science	MBPT	FFT, Linear Algebra	
CESM [2, 28]	Climate Modeling	Grid	Stencil(Multiple), Linear Algebra	WRF
Chombo-Crunch [42–44]	Multiple	AMR EB	EB Stencil(3D), Solver(AMG)	
Chroma [23, 29, 30]	Nuclear Physics	Lattice QCD	Stencil(4D), Solver(BiCGStab)	qlua
DWF	HEP	Lattice QCD	FFT, Stencil(5D), Solver(BiCGStab)	qlua
EMGeo [39, 38]	Geophysics	Grid	SpMM, Solver(IDR)	
GROMACS [4, 40]	Materials Science	Molecular Dynamics	Force Calculation	LAMMPS, NAMD
HISQ	HEP	Lattice QCD	Stencil(4D), Solver(BiCGStab)	
HMMER [5, 22]	Bioinformatics	Gene Annotation	Dynamic Programming(2D), Byteword Arithmetics	
MFDN [35, 36, 34, 18]	Nuclear Physics	Many Body	SpMM, Eigensolver(lanczos)	
MILC [6, 17]	HEP	Lattice QCD	Stencil(4D), Solver(BiCGStab)	qlua
MPAS-O [41, 37]	Climate Modeling	Unstructured Grid	Gather, Solver(RK4)	WRF
Nyx/BoxLib [14, 33, 24]	Multiple	AMR	Stencil(3D), Solver(GMG)	
Qbox [11, 26]	Materials Science	PW DFT	FFT, Linear Algebra, Eigensolver(lanczos)	cp2k
Quantum ESPRESSO [25, 16]	Materials Science	PW DFT	FFT, Linear Algebra, Eigensolver(lanczos)	
VASP [31]	Materials Science	PW DFT	FFT, Eigensolver(multiple)	
WARP [12, 45]	Accelerator Physics	PIC	Gather, Sort, FFT, Solver	osiris
XGC1 [13, 32, 27]	Fusion Research	PIC	Gather, Sort	gtc, gts, GYRO

Table 1. Overview of NESAP applications discussed in this paper including important references. The specified kernels represent the hot spots at the beginning of the NESAP effort. Due to optimization efforts, their importance relative to the rest of the code has decreased in general, but they still consume a significant fraction of the overall wall time.

3.1 Optimizations Summary

Historically, when a user is presented with a new architecture, they must often weigh the relative costs of porting and optimization effort against potential performance benefits. This is especially the case for Intel Xeon Phi$^{\text{TM}}$ as most x86-64 applications can run natively without modification. In the following sections we summarize the optimizations undertaken by the NESAP teams and quantify the performance benefits not only to KNL but also on traditional Xeons (Haswell, Ivy Bridge). We found the following techniques had the largest impact on a wide range of NESAP applications:

- *identifying and exploiting parallelism / creating more work for individual threads*: This maybe the most important thing to consider when switching from multi-core to many-core architectures. Small OpenMP sections that do not contain enough work for multiple threads will hurt performance significantly due to implicit barriers at the end of these sessions. Profiling usually highlights this as large `omp` or `kmp` sync/barrier overheads. Where possible, loop nests should be collapsed to maximize parallelism. Whereas perfect rectangular loop nests can be collapsed using the OpenMP 4 `collapse` clause, more complicated loop structures often require more manual transformations including data structure rearrangements such as extending array dimensions to allow for batched processing. We found the latter to be especially beneficial for batched node-local Fast Fourier Transforms (FFT).
- *loop tiling*: Cache blocking to achieve cache locality of heavily used arrays can be realized by reordering and tiling inner loops. This advice is not new as it is in general a good practice to optimize code for L1/L2 accesses. On Xeon Phi$^{\text{TM}}$ this is even more important as there is no L3 cache to mitigate the impact of L2 misses on application performance. Unfortunately, as this is a manual code transformation rather than a directive, code can become less readable and more brittle. Nevertheless, this technique benefits application performance on most architectures. In terms of loop tile sizes, we found that blocking to shared L2, i.e. 512KiB/core, performs best for most applications.
- *short loop unrolling*: Short loops do not provide sufficient work for either threads or the wide vector registers. Instead it is beneficial to unroll them using compiler directives or manual unrolling.
- *ensuring efficient vectorization*: This may sound obvious but can often result in a major challenge as it may entail loop reordering, loop restructuring, and/or data layout transformations. It is nevertheless desirable not only because there is a potential 16$\times$ loss in performance from not vectorizing (vs. 4$\times$ on Ivy Bridge), but it also affects memory and cache bandwidth as single element loads and stores are inefficient. Further compounding this challenge on scientific codes, efficient mathematical function implementations for square roots, exponentials, etc. are only available as vectorized variants. Where the compiler is deficient in auto-vectorizing parallel loops, compiler hints and the OpenMP 4 `simd` pragmas were found to be particularly useful.
- *using optimized mathematical functions*: This is related to the previous point that vectorization enables the compiler to utilize efficient implementations

