MILC Code Performance on High End CPU and GPU Supercomputer Clusters

Carleton DeTar[1], *Steven* Gottlieb[2,⋆], *Ruizi* Li[2,⋆⋆], and *Doug* Toussaint[3]

[1]*Department of Physics & Astronomy, University of Utah, Salt Lake City, UT 84112, U.S.A.*
[2]*Department of Physics, Indiana University, Bloomington, IN 47405, U.S.A.*
[3]*Department of Physics, University of Arizona, Tucson, AZ 85721, U.S.A.*

Abstract. With recent developments in parallel supercomputing architecture, many core, multi-core, and GPU processors are now commonplace, resulting in more levels of parallelism, memory hierarchy, and programming complexity. It has been necessary to adapt the MILC code to these new processors starting with NVIDIA GPUs, and more recently, the Intel Xeon Phi processors. We report on our efforts to port and optimize our code for the Intel Knights Landing architecture. We consider performance of the MILC code with MPI and OpenMP, and optimizations with QOPQDP and QPhiX. For the latter approach, we concentrate on the staggered conjugate gradient and gauge force. We also consider performance on recent NVIDIA GPUs using the QUDA library.

1 Introduction

The MILC code has been in production and freely available for over 20 years, with continual improvements to match our evolving physics goals and changing hardware. Currently a code of approximately 180,000 lines, it is used by several collaborations worldwide. Originally, there was a single level of parallelization based on message passing. With the advent of MPI, that became the main message passing library used, though there are some others as mentioned below. OpenMP parallelization was briefly tried around 2000, but is now more fully developed. The code has made increasing use of a library of specialized data-parallel linear algebra and I/O routines developed over the past several years with support from the DOE's Scientific Discovery through Advanced Computing (SciDAC) Program. In addition, we make use of the QUDA library for computers with NVIDIA GPUs. We have been porting the code to the Intel Xeon Phi many-integrated-core (MIC) architecture named Knights Landing (KNL). This architecture contains 512-bit SIMD vector processors that can do floating point arithmetic on 16 single precision or 8 double precision numbers at a time. To exploit the power of these vector units, we have been generalizing the QPhiX library [1] for Wilson/Clover quarks developed by Jefferson Lab and Intel to support staggered quarks. Currently, the staggered QPhiX library contains code for multi-shift and single mass conjugate gradient solvers. In addition, a Symanzik one-loop improved gauge force is available for gauge field generations. Routines to support smearing for HISQ quarks are under development. In addition, we

⋆Acknowledges financial support by Intel Parallel Computing Center at Indiana University.
⋆⋆Speaker, e-mail: ruizli@umail.iu.edu

are improving the OpenMP parallelization in the MILC code.

In the rest of this paper, we will briefly review the development of the MILC code, describe the three main libraries that the code can call to enhance performance, and then present a number of benchmarks. Finally, we present some conclusions.

2 Development of the MILC code

The MILC code was originally developed to allow single processor or multiprocessor (with message passing) running in a way that is transparent to the developer. Selecting a MILC communication file at compile time was the only difference between serial and parallel running. All of the application code was otherwise identical. Further, there were several message passing libraries at the time, so MILC had one for each message passing library, plus `com_vanilla.c` that did not need to pass messages but implemented the MILC calls like `start_gather`, `wait_gather`, and `cleanup_gather`. This isolated our users and developers from having to understand multiple message passing systems. Eventually, MPI won out. Later, OpenMP was supported to make use of symmetric multiprocessing (SMP) machines with the shared memory [2]; however, this required examining individual `FORALLSITES` loops and new `FORALLSITES_OMP`. Only with the advent of the Intel MIC architecture this did again become an area of focus.

Support for GPUs is based on the QUDA library [3], originally developed at Boston University (BU). The extension to staggered quarks began when one of us was on sabbatical at the National Center for Supercomputing Applications (NCSA) at the University of Illinois. The QUDA community has grown over the years and benefits greatly from the leadership of NVIDIA staff members Kate Clark, one of the original BU developers, and Mathias Wagner, a former MILC postdoc.

3 Libraries: SciDAC, QUDA, staggered QPhiX

The basic MILC code can call three packages to improve performance. USQCD's SciDAC-funded software package is used by the MILC code for runs on various CPU architectures. The software packages include QLA for basic linear algebra, QDP for data parallel routines, and QOPQDP, the high-level optimized library that supports functions such as CG solvers, gauge force, etc. The SciDAC package can use SSE2 instructions, but not newer SIMD instructions sets such as AVX512.

