ANL/ALCF/ESP-13/11

Lattice Quantum Chromodynamics (SPI, mapping, site ordering, and QPX in Lattice QCD code on Mira)

ALCF-2 Early Science Program Technical Report

Argonne Leadership Computing Facility

ANL/ALCF/ESP-13/11

Lattice Quantum Chromodynamics (SPI, mapping, site ordering, and QPX in Lattice QCD code on Mira)

ALCF-2 Early Science Program Technical Report

prepared by
Heechang Na and James Osborn
Argonne Leadership Computing Facility, Argonne National Laboratory

May 7, 2013

SPI, mapping, site ordering, and QPX in Lattice QCD code on Mira

Heechang Na and James Osborn
Argonne Leadership Computing Facility, ANL

April 3, 2013

1 Introduction

Lattice QCD is a numerical method to simulate QCD (Quantum Chromodynamics) including non-perturbative effects. Among other methods, lattice QCD is the only successful non-perturbative method that can be systematically improved from first principles. Lattice QCD plays an important role in High Energy Particle Physics (flavor physics, spectroscopy, and beyond the Standard Model physics) and Nuclear Physics (nucleon/nuclear spectrum & structure, quark-gluon plasma, and the QCD equation of state). Moreover, recently there are active developments in applying lattice gauge theory to quantum field theory in general, for example the theoretical description of graphene or cold atom systems.

We define the QCD action on a finite and discrete 'lattice' like grid, so that we can compute the path integral (partition function) on a computer. A current lattice size is $96^3 \times 192$, and the corresponding number of degrees of freedom for the path integral is about 6×10^8. Thus, it requires enormous computing resources. In this report, we describe our overall efforts to improve lattice QCD software on Mira, which include a new communication library using the Message Unit (MU) SPI (System Programming Interface), better node mapping strategy, lattice site reordering, and using the quad floating point (QPX) intrinsics.

2 Message Unit (MU) SPI

We apply a Hybrid Monte Carlo method to compute the QCD path integral. At every step in the Monte Carlo evolution, we must solve a large sparse linear system. The application of the sparse matrix requires communications at the boundary to the nearest neighbors, and all nodes need to communicate at the same time with the same manner. Because the problem size per node is relatively small, the latency is more important than it typically

is for other applications. We found that building a new communication library using the MU SPI would improve the communication time.

The MU SPI [1] is the lowest communication layer; it is lower than MPI or PAMI. It allows one to control the MU hardware directly. The MU SPI provides point-to-point and collective communications with memory fifo, direct put, and remote get methods for each. For our 'qspi' communication library, we utilize the point-to-point communications with the direct put.

A node on Mira has 16+1 groups with 4 subgroups per group. Since the seventeenth group is not used by a normal user, there are 64 useable subgroups per node. Each subgroup has 8 injection FIFOs and 8 reception FIFOs. In other words, there are 32 injection / reception FIFOs per group or 512 injection / reception FIFOs per node.

Mira has a 5D torus network, so that each node can access 10 optical cables with 1.8 GB/sec useable bandwidth. Therefore, 10 FIFOs per rank would be optimal in order to use all cables simultaneously. However, for c64 mode, one rank can have a maxim of 8 FIFOs. Furthermore, if one needs to use MPI with the MU SPI, one should reserve at least 1 FIFO for MPI. In this case, one should consider about the optimal running mode and optimal distribution of the resources.

The qspi communication library API consists of:

- `void qspi_init(void);`

  Allocate FIFOs and base address tables.

- `void qspi_set_send(int dest, void *buf, size_t size, qspi_msg_t send_msg);`

- `void qspi_set_recv(int src, void *buf, size_t size, qspi_msg_t recv_msg);`

  `qspi_set_send` and `qspi_set_recv` prepare the handle variables declared as `qspi_msg_t` type. `dest` and `src` are rank of the destination and source node, `*buf` is the address of the send or receive buffer, and `size` is the size of the messages.

- `void qspi_prepare(qspi_msg_t msgs[], int num);`

  Exchange the handles between senders and receivers using MPI, and sets the descriptors.

- `void qspi_start(qspi_msg_t msg);`

  Inject the descriptors.

- `void qspi_wait(qspi_msg_t msg);`

The process will wait until the receiver counter reaches zero.

- `void qspi_finalize(void);`

As one might notice, `MPI_Init()` should be called after `qspi_init()` for `qspi_prepare`. One important communication pattern in lattice QCD is repeatedly sending and receiving with the same data structure. Therefore, once the communication is prepared, we can use it over and over again. Using MPI for `qspi_prepare` does not affect the overall communication time. We also note that there is no global barrier in qspi, so one would use `MPI_Barrier`.