of expensive functions. Unfortunately, their generation may only be enabled by instructing the compiler to use a relaxed floating-point model. Under the same restrictions, compilers may not be able to factor a divide by a loop constant out of an inner loop. We found that manually factoring out the divide (by multiplying by the inverse of the loop-carried constant) could significantly improve performance.

This list is not meant to be a complete guide, and we recommend reviewing some of our NESAP case studies which discuss some of these topics [9] and a previous overview of the NESAP program results [15]. Nevertheless, this list can serve as a guideline for developers who aim at getting their current codes ready for Xeon Phi$^{\mathrm{TM}}$.

3.2 Optimized vs. Original

We will first compare the speedup achieved by optimizing the code for Xeon Phi$^{\mathrm{TM}}$ on the three different systems. The current state of this effort is displayed in Figure 2. It shows the speedup of optimized vs. original codes on all HPC systems at NERSC for a typical production partition size. Single node results represent capacity workloads with ideal weak scaling. The plot shows several important results: speedups of up to 17× (on KNL) to about 6× (on Haswell) could be achieved. The diagram shows that optimizations targeting Xeon Phi$^{\mathrm{TM}}$ can significantly benefit multi-core architectures as well. The main reason for that is that the optimized applications feature improved cache locality, contiguous aligned data access which facilitates vectorization and offers better thread-level parallelism. The improvements for some applications have an even bigger effect on Cori-Haswell. For example, this can happen if chunks of data are accessed in a random fashion but those chunks fit into the big L3 cache of the two multi-core architectures but not into L2 on Xeon Phi$^{\mathrm{TM}}$. These problems are usually memory latency bound and MCDRAM does not offer a significant advantage over conventional DRAM. An example for this is Chombo, which utilizes comparably large lookup tables in order to retrieve memory locations of the next relevant chunk of data. Another more obvious reason is that serial sections or sections with insufficient vectorization are hurting Xeon Phi$^{\mathrm{TM}}$ performance more than conventional multi-core architectures. The median speedup achieved on Xeon Phi$^{\mathrm{TM}}$ is 2.8×, which generated a median speedup of about 1.7× and 1.4× on Edison and Cori-Haswell respectively.

3.3 Manycore vs. Multicore

Perhaps one of the most fundamental questions is to quantify the performance advantage provided by manycore architectures like KNL compared to traditional multicore architectures like Ivy Bridge and Haswell. Figure 3 shows the speedup of the optimized codes on Cori-KNL with respect to Cori-Haswell and Edison. It shows that almost all applications on Xeon Phi$^{\mathrm{TM}}$ exceed Edison's performance (node for node) by at least 30% with similar power-requirements per node. We