To make use of GPUs on machines with NVIDIA hardware, the MILC code calls the QUDA library which has all of the routines needed for creating new gauge fields. Of course, that includes the solvers required to make quark propagators in analysis codes. QUDA supports mixed-precision calculation as well as more than one method of gauge-field compression to improve performance. The version benchmarked here is 0.8.0.

We have been developing the staggered QPhiX library as part of our effort to port the MILC code to the MIC architecture. QPhiX targets the first and second generations of this architecture. It also supports other instruction sets, e.g., AVX2, SSE. This library supports SMID vectorization via intrinsics, and OpenMP threading. As each MPI task can call the library, the resulting code supports all three levels of parallelization available on the Intel Xeon Phi, or the new Skylake chips. A detailed description of the QPhiX library framework can be found at [1]. The library uses a structure-of-array (SOA) data structure for improved cache reuse. The data layout is the same as in the Grid library [4].

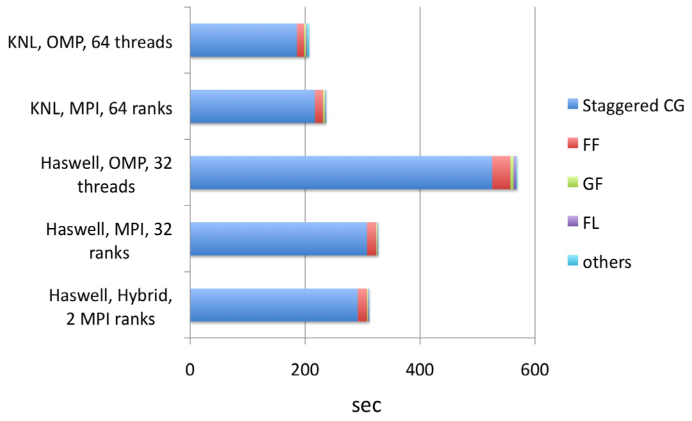


Figure 1. Runtime breakdown (in seconds) for the current version of the MILC code that also includes improved OpenMP parallelizations, on a single KNL 7250 or Haswell dual-socket 16-core Intel Xeon E5-2698 v3 node, with MPI, OpenMP, and hybrid MPI/OpenMP parallelism. The bars in red, green, and purple denote the HISQ fermion force (FF), Symanzik gauge force (GF), and HISQ link smearing (FL) time, respectively. The lattice size is 16^4. No hyper-threading was enabled, and the run on Haswell with hybrid parallelism has 16 threads per rank.

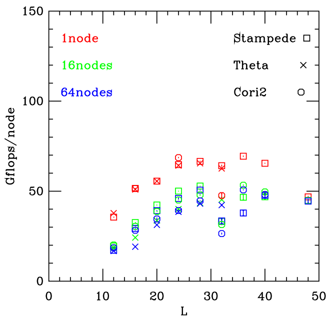
4 Benchmarks

The benchmarks shown here use the `su3_rhmd_hisq` application in the MILC code. This code implements gauge-field evolution using the RHMD algorithm for HISQ quarks. All results present here are for double precision. As expected, single-precision code performance is about twice as fast.

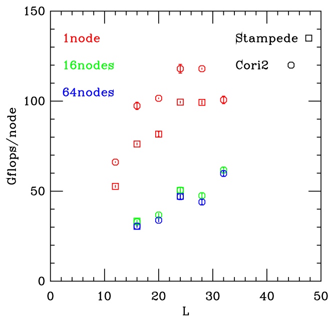
In Figure 1, we present a bar chart showing how time is spent in various parts of the code using various hardware and software combinations. Starting from a reasonably equilibrated 16^4 configuration, we see that about 90% of the time is spent in the CG routine, which is shown in blue. The HISQ fermion force, gauge force, and HISQ-link smearing are shown using other colors explained in the legend and caption. The top two bars are for runs on a KNL node with a single chip and the other three are for a dual socket Intel Haswell node. The slowest performance is from a pure OpenMP run on the dual-socket Haswell. However, a hybrid MPI/OpenMP run with one MPI task per socket beats a pure MPI run with 32 MPI tasks. On the KNL node, pure OpenMP with 64 threads is slightly faster than MPI with 64 ranks.