We measure the communication time with qspi and MPI with several different settings. First, we had a ping-pong test between a set of two nearest neighbors in c1 mode. The results are shown in Fig. 1. As one can see, there is a good speedup for the latency: about 0.6 micro-seconds latency for qspi while MPI gives about 3 micro-seconds latency. As the data size increases the communication time is saturated to the bandwidth limit, so there is almost no speedup for message sizes larger than around 100 KB.

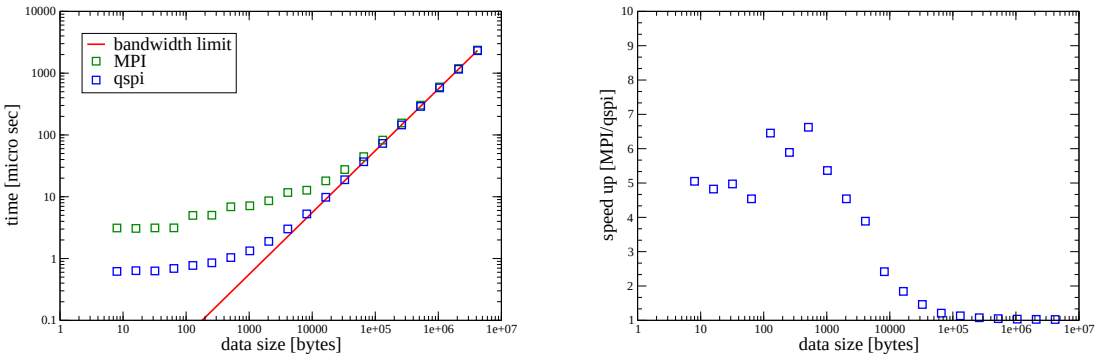

Figure 1: Single ping-pong benchmarks between qspi and MPI with c1 mode. The communication time vs. data size (left figure) and speedup comparing to MPI (right figure) are shown. The red line on the left figure represents the 1.8 GB/sec bandwidth limit.

Next, we tested with a more realistic situation that all ranks are sending and receiving messages to all 5 directions (5D halo exchange test) at the same time. The results are shown in Fig. 2. Latency is about 2.3 micro-seconds for qspi and 19 micro-seconds for MPI. The latency slows even for qspi, since there are much more communications needed, however the speedup comparing to MPI is much larger than for the single ping-pong tests. The speedup is around 9 to 20 times faster, while it was around 6 times faster for the single ping-pong tests.

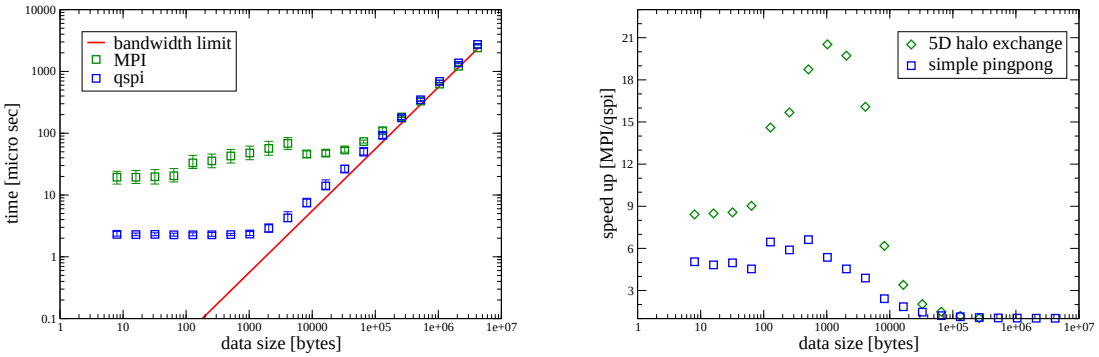

Figure 2: 5D halo exchange benchmarks between qspi and MPI with c1 mode. The communication time vs. data size (left figure) and speedup comparing to MPI (right figure) are shown. The red line on the left figure represents the 1.8 GB/sec bandwidth limit.

We ran the 5D halo exchange tests with c64 mode as well, and the results are shown in Fig. 3. In this case, one interesting question would be what an optimal number of FIFOs is. For the tests in Fig. 3, we assigned 4 FIFOs for each qspi process. We tested with 2 or 6 FIFOs as well, but we did not find any significant difference between the tests. This is because that the competition to get on the optical cable is higher than the competition to get a free FIFO. In c64 mode, there are 64 processes in a node trying to access 10 optical cables.

3 Mapping

Reducing the surface volume at the boundaries and the number of communication hops for each message are desirable, and one can get improvement with an optimal node mapping strategy. An example is shown in Fig. 4. In the figure, each orange dot represents a rank, and each blue box corresponds to a node. In the example, we assumed that we have 4 ranks per node. Lattice sites are allocated to each ranks, so that each rank needs to communicate to the nearest neighbors. As one can see, it has a smaller surface volume and smaller number of external hops with the mapping shown in the right figure of Fig. 4.