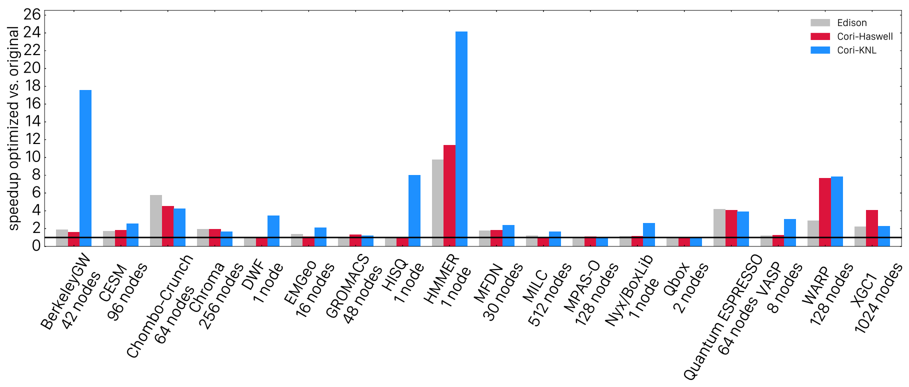

Fig. 2. Performance of optimized vs. original codes on the three major HPC systems/partitions at NERSC. The number of nodes mentioned below the application name are representative for a typical production run on the Cori-KNL system. The single node numbers represent embarrassingly parallel capacity workloads.

should mention that almost all original versions of NESAP applications, except for some heavily memory bound applications such as EMGeo, were initially significantly slower on Xeon Phi^{TM} than on Haswell and some even compared to Edison.

Compared to Cori-Haswell (a contemporaneous architecture), in many cases, the performance difference is not that significant. MFDn for example shows a huge speedup on Cori-KNL compared to Edison, but not to Cori-Haswell. This might look surprising as the architectural differences between Edison and Cori-Haswell are not very big, but there are three significant differences which can cause this behavior. MFDn constructs a huge sparse matrix at first. In this construction, vector instruction gather and broadcast routines are available on Haswell (AVX2) and, in an improved version on Xeon Phi^{TM} (AVX-512), that offer a significant advantage over individual loads and stores that might be used on Edison. Furthermore, the construction step used bitwise comparisons (XOR) that can be accelerated with AVX2(Haswell) and AVX-512(KNL), and the linear algebra part benefits from the fused multiply-add instructions also only available in AVX2 and AVX-512. The combination of all three effects can cause a significant architectural benefit for Haswell and Xeon Phi^{TM} over Ivy Bridge (Edison).

For the other applications the picture looks more consistent, where some applications favor Haswell over Xeon Phi^{TM} . Quantum ESPRESSO for example is very similar to BerkeleyGW and VASP but performs worse on Xeon Phi^{TM} than on Haswell. This is mainly due to inefficiencies in the eigenvalue solver: Quantum ESPRESSO can utilize SCALAPACK and ELPA but both libraries seem to have insufficient support for threading and/or vectorization. Other parts of the code, for example sections which heavily employ FFT and dense linear algebra, perform much better on Xeon Phi^{TM} than on Haswell.

The median overall speedups over Edison and Cori-Haswell are 1.8× and 1.1× respectively when running code optimized for the target machine.

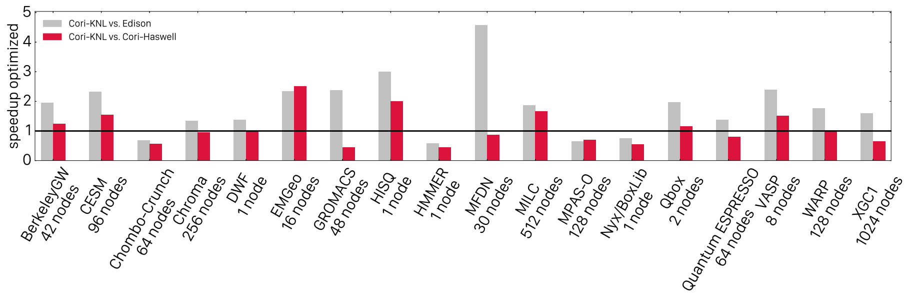

Fig. 3. Speedups of optimized NESAP codes on Cori-KNL vs. Cori-Haswell and Edison.