4.1 Staggered multi-shift CG

As mentioned, the staggered multi-shift CG is the most time-consuming part of the code in production runs. We carried out a set of weak-scaling benchmarks of this routine. All of our KNL benchmarks were collected on three clusters: Stampede 2 [5] at the Texas Advance Computing Center (TACC), Theta [6] at the Argonne Leadership Computing Center (ALCF), and Cori II [7] at the National



(a) Baseline code on three KNL clusters, shows peak performance at lattice size between $L = 24$ and 36.



(b) Code with QPhiX on two KNL clusters. One node runs are with OpenMP, and multe-node runs are with up to 16 ranks per node.

Figure 2. Staggered multi-shift CG weak-scaling performance on KNL clusters up to 64 nodes, with baseline code (left) and QPhiX (right). The horizontal axis encodes the lattice size per node L^4, and the vertical axis is the flop rate per node. The performance improvement with QPhiX can be over 50% on one node, but much less on multiple nodes.

Energy Research Supercomputing Center (NERSC). Each of these clusters has KNL nodes with either 64 or 68 cores per node, Theta and Cori II use the Cray Aries network, whereas Stampede 2 uses the Intel Omni-path network. The maximum number of nodes used was 2048, on Cori II, covering around half of the entire cluster at the time of run. Results for the baseline code below refer to the MILC code with MPI only, unless noted otherwise.

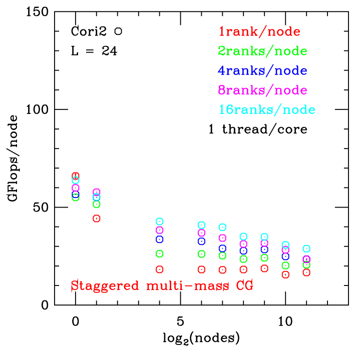
Figure 2 shows the staggered multi-shift CG performance on up to 64 KNL nodes on various clusters, comparing baseline code and optimization with QPhiX, as well as various lattice volumes L^4 per node. The overall performance improvement with QPhiX was over 50% on a single KNL node. We observed up to 120 Gflops/sec. On multiple nodes the performance gain with QPhiX was reduced, due to a network bottleneck for this routine, especially with L less than 30.

Another weak-scaling plot from the Cori II cluster is shown in figure 3. This exhibits the same pattern of the performance bottleneck for $L = 24$ due to network limitations and the need for cross-node communication. The issue is not as severe for a larger lattice size $L = 32$. We observed similar performance with hyper-threading of two threads per core, while increasing to four hyper-threads per core negatively impacted the performance.

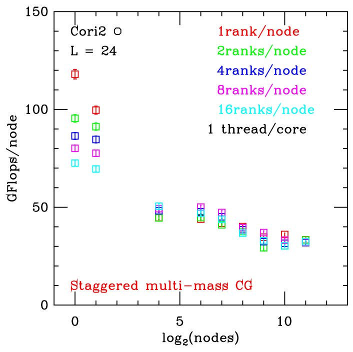
Figure 4 shows the performance of this routine on one GPU node using two different NVIDIA architectures, either K20 or P100. The kernel QUDA CG performance on one P100 is around 240 Gflops/sec, about five times of that of on single K20.

4.2 Symanzik one-loop gauge force and HISQ fermion force

The HISQ RHMC/RHMD algorithm also requires the calculation of the gauge force and the fermion force. These calculations are dominated by matrix-matrix multiplies, rather than by matrix-vector multiplies as in the CG. Thus, they have a higher arithmetic intensity and are less memory bandwidth



(a) Baseline code performance on multiple nodes is higher with more MPI ranks per node.



(b) Code with QPhiX performance on multiple nodes does not vary that much when varying the number of MPI ranks per node.

Figure 3. Staggered multi-shift CG weak-scaling performance on the Cori II KNL cluster using up to 2048 nodes, with baseline code (left) and QPhiX code (right). The vertical axis is the flop rate per node. The lattice volume is 24^4 per node. Runs use no hyper-threading (*i.e.*, 64 total threads per node), and various MPI/OpenMP combinations of up to 16 ranks per node.