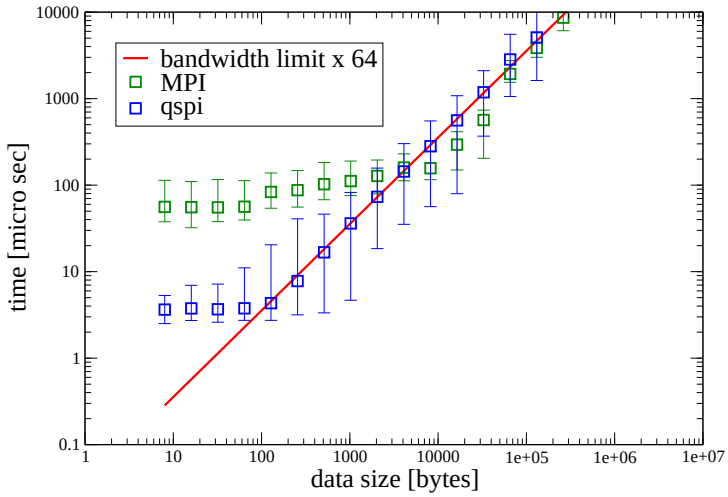

Figure 3: 5D halo exchange benchmarks between qspi and MPI with c64 mode. The communication time vs. data size (left figure) and speedup comparing to MPI (right figure) are shown. The red line represents the 1.8 GB/sec bandwidth limit divided over the 64 ranks per node.

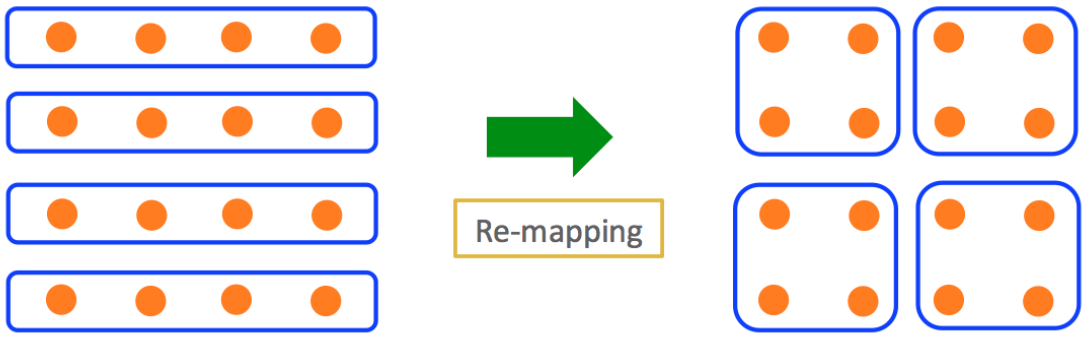

Figure 4: An example of a node mapping strategy in a 2D array with 4 ranks per node.

4 Site ordering

Lattice QCD calculations are typically based on a 4 dimensional lattice, and proper site ordering within a node can reduce the number of memory fragmentations in communications. An example of a 2D lattice site ordering strategy is shown in Fig. 5. The numbers in the figure indicate the order of data in memory. Normal site ordering is shown on the left figure. As one can see, the chunk of data at the boundary (inside of the blue rectangles) to be sent to left or right are not contiguous. However, with a better site ordering as on the right figure in Fig. 5, one can make all messages contiguous except the message to be sent to left.

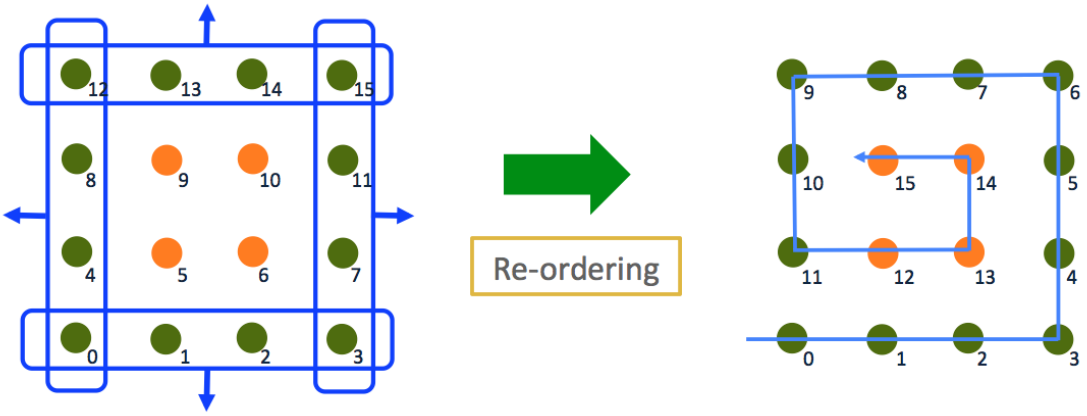

Figure 5: An example of site ordering strategy on a 2D lattice.