3.4 Value of Wider Vectors (AVX-512)

Another question we asked is whether AVX-512 offers a significant advantage over AVX2. Theoretically, the former offers a potential 2× speedup (ignoring bandwidth) because the vector units are twice as wide. However, it forces the developer to restructure for longer unit-stride access with no data dependencies and thus might restrict the application design in undesirable ways. Figure 4 shows the speedups achieved by running the application on Xeon Phi$^{\text{TM}}$ with either AVX-512 or AVX2 enabled. That is, for the same code, architecture, and compiler, what is the value of doubling the vector length. For applications that depend on libraries, we ensure the appropriate libraries were linked or environment variables were set (e.g. for selecting the instruction set in Intel MKL). In case of Chroma and MILC, which utilize QPhiX[29] which in turn uses a domain specific language to generate architecture dependent code [30], we made sure that the instruction level support was consistent. Figure 4 shows that the value of doubling the vector length varies significantly (naively, a 100% speed is expected because the vector lanes are twice as wide). Benefits lower than 100% can be attributed to multiple factors. The simplest reason for this speedup is that the code suffers from a low degree of vectorization. Another explanation is that code is memory bandwidth bound and thus cannot benefit fully from vectorization. However, it turns out that even bandwidth-bound codes such as MFDn or EMGeo or MILC can be significantly accelerated by using AVX-512. Although applying the Roofline Model [47, 46, 21] to such codes suggests there should be little gain, the reality is that AVX-512 instructions reduce contention in the pipeline and inject more parallelism into the memory subsystem thereby allowing for higher bandwidth. Codes such as DWF and EMGeo that observe more than 2× might benefit from advanced AVX-512 features such as masking.

This allows AVX-512 compilers to vectorize loops with certain types of conditionals which otherwise would not vectorize under AVX2. EMGeo vectorizes the solver over multiple right hand sides and relies on this masking for removing converged right hand sides from the solve. Additionally, AVX-512 provides 32 registers and thus twice as many as AVX2. For some codes, these extra registers likely mitigate register spill performance penalties. Finally, there are other advanced features such as optimized mathematical functions and broadcast operations which can give a gain exceeding the expected gain. Chroma is a special case as the performance for AVX-512 or AVX2 seems to be the same. At the time of writing, we could not find a satisfying explanation for that behavior but we have to note that about 90% of the time is spent in QPhiX and the rest in plain Chroma. The Chroma part utilizes LLVM with JIT and we had to disable AVX-512 JIT-support in because of an LLVM compiler bug. This means that this part of the code actually uses AVX2 in both cases. However, the time dominating part of the code should be sensitive to the instruction set and we cannot explain the differences here.

Ultimately, the median speedup achieved by using AVX-512 in lieu of AVX2 is approximately 1.2×.

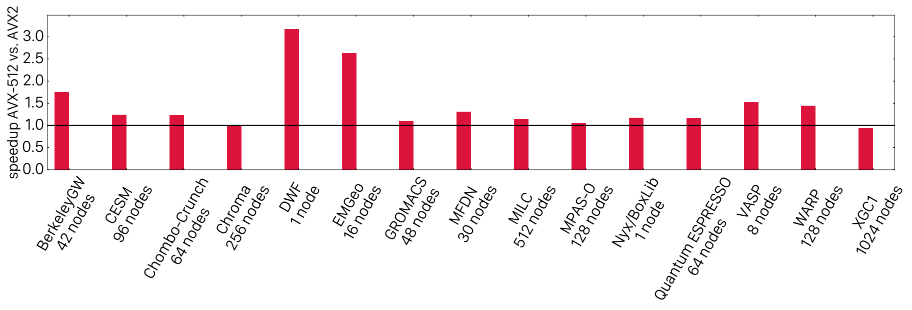

Fig. 4. Speedup from AVX-512 over AVX2 for optimized NESAP codes on Cori-KNL.