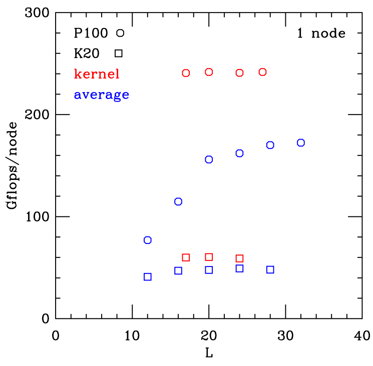
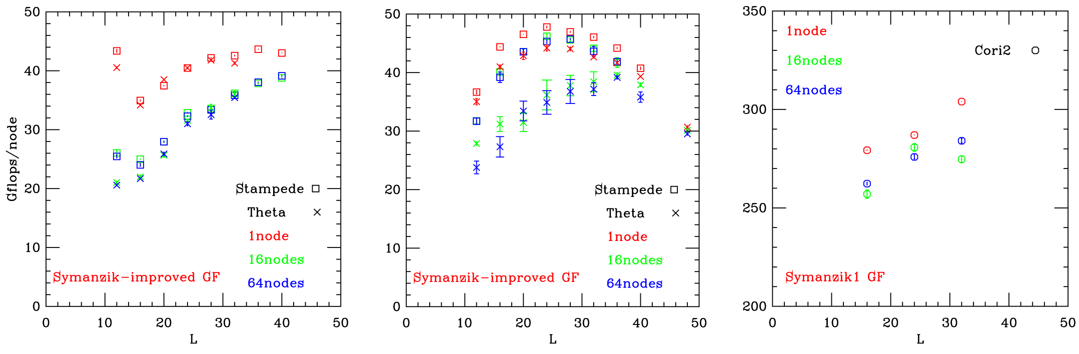


Figure 4. Staggered multi-shift CG performance on one GPU node for various lattice volumes L^4. The performance excludes routine overhead, e.g., the data reconstruction and transfer between the host and GPU device.

bound than the CG.

Figure 5 shows the gauge-force weak-scaling performance on up to 64 KNL nodes, *vs*. lattice size L. The QOPQDP code was compiled to use SSE2 instructions. Baseline MILC and QOPQDP have similar performance; however, there is over five times improvement with QPhiX. Using Intel VTune Amplifier to analyze the performance, we observed a higher cache reuse in the QPhiX algorithm. The QOPQDP algorithm also requires fewer flops than baseline MILC by virtue of reusing three-link staples. Its time was reduced to around one third of the baseline time. QPhiX also reuses three-link staples and its flop count is reduce by 60% compared to baseline MILC. Thus, the QPhiX timing in total was reduced to less than 6% of that of baseline MILC, excluding time for data remapping. While data remapping currently takes approximately the same amount of time as the routine itself, we expect to reduce its impact on the HMC performance after all parts of the algorithm are included in the library, as remapping will not be needed every time the routine is called.

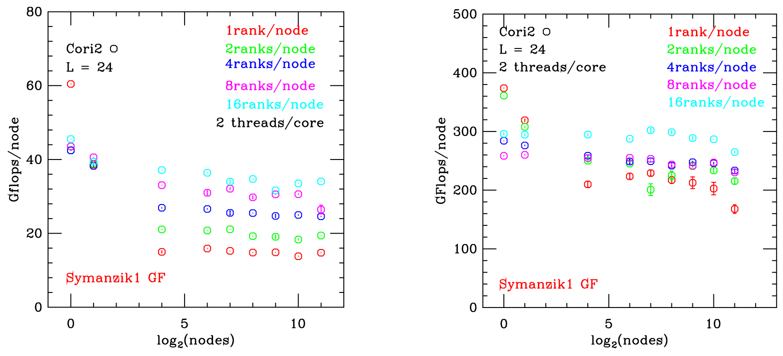


(a) Baseline code on two KNL clusters, with 64 ranks per node.

(b) Code with QOPQDP on two KNL clusters, with 64 ranks per node.

(c) Code with QPhiX on Cori II cluster. One node runs are with OpenMP and 64 threads, and multiple-node runs are with 16 ranks per node.

Figure 5. Weak scaling of Symanzik one-loop gauge force performance on KNL clusters up to 64 nodes, with baseline code (left), QOPQDP (center), and QPhiX (right). The horizontal axis is the lattice size per node, and vertical axis is the flop rate per node. Each run is with one thread per core.