5 QPX in QLA

We added a few QPX routines using XLC intrinsics in QLA. QLA is a linear algebra library in the USQCD SciDAC modules [2]. One of most important calculations in lattice QCD is to multiply an array of 3×3 complex matrixes by an array of pointers to length 3 vectors, for example with an array size of N:

```
for i=0 to N-1
  r[i] += M[i] * (*v[i])
```

where `r[i]` and `*v[i]` are the i-th vectors, and `M[i]` is the i-th 3×3 matrix. We combine two matrix-vector multiplications in order to preserve alignment of the arrays when loading them into QPX registers. Since the subelements are complex numbers, when the two subelements are coupled as in Fig. 6, each bunch will fit into a QPX register. This

requires only 9 QPX instructions per array element, which should be 4 times faster ideally than without QPX.

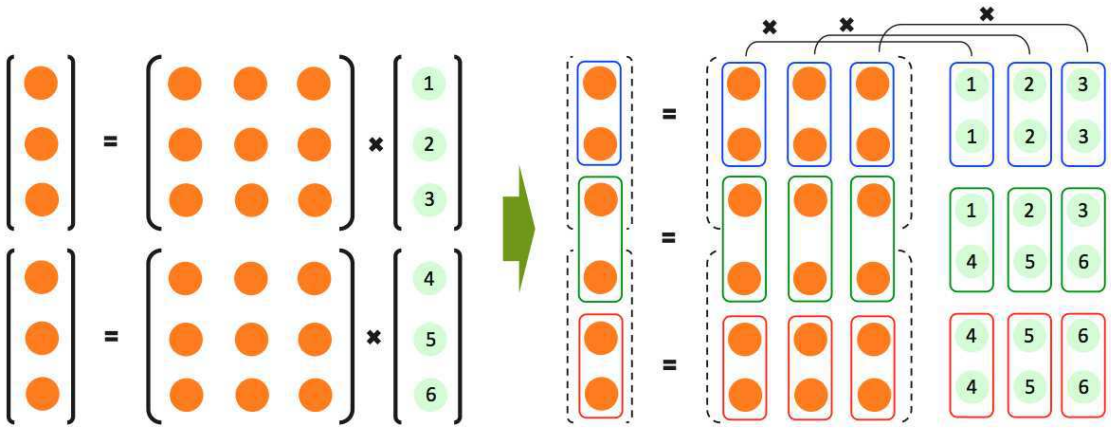

Figure 6: 3×3 matrix multiply by length 3 vector in QPX. All elements are complex numbers. In the right figure, each block with two complex numbers fits into a QPX register.

6 Overall performance and summary

We tested actual lattice QCD code with the improvements. We used the HISQ solver in the USQCD SciDAC modules [2] with a 12^4 lattice size per node on 128 BG/Q nodes. The HISQ solver is developed by the MILC collaboration [3]. The results are shown in Tab. 1. In these tests, we only use the site ordering with the qspi code. The first line of the table shows the result without any improvements. The next three lines represent results with only one improvement applied each. As one can see, all the improvements, QPX, qspi, and mapping, are equally improving the performance. One might notice that even we turned on the two of the improvements, the speedup is not so impressive (the next three lines). However, when we turn on all the three improvements with the site ordering, we obtain 2.3 times speedup as the last line of the table shows.

In this report, we presented several improvements in lattice QCD code on Mira and its impact. We developed a new communication library (qspi) using the MU SPI. With qspi, we got up to around 20 times faster latency than that with normal MPI in our 5D halo exchange tests. The optimal node mapping strategy helps to reduce the surface volume at the boundaries and the number of communication hops. In addition, we reduce the number of memory fragmentations in the communications by applying the optimal site ordering. Moreover, from using QPX in QLA, we gain significant speedup. With all the

Table 1: Overall benchmark results.

QPX	qspi	Mapping	mode	Gflops/node
			c32	11.7
		√	c64	12.9
	√		c32	13.8
√			c32	13.5
√	√		c32	17.6
√		√	c32	16.7
	√	√	c64	18.1
√	√	√	c32	26.7

improvements, we acquire 2.3 times speedup comparing to that without any improvements on BG/Q.

References

[1] Header files in /bgsys/drivers/ppcfloor/spi/include/mu and a html document /bgsys/drivers/ppcfloor/spi/doc/html/index.html in the BG/Q systems in ALCF.

[2] http://usqcd.jlab.org/usqcd-software/

[3] http://physics.indiana.edu/∼sg/milc.html

Argonne Leadership Computing Facility
Argonne National Laboratory
9700 South Cass Avenue, Bldg. 240
Argonne, IL 60439

www.anl.gov