3.5 Flat and Cache Memory Mode Comparison

No memory technology simultaneously provides high capacity, high bandwidth, and energy efficiency. Thus, KNL instantiates two distinct memories — an energy-efficient, high-capacity DDR, and a high-bandwidth MCDRAM. The KNL architecture can be configured to present these memories to the user as either two distinct memories (*flat mode*) or can be configured to treat the MCDRAM as a cache for DDR (*cache mode*). In this section we quantify the performance differences of using either cache or flat mode or not using MCDRAM at all; we do not consider hybrid modes as we have not identified any suitable use cases thus far.

Figure 5 shows the speedup attained with flat mode over the simpler cache mode as well as the benefit of MCDRAM over pure DDR. The figure clearly exhibits that MCDRAM should be used in any case as the performance was never worse than running from DDR for our selected applications. Furthermore, the use of MCDRAM can significantly speed up heavily memory bandwidth limited codes. For cache vs. flat the story is more complicated: we observe that the best performance gains for our codes are 15-20%. The codes that perform equally well in either mode have local problem sizes which fit into MCDRAM and thus suffer no MCDRAM cache capacity misses. Codes that show a significant performance penalty in flat mode (ChomboCrunch, DWF, and Qbox) feature local problem sizes that cannot entirely fit into MCDRAM. Instead of utilizing AutoHBW or compiler directives for selectively placing *hot* arrays into MCDRAM, they use `numactl -p 1` to prefer memory allocation in MCDRAM[4]. Unfortunately, this only places the first $\mathcal{O}(16\text{GiB})$ of allocated data in MCDRAM and the rest will be allocated in DDR. With that approach, a speedup can only be achieved if all *hot* arrays are allocated at the beginning and if they fit into MCDRAM. Nevertheless, codes that use pool allocators such as e.g. HISQ and Chroma can safely use this procedure. For all other codes we conclude that cache mode should be favored if one wishes minimal code modification.

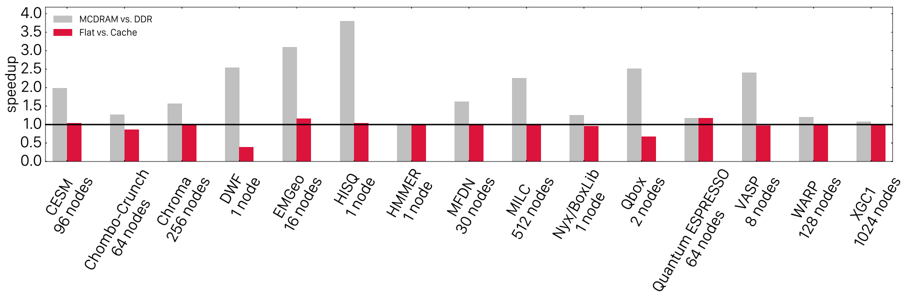

Fig. 5. Speedups of optimized NESAP codes achieved by running from MCDRAM vs. DDR and in flat vs. cache mode on Cori-KNL.

3.6 Total Savings in CPU Hours

We can now estimate the overall savings in units of CPU hours for the NERSC workload due to optimized applications and KNL architectural features. We assume that the CPU time fractions for the individual codes will be the same on Cori-KNL as those on Edison in 2015, and that the speedups are representative for the overall workload, the problem sizes are representative for the typical use

[4] `numactl -p 1` mimics the behavior of `numactl -m 1` but it is safer as it will not abort execution if there is no remaining free space in MCDRAM.

of the specific application at NERSC, and users actually use the KNL-optimized versions. Based on these assumptions we can combine the data from Fig. 1 with the speedups achieved in Figure 2. This yields an expected saving of $\sim$1.8B CPU hours by using the optimized code instead of the original code on Cori-KNL. This is about 23% of total available CPU hours. Since NERSC charges by the node-hour, the savings are real and can be used by the application teams to tackle more complicated science problems.