(a) Baseline code

(b) Code with QPhiX

Figure 6. Weak scaling study of Symanzik one-loop gauge force performance on the Cori II cluster with up to 2048 nodes, showing baseline code (left) and QPhiX (right). The vertical axis is the flop rate per node. The lattice size is 24^4 per node. The runs use two threads per core, and various MPI/OpenMP combinations of up to 16 ranks per node.

Weak scaling of the gauge force on Cori II is shown in figure 6. The scaling efficiency is higher compared with CG: over 40% with baseline code and more with QPhiX. Also, hyper-threading with two threads per core helps improve the performance. The QPhiX improvement in performance here is dramatic compared with that of the CG solver. As mentioned, this happens in part because the arithmetic intensity of the gauge-force algorithm is 1.1 in the baseline code and over 2.0 in the QPhiX algorithm, whereas with the CG, it is 0.26 for both baseline and QPhiX codes. We also rerranged the algorithm to avoid frequent communication, thus leading to a higher weak scaling efficiency. On the other hand, the performance of QPhiX at different MPI/OpenMP combinations varies more than the CG performance does.

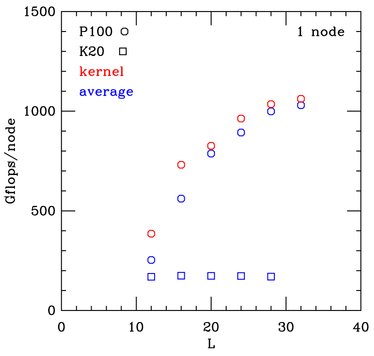


Figure 7. Symanzik one-loop gauge force performance on one GPU node and various lattice sizes L. The performance excludes data remapping overhead.

The gauge force performance on one GPU node is shown in figure 7, on various lattice sizes. Comparing P100 and K20, the performance ratio on these two architectures is again up to 5. On the other hand, the performance on one P100 is around 2.3 times that on one KNL, which is slightly higher than the ratio of CG on these two architectures. The algorithm in QUDA is the same as in the baseline code.

The HISQ fermion force routine was also benchmarked, comparing the performance of the baseline code and the QOPQDP library. We observed similar flop rates again for these two versions of the code, for which single node runs were around 40–50 Gflops/sec with the baseline code and 30–40 Gflops/sec with QOPQDP. The weak-scaling performance efficiency is 50–80% on up to 64 KNL nodes. Because of the reduced amount of computation in the QOPQDP routine, its timing was reduced to around 20% of that in the baseline code.

5 Conclusions

We explored performance of three major lattice QCD routines in the MILC code: the staggered multishift CG, Symanzik one-loop gauge-force, and the HISQ fermion force. We ran our benchmarks on KNL clusters located at three scientific supercomputer centers: ALCF, NERSC, and TACC. We compared the staggered multishift CG and Symanzik gauge force performance from optimizations with QPhiX, with that of the baseline MILC code and the QOPQDP library. We found the CG performance with QPhiX was improved by 1.5 times on one KNL, while the algorithm was network bandwidth bound on clusters. On the other hand, RHMD routines such as the gauge force and fermion force showed a weak scaling efficiency of up to over 80%. Comparing the performance improvement of these routines, we argued a higher arithmetic intensity of the gauge force calculations in QPhiX contributed to a higher performance improvement (over ten times) of this routine. As a comparison, we also showed the code performance on single GPU K20 and P100 nodes with the QUDA 0.8.0 library. In general, the optimized routine performance of a single P100 is about five times of that of a K20. We are developing QPhiX versions of other parts of the MILC code, including the HISQ fermion force and the HISQ link smearing, to add to the library.

References

[1] J. Jeffers, *et. al.*, Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition, Morgan Kaufmann pp. 581–597 (2016)

[2] S. Gottlieb, *et. al.*, Nucl.Phys.Proc.Suppl. **94**, 841 (2001)

[3] R. Babich, *et. al.*, *Parallelizing the QUDA Library for Multi-GPU Calculations in Lattice Quantum Chromodynamics*, SC'10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, 1710.01000

[4] P. Boyle, *et. al.*, *Grid: A next generation data parallel C++ QCD library*, 1512.03487

[5] *Stampede2 - Texas Advanced Computing Center*, https://www.tacc.utexas.edu/systems/stampede2

[6] *Theta - Argonne Leadership Computing Facility*, https://www.alcf.anl.gov/theta

[7] *Cori Intel Xeon Phi Nodes*, http://www.nersc.gov/users/computational-systems/cori/cori-intel-xeon-phi-nodes