4 Conclusions

We have presented overall and relative performance improvements of selected NESAP applications and discussed specifics of Xeon Phi$^{\text{TM}}$ that have to be considered when applications are optimized for this architecture. We further showed that improvements targeting Xeon Phi$^{\text{TM}}$ will usually benefit conventional multi-core architectures. Thus, it can be beneficial for developers to start adapting their codes to many-core systems even if they are still primarily targeting multi-core architectures. Those improvements mainly target memory locality by applying cache blocking to L2, and loop and data layout restructuring to exploit parallelism and facilitate vectorization. Using a combination of these techniques is essential if one is to outperform traditional multi-core architectures.

Acknowledgement

Research used resources of NERSC, a DOE Office of Science User Facility supported by the Office of Science of the U.S. DOE under Contract No. DE-AC02-05CH11231. This article has been authored at Lawrence Berkeley National Lab under Contract No. DE-AC02-05CH11231 and UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the United States Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan [3].

References

1. BerkeleyGW Website, `http://www.berkeleygw.org`
2. CESM Web Site, `http://www.cesm.ucar.edu`
3. DOE Public Access Plan, `https://energy.gov/downloads/doe-public-access-plan`
4. GROMACS Web Site, `http://www.gromacs.org`
5. HMMER Web Site, `http://hmmer.org/`

6. MILC Website, http://physics.indiana.edu/~sg/milc.html
7. NERSC and DOE Requirements Reviews Series, http://www.nersc.gov/science/hpc-requirements-reviews/
8. NERSC NESAP applications, http://www.nersc.gov/users/computational-systems/cori/nesap/nesap-projects/
9. NERSC NESAP case studies, http://www.nersc.gov/users/computational-systems/cori/application-porting-and-performance/application-case-studies/
10. NERSC Web Site, http://www.nersc.gov
11. QBox Web Site, http://qboxcode.org
12. Warp Web Site, http://warp.lbl.gov
13. XGC1 Web Site, http://epsi.pppl.gov/computing/xgc-1
14. Almgren, A.S., Bell, J.B., Lijewski, M.J., Lukić, Z., Van Andel, E.: Nyx: A Massively Parallel AMR Code for Computational Cosmology. The Astrophysical Journal 765, 39 (Mar 2013)
15. Barnes, T., Cook, B., Deslippe, J., Doerfler, D., Friesen, B., He, Y.H., Kurth, T., Koskela, T., Lobet, M., Malas, T., Oliker, L., Ovsyannikov, A., Sarje, A., Vay, J.L., Vincenti, H., Williams, S., Carrier, P., Wichmann, N., Wagner, M., Kent, P., Kerr, C., Dennis, J.: Evaluating and optimizing the nersc workload on knights landing. In: Proceedings of the 7th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computing Systems. pp. 43–53. PMBS '16, IEEE Press (2016)
16. Barnes, T.A., Kurth, T., Carrier, P., Wichmann, N., Prendergast, D., Kent, P.R., Deslippe, J.: Improved treatment of exact exchange in quantum espresso. Computer Physics Communications 214, 52–58 (2017)
17. Bauer, B., Gottlieb, S., Hoefler, T.: Performance modeling and comparative analysis of the MILC Lattice QCD application su3_rmd. In: Proc. CCGRID2012: IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (2012)
18. Binder, S., Calci, A., Epelbaum, E., Furnstahl, R.J., Golak, J., Hebeler, K., Kamada, H., Krebs, H., Langhammer, J., Liebig, S., Maris, P., Meißner, U.G., Minossi, D., Nogga, A., Potter, H., Roth, R., Skiniński, R., Topolnicki, K., Vary, J.P., Witała, H.: Few-nucleon systems with state-of-the-art chiral nucleon-nucleon forces. Phys. Rev. C 93(4), 044002 (2016)
19. Deslippe, J., Samsonidze, G., Strubbe, D.A., Jain, M., Cohen, M.L., Louie, S.G.: Berkeleygw: A massively parallel computer package for the calculation of the quasiparticle and optical properties of materials and nanostructures. Computer Physics Communications 183(6), 1269 – 1289 (2012), http://www.sciencedirect.com/science/article/pii/S0010465511003912
20. Doerfler, D., Austin, B., Cook, B., Deslippe, J., Kandalla, K., Mendygral, P.: Evaluating the networking characteristics of the cray xc-40 intel knights landing based cori supercomputer at nersc. In: Cray User Group Meeting (CUG) 2017 (May 2017)
21. Doerfler, D., Deslippe, J., WIlliams, S., Oliker, L., Cook, B., Kurth, T., Lobet, M., Malas, T., Vay, J.L., Vincenti, H.: Applying the Roofline Performance Model to the Intel Xeon Phi Knights Landing Processor. In: International Conference on High Performance Computing. vol. 9945, pp. 339–353. Springer (2016)
22. Eddy, S.R.: Accelerated profile hmm searches. PLOS Computational Biology 7(10), 1–16 (10 2011), https://doi.org/10.1371/journal.pcbi.1002195
23. Edwards, R.G., Joo, B.: The Chroma software system for lattice QCD. Nucl. Phys. Proc. Suppl. 140, 832 (2005)

24. Friesen, B., Almgren, A., Lukić, Z., Weber, G., Morozov, D., Beckner, V., Day, M.: In situ and in-transit analysis of cosmological simulations. Computational Astrophysics and Cosmology 3, 4 (Aug 2016)
25. Giannozzi, P., Baroni, S., Bonini, N., Calandra, M., Car, R., Cavazzoni, C., Ceresoli, D., Chiarotti, G.L., Cococcioni, M., Dabo, I., Corso, A.D., de Gironcoli, S., Fabris, S., Fratesi, G., Gebauer, R., Gerstmann, U., Gougoussis, C., Kokalj, A., Lazzeri, M., Martin-Samos, L., Marzari, N., Mauri, F., Mazzarello, R., Paolini, S., Pasquarello, A., Paulatto, L., Sbraccia, C., Scandolo, S., Sclauzero, G., Seitsonen, A.P., Smogunov, A., Umari, P., Wentzcovitch, R.M.: Quantum espresso: a modular and open-source software project for quantum simulations of materials. Journal of Physics: Condensed Matter 21(39), 395502 (2009), `http://stacks.iop.org/0953-8984/21/i=39/a=395502`
26. Gygi, F.: Architecture of qbox: A scalable first-principles molecular dynamics code. IBM J. Res. Dev. 52(1/2), 137–144 (Jan 2008), `http://dl.acm.org/citation.cfm?id=1375990.1376003`
27. Hager, R., Yoon, E., Ku, S., D'Azevedo, E., Worley, P., Chang, C.: A fully nonlinear multi-species fokkerplancklandau collision operator for simulation of fusion plasma. Journal of Computational Physics 315, 644 – 660 (2016), `http://www.sciencedirect.com/science/article/pii/S0021999116300298`
28. Hurrell, J., Holland, M., Gent, P., Ghan, S., Kay, J., Kushner, P., Lamarque, J.F., Large, W., Lawrence, D., Lindsay, K., Lipscomb, W., Long, M., Mahowald, N., Marsh, D., Neale, R., Rasch, P., Vavrus, S., Vertenstein, M., Bader, D., Collins, W., Hack, J., Kiehl, J., Marshall, S.: The Community Earth System Model: a framework for collaborative research. Bulletin of the American Meteorological Society 94, 1339–1360 (2013)
29. Joó, B.: `qphix` package web page. `http://jeffersonlab.github.io/qphix`
30. Joó, B.: `qphix-codegen` package web page. `http://jeffersonlab.github.io/qphix-codegen`
31. Kresse, G., Furthmueller, J.: Efficiency of ab-initio total energy calculations for metals and semiconductors using a plane-wave basis set. Computational Materials Science 6(1), 15 – 50 (1996), `http://www.sciencedirect.com/science/article/pii/0927025696000080`
32. Ku, S., Chang, C., Diamond, P.: Full-f gyrokinetic particle simulation of centrally heated global itg turbulence from magnetic axis to edge pedestal top in a realistic tokamak geometry. Nuclear Fusion 49(11), 115021 (2009)
33. Lukić, Z., Stark, C.W., Nugent, P., White, M., Meiksin, A.A., Almgren, A.: The Lyman α forest in optically thin hydrodynamical simulations. Monthly Notices of the Royal Astronomical Society 446, 3697–3724 (Feb 2015)
34. Maris, P., Caprio, M.A., Vary, J.P.: Emergence of rotational bands in ab initio no-core configuration interaction calculations of the Be isotopes. Phys. Rev. C 91(1), 014310 (2015)
35. Maris, P., Vary, J.P., Navratil, P., Ormand, W.E., Nam, H., Dean, D.J.: Origin of the anomalous long lifetime of ^{14}C. Phys. Rev. Lett. 106(20), 202502 (2011)
36. Maris, P., Vary, J.P., Gandolfi, S., Carlson, J., Pieper, S.C.: Properties of trapped neutrons interacting with realistic nuclear Hamiltonians. Phys. Rev. C 87(5), 054318 (2013)
37. Petersen, M.R., Jacobsen, D.W., Ringler, T.D., Hecht, M.W., Maltrud, M.E.: Evaluation of the arbitrary lagrangian-eulerian vertical coordinate method in the mpas-ocean model. Ocean Modelling 86, 93 – 113 (2015), `http://www.sciencedirect.com/science/article/pii/S1463500314001796`

38. Petrov, P.V., Newman, G.A.: Three-dimensional inverse modelling of damped elastic wave propagation in the fourier domain. Geophysical Journal International 198(3), 1599–1617 (2014)
39. Petrov, P.V., Newman, G.A.: 3d finite-difference modeling of elastic wave propagation in the laplace-fourier domain. GEOPHYSICS 77(4), T137–T155 (2012), `http://dx.doi.org/10.1190/geo2011-0238.1`
40. Pronk, S., Pll, S., Schulz, R., Larsson, P., Bjelkmar, P., Apostolov, R., Shirts, M.R., Smith, J.C., Kasson, P.M., van der Spoel, D., Hess, B., Lindahl, E.: Gromacs 4.5: a high-throughput and highly parallel open source molecular simulation toolkit. Bioinformatics 29(7), 845 (2013), `+http://dx.doi.org/10.1093/bioinformatics/btt055`
41. Ringler, T., Petersen, M., Higdon, R.L., Jacobsen, D., Jones, P.W., Maltrud, M.: A multi-resolution approach to global ocean modeling. Ocean Modelling 69, 211 – 232 (2013), `http://www.sciencedirect.com/science/article/pii/S1463500313000760`
42. Straalen, B.V., Trebotich, D., Ovsyannikov, A., Graves, D.T.: Exascale Scientific Applications: Programming Approaches for Scalability Performance and Portability, chap. Scalable Structured Adaptive Mesh Refinement with Complex Geometry. CRC Press (in press)
43. Trebotich, D., Adams, M.F., Molins, S., Steefel, C.I., Chaopeng, S.: High-resolution simulation of pore-scale reactive transport processes associated with carbon sequestration. Computing in Science & Engineering 16(6), 22–31 (2014)
44. Trebotich, D., Graves, D.: An adaptive finite volume method for the incompressible Navier–Stokes equations in complex geometries. Communications in Applied Mathematics and Computational Science 10(1), 43–82 (2015)
45. Vincenti, H., Lobet, M., Lehe, R., Sasanka, R., Vay, J.L.: An efficient and portable {SIMD} algorithm for charge/current deposition in particle-in-cell codes. Computer Physics Communications 210, 145 – 154 (2017), `http://www.sciencedirect.com/science/article/pii/S0010465516302764`
46. Williams, S., Waterman, A., Patterson, D.: Roofline: An Insightful Visual Performance Model for Multicore Architectures. Commun. ACM 52(4), 65–76 (Apr 2009), `http://doi.acm.org/10.1145/1498765.1498785`
47. Williams, S.W.: Auto-tuning Performance on Multicore Computers. Ph.D. thesis, EECS Department, University of California, Berkeley (Dec 2008), `http://www2.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-164.html`